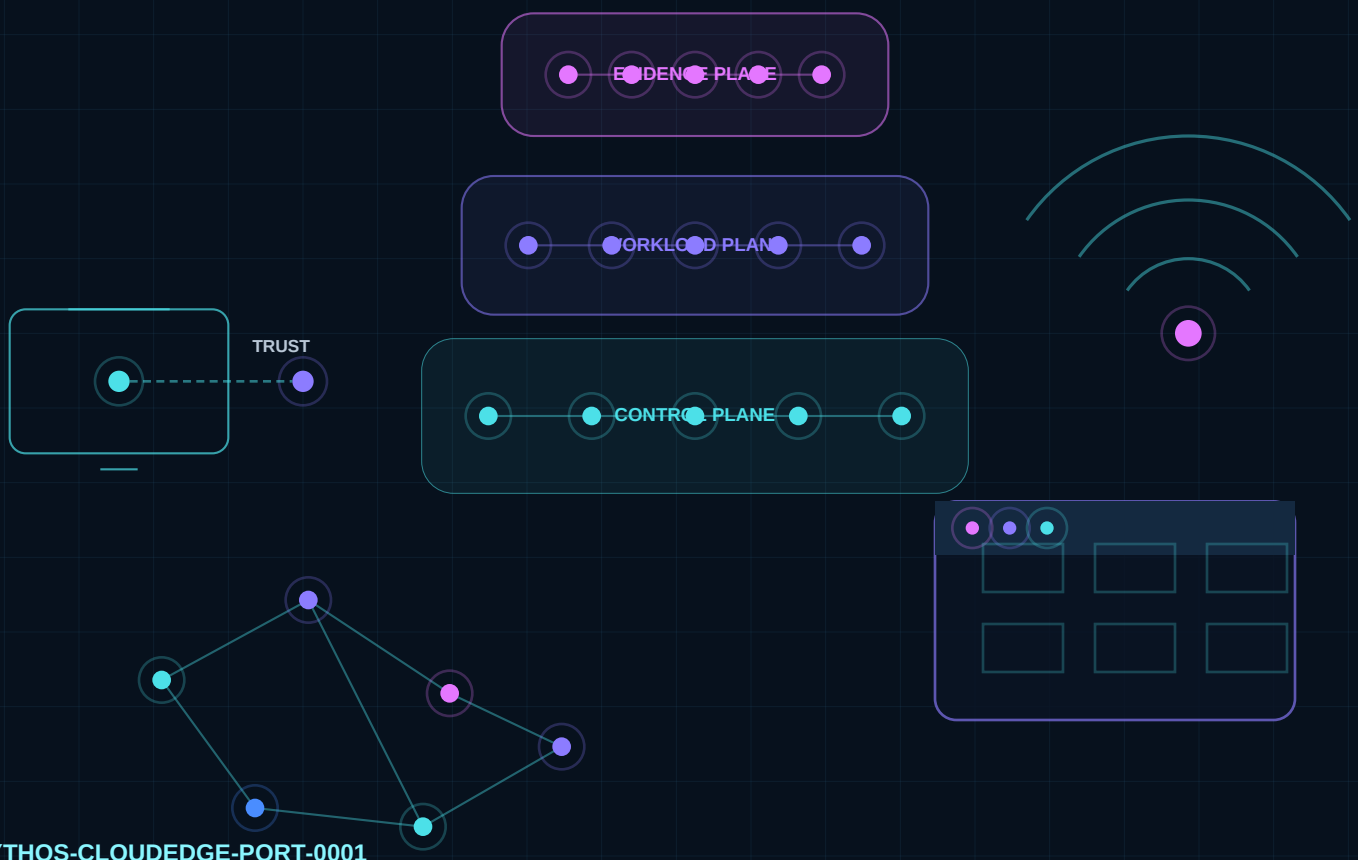
MYTHOS SYSTEMS

ENGINEERING STANDARDS LIBRARY

Cloud, Endpoint, OT, Telecom, Hardware, IoT, Media, and Web Library Portfolio

Permanent portfolio edition connecting cloud planes, endpoint trust, industrial zones, radio layers, compute fabrics, sensor graphs, realtime media paths, and browser/runtime isolation.



MYTHOS-CLOUDEDGE-PORT-0001

PERMANENT PUBLICATION IDENTITY / CANDIDATE PUBLICATION / PUBLIC

Library identity and scope

PERMANENT ID

MYTHOS-CLOUDEDGE-PORT-0001

STATUS

Candidate Portfolio Edition

NORMATIVE STANDARDS

21

ATOMIC CRITERIA

378

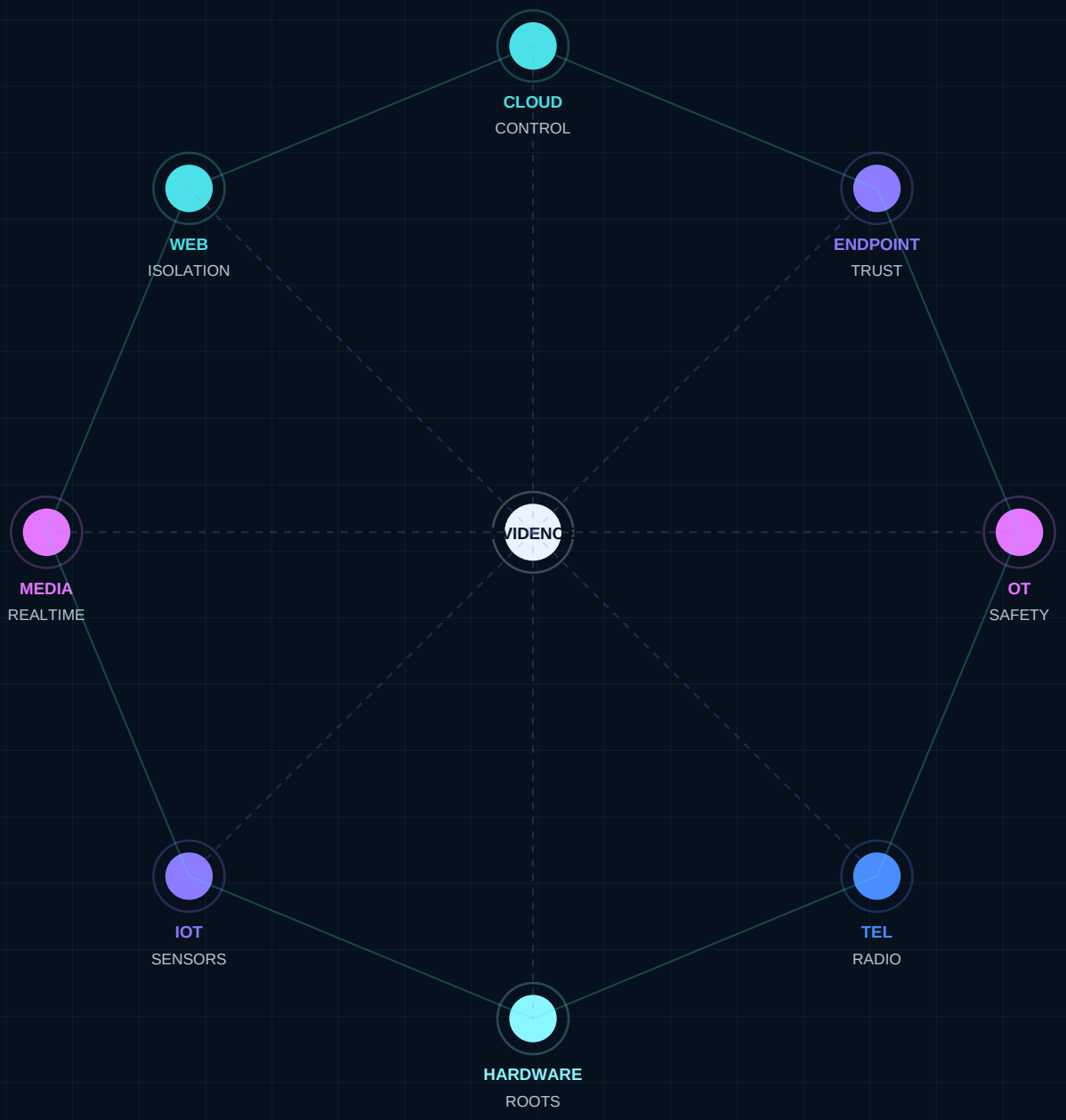
SERIES

CLD / END / OT / TEL / HW / IOT / MED / WEB

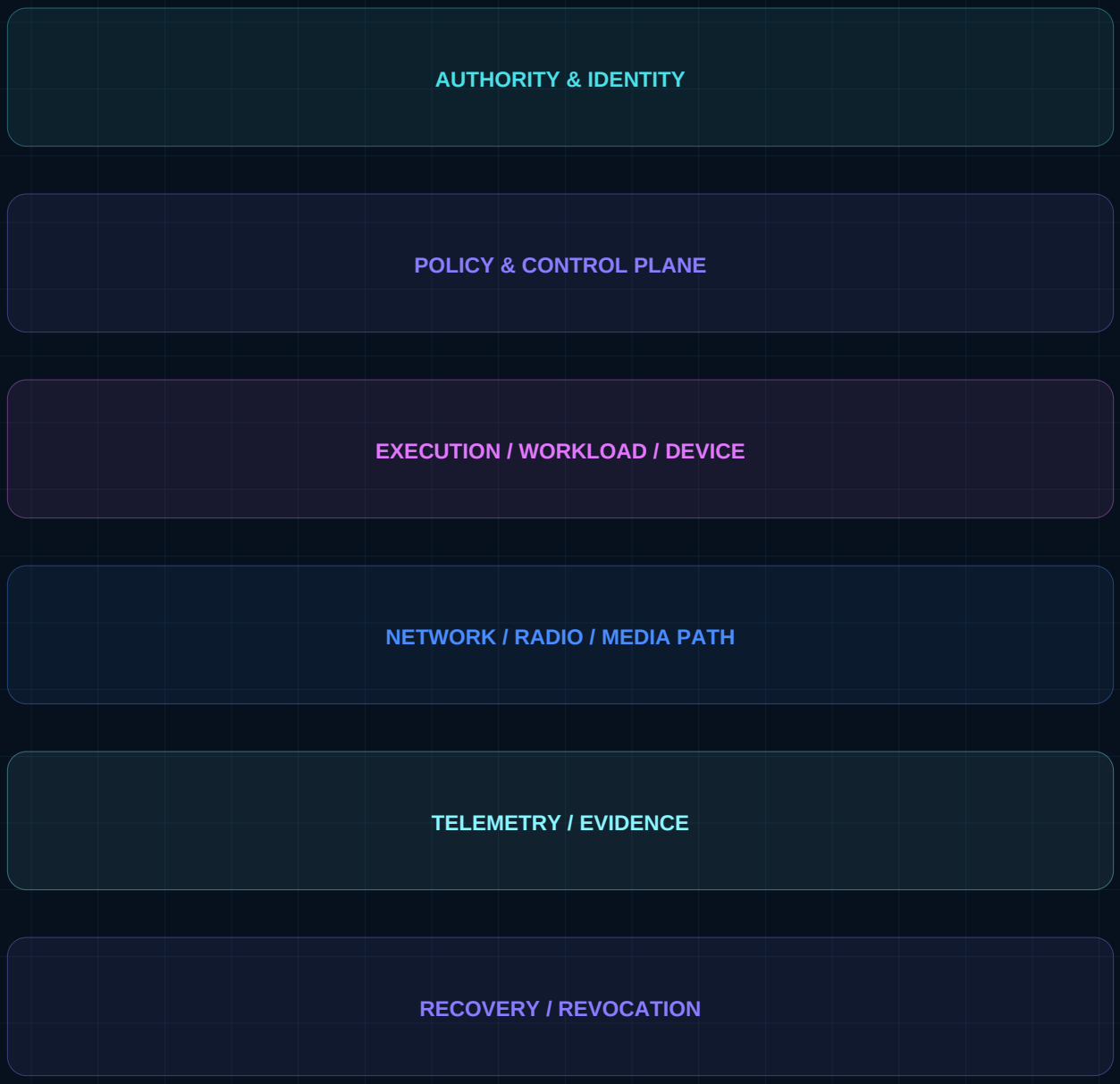
ASSURANCE

Foundation / Advanced / High-Assurance

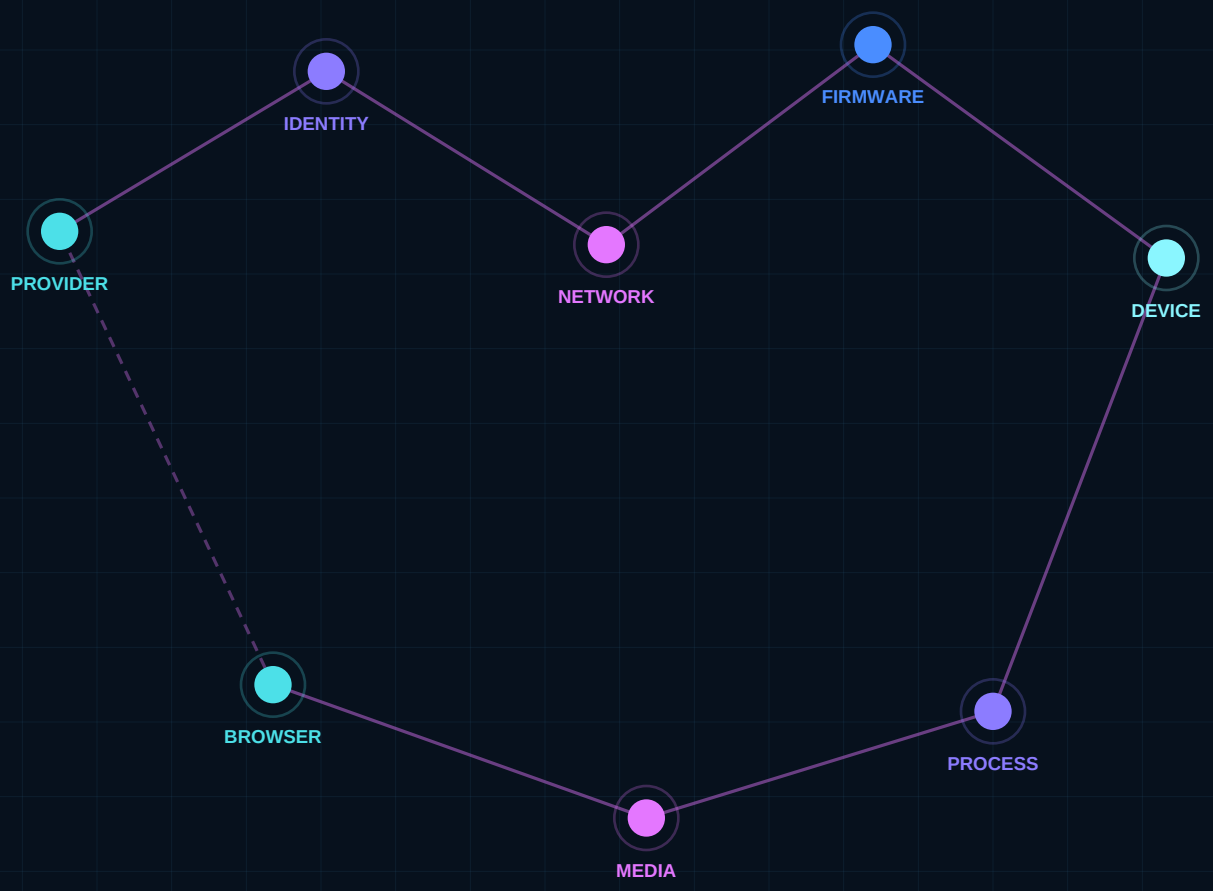
Infrastructure is a connected state machine



Separate authority, execution, observation, and recovery



Model failure as propagation, not isolated alerts



A reliable control system identifies the first divergence, contains blast radius, preserves evidence, restores service or safety, and proves the resulting state.

How to use the library

01

EXECUTIVE

Portfolio posture, scope, risk, and adoption gates.

02

STANDARDS

Normative requirements, evidence, assessment, profiles, and lifecycle.

03

GUIDES

Practical implementation and operational integration.

04

FIELD GUIDES

Focused cloud/platform teaching and implementation detail.

05

RESEARCH

Current evidence, uncertainty, readiness, and adoption posture.

06

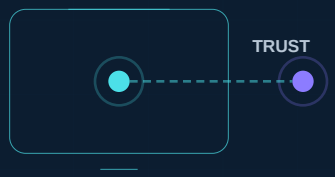
ARCHITECTURES

Deployable boundaries, flows, trust, and recovery patterns.

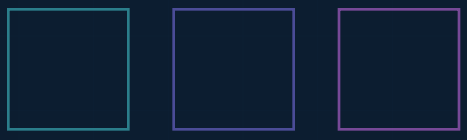
Eight engineering views

CLOUD PLANES

ENDPOINT TRUST



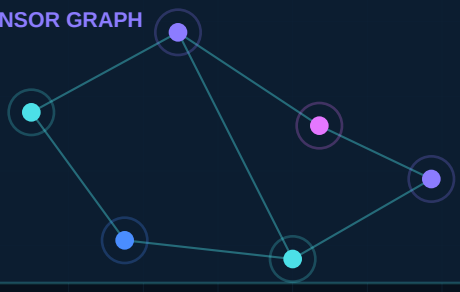
OT ZONES



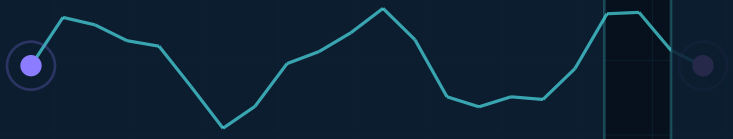
RADIO LAYERS

COMPUTE FABRIC

SENSOR GRAPH



MEDIA PATH



BROWSER SANDBOX

Volatile ecosystems require event-triggered review

KUBERNETES

1.37 current minor / 1.37-1.35 active branches

3GPP

Release 21 open / Release 20 open / Release 19 frozen

MATTER

1.6 released June 2026

WEBGPU

Candidate Recommendation Draft, August 2026

NIST OT

SP 800-82 Rev. 3 final; Rev. 4 remains draft

ZERO TRUST

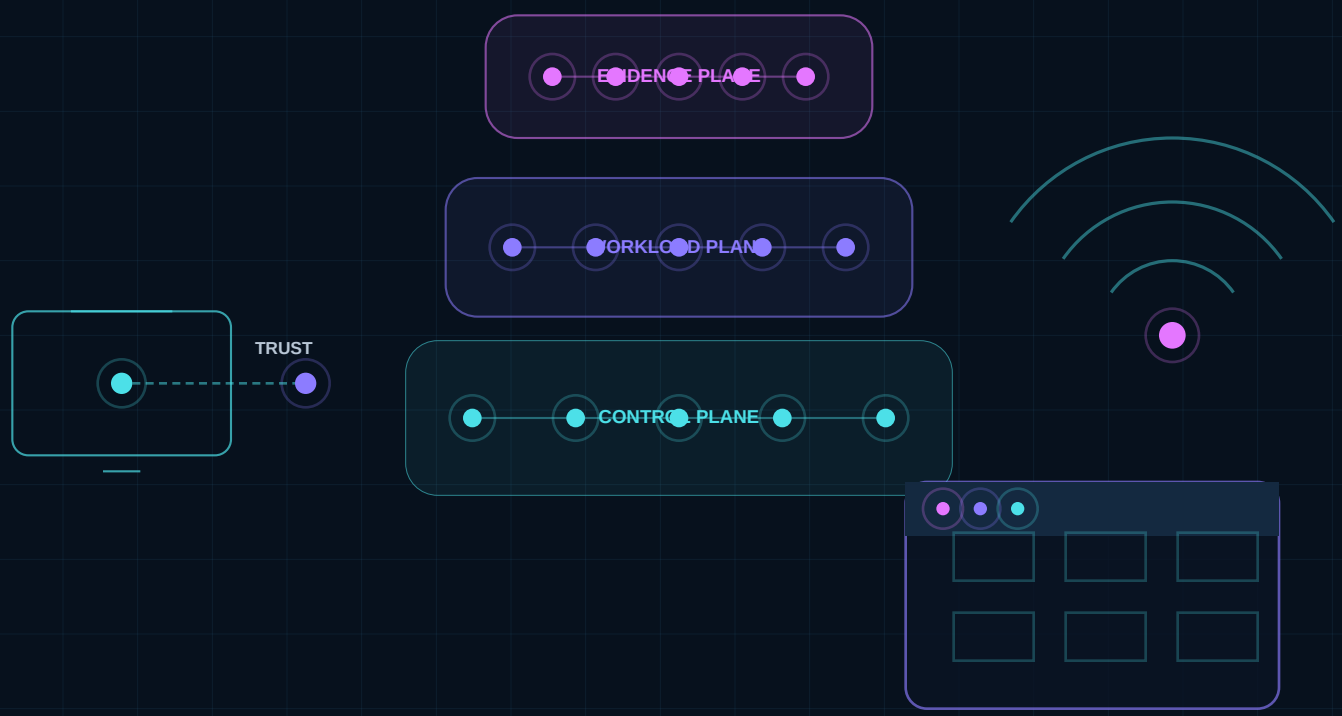
NIST SP 800-207 and 800-207A final

MYTHOS SYSTEMS

EXECUTIVE PORTFOLIO

Cloud, Edge, Hardware, Media, and Web Executive Portfolio

Leadership view of 21 permanent infrastructure standards spanning cloud control planes, managed endpoints, OT, telecommunications, trusted compute, IoT, realtime media, and browser isolation.



MYTHOS-CLOUDEDGE-EXEC-0001

PERMANENT PUBLICATION IDENTITY / CANDIDATE PUBLICATION / PUBLIC

Infrastructure assurance as one system

STANDARDS
21

ATOMIC CRITERIA
378

DOMAINS
8

GUIDES
3

FIELD GUIDES
8

RESEARCH
7

ARCHITECTURES
7

EVIDENCE MODEL
FOUNDATION

Eight domains, one control fabric

CLD	Cloud planes & workload placement
END	Endpoint trust & fleet state
OT	Zones, conduits & safety
TEL	RAN, RF, voice & timing
HW	Compute fabrics & firmware trust
IOT	Sensor identity & lifecycle
MED	Realtime media & provenance
WEB	Browser/runtime isolation

Policy gates every transition

Cloud, endpoint, OT, telecom, hardware, IoT, media, and browser systems are not separate risk islands. Identity, policy, state, telemetry, recovery, and evidence cross every boundary.

Failure propagation leaders must see

AUTHORITY DRIFT

Provider or device state diverges from approved policy.

HIDDEN DEPENDENCY

Cloud, firmware, carrier, browser, or supply-chain dependency fails outside local control.

UNVERIFIED RECOVERY

Failover exists on paper but restore, rollback, or degraded-mode behavior is untested.

TELEMETRY BLINDNESS

Observed health does not cover actual user, process, safety, or media outcome.

TRUST COLLAPSE

Identity, key, attestation, firmware, or update trust is compromised at scale.

Sequence the program around evidence

1

Inventory & classify

Assets, services, identities, zones, dependencies, versions, and owners.

2

Establish control planes

Policy, access, configuration, lifecycle, and source-of-truth boundaries.

3

Instrument reality

Telemetry across workload, device, process, RF, media, hardware, and browser state.

4

Prove failure behavior

Negative tests, overload, partition, rollback, restore, isolation, and safety cases.

5

Reconcile & govern

Compare intended, deployed, observed, historical, and exceptional states continuously.

Questions that gate material deployment

Q01

Can we name the authoritative control plane?

Q02

Can we prove identity and policy at the enforcement point?

Q03

Can we see degraded and unsafe states before users do?

Q04

Can we revoke, isolate, rollback, restore, and reconstruct?

Q05

Can we prove what happened from preserved evidence?

Q06

Can we exit the provider, platform, device, or protocol without losing control?

Candidate standards with bounded claims

This portfolio is a permanent Mythos library view. It does not replace the independently citable standards, guides, research reports, or reference architectures. Candidate status does not imply certification, regulator approval, or external validation.

PERMANENT IDENTITY

STABLE

Revisions retain the same public document identities unless scope changes create a genuinely new publication.

EVIDENCE POSTURE

REQUIRED

Claims are bounded by scope, version, environment, source currency, test period, and assessor confidence.

PRODUCTION METADATA

PRIVATE

Internal package or production sequencing never appears as public publication identity.

REVIEW TRIGGER

EVENT + CADENCE

Provider, protocol, firmware, browser, radio, or authority changes may trigger review before the scheduled date.

ENGINEERING STANDARDS LIBRARY / CLOUD ARCHITECTURE AND COMPUTE

Multi-Cloud Architecture and Control Plane Standard

Cross-provider portfolio governance, landing zones, identity, policy, sovereignty, resilience, cost, evidence, and exit.

PERMANENT DOCUMENT ID

MYTHOS-CLD-0001

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

Multi-Cloud Architecture and Control Plane Standard

DOCUMENT ID
MYTHOS-CLD-0001

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

18	6	4	1
ATOMIC CRITERIA	ASSURANCE VIEWS	IMPLEMENTATION PROFILES	PERMANENT IDENTITY

Publication doctrine. This publication establishes durable requirements for its defined scope. Interpretation must preserve the permanent document identity, required evidence, controlled exceptions, source currency, and formal revision history.

INFRASTRUCTURE AND CONTROL TOPOLOGY

Cloud Architecture and Compute

A trustworthy multi-cloud architecture and control plane system is a governed chain of ownership, identity, desired state, enforcement, workload or device behavior, telemetry, recovery, and independently reviewable evidence.

OWNERSHIP

Purpose, service boundary, accountable owner, suppliers, users, and retained responsibility remain explicit.

ENFORCEMENT

Identity, policy, segmentation, configuration, and local safety mechanisms constrain every trust transition.

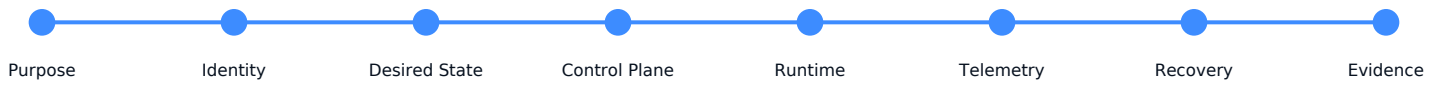
CONTINUITY

Capacity, dependency loss, safe degradation, backup, restoration, rollback, and exit are tested before reliance.

EVIDENCE

Inventory, desired state, runtime observation, tests, incidents, exceptions, and change records form the assurance chain.

GOVERNED INFRASTRUCTURE FLOW



Reconciliation, detection, response, restoration, and controlled evolution

INTERPRETATION AND ASSURANCE

How to apply this publication

Normative intent

Requirements define outcomes and constraints. Implementations may vary, but evidence must demonstrate equivalent control.

Evidence over assertion

Vendor documentation and design intent are not equivalent to reproduced behavior. Record evidence grade and uncertainty.

Risk-proportional depth

Higher consequence, autonomy, sensitivity, and irreversibility require stronger isolation, review, verification, and recovery.

Controlled exception

Exceptions require an owner, rationale, compensating control, residual-risk acceptance, expiration, and reevaluation trigger.

Normalized assessment scale

0	1	2	3	4	5
ABSENT	AD HOC	PARTIAL	OPERATIONAL	ADVANCED	LEADING

A numeric score must never hide a failed mandatory gate, missing evidence, untested workload, or unacceptable residual risk.

INTERACTIVE NAVIGATION

Contents

Navigation doctrine

Bookmarks and internal links are conveniences. Permanent document identity, criterion identifiers, evidence references, and revision history remain authoritative.

MYTHOS-CLD-0001 / CLOUD ARCHITECTURE

0. Executive Mandate

Purpose, strategic outcome, and non-negotiable operating doctrine

MANDATE

Organizations adopting multi-cloud architecture and control plane capabilities MUST define an owned service boundary, establish enforceable identity and configuration, protect data and safety, test failure and recovery, and preserve evidence sufficient for independent review.

Required outcomes

- Create a durable governance boundary for multi-cloud architecture and control plane across design, acquisition, deployment, operation, change, and retirement.
- Require risk-proportional segmentation, identity, configuration, telemetry, resilience, recovery, and supplier controls.
- Prevent central automation, administrative tooling, or provider services from becoming unbounded authority.
- Prove operation with representative workloads, devices, failure modes, and restoration exercises.
- Define measurable conformance, exceptions, reassessment triggers, and a viable exit path.

Decision boundary: conformance supports a documented assurance claim for the named scope. It does not prove universal availability, safety, regulatory acceptance, vendor fitness, or immunity from cyber-physical failure.

MYTHOS-CLD-0001 / CLOUD ARCHITECTURE

1. Scope and Applicability

Boundary definition, audience, exclusions, and triggering conditions

IN SCOPE

- Architecture, acquisition, configuration, integration, and operation of multi-cloud architecture and control plane capabilities.
- Human, workload, device, service, network, data, facility, provider, supplier, and evidence dependencies.
- Design, deployment, monitoring, support, incident response, recovery, migration, and retirement.
- On-premises, hosted, managed, carrier, manufacturer, integrator, and externally operated components.

OUT OF SCOPE

- Claims of legal, safety, carrier, or regulatory approval without the responsible authority.
- Vendor certification treated as proof of the adopter configuration or operating effectiveness.
- Theoretical or laboratory behavior presented as production evidence without a declared boundary.
- Inherited controls without documented scope, owner, period, limitations, and shared responsibility.

Applicability triggers

- The capability supports a business-critical, safety-relevant, regulated, public, or externally dependent service.
- The environment can expose sensitive data, identity, control, physical process, communications, or shared infrastructure.
- A management plane, automation system, carrier, provider, or supplier can affect broad operational scope.
- Failure, compromise, or change can be difficult to detect, reverse, isolate, or recover.
- The system depends on hardware, firmware, radio, browser, cloud, endpoint, IoT, OT, or real-time media behavior.

MYTHOS-CLD-0001 / CLOUD ARCHITECTURE

2. Definitions and Taxonomy

Controlled language for architecture, governance, and assessment

AUTHORITY

The right to approve, deny, constrain, or execute an action. Model output is not authority.

EVIDENCE

A reproducible source, trace, measurement, test, record, signed artifact, or documented observation supporting a claim.

ASSURANCE VIEW

A defined perspective such as governance, architecture, security, operations, privacy, or human oversight.

ATOMIC CRITERION

A uniquely identified requirement that can be assessed, evidenced, excepted, and revised independently.

IMPLEMENTATION PROFILE

A risk-proportional set of required depth, evidence, and gates for a class of use.

CONTROLLED EXCEPTION

A time-bounded deviation with owner, rationale, compensating control, residual risk, expiration, and review trigger.

DETERMINISTIC MECHANISM

A component whose inputs, policy, state transition, and side effects can be constrained and verified.

SOURCE AUTHORITY

The documented basis for treating a source as reliable for a defined claim, context, jurisdiction, and time.

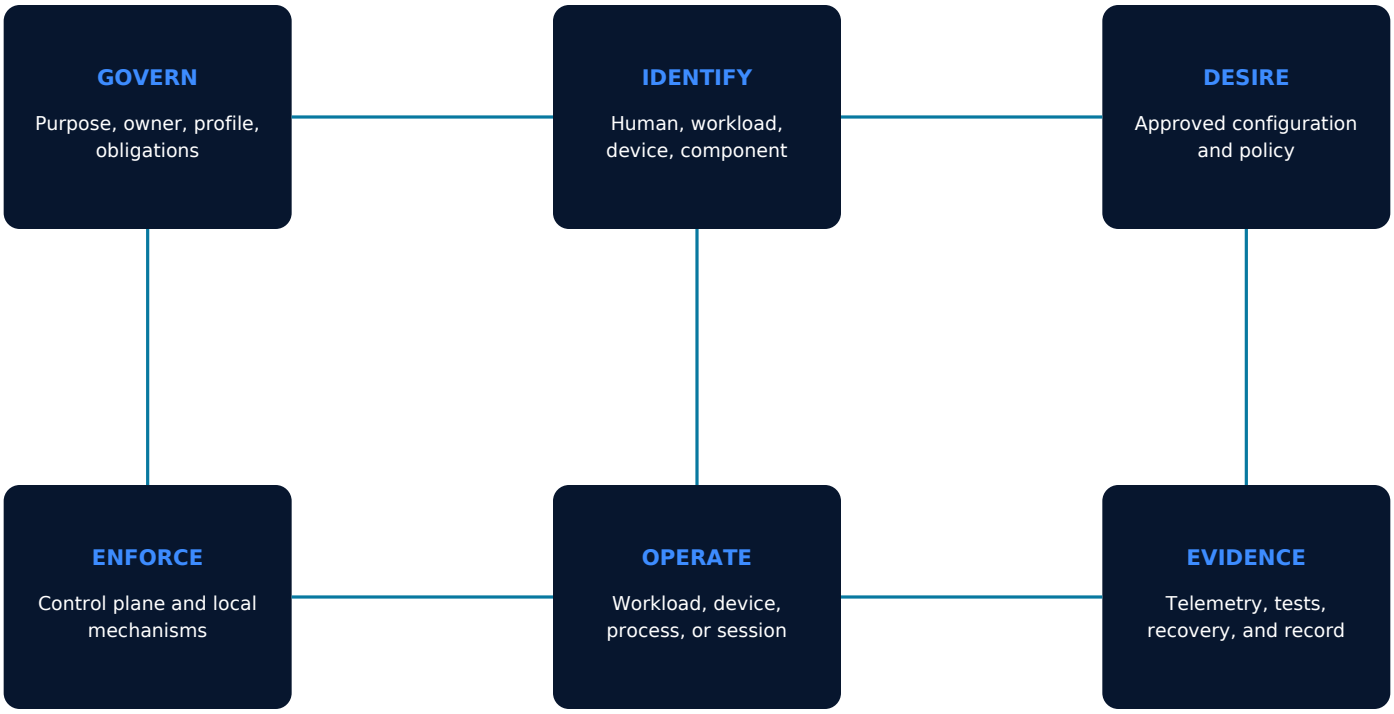
REASSESSMENT TRIGGER

A change, incident, drift signal, dependency revision, or time boundary requiring renewed evaluation.

MYTHOS-CLD-0001 / CLOUD ARCHITECTURE

3. Architecture and Trust Model

Separation of governance, management, enforcement, runtime, telemetry, and recovery



Mandatory trust boundaries

- Management planes MUST be isolated, strongly authenticated, monitored, and recoverable.
- Identity and policy MUST be evaluated at every provider, network, device, workload, and administrative transition.
- Runtime state MUST be compared with approved desired state and material drift MUST trigger response.
- Safety and continuity mechanisms MUST remain available when cloud, WAN, identity, DNS, time, carrier, or management dependencies fail.
- Evidence MUST be protected from the systems and administrators whose behavior it is intended to assess.

MYTHOS-CLD-0001 / CLOUD ARCHITECTURE

4. Governance and Accountability

Decision rights, separation of duties, and lifecycle ownership

ACCOUNTABLE OWNER

Accepts scope, risk tolerance, funding, and residual risk.

SYSTEM OWNER

Maintains architecture, inventory, operating boundaries, and change control.

SECURITY / PRIVACY

Defines threat, identity, data, isolation, monitoring, and incident requirements.

EVALUATION AUTHORITY

Designs representative tests, gates releases, and preserves reproducibility.

OPERATIONS

Maintains availability, rollback, recovery, evidence, and incident handling.

HUMAN OVERSIGHT

Reviews consequential decisions, exceptions, disputed outcomes, and harm signals.

DECISION PRINCIPLE

No single actor should be able to define the objective, grant authority, execute the consequential action, suppress evidence, and approve the result. Separation must be technical where consequence requires it.

MYTHOS-CLD-0001 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 01-04 of 18

MYTHOS-CLD-0001-C01 Cloud Portfolio Strategy and Service Boundary

The organization MUST define, implement, verify, and maintain cloud portfolio strategy and service boundary for in-scope multi-cloud architecture and control plane capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0001-C02 Provider, Region, and Service Approval

The organization MUST define, implement, verify, and maintain provider, region, and service approval for in-scope multi-cloud architecture and control plane capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0001-C03 Organization, Tenancy, and Account Hierarchy

The organization MUST define, implement, verify, and maintain organization, tenancy, and account hierarchy for in-scope multi-cloud architecture and control plane capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-CLD-0001-C04 Landing Zone and Environment Segmentation

The organization MUST define, implement, verify, and maintain landing zone and environment segmentation for in-scope multi-cloud architecture and control plane capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-CLD-0001 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 05-08 of 18

MYTHOS-CLD-0001-C05 Federated Identity and Privileged Administration

The organization MUST define, implement, verify, and maintain federated identity and privileged administration for in-scope multi-cloud architecture and control plane capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-CLD-0001-C06 Cross-Cloud Network and Egress Architecture

The organization MUST define, implement, verify, and maintain cross-cloud network and egress architecture for in-scope multi-cloud architecture and control plane capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-CLD-0001-C07 Data Classification, Residency, and Sovereignty

The organization MUST define, implement, verify, and maintain data classification, residency, and sovereignty for in-scope multi-cloud architecture and control plane capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-CLD-0001-C08 Encryption, Key Authority, and Secret Protection

The organization MUST define, implement, verify, and maintain encryption, key authority, and secret protection for in-scope multi-cloud architecture and control plane capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-CLD-0001 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 09-12 of 18

MYTHOS-CLD-0001-C09 Policy-as-Code and Preventive Guardrails

The organization MUST define, implement, verify, and maintain policy-as-code and preventive guardrails for in-scope multi-cloud architecture and control plane capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0001-C10 Workload Onboarding and Shared-Service Contract

The organization MUST define, implement, verify, and maintain workload onboarding and shared-service contract for in-scope multi-cloud architecture and control plane capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0001-C11 Telemetry, Inventory, and Evidence Normalization

The organization MUST define, implement, verify, and maintain telemetry, inventory, and evidence normalization for in-scope multi-cloud architecture and control plane capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-CLD-0001-C12 FinOps, Quota, and Capacity Governance

The organization MUST define, implement, verify, and maintain finops, quota, and capacity governance for in-scope multi-cloud architecture and control plane capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-CLD-0001 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 13-15 of 18

MYTHOS-CLD-0001-C13 Failure-Domain and Provider-Outage Strategy

The organization MUST define, implement, verify, and maintain failure-domain and provider-outage strategy for in-scope multi-cloud architecture and control plane capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-CLD-0001-C14 Backup, Restoration, and Cyber-Recovery

The organization MUST define, implement, verify, and maintain backup, restoration, and cyber-recovery for in-scope multi-cloud architecture and control plane capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-CLD-0001-C15 Cloud Incident Response and Escalation

The organization MUST define, implement, verify, and maintain cloud incident response and escalation for in-scope multi-cloud architecture and control plane capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-CLD-0001 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 16-18 of 18

MYTHOS-CLD-0001-C16 Supplier, Marketplace, and Dependency Governance

The organization MUST define, implement, verify, and maintain supplier, marketplace, and dependency governance for in-scope multi-cloud architecture and control plane capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-CLD-0001-C17 Portability, Interoperability, and Exit Testing

The organization MUST define, implement, verify, and maintain portability, interoperability, and exit testing for in-scope multi-cloud architecture and control plane capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0001-C18 Conformance Evidence and Reassessment

The organization MUST define, implement, verify, and maintain conformance evidence and reassessment for in-scope multi-cloud architecture and control plane capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0001 / CLOUD ARCHITECTURE

6. Evidence and Assessment

Examine, interview, test, observe, restore, and trace

EXAMINE

Policies, inventories, diagrams, configuration, code, firmware, contracts, provider evidence, logs, exceptions, and lifecycle records.

INTERVIEW

Accountable owners, architects, engineers, operators, safety personnel, security teams, vendors, carriers, integrators, and responders.

TEST

Identity, segmentation, policy, negative paths, malformed input, capacity stress, dependency loss, failover, rollback, and revocation.

RESTORE

Independent restoration of configuration, data, service, device, controller, cluster, media, or browser state from protected evidence.

OBSERVE

Production-like performance, quality, drift, resource use, physical behavior, alarms, user impact, and incident execution.

TRACE

End-to-end linkage from approved intent and identity through change, runtime behavior, telemetry, outcome, exception, and review.

Evidence grades

A

Independently reproducible, complete, current, integrity-protected, and representative.

B

Reproduced by the implementing organization with strong traceability and controlled scope.

C

Partially reproduced or indirect; limitations, inherited responsibility, and uncertainty recorded.

D

Assertion, diagram, design intent, certificate, or vendor statement without reproduced behavior.

MYTHOS-CLD-0001 / CLOUD ARCHITECTURE

7. Assurance Views

Six perspectives required for a complete assurance claim

GOVERNANCE

Ownership, purpose, risk tolerance, policy, exception, and decision rights.

ARCHITECTURE

Boundaries, dependencies, failure modes, state, data flow, and portability.

SECURITY

Identity, authorization, isolation, secrets, abuse resistance, and incident response.

OPERATIONS

Reliability, performance, cost, observability, change, recovery, and support.

PRIVACY / RIGHTS

Purpose limitation, minimization, consent, access, correction, deletion, and egress.

HUMAN / SOCIETAL

Usability, accessibility, disclosure, contestability, impact, and human control.

Conformance requires a defensible position across every applicable view. A strong aggregate score cannot compensate for an unowned system, a failed mandatory gate, untested recovery, or evidence below the accepted grade.

MYTHOS-CLD-0001 / CLOUD ARCHITECTURE

7.2 Evaluation Metrics and Decision Gates

Measures must expose service behavior, control effectiveness, failure, uncertainty, and cost

SERVICE

Availability, correctness, coverage, quality, latency, throughput, and user or process outcome.

CONTROL

Unauthorized attempt, policy denial, drift, segmentation escape, privilege, and exception exposure.

RESILIENCE

Failover success, recovery time, data loss, safe degradation, restore validity, and dependency survival.

SECURITY

Exploitability, credential exposure, supply-chain integrity, attack detection, containment, and revocation.

CAPACITY

Utilization, saturation, queueing, radio or fabric headroom, thermal margin, quota, and forecast error.

OPERATIONS

Change failure, mean time to detect, repair, support burden, vendor escalation, and evidence completeness.

PRIVACY / SAFETY

Unauthorized egress, retention breach, consent coverage, unsafe action, physical consequence, and contestability.

LIFECYCLE

Patch debt, end-of-support exposure, portability, replacement time, retirement completion, and reassessment age.

Decision gate: thresholds MUST be declared before assessment, cover representative load and failure conditions, and identify mandatory safety, identity, recovery, and evidence failures that block promotion.

MYTHOS-CLD-0001 / CLOUD ARCHITECTURE

8. Failure Modes and Adversarial Scenarios

Challenge assumptions before the system is trusted with consequential work

CONTROL-PLANE COMPROMISE

An administrative or orchestration plane can alter broad production scope.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

PROVIDER OR REGION DEPENDENCY

A provider, region, identity, DNS, or service dependency fails without a tested alternative.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

CONFIGURATION DRIFT

Runtime state diverges from approved desired state without timely detection.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

SOVEREIGNTY BREACH

Data or keys move through an unapproved jurisdiction or subprocess.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

COST AND QUOTA RUNAWAY

Unbounded scaling, abuse, or configuration error exhausts budget or service limits.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

EXIT FAILURE

Workloads, data, identity, or evidence cannot be migrated within the required time.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

Adversarial testing MUST protect people and production systems. Use isolated environments, synthetic or minimized data, bounded authority, kill switches, explicit legal and ethical review, and documented stop conditions.

MYTHOS-CLD-0001 / CLOUD ARCHITECTURE

9. Implementation Profiles

Risk-proportional implementation without weakening mandatory boundaries

FOUNDATIONAL

TYPICAL SCOPE

Low consequence, limited autonomy, non-sensitive data, reversible outputs.

MINIMUM DEPTH

Named owner; inventory; basic evaluation; access control; logging; rollback; annual review.

ADVANCED

TYPICAL SCOPE

Business-critical workflow, sensitive data, multiple tools or providers, moderate autonomy.

MINIMUM DEPTH

Formal threat model; representative evaluation; approval gates; isolated execution; tested recovery; quarterly review.

HIGH ASSURANCE

TYPICAL SCOPE

Safety, security, regulated, financial, identity, or broadly consequential use.

MINIMUM DEPTH

Independent challenge; strongest identity; deterministic enforcement; comprehensive evidence; canary release; continuous monitoring.

Profile selection rule

Choose the highest profile triggered by consequence, autonomy, sensitivity, scale, irreversibility, external dependency, or inability to rapidly detect and recover. Tailoring may strengthen controls; it may not erase a mandatory gate without a controlled exception.

MYTHOS-CLD-0001 / CLOUD ARCHITECTURE

9.2 Operational Lifecycle

Evidence-bearing operation from concept through retirement

1. **CONCEPT**
   Purpose, owner, impact, boundaries
2. **DESIGN**
   Architecture, threat, data, identity
3. **BUILD**
   Controlled implementation and provenance
4. **EVALUATE**
   Representative, adversarial, recovery tests
5. **RELEASE**
   Approval, canary, rollback, evidence
6. **OPERATE**
   Observe, support, detect, respond
7. **CHANGE**
   Reassess, compare, gate, reverse
8. **RETIRE**
   Revoke, export, dispose, preserve

LIFECYCLE INVARIANT

Every stage MUST leave sufficient evidence for the next authority to understand what was intended, what changed, what was tested, what failed, what was accepted, what remains uncertain, and how to recover or exit.

MYTHOS-CLD-0001 / CLOUD ARCHITECTURE

10. Adoption Roadmap

A sequenced path from uncontrolled capability to evidence-bearing operation

0-30 DAYS

Establish ownership and truth

Inventory systems and dependencies; classify risk; freeze unsupported claims; identify immediate privilege, data, and evidence gaps.

31-90 DAYS

Create enforceable boundaries

Define policy; isolate execution; implement identity and approval gates; establish representative evaluation and baseline telemetry.

3-6 MONTHS

Prove recovery and operating control

Exercise failure, rollback, revocation, incident, provider exit, and state-recovery scenarios; mature evidence packaging.

6-12 MONTHS

Scale assurance

Automate control checks; expand workload coverage; independent challenge; portfolio metrics; controlled exception expiry; continuous reassessment.

Adoption is complete only when operating evidence demonstrates the selected profile in the actual deployment context. Documentation alone is not conformance.

MYTHOS-CLD-0001 / CLOUD ARCHITECTURE

10.2 Conformance and Exception Statement

What an assurance claim must contain

SYSTEM / SERVICE

Named, versioned capability and deployment boundary.

PROFILE

Foundational, Advanced, or High Assurance with trigger rationale.

SCOPE

Workloads, users, data, tools, regions, providers, and exclusions.

CRITERIA

Pass, partial, fail, not applicable, and exception status for every criterion.

EVIDENCE

Artifact references, evidence grade, date, environment, and reproducibility.

LIMITATIONS

Known failure modes, unsupported workloads, uncertainty, and residual risk.

EXCEPTIONS

Owner, rationale, compensating control, expiration, and approval.

VALIDITY

Assessment date, change boundary, review date, and reassessment triggers.

Prohibited claim: “compliant” without the named scope, profile, evidence date, limitations, exceptions, and authority accepting residual risk.

MYTHOS-CLD-0001 / CLOUD ARCHITECTURE

11. Source Authority and Reference Register

Primary sources informing interpretation; current obligations and provider behavior must be verified at assessment t

NIST CSF 2.0	**Cybersecurity Framework 2.0**

https://doi.org/10.6028/NIST.CSWP.29

Enterprise governance and cybersecurity outcome framework.

NIST SP 800-53 Rev. 5	**Security and Privacy Controls for Information Systems and Organizations**

https://doi.org/10.6028/NIST.SP.800-53r5

Broad control baseline for organizational systems and services.

CSA CCM v4	**Cloud Controls Matrix**

https://cloudsecurityalliance.org/research/cloud-controls-matrix

Cloud control domains and shared-responsibility assessment structure.

FinOps Framework	**FinOps Framework**

https://www.finops.org/framework/

Cloud value, cost, usage, ownership, and operating-practice guidance.

NIST SP 800-204A	**Building Secure Microservices-based Applications Using Service-Mesh Architecture**

https://doi.org/10.6028/NIST.SP.800-204A

Microservice and service-mesh security architecture.

CNCF Cloud Native Security	**Cloud Native Security Whitepaper**

https://github.com/cncf/tag-security/tree/main/security-whitepaper

Cloud-native lifecycle, supply-chain, runtime, and operations guidance.

Source currency rule: authorities, specifications, legal obligations, provider services, firmware, browser behavior, carrier requirements, and operational evidence MUST be checked at assessment time. This publication records a 2026-07-24 source snapshot.

MYTHOS-CLD-0001 / CLOUD ARCHITECTURE

11.2 Revision History and Decision Register

Permanent identity, controlled change, and publication integrity

VERSION	DATE	STATE	SUMMARY
1.0.0-candidate	2026-07-24	Candidate	Permanent-publication edition; source refresh; expanded criteria, evidence, assurance, and lifecycle guidance.
Next review	2027-01-24	Scheduled	Review source currency, threat evolution, operating evidence, adoption feedback, and proposed revisions.

Publication decisions

PERMANENT IDENTITY The document ID remains stable across revisions; batch or production sequence metadata is not public identity.

CHANGE CONTROL Normative changes require rationale, impact analysis, affected-criterion mapping, and approval.

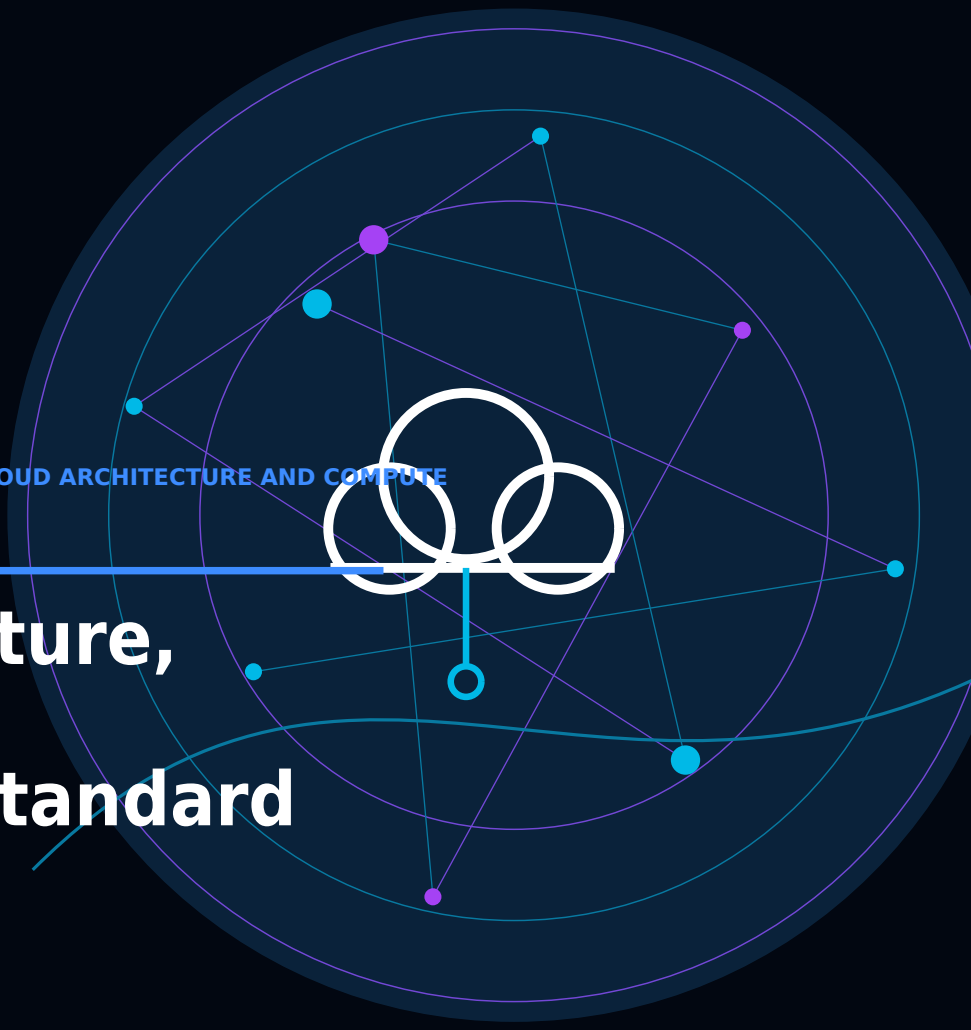
SUPERSESSION A superseded edition remains traceable; current routes and catalogs identify the active edition.

CORRECTION Material errors require a recorded correction, affected-evidence analysis, and notification appropriate to impact.

ARCHIVAL INTEGRITY Published artifacts receive hashes, metadata, and a release manifest sufficient for independent verification.

END OF PERMANENT PUBLICATION / MYTHOS-CLD-0001

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY



ENGINEERING STANDARDS LIBRARY / CLOUD ARCHITECTURE AND COMPUTE

AWS Architecture, Security, and Automation Standard

AWS organization design, landing zones, identity, networking, data, automation, resilience, operations, and evidence.

PERMANENT DOCUMENT ID

MYTHOS-CLD-0002

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

AWS Architecture, Security, and Automation Standard

DOCUMENT ID
MYTHOS-CLD-0002

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

18	6	4	1
ATOMIC CRITERIA	ASSURANCE VIEWS	IMPLEMENTATION PROFILES	PERMANENT IDENTITY

Publication doctrine. This publication establishes durable requirements for its defined scope. Interpretation must preserve the permanent document identity, required evidence, controlled exceptions, source currency, and formal revision history.

INFRASTRUCTURE AND CONTROL TOPOLOGY

Cloud Architecture and Compute

A trustworthy AWS cloud architecture system is a governed chain of ownership, identity, desired state, enforcement, workload or device behavior, telemetry, recovery, and independently reviewable evidence.

OWNERSHIP

Purpose, service boundary, accountable owner, suppliers, users, and retained responsibility remain explicit.

ENFORCEMENT

Identity, policy, segmentation, configuration, and local safety mechanisms constrain every trust transition.

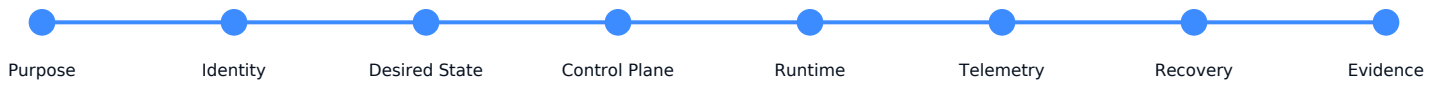
CONTINUITY

Capacity, dependency loss, safe degradation, backup, restoration, rollback, and exit are tested before reliance.

EVIDENCE

Inventory, desired state, runtime observation, tests, incidents, exceptions, and change records form the assurance chain.

GOVERNED INFRASTRUCTURE FLOW



Reconciliation, detection, response, restoration, and controlled evolution

INTERPRETATION AND ASSURANCE

How to apply this publication

Normative intent

Requirements define outcomes and constraints. Implementations may vary, but evidence must demonstrate equivalent control.

Evidence over assertion

Vendor documentation and design intent are not equivalent to reproduced behavior. Record evidence grade and uncertainty.

Risk-proportional depth

Higher consequence, autonomy, sensitivity, and irreversibility require stronger isolation, review, verification, and recovery.

Controlled exception

Exceptions require an owner, rationale, compensating control, residual-risk acceptance, expiration, and reevaluation trigger.

Normalized assessment scale

0	1	2	3	4	5
ABSENT	AD HOC	PARTIAL	OPERATIONAL	ADVANCED	LEADING

A numeric score must never hide a failed mandatory gate, missing evidence, untested workload, or unacceptable residual risk.

INTERACTIVE NAVIGATION

Contents

Navigation doctrine

Bookmarks and internal links are conveniences. Permanent document identity, criterion identifiers, evidence references, and revision history remain authoritative.

MYTHOS-CLD-0002 / CLOUD ARCHITECTURE

0. Executive Mandate

Purpose, strategic outcome, and non-negotiable operating doctrine

MANDATE

Organizations adopting AWS cloud architecture capabilities MUST define an owned service boundary, establish enforceable identity and configuration, protect data and safety, test failure and recovery, and preserve evidence sufficient for independent review.

Required outcomes

- Create a durable governance boundary for AWS cloud architecture across design, acquisition, deployment, operation, change, and retirement.
- Require risk-proportional segmentation, identity, configuration, telemetry, resilience, recovery, and supplier controls.
- Prevent central automation, administrative tooling, or provider services from becoming unbounded authority.
- Prove operation with representative workloads, devices, failure modes, and restoration exercises.
- Define measurable conformance, exceptions, reassessment triggers, and a viable exit path.

Decision boundary: conformance supports a documented assurance claim for the named scope. It does not prove universal availability, safety, regulatory acceptance, vendor fitness, or immunity from cyber-physical failure.

MYTHOS-CLD-0002 / CLOUD ARCHITECTURE

1. Scope and Applicability

Boundary definition, audience, exclusions, and triggering conditions

IN SCOPE

- Architecture, acquisition, configuration, integration, and operation of AWS cloud architecture capabilities.
- Human, workload, device, service, network, data, facility, provider, supplier, and evidence dependencies.
- Design, deployment, monitoring, support, incident response, recovery, migration, and retirement.
- On-premises, hosted, managed, carrier, manufacturer, integrator, and externally operated components.

OUT OF SCOPE

- Claims of legal, safety, carrier, or regulatory approval without the responsible authority.
- Vendor certification treated as proof of the adopter configuration or operating effectiveness.
- Theoretical or laboratory behavior presented as production evidence without a declared boundary.
- Inherited controls without documented scope, owner, period, limitations, and shared responsibility.

Applicability triggers

- The capability supports a business-critical, safety-relevant, regulated, public, or externally dependent service.
- The environment can expose sensitive data, identity, control, physical process, communications, or shared infrastructure.
- A management plane, automation system, carrier, provider, or supplier can affect broad operational scope.
- Failure, compromise, or change can be difficult to detect, reverse, isolate, or recover.
- The system depends on hardware, firmware, radio, browser, cloud, endpoint, IoT, OT, or real-time media behavior.

MYTHOS-CLD-0002 / CLOUD ARCHITECTURE

2. Definitions and Taxonomy

Controlled language for architecture, governance, and assessment

AUTHORITY

The right to approve, deny, constrain, or execute an action. Model output is not authority.

EVIDENCE

A reproducible source, trace, measurement, test, record, signed artifact, or documented observation supporting a claim.

ASSURANCE VIEW

A defined perspective such as governance, architecture, security, operations, privacy, or human oversight.

ATOMIC CRITERION

A uniquely identified requirement that can be assessed, evidenced, excepted, and revised independently.

IMPLEMENTATION PROFILE

A risk-proportional set of required depth, evidence, and gates for a class of use.

CONTROLLED EXCEPTION

A time-bounded deviation with owner, rationale, compensating control, residual risk, expiration, and review trigger.

DETERMINISTIC MECHANISM

A component whose inputs, policy, state transition, and side effects can be constrained and verified.

SOURCE AUTHORITY

The documented basis for treating a source as reliable for a defined claim, context, jurisdiction, and time.

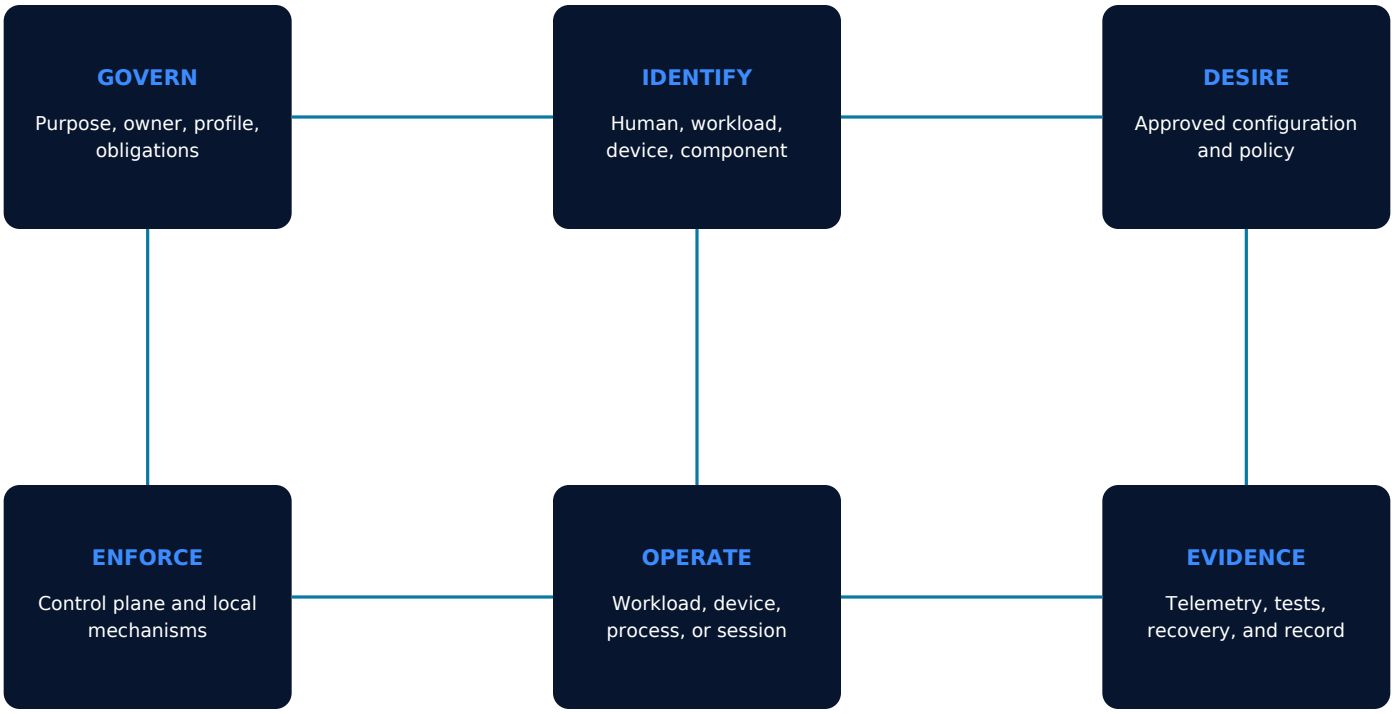
REASSESSMENT TRIGGER

A change, incident, drift signal, dependency revision, or time boundary requiring renewed evaluation.

MYTHOS-CLD-0002 / CLOUD ARCHITECTURE

3. Architecture and Trust Model

Separation of governance, management, enforcement, runtime, telemetry, and recovery



Mandatory trust boundaries

- Management planes MUST be isolated, strongly authenticated, monitored, and recoverable.
- Identity and policy MUST be evaluated at every provider, network, device, workload, and administrative transition.
- Runtime state MUST be compared with approved desired state and material drift MUST trigger response.
- Safety and continuity mechanisms MUST remain available when cloud, WAN, identity, DNS, time, carrier, or management dependencies fail.
- Evidence MUST be protected from the systems and administrators whose behavior it is intended to assess.

MYTHOS-CLD-0002 / CLOUD ARCHITECTURE

4. Governance and Accountability

Decision rights, separation of duties, and lifecycle ownership

ACCOUNTABLE OWNER

Accepts scope, risk tolerance, funding, and residual risk.

SYSTEM OWNER

Maintains architecture, inventory, operating boundaries, and change control.

SECURITY / PRIVACY

Defines threat, identity, data, isolation, monitoring, and incident requirements.

EVALUATION AUTHORITY

Designs representative tests, gates releases, and preserves reproducibility.

OPERATIONS

Maintains availability, rollback, recovery, evidence, and incident handling.

HUMAN OVERSIGHT

Reviews consequential decisions, exceptions, disputed outcomes, and harm signals.

DECISION PRINCIPLE

No single actor should be able to define the objective, grant authority, execute the consequential action, suppress evidence, and approve the result. Separation must be technical where consequence requires it.

MYTHOS-CLD-0002 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 01-04 of 18

MYTHOS-CLD-0002-C01 AWS Tenancy, Organization, and Account Hierarchy

The organization MUST define, implement, verify, and maintain aws tenancy, organization, and account hierarchy for in-scope AWS cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0002-C02 AWS Landing Zone and Environment Segmentation

The organization MUST define, implement, verify, and maintain aws landing zone and environment segmentation for in-scope AWS cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0002-C03 AWS Identity Federation and Privileged Administration

The organization MUST define, implement, verify, and maintain aws identity federation and privileged administration for in-scope AWS cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-CLD-0002-C04 AWS Resource Policy and Preventive Guardrails

The organization MUST define, implement, verify, and maintain aws resource policy and preventive guardrails for in-scope AWS cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-CLD-0002 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 05-08 of 18

MYTHOS-CLD-0002-C05 AWS Network Topology, Ingress, and Egress Control

The organization MUST define, implement, verify, and maintain aws network topology, ingress, and egress control for in-scope AWS cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-CLD-0002-C06 AWS Encryption, Key Authority, and Secret Protection

The organization MUST define, implement, verify, and maintain aws encryption, key authority, and secret protection for in-scope AWS cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-CLD-0002-C07 AWS Data Classification, Residency, and Service Selection

The organization MUST define, implement, verify, and maintain aws data classification, residency, and service selection for in-scope AWS cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-CLD-0002-C08 AWS Workload Identity and Service Authorization

The organization MUST define, implement, verify, and maintain aws workload identity and service authorization for in-scope AWS cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-CLD-0002 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 09-12 of 18

MYTHOS-CLD-0002-C09 AWS Declarative Provisioning and Change Control

The organization MUST define, implement, verify, and maintain aws declarative provisioning and change control for in-scope AWS cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0002-C10 AWS Artifact, Image, and Software Supply-Chain Assurance

The organization MUST define, implement, verify, and maintain aws artifact, image, and software supply-chain assurance for in-scope AWS cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0002-C11 AWS Logging, Telemetry, and Security Monitoring

The organization MUST define, implement, verify, and maintain aws logging, telemetry, and security monitoring for in-scope AWS cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-CLD-0002-C12 AWS Reliability, Availability-Zone, and Region Strategy

The organization MUST define, implement, verify, and maintain aws reliability, availability-zone, and region strategy for in-scope AWS cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-CLD-0002 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 13-15 of 18

MYTHOS-CLD-0002-C13 AWS Backup, Restoration, and Cyber-Recovery

The organization MUST define, implement, verify, and maintain aws backup, restoration, and cyber-recovery for in-scope AWS cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-CLD-0002-C14 AWS Vulnerability, Configuration, and Drift Management

The organization MUST define, implement, verify, and maintain aws vulnerability, configuration, and drift management for in-scope AWS cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-CLD-0002-C15 AWS Cost, Quota, Capacity, and FinOps Governance

The organization MUST define, implement, verify, and maintain aws cost, quota, capacity, and finops governance for in-scope AWS cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-CLD-0002 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 16-18 of 18

MYTHOS-CLD-0002-C16 AWS Incident Response and Provider Escalation

The organization MUST define, implement, verify, and maintain aws incident response and provider escalation for in-scope AWS cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-CLD-0002-C17 AWS Portability, Exit, and Data Disposition

The organization MUST define, implement, verify, and maintain aws portability, exit, and data disposition for in-scope AWS cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0002-C18 AWS Evidence Package and Periodic Reassessment

The organization MUST define, implement, verify, and maintain aws evidence package and periodic reassessment for in-scope AWS cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0002 / CLOUD ARCHITECTURE

6. Evidence and Assessment

Examine, interview, test, observe, restore, and trace

EXAMINE

Policies, inventories, diagrams, configuration, code, firmware, contracts, provider evidence, logs, exceptions, and lifecycle records.

INTERVIEW

Accountable owners, architects, engineers, operators, safety personnel, security teams, vendors, carriers, integrators, and responders.

TEST

Identity, segmentation, policy, negative paths, malformed input, capacity stress, dependency loss, failover, rollback, and revocation.

RESTORE

Independent restoration of configuration, data, service, device, controller, cluster, media, or browser state from protected evidence.

OBSERVE

Production-like performance, quality, drift, resource use, physical behavior, alarms, user impact, and incident execution.

TRACE

End-to-end linkage from approved intent and identity through change, runtime behavior, telemetry, outcome, exception, and review.

Evidence grades

A

Independently reproducible, complete, current, integrity-protected, and representative.

B

Reproduced by the implementing organization with strong traceability and controlled scope.

C

Partially reproduced or indirect; limitations, inherited responsibility, and uncertainty recorded.

D

Assertion, diagram, design intent, certificate, or vendor statement without reproduced behavior.

MYTHOS-CLD-0002 / CLOUD ARCHITECTURE

7. Assurance Views

Six perspectives required for a complete assurance claim

GOVERNANCE

Ownership, purpose, risk tolerance, policy, exception, and decision rights.

ARCHITECTURE

Boundaries, dependencies, failure modes, state, data flow, and portability.

SECURITY

Identity, authorization, isolation, secrets, abuse resistance, and incident response.

OPERATIONS

Reliability, performance, cost, observability, change, recovery, and support.

PRIVACY / RIGHTS

Purpose limitation, minimization, consent, access, correction, deletion, and egress.

HUMAN / SOCIETAL

Usability, accessibility, disclosure, contestability, impact, and human control.

Conformance requires a defensible position across every applicable view. A strong aggregate score cannot compensate for an unowned system, a failed mandatory gate, untested recovery, or evidence below the accepted grade.

MYTHOS-CLD-0002 / CLOUD ARCHITECTURE

7.2 Evaluation Metrics and Decision Gates

Measures must expose service behavior, control effectiveness, failure, uncertainty, and cost

SERVICE

Availability, correctness, coverage, quality, latency, throughput, and user or process outcome.

CONTROL

Unauthorized attempt, policy denial, drift, segmentation escape, privilege, and exception exposure.

RESILIENCE

Failover success, recovery time, data loss, safe degradation, restore validity, and dependency survival.

SECURITY

Exploitability, credential exposure, supply-chain integrity, attack detection, containment, and revocation.

CAPACITY

Utilization, saturation, queueing, radio or fabric headroom, thermal margin, quota, and forecast error.

OPERATIONS

Change failure, mean time to detect, repair, support burden, vendor escalation, and evidence completeness.

PRIVACY / SAFETY

Unauthorized egress, retention breach, consent coverage, unsafe action, physical consequence, and contestability.

LIFECYCLE

Patch debt, end-of-support exposure, portability, replacement time, retirement completion, and reassessment age.

Decision gate: thresholds MUST be declared before assessment, cover representative load and failure conditions, and identify mandatory safety, identity, recovery, and evidence failures that block promotion.

MYTHOS-CLD-0002 / CLOUD ARCHITECTURE

8. Failure Modes and Adversarial Scenarios

Challenge assumptions before the system is trusted with consequential work

CONTROL-PLANE COMPROMISE

An administrative or orchestration plane can alter broad production scope.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

PROVIDER OR REGION DEPENDENCY

A provider, region, identity, DNS, or service dependency fails without a tested alternative.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

CONFIGURATION DRIFT

Runtime state diverges from approved desired state without timely detection.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

SOVEREIGNTY BREACH

Data or keys move through an unapproved jurisdiction or subprocess.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

COST AND QUOTA RUNAWAY

Unbounded scaling, abuse, or configuration error exhausts budget or service limits.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

EXIT FAILURE

Workloads, data, identity, or evidence cannot be migrated within the required time.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

Adversarial testing MUST protect people and production systems. Use isolated environments, synthetic or minimized data, bounded authority, kill switches, explicit legal and ethical review, and documented stop conditions.

MYTHOS-CLD-0002 / CLOUD ARCHITECTURE

9. Implementation Profiles

Risk-proportional implementation without weakening mandatory boundaries

FOUNDATIONAL

TYPICAL SCOPE

Low consequence, limited autonomy, non-sensitive data, reversible outputs.

MINIMUM DEPTH

Named owner; inventory; basic evaluation; access control; logging; rollback; annual review.

ADVANCED

TYPICAL SCOPE

Business-critical workflow, sensitive data, multiple tools or providers, moderate autonomy.

MINIMUM DEPTH

Formal threat model; representative evaluation; approval gates; isolated execution; tested recovery; quarterly review.

HIGH ASSURANCE

TYPICAL SCOPE

Safety, security, regulated, financial, identity, or broadly consequential use.

MINIMUM DEPTH

Independent challenge; strongest identity; deterministic enforcement; comprehensive evidence; canary release; continuous monitoring.

Profile selection rule

Choose the highest profile triggered by consequence, autonomy, sensitivity, scale, irreversibility, external dependency, or inability to rapidly detect and recover. Tailoring may strengthen controls; it may not erase a mandatory gate without a controlled exception.

MYTHOS-CLD-0002 / CLOUD ARCHITECTURE

9.2 Operational Lifecycle

Evidence-bearing operation from concept through retirement

1. **CONCEPT**
   Purpose, owner, impact, boundaries
2. **DESIGN**
   Architecture, threat, data, identity
3. **BUILD**
   Controlled implementation and provenance
4. **EVALUATE**
   Representative, adversarial, recovery tests
5. **RELEASE**
   Approval, canary, rollback, evidence
6. **OPERATE**
   Observe, support, detect, respond
7. **CHANGE**
   Reassess, compare, gate, reverse
8. **RETIRE**
   Revoke, export, dispose, preserve

LIFECYCLE INVARIANT

Every stage MUST leave sufficient evidence for the next authority to understand what was intended, what changed, what was tested, what failed, what was accepted, what remains uncertain, and how to recover or exit.

MYTHOS-CLD-0002 / CLOUD ARCHITECTURE

10. Adoption Roadmap

A sequenced path from uncontrolled capability to evidence-bearing operation

0-30 DAYS

Establish ownership and truth

Inventory systems and dependencies; classify risk; freeze unsupported claims; identify immediate privilege, data, and evidence gaps.

31-90 DAYS

Create enforceable boundaries

Define policy; isolate execution; implement identity and approval gates; establish representative evaluation and baseline telemetry.

3-6 MONTHS

Prove recovery and operating control

Exercise failure, rollback, revocation, incident, provider exit, and state-recovery scenarios; mature evidence packaging.

6-12 MONTHS

Scale assurance

Automate control checks; expand workload coverage; independent challenge; portfolio metrics; controlled exception expiry; continuous reassessment.

Adoption is complete only when operating evidence demonstrates the selected profile in the actual deployment context. Documentation alone is not conformance.

MYTHOS-CLD-0002 / CLOUD ARCHITECTURE

10.2 Conformance and Exception Statement

What an assurance claim must contain

SYSTEM / SERVICE

Named, versioned capability and deployment boundary.

PROFILE

Foundational, Advanced, or High Assurance with trigger rationale.

SCOPE

Workloads, users, data, tools, regions, providers, and exclusions.

CRITERIA

Pass, partial, fail, not applicable, and exception status for every criterion.

EVIDENCE

Artifact references, evidence grade, date, environment, and reproducibility.

LIMITATIONS

Known failure modes, unsupported workloads, uncertainty, and residual risk.

EXCEPTIONS

Owner, rationale, compensating control, expiration, and approval.

VALIDITY

Assessment date, change boundary, review date, and reassessment triggers.

Prohibited claim: “compliant” without the named scope, profile, evidence date, limitations, exceptions, and authority accepting residual risk.

MYTHOS-CLD-0002 / CLOUD ARCHITECTURE

11. Source Authority and Reference Register

Primary sources informing interpretation; current obligations and provider behavior must be verified at assessment t

NIST CSF 2.0	**Cybersecurity Framework 2.0**

https://doi.org/10.6028/NIST.CSWP.29

Enterprise governance and cybersecurity outcome framework.

NIST SP 800-53 Rev. 5	**Security and Privacy Controls for Information Systems and Organizations**

https://doi.org/10.6028/NIST.SP.800-53r5

Broad control baseline for organizational systems and services.

CSA CCM v4	**Cloud Controls Matrix**

https://cloudsecurityalliance.org/research/cloud-controls-matrix

Cloud control domains and shared-responsibility assessment structure.

FinOps Framework	**FinOps Framework**

https://www.finops.org/framework/

Cloud value, cost, usage, ownership, and operating-practice guidance.

NIST SP 800-204A	**Building Secure Microservices-based Applications Using Service-Mesh Architecture**

https://doi.org/10.6028/NIST.SP.800-204A

Microservice and service-mesh security architecture.

CNCF Cloud Native Security	**Cloud Native Security Whitepaper**

https://github.com/cncf/tag-security/tree/main/security-whitepaper

Cloud-native lifecycle, supply-chain, runtime, and operations guidance.

Source currency rule: authorities, specifications, legal obligations, provider services, firmware, browser behavior, carrier requirements, and operational evidence MUST be checked at assessment time. This publication records a 2026-07-24 source snapshot.

MYTHOS-CLD-0002 / CLOUD ARCHITECTURE

11.2 Revision History and Decision Register

Permanent identity, controlled change, and publication integrity

VERSION	DATE	STATE	SUMMARY
1.0.0-candidate	2026-07-24	Candidate	Permanent-publication edition; source refresh; expanded criteria, evidence, assurance, and lifecycle guidance.
Next review	2027-01-24	Scheduled	Review source currency, threat evolution, operating evidence, adoption feedback, and proposed revisions.

Publication decisions

PERMANENT IDENTITY The document ID remains stable across revisions; batch or production sequence metadata is not public identity.

CHANGE CONTROL Normative changes require rationale, impact analysis, affected-criterion mapping, and approval.

SUPERSESSION A superseded edition remains traceable; current routes and catalogs identify the active edition.

CORRECTION Material errors require a recorded correction, affected-evidence analysis, and notification appropriate to impact.

ARCHIVAL INTEGRITY Published artifacts receive hashes, metadata, and a release manifest sufficient for independent verification.

END OF PERMANENT PUBLICATION / MYTHOS-CLD-0002

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY

ENGINEERING STANDARDS LIBRARY / CLOUD ARCHITECTURE AND COMPUTE

Microsoft Azure Architecture, Security, and Automation Standard

Azure tenant and management-group design, Entra identity, policy, networking, data, automation, resilience, and evidence.

PERMANENT DOCUMENT ID

MYTHOS-CLD-0003

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

Microsoft Azure Architecture, Security, and Automation Standard

DOCUMENT ID
MYTHOS-CLD-0003

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

18	6	4	1
ATOMIC CRITERIA	ASSURANCE VIEWS	IMPLEMENTATION PROFILES	PERMANENT IDENTITY

Publication doctrine. This publication establishes durable requirements for its defined scope. Interpretation must preserve the permanent document identity, required evidence, controlled exceptions, source currency, and formal revision history.

INFRASTRUCTURE AND CONTROL TOPOLOGY

Cloud Architecture and Compute

A trustworthy Microsoft Azure architecture system is a governed chain of ownership, identity, desired state, enforcement, workload or device behavior, telemetry, recovery, and independently reviewable evidence.

OWNERSHIP

Purpose, service boundary, accountable owner, suppliers, users, and retained responsibility remain explicit.

ENFORCEMENT

Identity, policy, segmentation, configuration, and local safety mechanisms constrain every trust transition.

CONTINUITY

Capacity, dependency loss, safe degradation, backup, restoration, rollback, and exit are tested before reliance.

EVIDENCE

Inventory, desired state, runtime observation, tests, incidents, exceptions, and change records form the assurance chain.

GOVERNED INFRASTRUCTURE FLOW

Reconciliation, detection, response, restoration, and controlled evolution

INTERPRETATION AND ASSURANCE

How to apply this publication

Normative intent

Requirements define outcomes and constraints. Implementations may vary, but evidence must demonstrate equivalent control.

Evidence over assertion

Vendor documentation and design intent are not equivalent to reproduced behavior. Record evidence grade and uncertainty.

Risk-proportional depth

Higher consequence, autonomy, sensitivity, and irreversibility require stronger isolation, review, verification, and recovery.

Controlled exception

Exceptions require an owner, rationale, compensating control, residual-risk acceptance, expiration, and reevaluation trigger.

Normalized assessment scale

0	1	2	3	4	5
ABSENT	AD HOC	PARTIAL	OPERATIONAL	ADVANCED	LEADING

A numeric score must never hide a failed mandatory gate, missing evidence, untested workload, or unacceptable residual risk.

INTERACTIVE NAVIGATION

Contents

Navigation doctrine

Bookmarks and internal links are conveniences. Permanent document identity, criterion identifiers, evidence references, and revision history remain authoritative.

MYTHOS-CLD-0003 / CLOUD ARCHITECTURE

0. Executive Mandate

Purpose, strategic outcome, and non-negotiable operating doctrine

MANDATE

Organizations adopting Microsoft Azure architecture capabilities MUST define an owned service boundary, establish enforceable identity and configuration, protect data and safety, test failure and recovery, and preserve evidence sufficient for independent review.

Required outcomes

- Create a durable governance boundary for Microsoft Azure architecture across design, acquisition, deployment, operation, change, and retirement.
- Require risk-proportional segmentation, identity, configuration, telemetry, resilience, recovery, and supplier controls.
- Prevent central automation, administrative tooling, or provider services from becoming unbounded authority.
- Prove operation with representative workloads, devices, failure modes, and restoration exercises.
- Define measurable conformance, exceptions, reassessment triggers, and a viable exit path.

Decision boundary: conformance supports a documented assurance claim for the named scope. It does not prove universal availability, safety, regulatory acceptance, vendor fitness, or immunity from cyber-physical failure.

MYTHOS-CLD-0003 / CLOUD ARCHITECTURE

1. Scope and Applicability

Boundary definition, audience, exclusions, and triggering conditions

IN SCOPE

- Architecture, acquisition, configuration, integration, and operation of Microsoft Azure architecture capabilities.
- Human, workload, device, service, network, data, facility, provider, supplier, and evidence dependencies.
- Design, deployment, monitoring, support, incident response, recovery, migration, and retirement.
- On-premises, hosted, managed, carrier, manufacturer, integrator, and externally operated components.

OUT OF SCOPE

- Claims of legal, safety, carrier, or regulatory approval without the responsible authority.
- Vendor certification treated as proof of the adopter configuration or operating effectiveness.
- Theoretical or laboratory behavior presented as production evidence without a declared boundary.
- Inherited controls without documented scope, owner, period, limitations, and shared responsibility.

Applicability triggers

- The capability supports a business-critical, safety-relevant, regulated, public, or externally dependent service.
- The environment can expose sensitive data, identity, control, physical process, communications, or shared infrastructure.
- A management plane, automation system, carrier, provider, or supplier can affect broad operational scope.
- Failure, compromise, or change can be difficult to detect, reverse, isolate, or recover.
- The system depends on hardware, firmware, radio, browser, cloud, endpoint, IoT, OT, or real-time media behavior.

MYTHOS-CLD-0003 / CLOUD ARCHITECTURE

2. Definitions and Taxonomy

Controlled language for architecture, governance, and assessment

AUTHORITY

The right to approve, deny, constrain, or execute an action. Model output is not authority.

EVIDENCE

A reproducible source, trace, measurement, test, record, signed artifact, or documented observation supporting a claim.

ASSURANCE VIEW

A defined perspective such as governance, architecture, security, operations, privacy, or human oversight.

ATOMIC CRITERION

A uniquely identified requirement that can be assessed, evidenced, excepted, and revised independently.

IMPLEMENTATION PROFILE

A risk-proportional set of required depth, evidence, and gates for a class of use.

CONTROLLED EXCEPTION

A time-bounded deviation with owner, rationale, compensating control, residual risk, expiration, and review trigger.

DETERMINISTIC MECHANISM

A component whose inputs, policy, state transition, and side effects can be constrained and verified.

SOURCE AUTHORITY

The documented basis for treating a source as reliable for a defined claim, context, jurisdiction, and time.

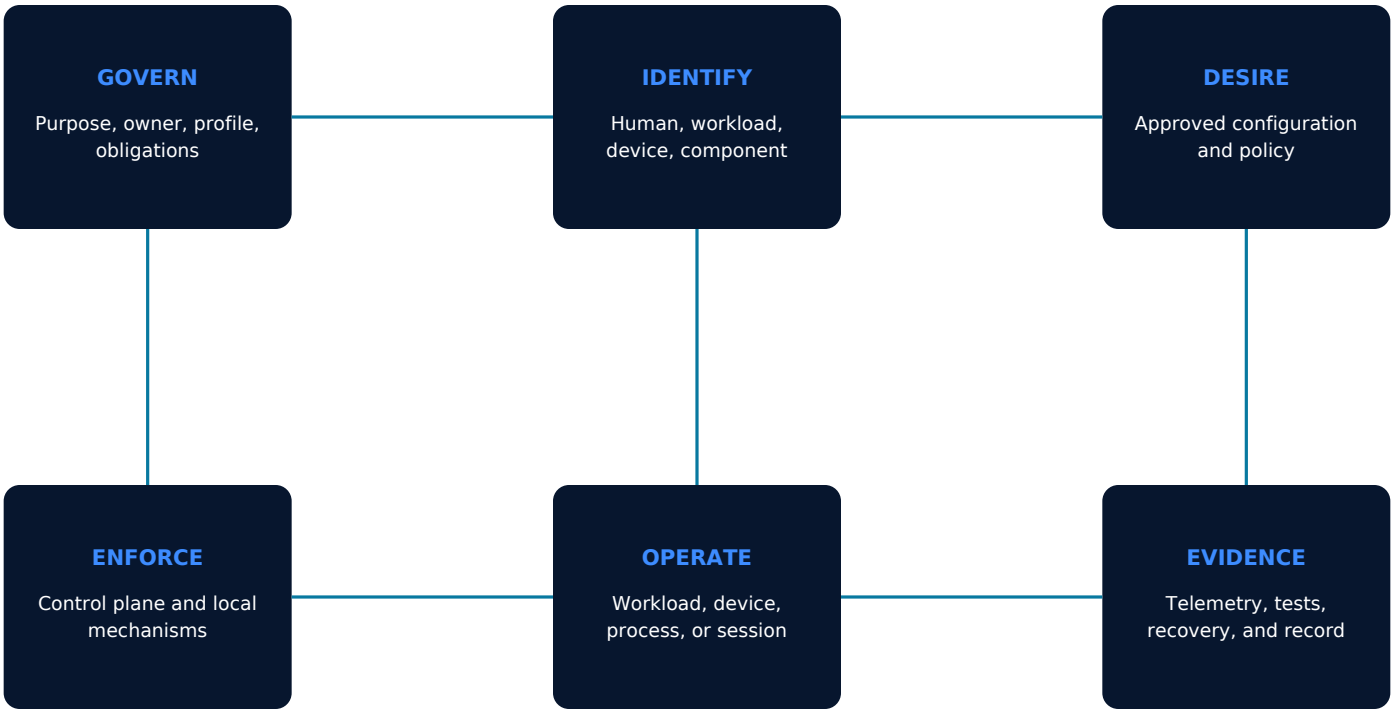
REASSESSMENT TRIGGER

A change, incident, drift signal, dependency revision, or time boundary requiring renewed evaluation.

MYTHOS-CLD-0003 / CLOUD ARCHITECTURE

3. Architecture and Trust Model

Separation of governance, management, enforcement, runtime, telemetry, and recovery



Mandatory trust boundaries

- Management planes MUST be isolated, strongly authenticated, monitored, and recoverable.
- Identity and policy MUST be evaluated at every provider, network, device, workload, and administrative transition.
- Runtime state MUST be compared with approved desired state and material drift MUST trigger response.
- Safety and continuity mechanisms MUST remain available when cloud, WAN, identity, DNS, time, carrier, or management dependencies fail.
- Evidence MUST be protected from the systems and administrators whose behavior it is intended to assess.

MYTHOS-CLD-0003 / CLOUD ARCHITECTURE

4. Governance and Accountability

Decision rights, separation of duties, and lifecycle ownership

ACCOUNTABLE OWNER

Accepts scope, risk tolerance, funding, and residual risk.

SYSTEM OWNER

Maintains architecture, inventory, operating boundaries, and change control.

SECURITY / PRIVACY

Defines threat, identity, data, isolation, monitoring, and incident requirements.

EVALUATION AUTHORITY

Designs representative tests, gates releases, and preserves reproducibility.

OPERATIONS

Maintains availability, rollback, recovery, evidence, and incident handling.

HUMAN OVERSIGHT

Reviews consequential decisions, exceptions, disputed outcomes, and harm signals.

DECISION PRINCIPLE

No single actor should be able to define the objective, grant authority, execute the consequential action, suppress evidence, and approve the result. Separation must be technical where consequence requires it.

MYTHOS-CLD-0003 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 01-04 of 18

MYTHOS-CLD-0003-C01 Microsoft Azure Tenancy, Organization, and Account Hierarchy

The organization MUST define, implement, verify, and maintain microsoft azure tenancy, organization, and account hierarchy for in-scope Microsoft Azure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0003-C02 Microsoft Azure Landing Zone and Environment Segmentation

The organization MUST define, implement, verify, and maintain microsoft azure landing zone and environment segmentation for in-scope Microsoft Azure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0003-C03 Microsoft Azure Identity Federation and Privileged Administration

The organization MUST define, implement, verify, and maintain microsoft azure identity federation and privileged administration for in-scope Microsoft Azure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-CLD-0003-C04 Microsoft Azure Resource Policy and Preventive Guardrails

The organization MUST define, implement, verify, and maintain microsoft azure resource policy and preventive guardrails for in-scope Microsoft Azure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-CLD-0003 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 05-08 of 18

MYTHOS-CLD-0003-C05 Microsoft Azure Network Topology, Ingress, and Egress Control

The organization MUST define, implement, verify, and maintain microsoft azure network topology, ingress, and egress control for in-scope Microsoft Azure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-CLD-0003-C06 Microsoft Azure Encryption, Key Authority, and Secret Protection

The organization MUST define, implement, verify, and maintain microsoft azure encryption, key authority, and secret protection for in-scope Microsoft Azure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-CLD-0003-C07 Microsoft Azure Data Classification, Residency, and Service Selection

The organization MUST define, implement, verify, and maintain microsoft azure data classification, residency, and service selection for in-scope Microsoft Azure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-CLD-0003-C08 Microsoft Azure Workload Identity and Service Authorization

The organization MUST define, implement, verify, and maintain microsoft azure workload identity and service authorization for in-scope Microsoft Azure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-CLD-0003 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 09-12 of 18

MYTHOS-CLD-0003-C09 Microsoft Azure Declarative Provisioning and Change Control

The organization MUST define, implement, verify, and maintain microsoft azure declarative provisioning and change control for in-scope Microsoft Azure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0003-C10 Microsoft Azure Artifact, Image, and Software Supply-Chain Assurance

The organization MUST define, implement, verify, and maintain microsoft azure artifact, image, and software supply-chain assurance for in-scope Microsoft Azure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0003-C11 Microsoft Azure Logging, Telemetry, and Security Monitoring

The organization MUST define, implement, verify, and maintain microsoft azure logging, telemetry, and security monitoring for in-scope Microsoft Azure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-CLD-0003-C12 Microsoft Azure Reliability, Availability-Zone, and Region Strategy

The organization MUST define, implement, verify, and maintain microsoft azure reliability, availability-zone, and region strategy for in-scope Microsoft Azure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-CLD-0003 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 13-15 of 18

MYTHOS-CLD-0003-C13 Microsoft Azure Backup, Restoration, and Cyber-Recovery

The organization MUST define, implement, verify, and maintain microsoft azure backup, restoration, and cyber-recovery for in-scope Microsoft Azure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-CLD-0003-C14 Microsoft Azure Vulnerability, Configuration, and Drift Management

The organization MUST define, implement, verify, and maintain microsoft azure vulnerability, configuration, and drift management for in-scope Microsoft Azure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-CLD-0003-C15 Microsoft Azure Cost, Quota, Capacity, and FinOps Governance

The organization MUST define, implement, verify, and maintain microsoft azure cost, quota, capacity, and finops governance for in-scope Microsoft Azure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-CLD-0003 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 16-18 of 18

MYTHOS-CLD-0003-C16 Microsoft Azure Incident Response and Provider Escalation

The organization MUST define, implement, verify, and maintain microsoft azure incident response and provider escalation for in-scope Microsoft Azure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-CLD-0003-C17 Microsoft Azure Portability, Exit, and Data Disposition

The organization MUST define, implement, verify, and maintain microsoft azure portability, exit, and data disposition for in-scope Microsoft Azure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0003-C18 Microsoft Azure Evidence Package and Periodic Reassessment

The organization MUST define, implement, verify, and maintain microsoft azure evidence package and periodic reassessment for in-scope Microsoft Azure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0003 / CLOUD ARCHITECTURE

6. Evidence and Assessment

Examine, interview, test, observe, restore, and trace

EXAMINE

Policies, inventories, diagrams, configuration, code, firmware, contracts, provider evidence, logs, exceptions, and lifecycle records.

INTERVIEW

Accountable owners, architects, engineers, operators, safety personnel, security teams, vendors, carriers, integrators, and responders.

TEST

Identity, segmentation, policy, negative paths, malformed input, capacity stress, dependency loss, failover, rollback, and revocation.

RESTORE

Independent restoration of configuration, data, service, device, controller, cluster, media, or browser state from protected evidence.

OBSERVE

Production-like performance, quality, drift, resource use, physical behavior, alarms, user impact, and incident execution.

TRACE

End-to-end linkage from approved intent and identity through change, runtime behavior, telemetry, outcome, exception, and review.

Evidence grades

A

Independently reproducible, complete, current, integrity-protected, and representative.

B

Reproduced by the implementing organization with strong traceability and controlled scope.

C

Partially reproduced or indirect; limitations, inherited responsibility, and uncertainty recorded.

D

Assertion, diagram, design intent, certificate, or vendor statement without reproduced behavior.

MYTHOS-CLD-0003 / CLOUD ARCHITECTURE

7. Assurance Views

Six perspectives required for a complete assurance claim

GOVERNANCE

Ownership, purpose, risk tolerance, policy, exception, and decision rights.

ARCHITECTURE

Boundaries, dependencies, failure modes, state, data flow, and portability.

SECURITY

Identity, authorization, isolation, secrets, abuse resistance, and incident response.

OPERATIONS

Reliability, performance, cost, observability, change, recovery, and support.

PRIVACY / RIGHTS

Purpose limitation, minimization, consent, access, correction, deletion, and egress.

HUMAN / SOCIETAL

Usability, accessibility, disclosure, contestability, impact, and human control.

Conformance requires a defensible position across every applicable view. A strong aggregate score cannot compensate for an unowned system, a failed mandatory gate, untested recovery, or evidence below the accepted grade.

MYTHOS-CLD-0003 / CLOUD ARCHITECTURE

7.2 Evaluation Metrics and Decision Gates

Measures must expose service behavior, control effectiveness, failure, uncertainty, and cost

SERVICE

Availability, correctness, coverage, quality, latency, throughput, and user or process outcome.

CONTROL

Unauthorized attempt, policy denial, drift, segmentation escape, privilege, and exception exposure.

RESILIENCE

Failover success, recovery time, data loss, safe degradation, restore validity, and dependency survival.

SECURITY

Exploitability, credential exposure, supply-chain integrity, attack detection, containment, and revocation.

CAPACITY

Utilization, saturation, queueing, radio or fabric headroom, thermal margin, quota, and forecast error.

OPERATIONS

Change failure, mean time to detect, repair, support burden, vendor escalation, and evidence completeness.

PRIVACY / SAFETY

Unauthorized egress, retention breach, consent coverage, unsafe action, physical consequence, and contestability.

LIFECYCLE

Patch debt, end-of-support exposure, portability, replacement time, retirement completion, and reassessment age.

Decision gate: thresholds MUST be declared before assessment, cover representative load and failure conditions, and identify mandatory safety, identity, recovery, and evidence failures that block promotion.

MYTHOS-CLD-0003 / CLOUD ARCHITECTURE

8. Failure Modes and Adversarial Scenarios

Challenge assumptions before the system is trusted with consequential work

CONTROL-PLANE COMPROMISE

An administrative or orchestration plane can alter broad production scope.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

PROVIDER OR REGION DEPENDENCY

A provider, region, identity, DNS, or service dependency fails without a tested alternative.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

CONFIGURATION DRIFT

Runtime state diverges from approved desired state without timely detection.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

SOVEREIGNTY BREACH

Data or keys move through an unapproved jurisdiction or subprocess.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

COST AND QUOTA RUNAWAY

Unbounded scaling, abuse, or configuration error exhausts budget or service limits.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

EXIT FAILURE

Workloads, data, identity, or evidence cannot be migrated within the required time.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

Adversarial testing MUST protect people and production systems. Use isolated environments, synthetic or minimized data, bounded authority, kill switches, explicit legal and ethical review, and documented stop conditions.

MYTHOS-CLD-0003 / CLOUD ARCHITECTURE

9. Implementation Profiles

Risk-proportional implementation without weakening mandatory boundaries

FOUNDATIONAL

TYPICAL SCOPE

Low consequence, limited autonomy, non-sensitive data, reversible outputs.

MINIMUM DEPTH

Named owner; inventory; basic evaluation; access control; logging; rollback; annual review.

ADVANCED

TYPICAL SCOPE

Business-critical workflow, sensitive data, multiple tools or providers, moderate autonomy.

MINIMUM DEPTH

Formal threat model; representative evaluation; approval gates; isolated execution; tested recovery; quarterly review.

HIGH ASSURANCE

TYPICAL SCOPE

Safety, security, regulated, financial, identity, or broadly consequential use.

MINIMUM DEPTH

Independent challenge; strongest identity; deterministic enforcement; comprehensive evidence; canary release; continuous monitoring.

Profile selection rule

Choose the highest profile triggered by consequence, autonomy, sensitivity, scale, irreversibility, external dependency, or inability to rapidly detect and recover. Tailoring may strengthen controls; it may not erase a mandatory gate without a controlled exception.

MYTHOS-CLD-0003 / CLOUD ARCHITECTURE

9.2 Operational Lifecycle

Evidence-bearing operation from concept through retirement

1. **CONCEPT**
   Purpose, owner, impact, boundaries
2. **DESIGN**
   Architecture, threat, data, identity
3. **BUILD**
   Controlled implementation and provenance
4. **EVALUATE**
   Representative, adversarial, recovery tests
5. **RELEASE**
   Approval, canary, rollback, evidence
6. **OPERATE**
   Observe, support, detect, respond
7. **CHANGE**
   Reassess, compare, gate, reverse
8. **RETIRE**
   Revoke, export, dispose, preserve

LIFECYCLE INVARIANT

Every stage MUST leave sufficient evidence for the next authority to understand what was intended, what changed, what was tested, what failed, what was accepted, what remains uncertain, and how to recover or exit.

MYTHOS-CLD-0003 / CLOUD ARCHITECTURE

10. Adoption Roadmap

A sequenced path from uncontrolled capability to evidence-bearing operation

0-30 DAYS

Establish ownership and truth

Inventory systems and dependencies; classify risk; freeze unsupported claims; identify immediate privilege, data, and evidence gaps.

31-90 DAYS

Create enforceable boundaries

Define policy; isolate execution; implement identity and approval gates; establish representative evaluation and baseline telemetry.

3-6 MONTHS

Prove recovery and operating control

Exercise failure, rollback, revocation, incident, provider exit, and state-recovery scenarios; mature evidence packaging.

6-12 MONTHS

Scale assurance

Automate control checks; expand workload coverage; independent challenge; portfolio metrics; controlled exception expiry; continuous reassessment.

Adoption is complete only when operating evidence demonstrates the selected profile in the actual deployment context. Documentation alone is not conformance.

MYTHOS-CLD-0003 / CLOUD ARCHITECTURE

10.2 Conformance and Exception Statement

What an assurance claim must contain

SYSTEM / SERVICE

Named, versioned capability and deployment boundary.

PROFILE

Foundational, Advanced, or High Assurance with trigger rationale.

SCOPE

Workloads, users, data, tools, regions, providers, and exclusions.

CRITERIA

Pass, partial, fail, not applicable, and exception status for every criterion.

EVIDENCE

Artifact references, evidence grade, date, environment, and reproducibility.

LIMITATIONS

Known failure modes, unsupported workloads, uncertainty, and residual risk.

EXCEPTIONS

Owner, rationale, compensating control, expiration, and approval.

VALIDITY

Assessment date, change boundary, review date, and reassessment triggers.

Prohibited claim: “compliant” without the named scope, profile, evidence date, limitations, exceptions, and authority accepting residual risk.

MYTHOS-CLD-0003 / CLOUD ARCHITECTURE

11. Source Authority and Reference Register

Primary sources informing interpretation; current obligations and provider behavior must be verified at assessment t

NIST CSF 2.0	**Cybersecurity Framework 2.0**

https://doi.org/10.6028/NIST.CSWP.29

Enterprise governance and cybersecurity outcome framework.

NIST SP 800-53 Rev. 5	**Security and Privacy Controls for Information Systems and Organizations**

https://doi.org/10.6028/NIST.SP.800-53r5

Broad control baseline for organizational systems and services.

CSA CCM v4	**Cloud Controls Matrix**

https://cloudsecurityalliance.org/research/cloud-controls-matrix

Cloud control domains and shared-responsibility assessment structure.

FinOps Framework	**FinOps Framework**

https://www.finops.org/framework/

Cloud value, cost, usage, ownership, and operating-practice guidance.

NIST SP 800-204A	**Building Secure Microservices-based Applications Using Service-Mesh Architecture**

https://doi.org/10.6028/NIST.SP.800-204A

Microservice and service-mesh security architecture.

CNCF Cloud Native Security	**Cloud Native Security Whitepaper**

https://github.com/cncf/tag-security/tree/main/security-whitepaper

Cloud-native lifecycle, supply-chain, runtime, and operations guidance.

Source currency rule: authorities, specifications, legal obligations, provider services, firmware, browser behavior, carrier requirements, and operational evidence MUST be checked at assessment time. This publication records a 2026-07-24 source snapshot.

MYTHOS-CLD-0003 / CLOUD ARCHITECTURE

11.2 Revision History and Decision Register

Permanent identity, controlled change, and publication integrity

VERSION	DATE	STATE	SUMMARY
1.0.0-candidate	2026-07-24	Candidate	Permanent-publication edition; source refresh; expanded criteria, evidence, assurance, and lifecycle guidance.
Next review	2027-01-24	Scheduled	Review source currency, threat evolution, operating evidence, adoption feedback, and proposed revisions.

Publication decisions

PERMANENT IDENTITY The document ID remains stable across revisions; batch or production sequence metadata is not public identity.

CHANGE CONTROL Normative changes require rationale, impact analysis, affected-criterion mapping, and approval.

SUPERSESSION A superseded edition remains traceable; current routes and catalogs identify the active edition.

CORRECTION Material errors require a recorded correction, affected-evidence analysis, and notification appropriate to impact.

ARCHIVAL INTEGRITY Published artifacts receive hashes, metadata, and a release manifest sufficient for independent verification.

END OF PERMANENT PUBLICATION / MYTHOS-CLD-0003

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY

ENGINEERING STANDARDS LIBRARY / CLOUD ARCHITECTURE AND COMPUTE

Google Cloud Architecture and Automation Standard

Google Cloud resource hierarchy, identity, policy, network, data, workload, resilience, automation, and evidence.

PERMANENT DOCUMENT ID

MYTHOS-CLD-0004

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

Google Cloud Architecture and Automation Standard

DOCUMENT ID
MYTHOS-CLD-0004

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

18	6	4	1
ATOMIC CRITERIA	ASSURANCE VIEWS	IMPLEMENTATION PROFILES	PERMANENT IDENTITY

Publication doctrine. This publication establishes durable requirements for its defined scope. Interpretation must preserve the permanent document identity, required evidence, controlled exceptions, source currency, and formal revision history.

INFRASTRUCTURE AND CONTROL TOPOLOGY

Cloud Architecture and Compute

A trustworthy Google Cloud architecture system is a governed chain of ownership, identity, desired state, enforcement, workload or device behavior, telemetry, recovery, and independently reviewable evidence.

OWNERSHIP

Purpose, service boundary, accountable owner, suppliers, users, and retained responsibility remain explicit.

ENFORCEMENT

Identity, policy, segmentation, configuration, and local safety mechanisms constrain every trust transition.

CONTINUITY

Capacity, dependency loss, safe degradation, backup, restoration, rollback, and exit are tested before reliance.

EVIDENCE

Inventory, desired state, runtime observation, tests, incidents, exceptions, and change records form the assurance chain.

GOVERNED INFRASTRUCTURE FLOW

Reconciliation, detection, response, restoration, and controlled evolution

INTERPRETATION AND ASSURANCE

How to apply this publication

Normative intent

Requirements define outcomes and constraints. Implementations may vary, but evidence must demonstrate equivalent control.

Evidence over assertion

Vendor documentation and design intent are not equivalent to reproduced behavior. Record evidence grade and uncertainty.

Risk-proportional depth

Higher consequence, autonomy, sensitivity, and irreversibility require stronger isolation, review, verification, and recovery.

Controlled exception

Exceptions require an owner, rationale, compensating control, residual-risk acceptance, expiration, and reevaluation trigger.

Normalized assessment scale

0	1	2	3	4	5
ABSENT	AD HOC	PARTIAL	OPERATIONAL	ADVANCED	LEADING

A numeric score must never hide a failed mandatory gate, missing evidence, untested workload, or unacceptable residual risk.

INTERACTIVE NAVIGATION

Contents

Navigation doctrine

Bookmarks and internal links are conveniences. Permanent document identity, criterion identifiers, evidence references, and revision history remain authoritative.

MYTHOS-CLD-0004 / CLOUD ARCHITECTURE

0. Executive Mandate

Purpose, strategic outcome, and non-negotiable operating doctrine

MANDATE

Organizations adopting Google Cloud architecture capabilities MUST define an owned service boundary, establish enforceable identity and configuration, protect data and safety, test failure and recovery, and preserve evidence sufficient for independent review.

Required outcomes

- Create a durable governance boundary for Google Cloud architecture across design, acquisition, deployment, operation, change, and retirement.
- Require risk-proportional segmentation, identity, configuration, telemetry, resilience, recovery, and supplier controls.
- Prevent central automation, administrative tooling, or provider services from becoming unbounded authority.
- Prove operation with representative workloads, devices, failure modes, and restoration exercises.
- Define measurable conformance, exceptions, reassessment triggers, and a viable exit path.

Decision boundary: conformance supports a documented assurance claim for the named scope. It does not prove universal availability, safety, regulatory acceptance, vendor fitness, or immunity from cyber-physical failure.

MYTHOS-CLD-0004 / CLOUD ARCHITECTURE

1. Scope and Applicability

Boundary definition, audience, exclusions, and triggering conditions

IN SCOPE

- Architecture, acquisition, configuration, integration, and operation of Google Cloud architecture capabilities.
- Human, workload, device, service, network, data, facility, provider, supplier, and evidence dependencies.
- Design, deployment, monitoring, support, incident response, recovery, migration, and retirement.
- On-premises, hosted, managed, carrier, manufacturer, integrator, and externally operated components.

OUT OF SCOPE

- Claims of legal, safety, carrier, or regulatory approval without the responsible authority.
- Vendor certification treated as proof of the adopter configuration or operating effectiveness.
- Theoretical or laboratory behavior presented as production evidence without a declared boundary.
- Inherited controls without documented scope, owner, period, limitations, and shared responsibility.

Applicability triggers

- The capability supports a business-critical, safety-relevant, regulated, public, or externally dependent service.
- The environment can expose sensitive data, identity, control, physical process, communications, or shared infrastructure.
- A management plane, automation system, carrier, provider, or supplier can affect broad operational scope.
- Failure, compromise, or change can be difficult to detect, reverse, isolate, or recover.
- The system depends on hardware, firmware, radio, browser, cloud, endpoint, IoT, OT, or real-time media behavior.

MYTHOS-CLD-0004 / CLOUD ARCHITECTURE

2. Definitions and Taxonomy

Controlled language for architecture, governance, and assessment

AUTHORITY

The right to approve, deny, constrain, or execute an action. Model output is not authority.

EVIDENCE

A reproducible source, trace, measurement, test, record, signed artifact, or documented observation supporting a claim.

ASSURANCE VIEW

A defined perspective such as governance, architecture, security, operations, privacy, or human oversight.

ATOMIC CRITERION

A uniquely identified requirement that can be assessed, evidenced, excepted, and revised independently.

IMPLEMENTATION PROFILE

A risk-proportional set of required depth, evidence, and gates for a class of use.

CONTROLLED EXCEPTION

A time-bounded deviation with owner, rationale, compensating control, residual risk, expiration, and review trigger.

DETERMINISTIC MECHANISM

A component whose inputs, policy, state transition, and side effects can be constrained and verified.

SOURCE AUTHORITY

The documented basis for treating a source as reliable for a defined claim, context, jurisdiction, and time.

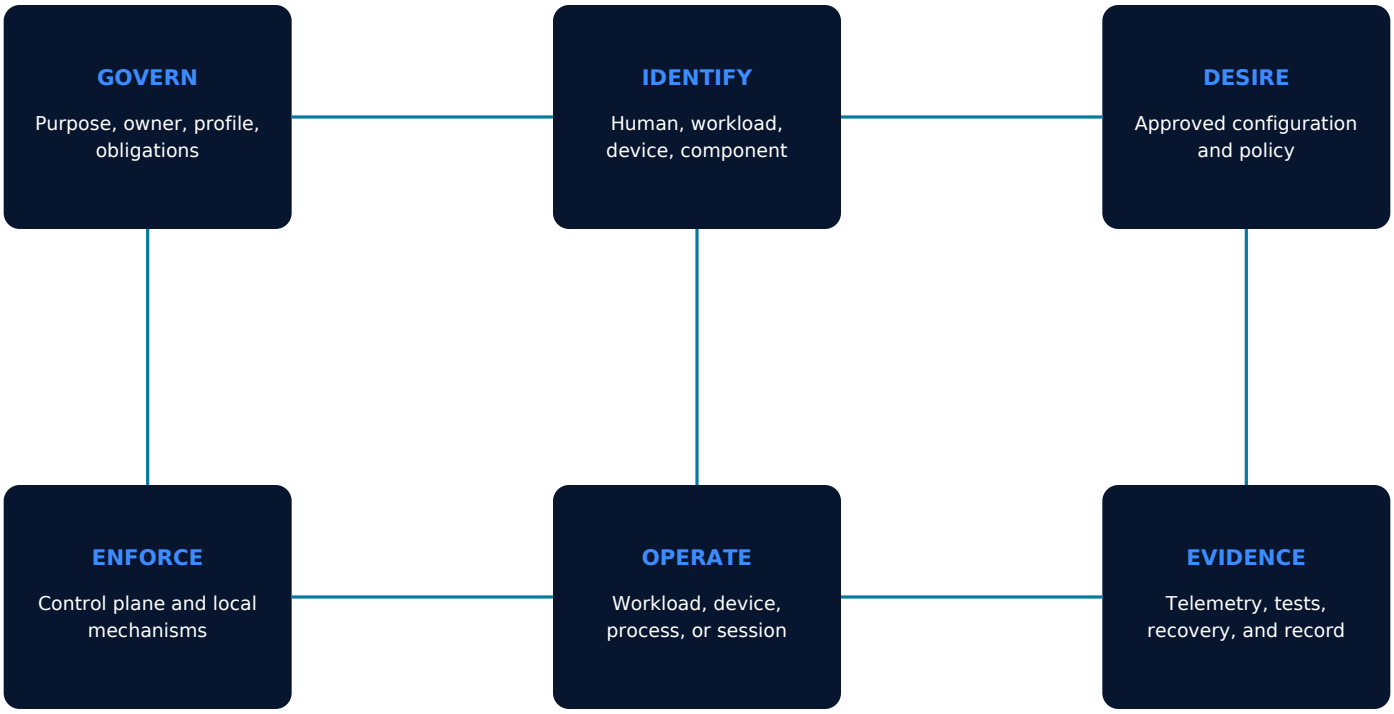
REASSESSMENT TRIGGER

A change, incident, drift signal, dependency revision, or time boundary requiring renewed evaluation.

MYTHOS-CLD-0004 / CLOUD ARCHITECTURE

3. Architecture and Trust Model

Separation of governance, management, enforcement, runtime, telemetry, and recovery



Mandatory trust boundaries

- Management planes MUST be isolated, strongly authenticated, monitored, and recoverable.
- Identity and policy MUST be evaluated at every provider, network, device, workload, and administrative transition.
- Runtime state MUST be compared with approved desired state and material drift MUST trigger response.
- Safety and continuity mechanisms MUST remain available when cloud, WAN, identity, DNS, time, carrier, or management dependencies fail.
- Evidence MUST be protected from the systems and administrators whose behavior it is intended to assess.

MYTHOS-CLD-0004 / CLOUD ARCHITECTURE

4. Governance and Accountability

Decision rights, separation of duties, and lifecycle ownership

ACCOUNTABLE OWNER

Accepts scope, risk tolerance, funding, and residual risk.

SYSTEM OWNER

Maintains architecture, inventory, operating boundaries, and change control.

SECURITY / PRIVACY

Defines threat, identity, data, isolation, monitoring, and incident requirements.

EVALUATION AUTHORITY

Designs representative tests, gates releases, and preserves reproducibility.

OPERATIONS

Maintains availability, rollback, recovery, evidence, and incident handling.

HUMAN OVERSIGHT

Reviews consequential decisions, exceptions, disputed outcomes, and harm signals.

DECISION PRINCIPLE

No single actor should be able to define the objective, grant authority, execute the consequential action, suppress evidence, and approve the result. Separation must be technical where consequence requires it.

MYTHOS-CLD-0004 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 01-04 of 18

MYTHOS-CLD-0004-C01 Google Cloud Tenancy, Organization, and Account Hierarchy

The organization MUST define, implement, verify, and maintain google cloud tenancy, organization, and account hierarchy for in-scope Google Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0004-C02 Google Cloud Landing Zone and Environment Segmentation

The organization MUST define, implement, verify, and maintain google cloud landing zone and environment segmentation for in-scope Google Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0004-C03 Google Cloud Identity Federation and Privileged Administration

The organization MUST define, implement, verify, and maintain google cloud identity federation and privileged administration for in-scope Google Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-CLD-0004-C04 Google Cloud Resource Policy and Preventive Guardrails

The organization MUST define, implement, verify, and maintain google cloud resource policy and preventive guardrails for in-scope Google Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-CLD-0004 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 05-08 of 18

MYTHOS-CLD-0004-C05 Google Cloud Network Topology, Ingress, and Egress Control

The organization MUST define, implement, verify, and maintain google cloud network topology, ingress, and egress control for in-scope Google Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-CLD-0004-C06 Google Cloud Encryption, Key Authority, and Secret Protection

The organization MUST define, implement, verify, and maintain google cloud encryption, key authority, and secret protection for in-scope Google Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-CLD-0004-C07 Google Cloud Data Classification, Residency, and Service Selection

The organization MUST define, implement, verify, and maintain google cloud data classification, residency, and service selection for in-scope Google Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-CLD-0004-C08 Google Cloud Workload Identity and Service Authorization

The organization MUST define, implement, verify, and maintain google cloud workload identity and service authorization for in-scope Google Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-CLD-0004 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 09-12 of 18

MYTHOS-CLD-0004-C09 Google Cloud Declarative Provisioning and Change Control

The organization MUST define, implement, verify, and maintain google cloud declarative provisioning and change control for in-scope Google Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0004-C10 Google Cloud Artifact, Image, and Software Supply-Chain Assurance

The organization MUST define, implement, verify, and maintain google cloud artifact, image, and software supply-chain assurance for in-scope Google Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0004-C11 Google Cloud Logging, Telemetry, and Security Monitoring

The organization MUST define, implement, verify, and maintain google cloud logging, telemetry, and security monitoring for in-scope Google Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-CLD-0004-C12 Google Cloud Reliability, Availability-Zone, and Region Strategy

The organization MUST define, implement, verify, and maintain google cloud reliability, availability-zone, and region strategy for in-scope Google Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-CLD-0004 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 13-15 of 18

MYTHOS-CLD-0004-C13 Google Cloud Backup, Restoration, and Cyber-Recovery

The organization MUST define, implement, verify, and maintain google cloud backup, restoration, and cyber-recovery for in-scope Google Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-CLD-0004-C14 Google Cloud Vulnerability, Configuration, and Drift Management

The organization MUST define, implement, verify, and maintain google cloud vulnerability, configuration, and drift management for in-scope Google Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-CLD-0004-C15 Google Cloud Cost, Quota, Capacity, and FinOps Governance

The organization MUST define, implement, verify, and maintain google cloud cost, quota, capacity, and finops governance for in-scope Google Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-CLD-0004 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 16-18 of 18

MYTHOS-CLD-0004-C16 Google Cloud Incident Response and Provider Escalation

The organization MUST define, implement, verify, and maintain google cloud incident response and provider escalation for in-scope Google Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-CLD-0004-C17 Google Cloud Portability, Exit, and Data Disposition

The organization MUST define, implement, verify, and maintain google cloud portability, exit, and data disposition for in-scope Google Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0004-C18 Google Cloud Evidence Package and Periodic Reassessment

The organization MUST define, implement, verify, and maintain google cloud evidence package and periodic reassessment for in-scope Google Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0004 / CLOUD ARCHITECTURE

6. Evidence and Assessment

Examine, interview, test, observe, restore, and trace

EXAMINE

Policies, inventories, diagrams, configuration, code, firmware, contracts, provider evidence, logs, exceptions, and lifecycle records.

INTERVIEW

Accountable owners, architects, engineers, operators, safety personnel, security teams, vendors, carriers, integrators, and responders.

TEST

Identity, segmentation, policy, negative paths, malformed input, capacity stress, dependency loss, failover, rollback, and revocation.

RESTORE

Independent restoration of configuration, data, service, device, controller, cluster, media, or browser state from protected evidence.

OBSERVE

Production-like performance, quality, drift, resource use, physical behavior, alarms, user impact, and incident execution.

TRACE

End-to-end linkage from approved intent and identity through change, runtime behavior, telemetry, outcome, exception, and review.

Evidence grades

A

Independently reproducible, complete, current, integrity-protected, and representative.

B

Reproduced by the implementing organization with strong traceability and controlled scope.

C

Partially reproduced or indirect; limitations, inherited responsibility, and uncertainty recorded.

D

Assertion, diagram, design intent, certificate, or vendor statement without reproduced behavior.

MYTHOS-CLD-0004 / CLOUD ARCHITECTURE

7. Assurance Views

Six perspectives required for a complete assurance claim

GOVERNANCE

Ownership, purpose, risk tolerance, policy, exception, and decision rights.

ARCHITECTURE

Boundaries, dependencies, failure modes, state, data flow, and portability.

SECURITY

Identity, authorization, isolation, secrets, abuse resistance, and incident response.

OPERATIONS

Reliability, performance, cost, observability, change, recovery, and support.

PRIVACY / RIGHTS

Purpose limitation, minimization, consent, access, correction, deletion, and egress.

HUMAN / SOCIETAL

Usability, accessibility, disclosure, contestability, impact, and human control.

Conformance requires a defensible position across every applicable view. A strong aggregate score cannot compensate for an unowned system, a failed mandatory gate, untested recovery, or evidence below the accepted grade.

MYTHOS-CLD-0004 / CLOUD ARCHITECTURE

7.2 Evaluation Metrics and Decision Gates

Measures must expose service behavior, control effectiveness, failure, uncertainty, and cost

SERVICE

Availability, correctness, coverage, quality, latency, throughput, and user or process outcome.

CONTROL

Unauthorized attempt, policy denial, drift, segmentation escape, privilege, and exception exposure.

RESILIENCE

Failover success, recovery time, data loss, safe degradation, restore validity, and dependency survival.

SECURITY

Exploitability, credential exposure, supply-chain integrity, attack detection, containment, and revocation.

CAPACITY

Utilization, saturation, queueing, radio or fabric headroom, thermal margin, quota, and forecast error.

OPERATIONS

Change failure, mean time to detect, repair, support burden, vendor escalation, and evidence completeness.

PRIVACY / SAFETY

Unauthorized egress, retention breach, consent coverage, unsafe action, physical consequence, and contestability.

LIFECYCLE

Patch debt, end-of-support exposure, portability, replacement time, retirement completion, and reassessment age.

Decision gate: thresholds MUST be declared before assessment, cover representative load and failure conditions, and identify mandatory safety, identity, recovery, and evidence failures that block promotion.

MYTHOS-CLD-0004 / CLOUD ARCHITECTURE

8. Failure Modes and Adversarial Scenarios

Challenge assumptions before the system is trusted with consequential work

CONTROL-PLANE COMPROMISE

An administrative or orchestration plane can alter broad production scope.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

PROVIDER OR REGION DEPENDENCY

A provider, region, identity, DNS, or service dependency fails without a tested alternative.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

CONFIGURATION DRIFT

Runtime state diverges from approved desired state without timely detection.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

SOVEREIGNTY BREACH

Data or keys move through an unapproved jurisdiction or subprocess.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

COST AND QUOTA RUNAWAY

Unbounded scaling, abuse, or configuration error exhausts budget or service limits.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

EXIT FAILURE

Workloads, data, identity, or evidence cannot be migrated within the required time.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

Adversarial testing MUST protect people and production systems. Use isolated environments, synthetic or minimized data, bounded authority, kill switches, explicit legal and ethical review, and documented stop conditions.

MYTHOS-CLD-0004 / CLOUD ARCHITECTURE

9. Implementation Profiles

Risk-proportional implementation without weakening mandatory boundaries

FOUNDATIONAL

TYPICAL SCOPE

Low consequence, limited autonomy, non-sensitive data, reversible outputs.

MINIMUM DEPTH

Named owner; inventory; basic evaluation; access control; logging; rollback; annual review.

ADVANCED

TYPICAL SCOPE

Business-critical workflow, sensitive data, multiple tools or providers, moderate autonomy.

MINIMUM DEPTH

Formal threat model; representative evaluation; approval gates; isolated execution; tested recovery; quarterly review.

HIGH ASSURANCE

TYPICAL SCOPE

Safety, security, regulated, financial, identity, or broadly consequential use.

MINIMUM DEPTH

Independent challenge; strongest identity; deterministic enforcement; comprehensive evidence; canary release; continuous monitoring.

Profile selection rule

Choose the highest profile triggered by consequence, autonomy, sensitivity, scale, irreversibility, external dependency, or inability to rapidly detect and recover. Tailoring may strengthen controls; it may not erase a mandatory gate without a controlled exception.

MYTHOS-CLD-0004 / CLOUD ARCHITECTURE

9.2 Operational Lifecycle

Evidence-bearing operation from concept through retirement

1. **CONCEPT**
   Purpose, owner, impact, boundaries
2. **DESIGN**
   Architecture, threat, data, identity
3. **BUILD**
   Controlled implementation and provenance
4. **EVALUATE**
   Representative, adversarial, recovery tests
5. **RELEASE**
   Approval, canary, rollback, evidence
6. **OPERATE**
   Observe, support, detect, respond
7. **CHANGE**
   Reassess, compare, gate, reverse
8. **RETIRE**
   Revoke, export, dispose, preserve

LIFECYCLE INVARIANT

Every stage MUST leave sufficient evidence for the next authority to understand what was intended, what changed, what was tested, what failed, what was accepted, what remains uncertain, and how to recover or exit.

MYTHOS-CLD-0004 / CLOUD ARCHITECTURE

10. Adoption Roadmap

A sequenced path from uncontrolled capability to evidence-bearing operation

0-30 DAYS

Establish ownership and truth

Inventory systems and dependencies; classify risk; freeze unsupported claims; identify immediate privilege, data, and evidence gaps.

31-90 DAYS

Create enforceable boundaries

Define policy; isolate execution; implement identity and approval gates; establish representative evaluation and baseline telemetry.

3-6 MONTHS

Prove recovery and operating control

Exercise failure, rollback, revocation, incident, provider exit, and state-recovery scenarios; mature evidence packaging.

6-12 MONTHS

Scale assurance

Automate control checks; expand workload coverage; independent challenge; portfolio metrics; controlled exception expiry; continuous reassessment.

Adoption is complete only when operating evidence demonstrates the selected profile in the actual deployment context. Documentation alone is not conformance.

MYTHOS-CLD-0004 / CLOUD ARCHITECTURE

10.2 Conformance and Exception Statement

What an assurance claim must contain

SYSTEM / SERVICE

Named, versioned capability and deployment boundary.

PROFILE

Foundational, Advanced, or High Assurance with trigger rationale.

SCOPE

Workloads, users, data, tools, regions, providers, and exclusions.

CRITERIA

Pass, partial, fail, not applicable, and exception status for every criterion.

EVIDENCE

Artifact references, evidence grade, date, environment, and reproducibility.

LIMITATIONS

Known failure modes, unsupported workloads, uncertainty, and residual risk.

EXCEPTIONS

Owner, rationale, compensating control, expiration, and approval.

VALIDITY

Assessment date, change boundary, review date, and reassessment triggers.

Prohibited claim: “compliant” without the named scope, profile, evidence date, limitations, exceptions, and authority accepting residual risk.

MYTHOS-CLD-0004 / CLOUD ARCHITECTURE

11. Source Authority and Reference Register

Primary sources informing interpretation; current obligations and provider behavior must be verified at assessment t

NIST CSF 2.0	**Cybersecurity Framework 2.0**

https://doi.org/10.6028/NIST.CSWP.29

Enterprise governance and cybersecurity outcome framework.

NIST SP 800-53 Rev. 5	**Security and Privacy Controls for Information Systems and Organizations**

https://doi.org/10.6028/NIST.SP.800-53r5

Broad control baseline for organizational systems and services.

CSA CCM v4	**Cloud Controls Matrix**

https://cloudsecurityalliance.org/research/cloud-controls-matrix

Cloud control domains and shared-responsibility assessment structure.

FinOps Framework	**FinOps Framework**

https://www.finops.org/framework/

Cloud value, cost, usage, ownership, and operating-practice guidance.

NIST SP 800-204A	**Building Secure Microservices-based Applications Using Service-Mesh Architecture**

https://doi.org/10.6028/NIST.SP.800-204A

Microservice and service-mesh security architecture.

CNCF Cloud Native Security	**Cloud Native Security Whitepaper**

https://github.com/cncf/tag-security/tree/main/security-whitepaper

Cloud-native lifecycle, supply-chain, runtime, and operations guidance.

Source currency rule: authorities, specifications, legal obligations, provider services, firmware, browser behavior, carrier requirements, and operational evidence MUST be checked at assessment time. This publication records a 2026-07-24 source snapshot.

MYTHOS-CLD-0004 / CLOUD ARCHITECTURE

11.2 Revision History and Decision Register

Permanent identity, controlled change, and publication integrity

VERSION	DATE	STATE	SUMMARY
1.0.0-candidate	2026-07-24	Candidate	Permanent-publication edition; source refresh; expanded criteria, evidence, assurance, and lifecycle guidance.
Next review	2027-01-24	Scheduled	Review source currency, threat evolution, operating evidence, adoption feedback, and proposed revisions.

Publication decisions

PERMANENT IDENTITY The document ID remains stable across revisions; batch or production sequence metadata is not public identity.

CHANGE CONTROL Normative changes require rationale, impact analysis, affected-criterion mapping, and approval.

SUPERSESSION A superseded edition remains traceable; current routes and catalogs identify the active edition.

CORRECTION Material errors require a recorded correction, affected-evidence analysis, and notification appropriate to impact.

ARCHIVAL INTEGRITY Published artifacts receive hashes, metadata, and a release manifest sufficient for independent verification.

END OF PERMANENT PUBLICATION / MYTHOS-CLD-0004

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY

ENGINEERING STANDARDS LIBRARY / CLOUD ARCHITECTURE AND COMPUTE

Oracle Cloud Infrastructure Architecture and Automation Standard

OCI tenancy, compartments, identity domains, network, data, automation, resilience, operations, and evidence.

PERMANENT DOCUMENT ID

MYTHOS-CLD-0005

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

Oracle Cloud Infrastructure Architecture and Automation Standard

DOCUMENT ID
MYTHOS-CLD-0005

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

18	6	4	1
ATOMIC CRITERIA	ASSURANCE VIEWS	IMPLEMENTATION PROFILES	PERMANENT IDENTITY

Publication doctrine. This publication establishes durable requirements for its defined scope. Interpretation must preserve the permanent document identity, required evidence, controlled exceptions, source currency, and formal revision history.

INFRASTRUCTURE AND CONTROL TOPOLOGY

Cloud Architecture and Compute

A trustworthy Oracle Cloud Infrastructure architecture system is a governed chain of ownership, identity, desired state, enforcement, workload or device behavior, telemetry, recovery, and independently reviewable evidence.

OWNERSHIP

Purpose, service boundary, accountable owner, suppliers, users, and retained responsibility remain explicit.

ENFORCEMENT

Identity, policy, segmentation, configuration, and local safety mechanisms constrain every trust transition.

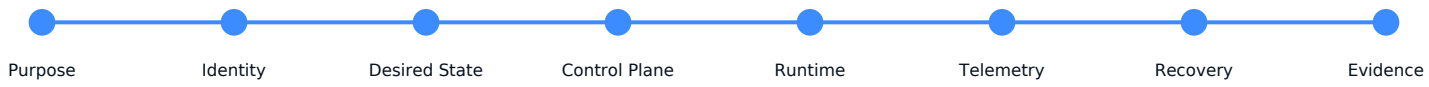
CONTINUITY

Capacity, dependency loss, safe degradation, backup, restoration, rollback, and exit are tested before reliance.

EVIDENCE

Inventory, desired state, runtime observation, tests, incidents, exceptions, and change records form the assurance chain.

GOVERNED INFRASTRUCTURE FLOW



Reconciliation, detection, response, restoration, and controlled evolution

INTERPRETATION AND ASSURANCE

How to apply this publication

Normative intent

Requirements define outcomes and constraints. Implementations may vary, but evidence must demonstrate equivalent control.

Evidence over assertion

Vendor documentation and design intent are not equivalent to reproduced behavior. Record evidence grade and uncertainty.

Risk-proportional depth

Higher consequence, autonomy, sensitivity, and irreversibility require stronger isolation, review, verification, and recovery.

Controlled exception

Exceptions require an owner, rationale, compensating control, residual-risk acceptance, expiration, and reevaluation trigger.

Normalized assessment scale

0	1	2	3	4	5
ABSENT	AD HOC	PARTIAL	OPERATIONAL	ADVANCED	LEADING

A numeric score must never hide a failed mandatory gate, missing evidence, untested workload, or unacceptable residual risk.

INTERACTIVE NAVIGATION

Contents

Navigation doctrine

Bookmarks and internal links are conveniences. Permanent document identity, criterion identifiers, evidence references, and revision history remain authoritative.

MYTHOS-CLD-0005 / CLOUD ARCHITECTURE

0. Executive Mandate

Purpose, strategic outcome, and non-negotiable operating doctrine

MANDATE

Organizations adopting Oracle Cloud Infrastructure architecture capabilities MUST define an owned service boundary, establish enforceable identity and configuration, protect data and safety, test failure and recovery, and preserve evidence sufficient for independent review.

Required outcomes

- Create a durable governance boundary for Oracle Cloud Infrastructure architecture across design, acquisition, deployment, operation, change, and retirement.
- Require risk-proportional segmentation, identity, configuration, telemetry, resilience, recovery, and supplier controls.
- Prevent central automation, administrative tooling, or provider services from becoming unbounded authority.
- Prove operation with representative workloads, devices, failure modes, and restoration exercises.
- Define measurable conformance, exceptions, reassessment triggers, and a viable exit path.

Decision boundary: conformance supports a documented assurance claim for the named scope. It does not prove universal availability, safety, regulatory acceptance, vendor fitness, or immunity from cyber-physical failure.

MYTHOS-CLD-0005 / CLOUD ARCHITECTURE

1. Scope and Applicability

Boundary definition, audience, exclusions, and triggering conditions

IN SCOPE

- Architecture, acquisition, configuration, integration, and operation of Oracle Cloud Infrastructure architecture capabilities.
- Human, workload, device, service, network, data, facility, provider, supplier, and evidence dependencies.
- Design, deployment, monitoring, support, incident response, recovery, migration, and retirement.
- On-premises, hosted, managed, carrier, manufacturer, integrator, and externally operated components.

OUT OF SCOPE

- Claims of legal, safety, carrier, or regulatory approval without the responsible authority.
- Vendor certification treated as proof of the adopter configuration or operating effectiveness.
- Theoretical or laboratory behavior presented as production evidence without a declared boundary.
- Inherited controls without documented scope, owner, period, limitations, and shared responsibility.

Applicability triggers

- The capability supports a business-critical, safety-relevant, regulated, public, or externally dependent service.
- The environment can expose sensitive data, identity, control, physical process, communications, or shared infrastructure.
- A management plane, automation system, carrier, provider, or supplier can affect broad operational scope.
- Failure, compromise, or change can be difficult to detect, reverse, isolate, or recover.
- The system depends on hardware, firmware, radio, browser, cloud, endpoint, IoT, OT, or real-time media behavior.

MYTHOS-CLD-0005 / CLOUD ARCHITECTURE

2. Definitions and Taxonomy

Controlled language for architecture, governance, and assessment

AUTHORITY

The right to approve, deny, constrain, or execute an action. Model output is not authority.

EVIDENCE

A reproducible source, trace, measurement, test, record, signed artifact, or documented observation supporting a claim.

ASSURANCE VIEW

A defined perspective such as governance, architecture, security, operations, privacy, or human oversight.

ATOMIC CRITERION

A uniquely identified requirement that can be assessed, evidenced, excepted, and revised independently.

IMPLEMENTATION PROFILE

A risk-proportional set of required depth, evidence, and gates for a class of use.

CONTROLLED EXCEPTION

A time-bounded deviation with owner, rationale, compensating control, residual risk, expiration, and review trigger.

DETERMINISTIC MECHANISM

A component whose inputs, policy, state transition, and side effects can be constrained and verified.

SOURCE AUTHORITY

The documented basis for treating a source as reliable for a defined claim, context, jurisdiction, and time.

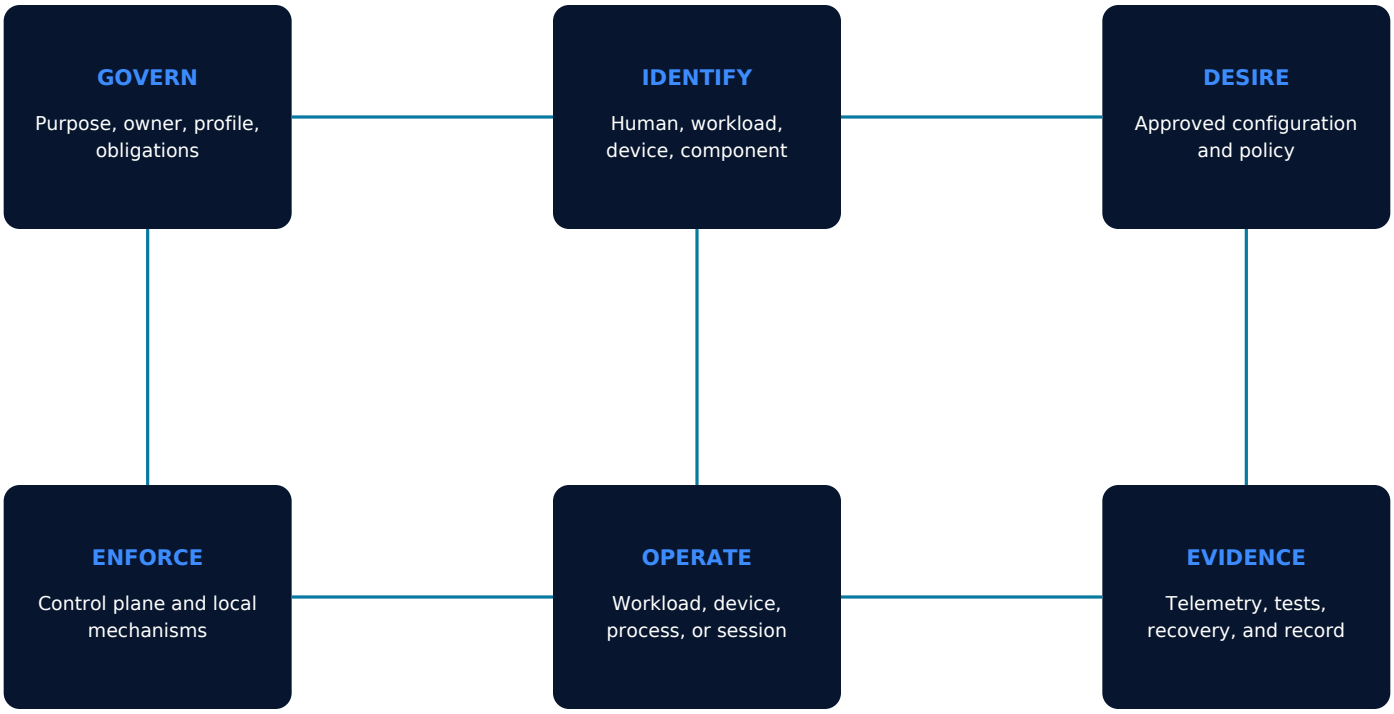
REASSESSMENT TRIGGER

A change, incident, drift signal, dependency revision, or time boundary requiring renewed evaluation.

MYTHOS-CLD-0005 / CLOUD ARCHITECTURE

3. Architecture and Trust Model

Separation of governance, management, enforcement, runtime, telemetry, and recovery



Mandatory trust boundaries

- Management planes MUST be isolated, strongly authenticated, monitored, and recoverable.
- Identity and policy MUST be evaluated at every provider, network, device, workload, and administrative transition.
- Runtime state MUST be compared with approved desired state and material drift MUST trigger response.
- Safety and continuity mechanisms MUST remain available when cloud, WAN, identity, DNS, time, carrier, or management dependencies fail.
- Evidence MUST be protected from the systems and administrators whose behavior it is intended to assess.

MYTHOS-CLD-0005 / CLOUD ARCHITECTURE

4. Governance and Accountability

Decision rights, separation of duties, and lifecycle ownership

ACCOUNTABLE OWNER

Accepts scope, risk tolerance, funding, and residual risk.

SYSTEM OWNER

Maintains architecture, inventory, operating boundaries, and change control.

SECURITY / PRIVACY

Defines threat, identity, data, isolation, monitoring, and incident requirements.

EVALUATION AUTHORITY

Designs representative tests, gates releases, and preserves reproducibility.

OPERATIONS

Maintains availability, rollback, recovery, evidence, and incident handling.

HUMAN OVERSIGHT

Reviews consequential decisions, exceptions, disputed outcomes, and harm signals.

DECISION PRINCIPLE

No single actor should be able to define the objective, grant authority, execute the consequential action, suppress evidence, and approve the result. Separation must be technical where consequence requires it.

MYTHOS-CLD-0005 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 01-04 of 18

MYTHOS-CLD-0005-C01 Oracle Cloud Infrastructure Tenancy, Organization, and Account Hierarchy

The organization MUST define, implement, verify, and maintain oracle cloud infrastructure tenancy, organization, and account hierarchy for in-scope Oracle Cloud Infrastructure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0005-C02 Oracle Cloud Infrastructure Landing Zone and Environment Segmentation

The organization MUST define, implement, verify, and maintain oracle cloud infrastructure landing zone and environment segmentation for in-scope Oracle Cloud Infrastructure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0005-C03 Oracle Cloud Infrastructure Identity Federation and Privileged Administration

The organization MUST define, implement, verify, and maintain oracle cloud infrastructure identity federation and privileged administration for in-scope Oracle Cloud Infrastructure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-CLD-0005-C04 Oracle Cloud Infrastructure Resource Policy and Preventive Guardrails

The organization MUST define, implement, verify, and maintain oracle cloud infrastructure resource policy and preventive guardrails for in-scope Oracle Cloud Infrastructure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-CLD-0005 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 05-08 of 18

MYTHOS-CLD-0005-C05 Oracle Cloud Infrastructure Network Topology, Ingress, and Egress Control

The organization MUST define, implement, verify, and maintain oracle cloud infrastructure network topology, ingress, and egress control for in-scope Oracle Cloud Infrastructure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-CLD-0005-C06 Oracle Cloud Infrastructure Encryption, Key Authority, and Secret Protection

The organization MUST define, implement, verify, and maintain oracle cloud infrastructure encryption, key authority, and secret protection for in-scope Oracle Cloud Infrastructure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-CLD-0005-C07 Oracle Cloud Infrastructure Data Classification, Residency, and Service Selection

The organization MUST define, implement, verify, and maintain oracle cloud infrastructure data classification, residency, and service selection for in-scope Oracle Cloud Infrastructure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-CLD-0005-C08 Oracle Cloud Infrastructure Workload Identity and Service Authorization

The organization MUST define, implement, verify, and maintain oracle cloud infrastructure workload identity and service authorization for in-scope Oracle Cloud Infrastructure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-CLD-0005 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 09-12 of 18

MYTHOS-CLD-0005-C09 Oracle Cloud Infrastructure Declarative Provisioning and Change Control

The organization MUST define, implement, verify, and maintain oracle cloud infrastructure declarative provisioning and change control for in-scope Oracle Cloud Infrastructure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0005-C10 Oracle Cloud Infrastructure Artifact, Image, and Software Supply-Chain Assurance

The organization MUST define, implement, verify, and maintain oracle cloud infrastructure artifact, image, and software supply-chain assurance for in-scope Oracle Cloud Infrastructure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0005-C11 Oracle Cloud Infrastructure Logging, Telemetry, and Security Monitoring

The organization MUST define, implement, verify, and maintain oracle cloud infrastructure logging, telemetry, and security monitoring for in-scope Oracle Cloud Infrastructure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-CLD-0005-C12 Oracle Cloud Infrastructure Reliability, Availability-Zone, and Region Strategy

The organization MUST define, implement, verify, and maintain oracle cloud infrastructure reliability, availability-zone, and region strategy for in-scope Oracle Cloud Infrastructure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-CLD-0005 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 13-15 of 18

MYTHOS-CLD-0005-C13 Oracle Cloud Infrastructure Backup, Restoration, and Cyber-Recovery

The organization MUST define, implement, verify, and maintain oracle cloud infrastructure backup, restoration, and cyber-recovery for in-scope Oracle Cloud Infrastructure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-CLD-0005-C14 Oracle Cloud Infrastructure Vulnerability, Configuration, and Drift Management

The organization MUST define, implement, verify, and maintain oracle cloud infrastructure vulnerability, configuration, and drift management for in-scope Oracle Cloud Infrastructure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-CLD-0005-C15 Oracle Cloud Infrastructure Cost, Quota, Capacity, and FinOps Governance

The organization MUST define, implement, verify, and maintain oracle cloud infrastructure cost, quota, capacity, and finops governance for in-scope Oracle Cloud Infrastructure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-CLD-0005 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 16-18 of 18

MYTHOS-CLD-0005-C16 Oracle Cloud Infrastructure Incident Response and Provider Escalation

The organization MUST define, implement, verify, and maintain oracle cloud infrastructure incident response and provider escalation for in-scope Oracle Cloud Infrastructure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-CLD-0005-C17 Oracle Cloud Infrastructure Portability, Exit, and Data Disposition

The organization MUST define, implement, verify, and maintain oracle cloud infrastructure portability, exit, and data disposition for in-scope Oracle Cloud Infrastructure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0005-C18 Oracle Cloud Infrastructure Evidence Package and Periodic Reassessment

The organization MUST define, implement, verify, and maintain oracle cloud infrastructure evidence package and periodic reassessment for in-scope Oracle Cloud Infrastructure architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0005 / CLOUD ARCHITECTURE

6. Evidence and Assessment

Examine, interview, test, observe, restore, and trace

EXAMINE

Policies, inventories, diagrams, configuration, code, firmware, contracts, provider evidence, logs, exceptions, and lifecycle records.

INTERVIEW

Accountable owners, architects, engineers, operators, safety personnel, security teams, vendors, carriers, integrators, and responders.

TEST

Identity, segmentation, policy, negative paths, malformed input, capacity stress, dependency loss, failover, rollback, and revocation.

RESTORE

Independent restoration of configuration, data, service, device, controller, cluster, media, or browser state from protected evidence.

OBSERVE

Production-like performance, quality, drift, resource use, physical behavior, alarms, user impact, and incident execution.

TRACE

End-to-end linkage from approved intent and identity through change, runtime behavior, telemetry, outcome, exception, and review.

Evidence grades

A

Independently reproducible, complete, current, integrity-protected, and representative.

B

Reproduced by the implementing organization with strong traceability and controlled scope.

C

Partially reproduced or indirect; limitations, inherited responsibility, and uncertainty recorded.

D

Assertion, diagram, design intent, certificate, or vendor statement without reproduced behavior.

MYTHOS-CLD-0005 / CLOUD ARCHITECTURE

7. Assurance Views

Six perspectives required for a complete assurance claim

GOVERNANCE

Ownership, purpose, risk tolerance, policy, exception, and decision rights.

ARCHITECTURE

Boundaries, dependencies, failure modes, state, data flow, and portability.

SECURITY

Identity, authorization, isolation, secrets, abuse resistance, and incident response.

OPERATIONS

Reliability, performance, cost, observability, change, recovery, and support.

PRIVACY / RIGHTS

Purpose limitation, minimization, consent, access, correction, deletion, and egress.

HUMAN / SOCIETAL

Usability, accessibility, disclosure, contestability, impact, and human control.

Conformance requires a defensible position across every applicable view. A strong aggregate score cannot compensate for an unowned system, a failed mandatory gate, untested recovery, or evidence below the accepted grade.

MYTHOS-CLD-0005 / CLOUD ARCHITECTURE

7.2 Evaluation Metrics and Decision Gates

Measures must expose service behavior, control effectiveness, failure, uncertainty, and cost

SERVICE

Availability, correctness, coverage, quality, latency, throughput, and user or process outcome.

CONTROL

Unauthorized attempt, policy denial, drift, segmentation escape, privilege, and exception exposure.

RESILIENCE

Failover success, recovery time, data loss, safe degradation, restore validity, and dependency survival.

SECURITY

Exploitability, credential exposure, supply-chain integrity, attack detection, containment, and revocation.

CAPACITY

Utilization, saturation, queueing, radio or fabric headroom, thermal margin, quota, and forecast error.

OPERATIONS

Change failure, mean time to detect, repair, support burden, vendor escalation, and evidence completeness.

PRIVACY / SAFETY

Unauthorized egress, retention breach, consent coverage, unsafe action, physical consequence, and contestability.

LIFECYCLE

Patch debt, end-of-support exposure, portability, replacement time, retirement completion, and reassessment age.

Decision gate: thresholds MUST be declared before assessment, cover representative load and failure conditions, and identify mandatory safety, identity, recovery, and evidence failures that block promotion.

MYTHOS-CLD-0005 / CLOUD ARCHITECTURE

8. Failure Modes and Adversarial Scenarios

Challenge assumptions before the system is trusted with consequential work

CONTROL-PLANE COMPROMISE

An administrative or orchestration plane can alter broad production scope.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

PROVIDER OR REGION DEPENDENCY

A provider, region, identity, DNS, or service dependency fails without a tested alternative.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

CONFIGURATION DRIFT

Runtime state diverges from approved desired state without timely detection.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

SOVEREIGNTY BREACH

Data or keys move through an unapproved jurisdiction or subprocess.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

COST AND QUOTA RUNAWAY

Unbounded scaling, abuse, or configuration error exhausts budget or service limits.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

EXIT FAILURE

Workloads, data, identity, or evidence cannot be migrated within the required time.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

Adversarial testing MUST protect people and production systems. Use isolated environments, synthetic or minimized data, bounded authority, kill switches, explicit legal and ethical review, and documented stop conditions.

MYTHOS-CLD-0005 / CLOUD ARCHITECTURE

9. Implementation Profiles

Risk-proportional implementation without weakening mandatory boundaries

FOUNDATIONAL

TYPICAL SCOPE

Low consequence, limited autonomy, non-sensitive data, reversible outputs.

MINIMUM DEPTH

Named owner; inventory; basic evaluation; access control; logging; rollback; annual review.

ADVANCED

TYPICAL SCOPE

Business-critical workflow, sensitive data, multiple tools or providers, moderate autonomy.

MINIMUM DEPTH

Formal threat model; representative evaluation; approval gates; isolated execution; tested recovery; quarterly review.

HIGH ASSURANCE

TYPICAL SCOPE

Safety, security, regulated, financial, identity, or broadly consequential use.

MINIMUM DEPTH

Independent challenge; strongest identity; deterministic enforcement; comprehensive evidence; canary release; continuous monitoring.

Profile selection rule

Choose the highest profile triggered by consequence, autonomy, sensitivity, scale, irreversibility, external dependency, or inability to rapidly detect and recover. Tailoring may strengthen controls; it may not erase a mandatory gate without a controlled exception.

MYTHOS-CLD-0005 / CLOUD ARCHITECTURE

9.2 Operational Lifecycle

Evidence-bearing operation from concept through retirement

1. **CONCEPT**
   Purpose, owner, impact, boundaries
2. **DESIGN**
   Architecture, threat, data, identity
3. **BUILD**
   Controlled implementation and provenance
4. **EVALUATE**
   Representative, adversarial, recovery tests
5. **RELEASE**
   Approval, canary, rollback, evidence
6. **OPERATE**
   Observe, support, detect, respond
7. **CHANGE**
   Reassess, compare, gate, reverse
8. **RETIRE**
   Revoke, export, dispose, preserve

LIFECYCLE INVARIANT

Every stage MUST leave sufficient evidence for the next authority to understand what was intended, what changed, what was tested, what failed, what was accepted, what remains uncertain, and how to recover or exit.

MYTHOS-CLD-0005 / CLOUD ARCHITECTURE

10. Adoption Roadmap

A sequenced path from uncontrolled capability to evidence-bearing operation

0-30 DAYS

Establish ownership and truth

Inventory systems and dependencies; classify risk; freeze unsupported claims; identify immediate privilege, data, and evidence gaps.

31-90 DAYS

Create enforceable boundaries

Define policy; isolate execution; implement identity and approval gates; establish representative evaluation and baseline telemetry.

3-6 MONTHS

Prove recovery and operating control

Exercise failure, rollback, revocation, incident, provider exit, and state-recovery scenarios; mature evidence packaging.

6-12 MONTHS

Scale assurance

Automate control checks; expand workload coverage; independent challenge; portfolio metrics; controlled exception expiry; continuous reassessment.

Adoption is complete only when operating evidence demonstrates the selected profile in the actual deployment context. Documentation alone is not conformance.

MYTHOS-CLD-0005 / CLOUD ARCHITECTURE

10.2 Conformance and Exception Statement

What an assurance claim must contain

SYSTEM / SERVICE

Named, versioned capability and deployment boundary.

PROFILE

Foundational, Advanced, or High Assurance with trigger rationale.

SCOPE

Workloads, users, data, tools, regions, providers, and exclusions.

CRITERIA

Pass, partial, fail, not applicable, and exception status for every criterion.

EVIDENCE

Artifact references, evidence grade, date, environment, and reproducibility.

LIMITATIONS

Known failure modes, unsupported workloads, uncertainty, and residual risk.

EXCEPTIONS

Owner, rationale, compensating control, expiration, and approval.

VALIDITY

Assessment date, change boundary, review date, and reassessment triggers.

Prohibited claim: “compliant” without the named scope, profile, evidence date, limitations, exceptions, and authority accepting residual risk.

MYTHOS-CLD-0005 / CLOUD ARCHITECTURE

11. Source Authority and Reference Register

Primary sources informing interpretation; current obligations and provider behavior must be verified at assessment t

NIST CSF 2.0	**Cybersecurity Framework 2.0**

https://doi.org/10.6028/NIST.CSWP.29

Enterprise governance and cybersecurity outcome framework.

NIST SP 800-53 Rev. 5	**Security and Privacy Controls for Information Systems and Organizations**

https://doi.org/10.6028/NIST.SP.800-53r5

Broad control baseline for organizational systems and services.

CSA CCM v4	**Cloud Controls Matrix**

https://cloudsecurityalliance.org/research/cloud-controls-matrix

Cloud control domains and shared-responsibility assessment structure.

FinOps Framework	**FinOps Framework**

https://www.finops.org/framework/

Cloud value, cost, usage, ownership, and operating-practice guidance.

NIST SP 800-204A	**Building Secure Microservices-based Applications Using Service-Mesh Architecture**

https://doi.org/10.6028/NIST.SP.800-204A

Microservice and service-mesh security architecture.

CNCF Cloud Native Security	**Cloud Native Security Whitepaper**

https://github.com/cncf/tag-security/tree/main/security-whitepaper

Cloud-native lifecycle, supply-chain, runtime, and operations guidance.

Source currency rule: authorities, specifications, legal obligations, provider services, firmware, browser behavior, carrier requirements, and operational evidence MUST be checked at assessment time. This publication records a 2026-07-24 source snapshot.

MYTHOS-CLD-0005 / CLOUD ARCHITECTURE

11.2 Revision History and Decision Register

Permanent identity, controlled change, and publication integrity

VERSION	DATE	STATE	SUMMARY
1.0.0-candidate	2026-07-24	Candidate	Permanent-publication edition; source refresh; expanded criteria, evidence, assurance, and lifecycle guidance.
Next review	2027-01-24	Scheduled	Review source currency, threat evolution, operating evidence, adoption feedback, and proposed revisions.

Publication decisions

PERMANENT IDENTITY The document ID remains stable across revisions; batch or production sequence metadata is not public identity.

CHANGE CONTROL Normative changes require rationale, impact analysis, affected-criterion mapping, and approval.

SUPERSESSION A superseded edition remains traceable; current routes and catalogs identify the active edition.

CORRECTION Material errors require a recorded correction, affected-evidence analysis, and notification appropriate to impact.

ARCHIVAL INTEGRITY Published artifacts receive hashes, metadata, and a release manifest sufficient for independent verification.

END OF PERMANENT PUBLICATION / MYTHOS-CLD-0005

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY

ENGINEERING STANDARDS LIBRARY / CLOUD ARCHITECTURE AND COMPUTE

Alibaba Cloud Architecture and Automation Standard

Alibaba Cloud resource hierarchy, RAM identity, network, data residency, automation, resilience, operations, and evidence.

PERMANENT DOCUMENT ID

MYTHOS-CLD-0006

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

Alibaba Cloud Architecture and Automation Standard

DOCUMENT ID
MYTHOS-CLD-0006

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

18	6	4	1
ATOMIC CRITERIA	ASSURANCE VIEWS	IMPLEMENTATION PROFILES	PERMANENT IDENTITY

Publication doctrine. This publication establishes durable requirements for its defined scope. Interpretation must preserve the permanent document identity, required evidence, controlled exceptions, source currency, and formal revision history.

INFRASTRUCTURE AND CONTROL TOPOLOGY

Cloud Architecture and Compute

A trustworthy Alibaba Cloud architecture system is a governed chain of ownership, identity, desired state, enforcement, workload or device behavior, telemetry, recovery, and independently reviewable evidence.

OWNERSHIP

Purpose, service boundary, accountable owner, suppliers, users, and retained responsibility remain explicit.

ENFORCEMENT

Identity, policy, segmentation, configuration, and local safety mechanisms constrain every trust transition.

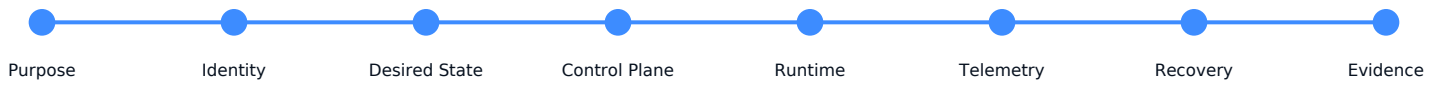
CONTINUITY

Capacity, dependency loss, safe degradation, backup, restoration, rollback, and exit are tested before reliance.

EVIDENCE

Inventory, desired state, runtime observation, tests, incidents, exceptions, and change records form the assurance chain.

GOVERNED INFRASTRUCTURE FLOW



Reconciliation, detection, response, restoration, and controlled evolution

INTERPRETATION AND ASSURANCE

How to apply this publication

Normative intent

Requirements define outcomes and constraints. Implementations may vary, but evidence must demonstrate equivalent control.

Evidence over assertion

Vendor documentation and design intent are not equivalent to reproduced behavior. Record evidence grade and uncertainty.

Risk-proportional depth

Higher consequence, autonomy, sensitivity, and irreversibility require stronger isolation, review, verification, and recovery.

Controlled exception

Exceptions require an owner, rationale, compensating control, residual-risk acceptance, expiration, and reevaluation trigger.

Normalized assessment scale

0	1	2	3	4	5
ABSENT	AD HOC	PARTIAL	OPERATIONAL	ADVANCED	LEADING

A numeric score must never hide a failed mandatory gate, missing evidence, untested workload, or unacceptable residual risk.

INTERACTIVE NAVIGATION

Contents

Navigation doctrine

Bookmarks and internal links are conveniences. Permanent document identity, criterion identifiers, evidence references, and revision history remain authoritative.

MYTHOS-CLD-0006 / CLOUD ARCHITECTURE

0. Executive Mandate

Purpose, strategic outcome, and non-negotiable operating doctrine

MANDATE

Organizations adopting Alibaba Cloud architecture capabilities MUST define an owned service boundary, establish enforceable identity and configuration, protect data and safety, test failure and recovery, and preserve evidence sufficient for independent review.

Required outcomes

- Create a durable governance boundary for Alibaba Cloud architecture across design, acquisition, deployment, operation, change, and retirement.
- Require risk-proportional segmentation, identity, configuration, telemetry, resilience, recovery, and supplier controls.
- Prevent central automation, administrative tooling, or provider services from becoming unbounded authority.
- Prove operation with representative workloads, devices, failure modes, and restoration exercises.
- Define measurable conformance, exceptions, reassessment triggers, and a viable exit path.

Decision boundary: conformance supports a documented assurance claim for the named scope. It does not prove universal availability, safety, regulatory acceptance, vendor fitness, or immunity from cyber-physical failure.

MYTHOS-CLD-0006 / CLOUD ARCHITECTURE

1. Scope and Applicability

Boundary definition, audience, exclusions, and triggering conditions

IN SCOPE

- Architecture, acquisition, configuration, integration, and operation of Alibaba Cloud architecture capabilities.
- Human, workload, device, service, network, data, facility, provider, supplier, and evidence dependencies.
- Design, deployment, monitoring, support, incident response, recovery, migration, and retirement.
- On-premises, hosted, managed, carrier, manufacturer, integrator, and externally operated components.

OUT OF SCOPE

- Claims of legal, safety, carrier, or regulatory approval without the responsible authority.
- Vendor certification treated as proof of the adopter configuration or operating effectiveness.
- Theoretical or laboratory behavior presented as production evidence without a declared boundary.
- Inherited controls without documented scope, owner, period, limitations, and shared responsibility.

Applicability triggers

- The capability supports a business-critical, safety-relevant, regulated, public, or externally dependent service.
- The environment can expose sensitive data, identity, control, physical process, communications, or shared infrastructure.
- A management plane, automation system, carrier, provider, or supplier can affect broad operational scope.
- Failure, compromise, or change can be difficult to detect, reverse, isolate, or recover.
- The system depends on hardware, firmware, radio, browser, cloud, endpoint, IoT, OT, or real-time media behavior.

MYTHOS-CLD-0006 / CLOUD ARCHITECTURE

2. Definitions and Taxonomy

Controlled language for architecture, governance, and assessment

AUTHORITY

The right to approve, deny, constrain, or execute an action. Model output is not authority.

EVIDENCE

A reproducible source, trace, measurement, test, record, signed artifact, or documented observation supporting a claim.

ASSURANCE VIEW

A defined perspective such as governance, architecture, security, operations, privacy, or human oversight.

ATOMIC CRITERION

A uniquely identified requirement that can be assessed, evidenced, excepted, and revised independently.

IMPLEMENTATION PROFILE

A risk-proportional set of required depth, evidence, and gates for a class of use.

CONTROLLED EXCEPTION

A time-bounded deviation with owner, rationale, compensating control, residual risk, expiration, and review trigger.

DETERMINISTIC MECHANISM

A component whose inputs, policy, state transition, and side effects can be constrained and verified.

SOURCE AUTHORITY

The documented basis for treating a source as reliable for a defined claim, context, jurisdiction, and time.

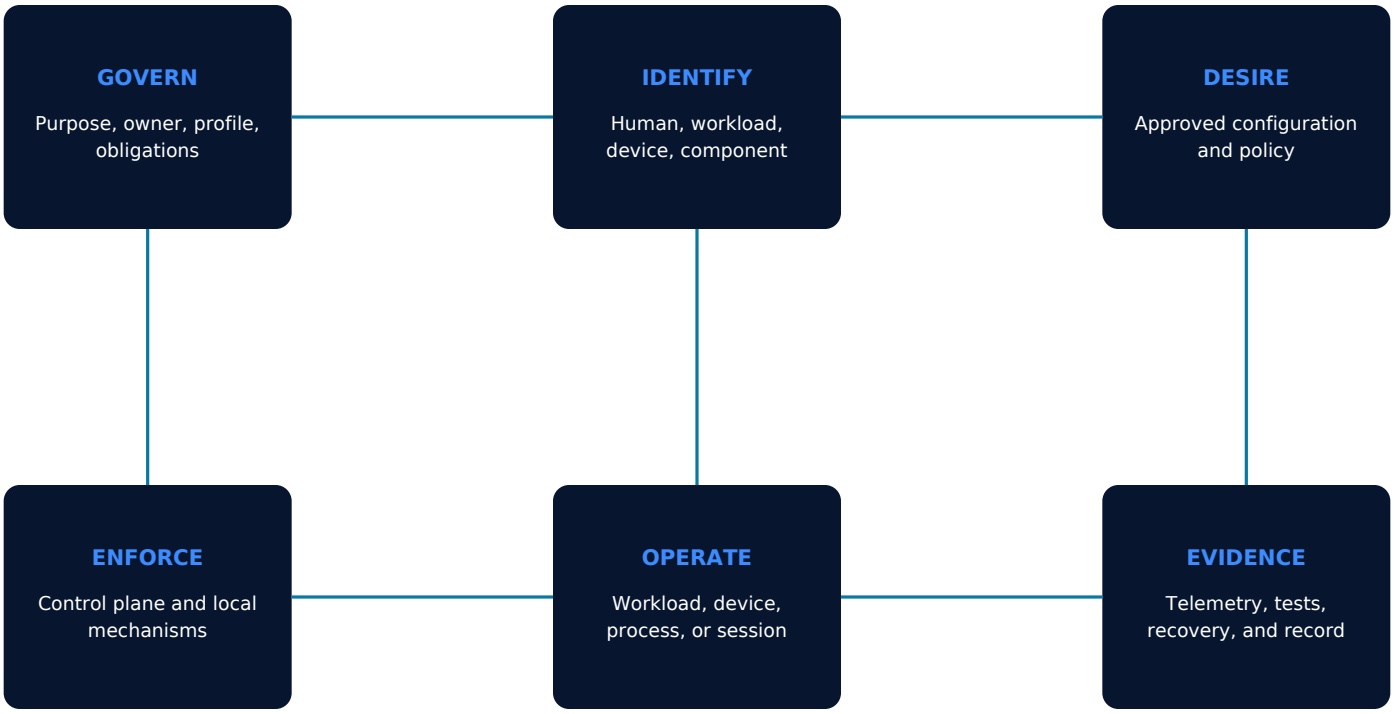
REASSESSMENT TRIGGER

A change, incident, drift signal, dependency revision, or time boundary requiring renewed evaluation.

MYTHOS-CLD-0006 / CLOUD ARCHITECTURE

3. Architecture and Trust Model

Separation of governance, management, enforcement, runtime, telemetry, and recovery



Mandatory trust boundaries

- Management planes MUST be isolated, strongly authenticated, monitored, and recoverable.
- Identity and policy MUST be evaluated at every provider, network, device, workload, and administrative transition.
- Runtime state MUST be compared with approved desired state and material drift MUST trigger response.
- Safety and continuity mechanisms MUST remain available when cloud, WAN, identity, DNS, time, carrier, or management dependencies fail.
- Evidence MUST be protected from the systems and administrators whose behavior it is intended to assess.

MYTHOS-CLD-0006 / CLOUD ARCHITECTURE

4. Governance and Accountability

Decision rights, separation of duties, and lifecycle ownership

ACCOUNTABLE OWNER

Accepts scope, risk tolerance, funding, and residual risk.

SYSTEM OWNER

Maintains architecture, inventory, operating boundaries, and change control.

SECURITY / PRIVACY

Defines threat, identity, data, isolation, monitoring, and incident requirements.

EVALUATION AUTHORITY

Designs representative tests, gates releases, and preserves reproducibility.

OPERATIONS

Maintains availability, rollback, recovery, evidence, and incident handling.

HUMAN OVERSIGHT

Reviews consequential decisions, exceptions, disputed outcomes, and harm signals.

DECISION PRINCIPLE

No single actor should be able to define the objective, grant authority, execute the consequential action, suppress evidence, and approve the result. Separation must be technical where consequence requires it.

MYTHOS-CLD-0006 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 01-04 of 18

MYTHOS-CLD-0006-C01 Alibaba Cloud Tenancy, Organization, and Account Hierarchy

The organization MUST define, implement, verify, and maintain alibaba cloud tenancy, organization, and account hierarchy for in-scope Alibaba Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0006-C02 Alibaba Cloud Landing Zone and Environment Segmentation

The organization MUST define, implement, verify, and maintain alibaba cloud landing zone and environment segmentation for in-scope Alibaba Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0006-C03 Alibaba Cloud Identity Federation and Privileged Administration

The organization MUST define, implement, verify, and maintain alibaba cloud identity federation and privileged administration for in-scope Alibaba Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-CLD-0006-C04 Alibaba Cloud Resource Policy and Preventive Guardrails

The organization MUST define, implement, verify, and maintain alibaba cloud resource policy and preventive guardrails for in-scope Alibaba Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-CLD-0006 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 05-08 of 18

MYTHOS-CLD-0006-C05 Alibaba Cloud Network Topology, Ingress, and Egress Control

The organization MUST define, implement, verify, and maintain alibaba cloud network topology, ingress, and egress control for in-scope Alibaba Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-CLD-0006-C06 Alibaba Cloud Encryption, Key Authority, and Secret Protection

The organization MUST define, implement, verify, and maintain alibaba cloud encryption, key authority, and secret protection for in-scope Alibaba Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-CLD-0006-C07 Alibaba Cloud Data Classification, Residency, and Service Selection

The organization MUST define, implement, verify, and maintain alibaba cloud data classification, residency, and service selection for in-scope Alibaba Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-CLD-0006-C08 Alibaba Cloud Workload Identity and Service Authorization

The organization MUST define, implement, verify, and maintain alibaba cloud workload identity and service authorization for in-scope Alibaba Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-CLD-0006 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 09-12 of 18

MYTHOS-CLD-0006-C09 Alibaba Cloud Declarative Provisioning and Change Control

The organization MUST define, implement, verify, and maintain alibaba cloud declarative provisioning and change control for in-scope Alibaba Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0006-C10 Alibaba Cloud Artifact, Image, and Software Supply-Chain Assurance

The organization MUST define, implement, verify, and maintain alibaba cloud artifact, image, and software supply-chain assurance for in-scope Alibaba Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0006-C11 Alibaba Cloud Logging, Telemetry, and Security Monitoring

The organization MUST define, implement, verify, and maintain alibaba cloud logging, telemetry, and security monitoring for in-scope Alibaba Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-CLD-0006-C12 Alibaba Cloud Reliability, Availability-Zone, and Region Strategy

The organization MUST define, implement, verify, and maintain alibaba cloud reliability, availability-zone, and region strategy for in-scope Alibaba Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-CLD-0006 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 13-15 of 18

MYTHOS-CLD-0006-C13 Alibaba Cloud Backup, Restoration, and Cyber-Recovery

The organization MUST define, implement, verify, and maintain alibaba cloud backup, restoration, and cyber-recovery for in-scope Alibaba Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-CLD-0006-C14 Alibaba Cloud Vulnerability, Configuration, and Drift Management

The organization MUST define, implement, verify, and maintain alibaba cloud vulnerability, configuration, and drift management for in-scope Alibaba Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-CLD-0006-C15 Alibaba Cloud Cost, Quota, Capacity, and FinOps Governance

The organization MUST define, implement, verify, and maintain alibaba cloud cost, quota, capacity, and finops governance for in-scope Alibaba Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-CLD-0006 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 16-18 of 18

MYTHOS-CLD-0006-C16 Alibaba Cloud Incident Response and Provider Escalation

The organization MUST define, implement, verify, and maintain alibaba cloud incident response and provider escalation for in-scope Alibaba Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-CLD-0006-C17 Alibaba Cloud Portability, Exit, and Data Disposition

The organization MUST define, implement, verify, and maintain alibaba cloud portability, exit, and data disposition for in-scope Alibaba Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0006-C18 Alibaba Cloud Evidence Package and Periodic Reassessment

The organization MUST define, implement, verify, and maintain alibaba cloud evidence package and periodic reassessment for in-scope Alibaba Cloud architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0006 / CLOUD ARCHITECTURE

6. Evidence and Assessment

Examine, interview, test, observe, restore, and trace

EXAMINE

Policies, inventories, diagrams, configuration, code, firmware, contracts, provider evidence, logs, exceptions, and lifecycle records.

INTERVIEW

Accountable owners, architects, engineers, operators, safety personnel, security teams, vendors, carriers, integrators, and responders.

TEST

Identity, segmentation, policy, negative paths, malformed input, capacity stress, dependency loss, failover, rollback, and revocation.

RESTORE

Independent restoration of configuration, data, service, device, controller, cluster, media, or browser state from protected evidence.

OBSERVE

Production-like performance, quality, drift, resource use, physical behavior, alarms, user impact, and incident execution.

TRACE

End-to-end linkage from approved intent and identity through change, runtime behavior, telemetry, outcome, exception, and review.

Evidence grades

A

Independently reproducible, complete, current, integrity-protected, and representative.

B

Reproduced by the implementing organization with strong traceability and controlled scope.

C

Partially reproduced or indirect; limitations, inherited responsibility, and uncertainty recorded.

D

Assertion, diagram, design intent, certificate, or vendor statement without reproduced behavior.

MYTHOS-CLD-0006 / CLOUD ARCHITECTURE

7. Assurance Views

Six perspectives required for a complete assurance claim

GOVERNANCE

Ownership, purpose, risk tolerance, policy, exception, and decision rights.

ARCHITECTURE

Boundaries, dependencies, failure modes, state, data flow, and portability.

SECURITY

Identity, authorization, isolation, secrets, abuse resistance, and incident response.

OPERATIONS

Reliability, performance, cost, observability, change, recovery, and support.

PRIVACY / RIGHTS

Purpose limitation, minimization, consent, access, correction, deletion, and egress.

HUMAN / SOCIETAL

Usability, accessibility, disclosure, contestability, impact, and human control.

Conformance requires a defensible position across every applicable view. A strong aggregate score cannot compensate for an unowned system, a failed mandatory gate, untested recovery, or evidence below the accepted grade.

MYTHOS-CLD-0006 / CLOUD ARCHITECTURE

7.2 Evaluation Metrics and Decision Gates

Measures must expose service behavior, control effectiveness, failure, uncertainty, and cost

SERVICE

Availability, correctness, coverage, quality, latency, throughput, and user or process outcome.

CONTROL

Unauthorized attempt, policy denial, drift, segmentation escape, privilege, and exception exposure.

RESILIENCE

Failover success, recovery time, data loss, safe degradation, restore validity, and dependency survival.

SECURITY

Exploitability, credential exposure, supply-chain integrity, attack detection, containment, and revocation.

CAPACITY

Utilization, saturation, queueing, radio or fabric headroom, thermal margin, quota, and forecast error.

OPERATIONS

Change failure, mean time to detect, repair, support burden, vendor escalation, and evidence completeness.

PRIVACY / SAFETY

Unauthorized egress, retention breach, consent coverage, unsafe action, physical consequence, and contestability.

LIFECYCLE

Patch debt, end-of-support exposure, portability, replacement time, retirement completion, and reassessment age.

Decision gate: thresholds MUST be declared before assessment, cover representative load and failure conditions, and identify mandatory safety, identity, recovery, and evidence failures that block promotion.

MYTHOS-CLD-0006 / CLOUD ARCHITECTURE

8. Failure Modes and Adversarial Scenarios

Challenge assumptions before the system is trusted with consequential work

CONTROL-PLANE COMPROMISE

An administrative or orchestration plane can alter broad production scope.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

PROVIDER OR REGION DEPENDENCY

A provider, region, identity, DNS, or service dependency fails without a tested alternative.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

CONFIGURATION DRIFT

Runtime state diverges from approved desired state without timely detection.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

SOVEREIGNTY BREACH

Data or keys move through an unapproved jurisdiction or subprocess.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

COST AND QUOTA RUNAWAY

Unbounded scaling, abuse, or configuration error exhausts budget or service limits.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

EXIT FAILURE

Workloads, data, identity, or evidence cannot be migrated within the required time.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

Adversarial testing MUST protect people and production systems. Use isolated environments, synthetic or minimized data, bounded authority, kill switches, explicit legal and ethical review, and documented stop conditions.

MYTHOS-CLD-0006 / CLOUD ARCHITECTURE

9. Implementation Profiles

Risk-proportional implementation without weakening mandatory boundaries

FOUNDATIONAL

TYPICAL SCOPE

Low consequence, limited autonomy, non-sensitive data, reversible outputs.

MINIMUM DEPTH

Named owner; inventory; basic evaluation; access control; logging; rollback; annual review.

ADVANCED

TYPICAL SCOPE

Business-critical workflow, sensitive data, multiple tools or providers, moderate autonomy.

MINIMUM DEPTH

Formal threat model; representative evaluation; approval gates; isolated execution; tested recovery; quarterly review.

HIGH ASSURANCE

TYPICAL SCOPE

Safety, security, regulated, financial, identity, or broadly consequential use.

MINIMUM DEPTH

Independent challenge; strongest identity; deterministic enforcement; comprehensive evidence; canary release; continuous monitoring.

Profile selection rule

Choose the highest profile triggered by consequence, autonomy, sensitivity, scale, irreversibility, external dependency, or inability to rapidly detect and recover. Tailoring may strengthen controls; it may not erase a mandatory gate without a controlled exception.

MYTHOS-CLD-0006 / CLOUD ARCHITECTURE

9.2 Operational Lifecycle

Evidence-bearing operation from concept through retirement

1. **CONCEPT**
   Purpose, owner, impact, boundaries
2. **DESIGN**
   Architecture, threat, data, identity
3. **BUILD**
   Controlled implementation and provenance
4. **EVALUATE**
   Representative, adversarial, recovery tests
5. **RELEASE**
   Approval, canary, rollback, evidence
6. **OPERATE**
   Observe, support, detect, respond
7. **CHANGE**
   Reassess, compare, gate, reverse
8. **RETIRE**
   Revoke, export, dispose, preserve

LIFECYCLE INVARIANT

Every stage MUST leave sufficient evidence for the next authority to understand what was intended, what changed, what was tested, what failed, what was accepted, what remains uncertain, and how to recover or exit.

MYTHOS-CLD-0006 / CLOUD ARCHITECTURE

10. Adoption Roadmap

A sequenced path from uncontrolled capability to evidence-bearing operation

0-30 DAYS

Establish ownership and truth

Inventory systems and dependencies; classify risk; freeze unsupported claims; identify immediate privilege, data, and evidence gaps.

31-90 DAYS

Create enforceable boundaries

Define policy; isolate execution; implement identity and approval gates; establish representative evaluation and baseline telemetry.

3-6 MONTHS

Prove recovery and operating control

Exercise failure, rollback, revocation, incident, provider exit, and state-recovery scenarios; mature evidence packaging.

6-12 MONTHS

Scale assurance

Automate control checks; expand workload coverage; independent challenge; portfolio metrics; controlled exception expiry; continuous reassessment.

Adoption is complete only when operating evidence demonstrates the selected profile in the actual deployment context. Documentation alone is not conformance.

MYTHOS-CLD-0006 / CLOUD ARCHITECTURE

10.2 Conformance and Exception Statement

What an assurance claim must contain

SYSTEM / SERVICE

Named, versioned capability and deployment boundary.

PROFILE

Foundational, Advanced, or High Assurance with trigger rationale.

SCOPE

Workloads, users, data, tools, regions, providers, and exclusions.

CRITERIA

Pass, partial, fail, not applicable, and exception status for every criterion.

EVIDENCE

Artifact references, evidence grade, date, environment, and reproducibility.

LIMITATIONS

Known failure modes, unsupported workloads, uncertainty, and residual risk.

EXCEPTIONS

Owner, rationale, compensating control, expiration, and approval.

VALIDITY

Assessment date, change boundary, review date, and reassessment triggers.

Prohibited claim: “compliant” without the named scope, profile, evidence date, limitations, exceptions, and authority accepting residual risk.

MYTHOS-CLD-0006 / CLOUD ARCHITECTURE

11. Source Authority and Reference Register

Primary sources informing interpretation; current obligations and provider behavior must be verified at assessment t

NIST CSF 2.0	**Cybersecurity Framework 2.0**

https://doi.org/10.6028/NIST.CSWP.29

Enterprise governance and cybersecurity outcome framework.

NIST SP 800-53 Rev. 5	**Security and Privacy Controls for Information Systems and Organizations**

https://doi.org/10.6028/NIST.SP.800-53r5

Broad control baseline for organizational systems and services.

CSA CCM v4	**Cloud Controls Matrix**

https://cloudsecurityalliance.org/research/cloud-controls-matrix

Cloud control domains and shared-responsibility assessment structure.

FinOps Framework	**FinOps Framework**

https://www.finops.org/framework/

Cloud value, cost, usage, ownership, and operating-practice guidance.

NIST SP 800-204A	**Building Secure Microservices-based Applications Using Service-Mesh Architecture**

https://doi.org/10.6028/NIST.SP.800-204A

Microservice and service-mesh security architecture.

CNCF Cloud Native Security	**Cloud Native Security Whitepaper**

https://github.com/cncf/tag-security/tree/main/security-whitepaper

Cloud-native lifecycle, supply-chain, runtime, and operations guidance.

Source currency rule: authorities, specifications, legal obligations, provider services, firmware, browser behavior, carrier requirements, and operational evidence MUST be checked at assessment time. This publication records a 2026-07-24 source snapshot.

MYTHOS-CLD-0006 / CLOUD ARCHITECTURE

11.2 Revision History and Decision Register

Permanent identity, controlled change, and publication integrity

VERSION	DATE	STATE	SUMMARY
1.0.0-candidate	2026-07-24	Candidate	Permanent-publication edition; source refresh; expanded criteria, evidence, assurance, and lifecycle guidance.
Next review	2027-01-24	Scheduled	Review source currency, threat evolution, operating evidence, adoption feedback, and proposed revisions.

Publication decisions

PERMANENT IDENTITY The document ID remains stable across revisions; batch or production sequence metadata is not public identity.

CHANGE CONTROL Normative changes require rationale, impact analysis, affected-criterion mapping, and approval.

SUPERSESSION A superseded edition remains traceable; current routes and catalogs identify the active edition.

CORRECTION Material errors require a recorded correction, affected-evidence analysis, and notification appropriate to impact.

ARCHIVAL INTEGRITY Published artifacts receive hashes, metadata, and a release manifest sufficient for independent verification.

END OF PERMANENT PUBLICATION / MYTHOS-CLD-0006

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY

ENGINEERING STANDARDS LIBRARY / CLOUD ARCHITECTURE AND COMPUTE

Kubernetes, Container Orchestration, and Operator Standard

Cluster governance, API and admission control, workload isolation, image provenance, operators, fleet lifecycle, recovery, and evidence.

PERMANENT DOCUMENT ID

MYTHOS-CLD-0007

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

Kubernetes, Container Orchestration, and Operator Standard

DOCUMENT ID
MYTHOS-CLD-0007

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

18	6	4	1
ATOMIC CRITERIA	ASSURANCE VIEWS	IMPLEMENTATION PROFILES	PERMANENT IDENTITY

Publication doctrine. This publication establishes durable requirements for its defined scope. Interpretation must preserve the permanent document identity, required evidence, controlled exceptions, source currency, and formal revision history.

INFRASTRUCTURE AND CONTROL TOPOLOGY

Cloud Architecture and Compute

A trustworthy Kubernetes and container orchestration system is a governed chain of ownership, identity, desired state, enforcement, workload or device behavior, telemetry, recovery, and independently reviewable evidence.

OWNERSHIP

Purpose, service boundary, accountable owner, suppliers, users, and retained responsibility remain explicit.

ENFORCEMENT

Identity, policy, segmentation, configuration, and local safety mechanisms constrain every trust transition.

CONTINUITY

Capacity, dependency loss, safe degradation, backup, restoration, rollback, and exit are tested before reliance.

EVIDENCE

Inventory, desired state, runtime observation, tests, incidents, exceptions, and change records form the assurance chain.

GOVERNED INFRASTRUCTURE FLOW

Reconciliation, detection, response, restoration, and controlled evolution

INTERPRETATION AND ASSURANCE

How to apply this publication

Normative intent

Requirements define outcomes and constraints. Implementations may vary, but evidence must demonstrate equivalent control.

Evidence over assertion

Vendor documentation and design intent are not equivalent to reproduced behavior. Record evidence grade and uncertainty.

Risk-proportional depth

Higher consequence, autonomy, sensitivity, and irreversibility require stronger isolation, review, verification, and recovery.

Controlled exception

Exceptions require an owner, rationale, compensating control, residual-risk acceptance, expiration, and reevaluation trigger.

Normalized assessment scale

0	1	2	3	4	5
ABSENT	AD HOC	PARTIAL	OPERATIONAL	ADVANCED	LEADING

A numeric score must never hide a failed mandatory gate, missing evidence, untested workload, or unacceptable residual risk.

INTERACTIVE NAVIGATION

Contents

Navigation doctrine

Bookmarks and internal links are conveniences. Permanent document identity, criterion identifiers, evidence references, and revision history remain authoritative.

MYTHOS-CLD-0007 / CLOUD ARCHITECTURE

0. Executive Mandate

Purpose, strategic outcome, and non-negotiable operating doctrine

MANDATE

Organizations adopting Kubernetes and container orchestration capabilities MUST define an owned service boundary, establish enforceable identity and configuration, protect data and safety, test failure and recovery, and preserve evidence sufficient for independent review.

Required outcomes

- Create a durable governance boundary for Kubernetes and container orchestration across design, acquisition, deployment, operation, change, and retirement.
- Require risk-proportional segmentation, identity, configuration, telemetry, resilience, recovery, and supplier controls.
- Prevent central automation, administrative tooling, or provider services from becoming unbounded authority.
- Prove operation with representative workloads, devices, failure modes, and restoration exercises.
- Define measurable conformance, exceptions, reassessment triggers, and a viable exit path.

Decision boundary: conformance supports a documented assurance claim for the named scope. It does not prove universal availability, safety, regulatory acceptance, vendor fitness, or immunity from cyber-physical failure.

MYTHOS-CLD-0007 / CLOUD ARCHITECTURE

1. Scope and Applicability

Boundary definition, audience, exclusions, and triggering conditions

IN SCOPE

- Architecture, acquisition, configuration, integration, and operation of Kubernetes and container orchestration capabilities.
- Human, workload, device, service, network, data, facility, provider, supplier, and evidence dependencies.
- Design, deployment, monitoring, support, incident response, recovery, migration, and retirement.
- On-premises, hosted, managed, carrier, manufacturer, integrator, and externally operated components.

OUT OF SCOPE

- Claims of legal, safety, carrier, or regulatory approval without the responsible authority.
- Vendor certification treated as proof of the adopter configuration or operating effectiveness.
- Theoretical or laboratory behavior presented as production evidence without a declared boundary.
- Inherited controls without documented scope, owner, period, limitations, and shared responsibility.

Applicability triggers

- The capability supports a business-critical, safety-relevant, regulated, public, or externally dependent service.
- The environment can expose sensitive data, identity, control, physical process, communications, or shared infrastructure.
- A management plane, automation system, carrier, provider, or supplier can affect broad operational scope.
- Failure, compromise, or change can be difficult to detect, reverse, isolate, or recover.
- The system depends on hardware, firmware, radio, browser, cloud, endpoint, IoT, OT, or real-time media behavior.

MYTHOS-CLD-0007 / CLOUD ARCHITECTURE

2. Definitions and Taxonomy

Controlled language for architecture, governance, and assessment

AUTHORITY

The right to approve, deny, constrain, or execute an action. Model output is not authority.

EVIDENCE

A reproducible source, trace, measurement, test, record, signed artifact, or documented observation supporting a claim.

ASSURANCE VIEW

A defined perspective such as governance, architecture, security, operations, privacy, or human oversight.

ATOMIC CRITERION

A uniquely identified requirement that can be assessed, evidenced, excepted, and revised independently.

IMPLEMENTATION PROFILE

A risk-proportional set of required depth, evidence, and gates for a class of use.

CONTROLLED EXCEPTION

A time-bounded deviation with owner, rationale, compensating control, residual risk, expiration, and review trigger.

DETERMINISTIC MECHANISM

A component whose inputs, policy, state transition, and side effects can be constrained and verified.

SOURCE AUTHORITY

The documented basis for treating a source as reliable for a defined claim, context, jurisdiction, and time.

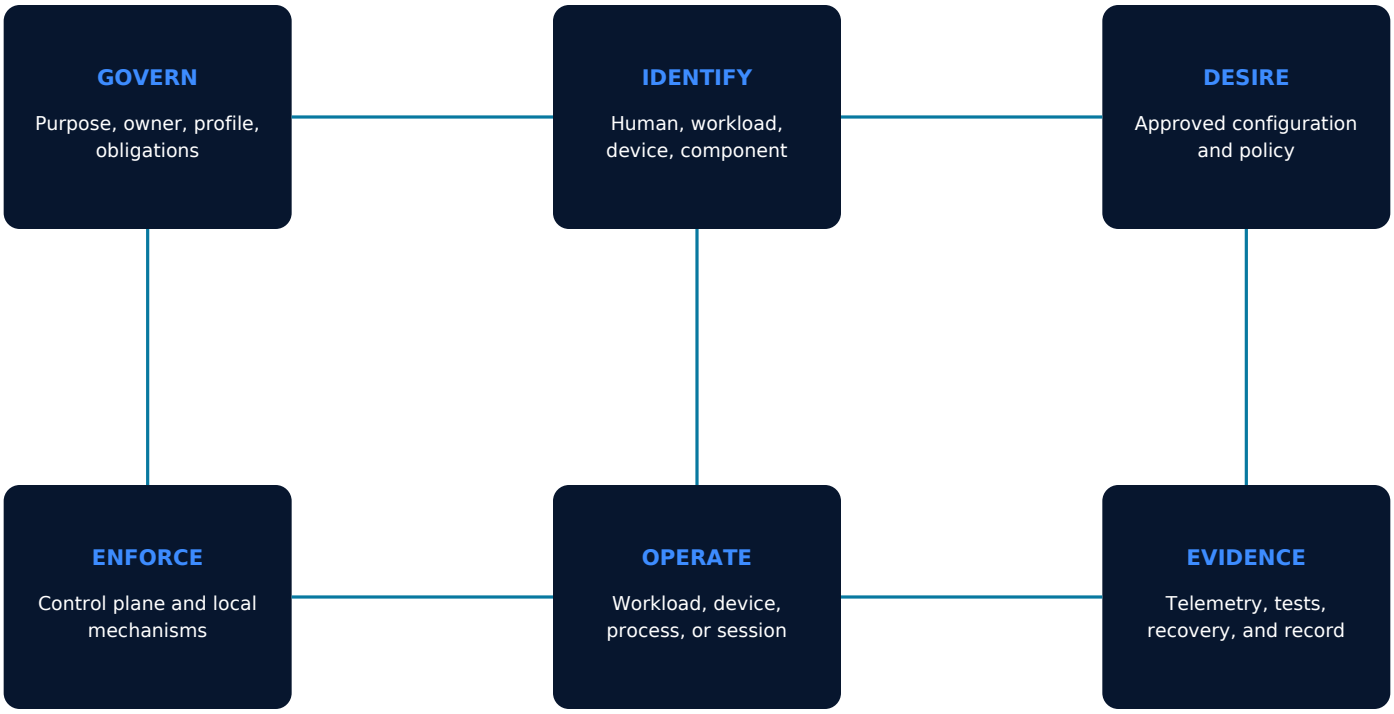
REASSESSMENT TRIGGER

A change, incident, drift signal, dependency revision, or time boundary requiring renewed evaluation.

MYTHOS-CLD-0007 / CLOUD ARCHITECTURE

3. Architecture and Trust Model

Separation of governance, management, enforcement, runtime, telemetry, and recovery



Mandatory trust boundaries

- Management planes MUST be isolated, strongly authenticated, monitored, and recoverable.
- Identity and policy MUST be evaluated at every provider, network, device, workload, and administrative transition.
- Runtime state MUST be compared with approved desired state and material drift MUST trigger response.
- Safety and continuity mechanisms MUST remain available when cloud, WAN, identity, DNS, time, carrier, or management dependencies fail.
- Evidence MUST be protected from the systems and administrators whose behavior it is intended to assess.

MYTHOS-CLD-0007 / CLOUD ARCHITECTURE

4. Governance and Accountability

Decision rights, separation of duties, and lifecycle ownership

ACCOUNTABLE OWNER

Accepts scope, risk tolerance, funding, and residual risk.

SYSTEM OWNER

Maintains architecture, inventory, operating boundaries, and change control.

SECURITY / PRIVACY

Defines threat, identity, data, isolation, monitoring, and incident requirements.

EVALUATION AUTHORITY

Designs representative tests, gates releases, and preserves reproducibility.

OPERATIONS

Maintains availability, rollback, recovery, evidence, and incident handling.

HUMAN OVERSIGHT

Reviews consequential decisions, exceptions, disputed outcomes, and harm signals.

DECISION PRINCIPLE

No single actor should be able to define the objective, grant authority, execute the consequential action, suppress evidence, and approve the result. Separation must be technical where consequence requires it.

MYTHOS-CLD-0007 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 01-04 of 18

MYTHOS-CLD-0007-C01 Cluster Inventory, Ownership, and Criticality

The organization MUST define, implement, verify, and maintain cluster inventory, ownership, and criticality for in-scope Kubernetes and container orchestration capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0007-C02 API Server Exposure and Administrative Access

The organization MUST define, implement, verify, and maintain api server exposure and administrative access for in-scope Kubernetes and container orchestration capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0007-C03 Authentication, RBAC, and Service-Account Governance

The organization MUST define, implement, verify, and maintain authentication, rbac, and service-account governance for in-scope Kubernetes and container orchestration capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-CLD-0007-C04 Admission Control and Policy Enforcement

The organization MUST define, implement, verify, and maintain admission control and policy enforcement for in-scope Kubernetes and container orchestration capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-CLD-0007 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 05-08 of 18

MYTHOS-CLD-0007-C05 Pod Security and Workload Isolation

The organization MUST define, implement, verify, and maintain pod security and workload isolation for in-scope Kubernetes and container orchestration capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-CLD-0007-C06 Network Policy, Service Mesh, and Egress Control

The organization MUST define, implement, verify, and maintain network policy, service mesh, and egress control for in-scope Kubernetes and container orchestration capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-CLD-0007-C07 Secrets, Certificates, and Key Lifecycle

The organization MUST define, implement, verify, and maintain secrets, certificates, and key lifecycle for in-scope Kubernetes and container orchestration capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-CLD-0007-C08 Image Provenance, Signing, and Registry Governance

The organization MUST define, implement, verify, and maintain image provenance, signing, and registry governance for in-scope Kubernetes and container orchestration capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-CLD-0007 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 09-12 of 18

MYTHOS-CLD-0007-C09 Node Operating System and Runtime Hardening

The organization MUST define, implement, verify, and maintain node operating system and runtime hardening for in-scope Kubernetes and container orchestration capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0007-C10 Storage Classes, Persistent Data, and Encryption

The organization MUST define, implement, verify, and maintain storage classes, persistent data, and encryption for in-scope Kubernetes and container orchestration capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0007-C11 Operator, Controller, and Custom-Resource Governance

The organization MUST define, implement, verify, and maintain operator, controller, and custom-resource governance for in-scope Kubernetes and container orchestration capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-CLD-0007-C12 Multi-Tenancy and Namespace Boundary Assurance

The organization MUST define, implement, verify, and maintain multi-tenancy and namespace boundary assurance for in-scope Kubernetes and container orchestration capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-CLD-0007 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 13-15 of 18

MYTHOS-CLD-0007-C13 GitOps, Desired State, and Drift Reconciliation

The organization MUST define, implement, verify, and maintain gitops, desired state, and drift reconciliation for in-scope Kubernetes and container orchestration capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-CLD-0007-C14 Observability, Audit, and Runtime Detection

The organization MUST define, implement, verify, and maintain observability, audit, and runtime detection for in-scope Kubernetes and container orchestration capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-CLD-0007-C15 Backup, Restore, and Cluster Reconstitution

The organization MUST define, implement, verify, and maintain backup, restore, and cluster reconstitution for in-scope Kubernetes and container orchestration capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-CLD-0007 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 16-18 of 18

MYTHOS-CLD-0007-C16 Version, Upgrade, and Deprecation Management

The organization MUST define, implement, verify, and maintain version, upgrade, and deprecation management for in-scope Kubernetes and container orchestration capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-CLD-0007-C17 Incident Containment and Forensic Preservation

The organization MUST define, implement, verify, and maintain incident containment and forensic preservation for in-scope Kubernetes and container orchestration capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0007-C18 Fleet Evidence and Periodic Conformance

The organization MUST define, implement, verify, and maintain fleet evidence and periodic conformance for in-scope Kubernetes and container orchestration capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0007 / CLOUD ARCHITECTURE

6. Evidence and Assessment

Examine, interview, test, observe, restore, and trace

EXAMINE

Policies, inventories, diagrams, configuration, code, firmware, contracts, provider evidence, logs, exceptions, and lifecycle records.

INTERVIEW

Accountable owners, architects, engineers, operators, safety personnel, security teams, vendors, carriers, integrators, and responders.

TEST

Identity, segmentation, policy, negative paths, malformed input, capacity stress, dependency loss, failover, rollback, and revocation.

RESTORE

Independent restoration of configuration, data, service, device, controller, cluster, media, or browser state from protected evidence.

OBSERVE

Production-like performance, quality, drift, resource use, physical behavior, alarms, user impact, and incident execution.

TRACE

End-to-end linkage from approved intent and identity through change, runtime behavior, telemetry, outcome, exception, and review.

Evidence grades

A

Independently reproducible, complete, current, integrity-protected, and representative.

B

Reproduced by the implementing organization with strong traceability and controlled scope.

C

Partially reproduced or indirect; limitations, inherited responsibility, and uncertainty recorded.

D

Assertion, diagram, design intent, certificate, or vendor statement without reproduced behavior.

MYTHOS-CLD-0007 / CLOUD ARCHITECTURE

7. Assurance Views

Six perspectives required for a complete assurance claim

GOVERNANCE

Ownership, purpose, risk tolerance, policy, exception, and decision rights.

ARCHITECTURE

Boundaries, dependencies, failure modes, state, data flow, and portability.

SECURITY

Identity, authorization, isolation, secrets, abuse resistance, and incident response.

OPERATIONS

Reliability, performance, cost, observability, change, recovery, and support.

PRIVACY / RIGHTS

Purpose limitation, minimization, consent, access, correction, deletion, and egress.

HUMAN / SOCIETAL

Usability, accessibility, disclosure, contestability, impact, and human control.

Conformance requires a defensible position across every applicable view. A strong aggregate score cannot compensate for an unowned system, a failed mandatory gate, untested recovery, or evidence below the accepted grade.

MYTHOS-CLD-0007 / CLOUD ARCHITECTURE

7.2 Evaluation Metrics and Decision Gates

Measures must expose service behavior, control effectiveness, failure, uncertainty, and cost

SERVICE

Availability, correctness, coverage, quality, latency, throughput, and user or process outcome.

CONTROL

Unauthorized attempt, policy denial, drift, segmentation escape, privilege, and exception exposure.

RESILIENCE

Failover success, recovery time, data loss, safe degradation, restore validity, and dependency survival.

SECURITY

Exploitability, credential exposure, supply-chain integrity, attack detection, containment, and revocation.

CAPACITY

Utilization, saturation, queueing, radio or fabric headroom, thermal margin, quota, and forecast error.

OPERATIONS

Change failure, mean time to detect, repair, support burden, vendor escalation, and evidence completeness.

PRIVACY / SAFETY

Unauthorized egress, retention breach, consent coverage, unsafe action, physical consequence, and contestability.

LIFECYCLE

Patch debt, end-of-support exposure, portability, replacement time, retirement completion, and reassessment age.

Decision gate: thresholds MUST be declared before assessment, cover representative load and failure conditions, and identify mandatory safety, identity, recovery, and evidence failures that block promotion.

MYTHOS-CLD-0007 / CLOUD ARCHITECTURE

8. Failure Modes and Adversarial Scenarios

Challenge assumptions before the system is trusted with consequential work

CONTROL-PLANE COMPROMISE

An administrative or orchestration plane can alter broad production scope.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

PROVIDER OR REGION DEPENDENCY

A provider, region, identity, DNS, or service dependency fails without a tested alternative.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

CONFIGURATION DRIFT

Runtime state diverges from approved desired state without timely detection.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

SOVEREIGNTY BREACH

Data or keys move through an unapproved jurisdiction or subprocess.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

COST AND QUOTA RUNAWAY

Unbounded scaling, abuse, or configuration error exhausts budget or service limits.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

EXIT FAILURE

Workloads, data, identity, or evidence cannot be migrated within the required time.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

Adversarial testing MUST protect people and production systems. Use isolated environments, synthetic or minimized data, bounded authority, kill switches, explicit legal and ethical review, and documented stop conditions.

MYTHOS-CLD-0007 / CLOUD ARCHITECTURE

9. Implementation Profiles

Risk-proportional implementation without weakening mandatory boundaries

FOUNDATIONAL

TYPICAL SCOPE

Low consequence, limited autonomy, non-sensitive data, reversible outputs.

MINIMUM DEPTH

Named owner; inventory; basic evaluation; access control; logging; rollback; annual review.

ADVANCED

TYPICAL SCOPE

Business-critical workflow, sensitive data, multiple tools or providers, moderate autonomy.

MINIMUM DEPTH

Formal threat model; representative evaluation; approval gates; isolated execution; tested recovery; quarterly review.

HIGH ASSURANCE

TYPICAL SCOPE

Safety, security, regulated, financial, identity, or broadly consequential use.

MINIMUM DEPTH

Independent challenge; strongest identity; deterministic enforcement; comprehensive evidence; canary release; continuous monitoring.

Profile selection rule

Choose the highest profile triggered by consequence, autonomy, sensitivity, scale, irreversibility, external dependency, or inability to rapidly detect and recover. Tailoring may strengthen controls; it may not erase a mandatory gate without a controlled exception.

MYTHOS-CLD-0007 / CLOUD ARCHITECTURE

9.2 Operational Lifecycle

Evidence-bearing operation from concept through retirement

1. **CONCEPT**
   Purpose, owner, impact, boundaries
2. **DESIGN**
   Architecture, threat, data, identity
3. **BUILD**
   Controlled implementation and provenance
4. **EVALUATE**
   Representative, adversarial, recovery tests
5. **RELEASE**
   Approval, canary, rollback, evidence
6. **OPERATE**
   Observe, support, detect, respond
7. **CHANGE**
   Reassess, compare, gate, reverse
8. **RETIRE**
   Revoke, export, dispose, preserve

LIFECYCLE INVARIANT

Every stage MUST leave sufficient evidence for the next authority to understand what was intended, what changed, what was tested, what failed, what was accepted, what remains uncertain, and how to recover or exit.

MYTHOS-CLD-0007 / CLOUD ARCHITECTURE

10. Adoption Roadmap

A sequenced path from uncontrolled capability to evidence-bearing operation

0-30 DAYS

Establish ownership and truth

Inventory systems and dependencies; classify risk; freeze unsupported claims; identify immediate privilege, data, and evidence gaps.

31-90 DAYS

Create enforceable boundaries

Define policy; isolate execution; implement identity and approval gates; establish representative evaluation and baseline telemetry.

3-6 MONTHS

Prove recovery and operating control

Exercise failure, rollback, revocation, incident, provider exit, and state-recovery scenarios; mature evidence packaging.

6-12 MONTHS

Scale assurance

Automate control checks; expand workload coverage; independent challenge; portfolio metrics; controlled exception expiry; continuous reassessment.

Adoption is complete only when operating evidence demonstrates the selected profile in the actual deployment context. Documentation alone is not conformance.

MYTHOS-CLD-0007 / CLOUD ARCHITECTURE

10.2 Conformance and Exception Statement

What an assurance claim must contain

SYSTEM / SERVICE

Named, versioned capability and deployment boundary.

PROFILE

Foundational, Advanced, or High Assurance with trigger rationale.

SCOPE

Workloads, users, data, tools, regions, providers, and exclusions.

CRITERIA

Pass, partial, fail, not applicable, and exception status for every criterion.

EVIDENCE

Artifact references, evidence grade, date, environment, and reproducibility.

LIMITATIONS

Known failure modes, unsupported workloads, uncertainty, and residual risk.

EXCEPTIONS

Owner, rationale, compensating control, expiration, and approval.

VALIDITY

Assessment date, change boundary, review date, and reassessment triggers.

Prohibited claim: “compliant” without the named scope, profile, evidence date, limitations, exceptions, and authority accepting residual risk.

MYTHOS-CLD-0007 / CLOUD ARCHITECTURE

11. Source Authority and Reference Register

Primary sources informing interpretation; current obligations and provider behavior must be verified at assessment t

NIST CSF 2.0	**Cybersecurity Framework 2.0**

https://doi.org/10.6028/NIST.CSWP.29

Enterprise governance and cybersecurity outcome framework.

NIST SP 800-53 Rev. 5	**Security and Privacy Controls for Information Systems and Organizations**

https://doi.org/10.6028/NIST.SP.800-53r5

Broad control baseline for organizational systems and services.

CSA CCM v4	**Cloud Controls Matrix**

https://cloudsecurityalliance.org/research/cloud-controls-matrix

Cloud control domains and shared-responsibility assessment structure.

FinOps Framework	**FinOps Framework**

https://www.finops.org/framework/

Cloud value, cost, usage, ownership, and operating-practice guidance.

NIST SP 800-204A	**Building Secure Microservices-based Applications Using Service-Mesh Architecture**

https://doi.org/10.6028/NIST.SP.800-204A

Microservice and service-mesh security architecture.

CNCF Cloud Native Security	**Cloud Native Security Whitepaper**

https://github.com/cncf/tag-security/tree/main/security-whitepaper

Cloud-native lifecycle, supply-chain, runtime, and operations guidance.

Source currency rule: authorities, specifications, legal obligations, provider services, firmware, browser behavior, carrier requirements, and operational evidence MUST be checked at assessment time. This publication records a 2026-07-24 source snapshot.

MYTHOS-CLD-0007 / CLOUD ARCHITECTURE

11.2 Revision History and Decision Register

Permanent identity, controlled change, and publication integrity

VERSION	DATE	STATE	SUMMARY
1.0.0-candidate	2026-07-24	Candidate	Permanent-publication edition; source refresh; expanded criteria, evidence, assurance, and lifecycle guidance.
Next review	2027-01-24	Scheduled	Review source currency, threat evolution, operating evidence, adoption feedback, and proposed revisions.

Publication decisions

PERMANENT IDENTITY The document ID remains stable across revisions; batch or production sequence metadata is not public identity.

CHANGE CONTROL Normative changes require rationale, impact analysis, affected-criterion mapping, and approval.

SUPERSESSION A superseded edition remains traceable; current routes and catalogs identify the active edition.

CORRECTION Material errors require a recorded correction, affected-evidence analysis, and notification appropriate to impact.

ARCHIVAL INTEGRITY Published artifacts receive hashes, metadata, and a release manifest sufficient for independent verification.

END OF PERMANENT PUBLICATION / MYTHOS-CLD-0007

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY

ENGINEERING STANDARDS LIBRARY / CLOUD ARCHITECTURE AND COMPUTE

Serverless and Event-Driven Compute Standard

Function and event architecture, identity, event contracts, replay, backpressure, observability, recovery, and cost controls.

PERMANENT DOCUMENT ID

MYTHOS-CLD-0008

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

Serverless and Event-Driven Compute Standard

DOCUMENT ID
MYTHOS-CLD-0008

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

18	6	4	1
ATOMIC CRITERIA	ASSURANCE VIEWS	IMPLEMENTATION PROFILES	PERMANENT IDENTITY

Publication doctrine. This publication establishes durable requirements for its defined scope. Interpretation must preserve the permanent document identity, required evidence, controlled exceptions, source currency, and formal revision history.

INFRASTRUCTURE AND CONTROL TOPOLOGY

Cloud Architecture and Compute

A trustworthy serverless and event-driven compute system is a governed chain of ownership, identity, desired state, enforcement, workload or device behavior, telemetry, recovery, and independently reviewable evidence.

OWNERSHIP

Purpose, service boundary, accountable owner, suppliers, users, and retained responsibility remain explicit.

ENFORCEMENT

Identity, policy, segmentation, configuration, and local safety mechanisms constrain every trust transition.

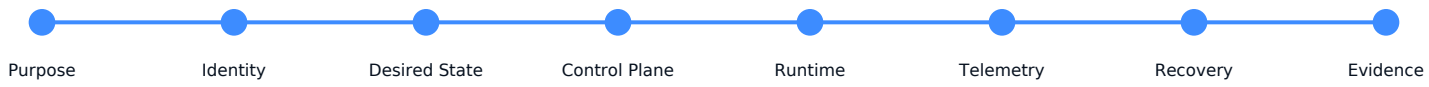
CONTINUITY

Capacity, dependency loss, safe degradation, backup, restoration, rollback, and exit are tested before reliance.

EVIDENCE

Inventory, desired state, runtime observation, tests, incidents, exceptions, and change records form the assurance chain.

GOVERNED INFRASTRUCTURE FLOW



Reconciliation, detection, response, restoration, and controlled evolution

INTERPRETATION AND ASSURANCE

How to apply this publication

Normative intent

Requirements define outcomes and constraints. Implementations may vary, but evidence must demonstrate equivalent control.

Evidence over assertion

Vendor documentation and design intent are not equivalent to reproduced behavior. Record evidence grade and uncertainty.

Risk-proportional depth

Higher consequence, autonomy, sensitivity, and irreversibility require stronger isolation, review, verification, and recovery.

Controlled exception

Exceptions require an owner, rationale, compensating control, residual-risk acceptance, expiration, and reevaluation trigger.

Normalized assessment scale

0	1	2	3	4	5
ABSENT	AD HOC	PARTIAL	OPERATIONAL	ADVANCED	LEADING

A numeric score must never hide a failed mandatory gate, missing evidence, untested workload, or unacceptable residual risk.

INTERACTIVE NAVIGATION

Contents

Navigation doctrine

Bookmarks and internal links are conveniences. Permanent document identity, criterion identifiers, evidence references, and revision history remain authoritative.

MYTHOS-CLD-0008 / CLOUD ARCHITECTURE

0. Executive Mandate

Purpose, strategic outcome, and non-negotiable operating doctrine

MANDATE

Organizations adopting serverless and event-driven compute capabilities MUST define an owned service boundary, establish enforceable identity and configuration, protect data and safety, test failure and recovery, and preserve evidence sufficient for independent review.

Required outcomes

- Create a durable governance boundary for serverless and event-driven compute across design, acquisition, deployment, operation, change, and retirement.
- Require risk-proportional segmentation, identity, configuration, telemetry, resilience, recovery, and supplier controls.
- Prevent central automation, administrative tooling, or provider services from becoming unbounded authority.
- Prove operation with representative workloads, devices, failure modes, and restoration exercises.
- Define measurable conformance, exceptions, reassessment triggers, and a viable exit path.

Decision boundary: conformance supports a documented assurance claim for the named scope. It does not prove universal availability, safety, regulatory acceptance, vendor fitness, or immunity from cyber-physical failure.

MYTHOS-CLD-0008 / CLOUD ARCHITECTURE

1. Scope and Applicability

Boundary definition, audience, exclusions, and triggering conditions

IN SCOPE

- Architecture, acquisition, configuration, integration, and operation of serverless and event-driven compute capabilities.
- Human, workload, device, service, network, data, facility, provider, supplier, and evidence dependencies.
- Design, deployment, monitoring, support, incident response, recovery, migration, and retirement.
- On-premises, hosted, managed, carrier, manufacturer, integrator, and externally operated components.

OUT OF SCOPE

- Claims of legal, safety, carrier, or regulatory approval without the responsible authority.
- Vendor certification treated as proof of the adopter configuration or operating effectiveness.
- Theoretical or laboratory behavior presented as production evidence without a declared boundary.
- Inherited controls without documented scope, owner, period, limitations, and shared responsibility.

Applicability triggers

- The capability supports a business-critical, safety-relevant, regulated, public, or externally dependent service.
- The environment can expose sensitive data, identity, control, physical process, communications, or shared infrastructure.
- A management plane, automation system, carrier, provider, or supplier can affect broad operational scope.
- Failure, compromise, or change can be difficult to detect, reverse, isolate, or recover.
- The system depends on hardware, firmware, radio, browser, cloud, endpoint, IoT, OT, or real-time media behavior.

MYTHOS-CLD-0008 / CLOUD ARCHITECTURE

2. Definitions and Taxonomy

Controlled language for architecture, governance, and assessment

AUTHORITY

The right to approve, deny, constrain, or execute an action. Model output is not authority.

EVIDENCE

A reproducible source, trace, measurement, test, record, signed artifact, or documented observation supporting a claim.

ASSURANCE VIEW

A defined perspective such as governance, architecture, security, operations, privacy, or human oversight.

ATOMIC CRITERION

A uniquely identified requirement that can be assessed, evidenced, excepted, and revised independently.

IMPLEMENTATION PROFILE

A risk-proportional set of required depth, evidence, and gates for a class of use.

CONTROLLED EXCEPTION

A time-bounded deviation with owner, rationale, compensating control, residual risk, expiration, and review trigger.

DETERMINISTIC MECHANISM

A component whose inputs, policy, state transition, and side effects can be constrained and verified.

SOURCE AUTHORITY

The documented basis for treating a source as reliable for a defined claim, context, jurisdiction, and time.

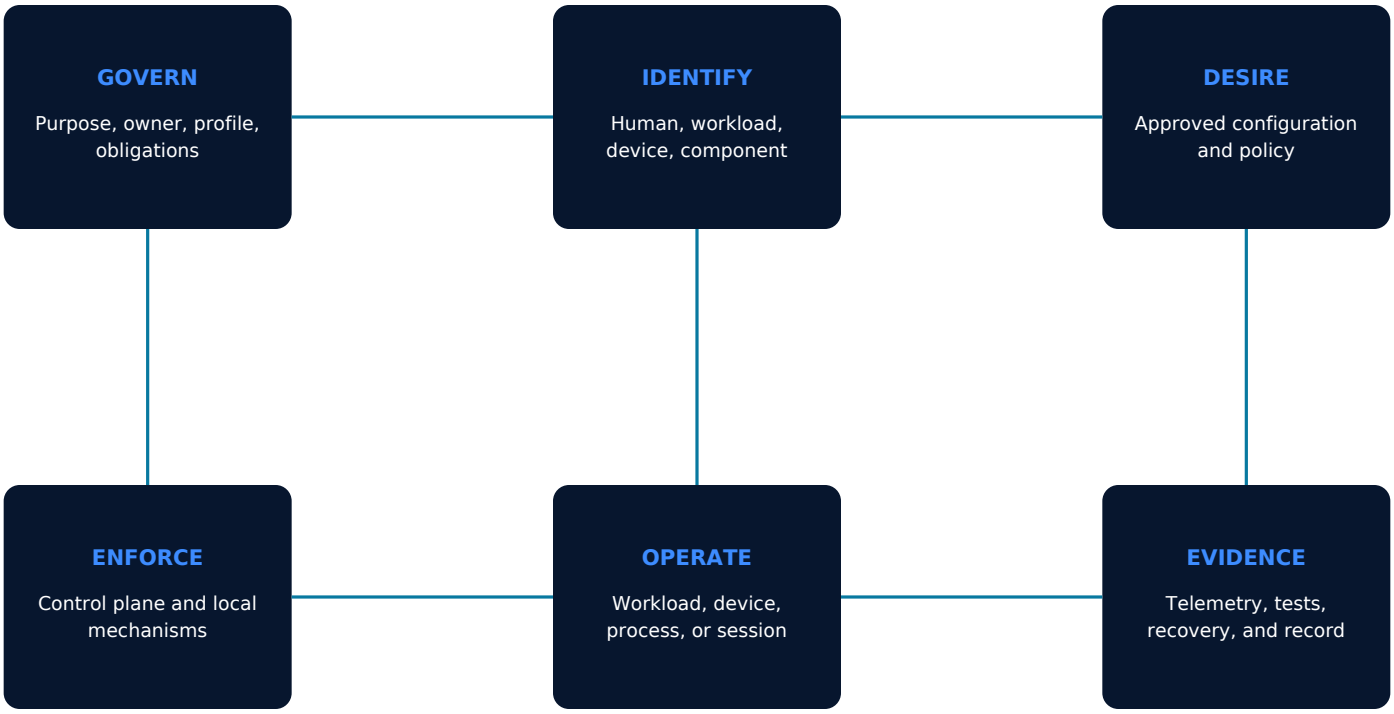
REASSESSMENT TRIGGER

A change, incident, drift signal, dependency revision, or time boundary requiring renewed evaluation.

MYTHOS-CLD-0008 / CLOUD ARCHITECTURE

3. Architecture and Trust Model

Separation of governance, management, enforcement, runtime, telemetry, and recovery



Mandatory trust boundaries

- Management planes MUST be isolated, strongly authenticated, monitored, and recoverable.
- Identity and policy MUST be evaluated at every provider, network, device, workload, and administrative transition.
- Runtime state MUST be compared with approved desired state and material drift MUST trigger response.
- Safety and continuity mechanisms MUST remain available when cloud, WAN, identity, DNS, time, carrier, or management dependencies fail.
- Evidence MUST be protected from the systems and administrators whose behavior it is intended to assess.

MYTHOS-CLD-0008 / CLOUD ARCHITECTURE

4. Governance and Accountability

Decision rights, separation of duties, and lifecycle ownership

ACCOUNTABLE OWNER

Accepts scope, risk tolerance, funding, and residual risk.

SYSTEM OWNER

Maintains architecture, inventory, operating boundaries, and change control.

SECURITY / PRIVACY

Defines threat, identity, data, isolation, monitoring, and incident requirements.

EVALUATION AUTHORITY

Designs representative tests, gates releases, and preserves reproducibility.

OPERATIONS

Maintains availability, rollback, recovery, evidence, and incident handling.

HUMAN OVERSIGHT

Reviews consequential decisions, exceptions, disputed outcomes, and harm signals.

DECISION PRINCIPLE

No single actor should be able to define the objective, grant authority, execute the consequential action, suppress evidence, and approve the result. Separation must be technical where consequence requires it.

MYTHOS-CLD-0008 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 01-04 of 18

MYTHOS-CLD-0008-C01 Function, Service, and Event Inventory

The organization MUST define, implement, verify, and maintain function, service, and event inventory for in-scope serverless and event-driven compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0008-C02 Event Contract and Schema Governance

The organization MUST define, implement, verify, and maintain event contract and schema governance for in-scope serverless and event-driven compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0008-C03 Trigger Identity and Authorization

The organization MUST define, implement, verify, and maintain trigger identity and authorization for in-scope serverless and event-driven compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-CLD-0008-C04 Workload Identity and Least Privilege

The organization MUST define, implement, verify, and maintain workload identity and least privilege for in-scope serverless and event-driven compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-CLD-0008 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 05-08 of 18

MYTHOS-CLD-0008-C05 Idempotency and Duplicate-Delivery Handling

The organization MUST define, implement, verify, and maintain idempotency and duplicate-delivery handling for in-scope serverless and event-driven compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-CLD-0008-C06 Ordering, Concurrency, and Consistency Semantics

The organization MUST define, implement, verify, and maintain ordering, concurrency, and consistency semantics for in-scope serverless and event-driven compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-CLD-0008-C07 Retry, Timeout, and Dead-Letter Governance

The organization MUST define, implement, verify, and maintain retry, timeout, and dead-letter governance for in-scope serverless and event-driven compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-CLD-0008-C08 Replay, Reprocessing, and Side-Effect Safety

The organization MUST define, implement, verify, and maintain replay, reprocessing, and side-effect safety for in-scope serverless and event-driven compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-CLD-0008 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 09-12 of 18

MYTHOS-CLD-0008-C09 Data Classification and Payload Protection

The organization MUST define, implement, verify, and maintain data classification and payload protection for in-scope serverless and event-driven compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0008-C10 Secrets, Configuration, and Environment Isolation

The organization MUST define, implement, verify, and maintain secrets, configuration, and environment isolation for in-scope serverless and event-driven compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0008-C11 Network Egress and Service Boundary Control

The organization MUST define, implement, verify, and maintain network egress and service boundary control for in-scope serverless and event-driven compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-CLD-0008-C12 Runtime, Dependency, and Artifact Provenance

The organization MUST define, implement, verify, and maintain runtime, dependency, and artifact provenance for in-scope serverless and event-driven compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-CLD-0008 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 13-15 of 18

MYTHOS-CLD-0008-C13 Scaling, Backpressure, and Saturation Protection

The organization MUST define, implement, verify, and maintain scaling, backpressure, and saturation protection for in-scope serverless and event-driven compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-CLD-0008-C14 Tracing, Correlation, and Operational Evidence

The organization MUST define, implement, verify, and maintain tracing, correlation, and operational evidence for in-scope serverless and event-driven compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-CLD-0008-C15 Cost, Quota, and Invocation-Abuse Controls

The organization MUST define, implement, verify, and maintain cost, quota, and invocation-abuse controls for in-scope serverless and event-driven compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-CLD-0008 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 16-18 of 18

MYTHOS-CLD-0008-C16 Failure Recovery and Regional Continuity

The organization MUST define, implement, verify, and maintain failure recovery and regional continuity for in-scope serverless and event-driven compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-CLD-0008-C17 Representative Testing and Failure Injection

The organization MUST define, implement, verify, and maintain representative testing and failure injection for in-scope serverless and event-driven compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0008-C18 Retirement, Event Drainage, and Data Disposition

The organization MUST define, implement, verify, and maintain retirement, event drainage, and data disposition for in-scope serverless and event-driven compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0008 / CLOUD ARCHITECTURE

6. Evidence and Assessment

Examine, interview, test, observe, restore, and trace

EXAMINE

Policies, inventories, diagrams, configuration, code, firmware, contracts, provider evidence, logs, exceptions, and lifecycle records.

INTERVIEW

Accountable owners, architects, engineers, operators, safety personnel, security teams, vendors, carriers, integrators, and responders.

TEST

Identity, segmentation, policy, negative paths, malformed input, capacity stress, dependency loss, failover, rollback, and revocation.

RESTORE

Independent restoration of configuration, data, service, device, controller, cluster, media, or browser state from protected evidence.

OBSERVE

Production-like performance, quality, drift, resource use, physical behavior, alarms, user impact, and incident execution.

TRACE

End-to-end linkage from approved intent and identity through change, runtime behavior, telemetry, outcome, exception, and review.

Evidence grades

A

Independently reproducible, complete, current, integrity-protected, and representative.

B

Reproduced by the implementing organization with strong traceability and controlled scope.

C

Partially reproduced or indirect; limitations, inherited responsibility, and uncertainty recorded.

D

Assertion, diagram, design intent, certificate, or vendor statement without reproduced behavior.

MYTHOS-CLD-0008 / CLOUD ARCHITECTURE

7. Assurance Views

Six perspectives required for a complete assurance claim

GOVERNANCE

Ownership, purpose, risk tolerance, policy, exception, and decision rights.

ARCHITECTURE

Boundaries, dependencies, failure modes, state, data flow, and portability.

SECURITY

Identity, authorization, isolation, secrets, abuse resistance, and incident response.

OPERATIONS

Reliability, performance, cost, observability, change, recovery, and support.

PRIVACY / RIGHTS

Purpose limitation, minimization, consent, access, correction, deletion, and egress.

HUMAN / SOCIETAL

Usability, accessibility, disclosure, contestability, impact, and human control.

Conformance requires a defensible position across every applicable view. A strong aggregate score cannot compensate for an unowned system, a failed mandatory gate, untested recovery, or evidence below the accepted grade.

MYTHOS-CLD-0008 / CLOUD ARCHITECTURE

7.2 Evaluation Metrics and Decision Gates

Measures must expose service behavior, control effectiveness, failure, uncertainty, and cost

SERVICE

Availability, correctness, coverage, quality, latency, throughput, and user or process outcome.

CONTROL

Unauthorized attempt, policy denial, drift, segmentation escape, privilege, and exception exposure.

RESILIENCE

Failover success, recovery time, data loss, safe degradation, restore validity, and dependency survival.

SECURITY

Exploitability, credential exposure, supply-chain integrity, attack detection, containment, and revocation.

CAPACITY

Utilization, saturation, queueing, radio or fabric headroom, thermal margin, quota, and forecast error.

OPERATIONS

Change failure, mean time to detect, repair, support burden, vendor escalation, and evidence completeness.

PRIVACY / SAFETY

Unauthorized egress, retention breach, consent coverage, unsafe action, physical consequence, and contestability.

LIFECYCLE

Patch debt, end-of-support exposure, portability, replacement time, retirement completion, and reassessment age.

Decision gate: thresholds MUST be declared before assessment, cover representative load and failure conditions, and identify mandatory safety, identity, recovery, and evidence failures that block promotion.

MYTHOS-CLD-0008 / CLOUD ARCHITECTURE

8. Failure Modes and Adversarial Scenarios

Challenge assumptions before the system is trusted with consequential work

CONTROL-PLANE COMPROMISE

An administrative or orchestration plane can alter broad production scope.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

PROVIDER OR REGION DEPENDENCY

A provider, region, identity, DNS, or service dependency fails without a tested alternative.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

CONFIGURATION DRIFT

Runtime state diverges from approved desired state without timely detection.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

SOVEREIGNTY BREACH

Data or keys move through an unapproved jurisdiction or subprocess.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

COST AND QUOTA RUNAWAY

Unbounded scaling, abuse, or configuration error exhausts budget or service limits.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

EXIT FAILURE

Workloads, data, identity, or evidence cannot be migrated within the required time.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

Adversarial testing MUST protect people and production systems. Use isolated environments, synthetic or minimized data, bounded authority, kill switches, explicit legal and ethical review, and documented stop conditions.

MYTHOS-CLD-0008 / CLOUD ARCHITECTURE

9. Implementation Profiles

Risk-proportional implementation without weakening mandatory boundaries

FOUNDATIONAL

TYPICAL SCOPE

Low consequence, limited autonomy, non-sensitive data, reversible outputs.

MINIMUM DEPTH

Named owner; inventory; basic evaluation; access control; logging; rollback; annual review.

ADVANCED

TYPICAL SCOPE

Business-critical workflow, sensitive data, multiple tools or providers, moderate autonomy.

MINIMUM DEPTH

Formal threat model; representative evaluation; approval gates; isolated execution; tested recovery; quarterly review.

HIGH ASSURANCE

TYPICAL SCOPE

Safety, security, regulated, financial, identity, or broadly consequential use.

MINIMUM DEPTH

Independent challenge; strongest identity; deterministic enforcement; comprehensive evidence; canary release; continuous monitoring.

Profile selection rule

Choose the highest profile triggered by consequence, autonomy, sensitivity, scale, irreversibility, external dependency, or inability to rapidly detect and recover. Tailoring may strengthen controls; it may not erase a mandatory gate without a controlled exception.

MYTHOS-CLD-0008 / CLOUD ARCHITECTURE

9.2 Operational Lifecycle

Evidence-bearing operation from concept through retirement

1. **CONCEPT**
   Purpose, owner, impact, boundaries
2. **DESIGN**
   Architecture, threat, data, identity
3. **BUILD**
   Controlled implementation and provenance
4. **EVALUATE**
   Representative, adversarial, recovery tests
5. **RELEASE**
   Approval, canary, rollback, evidence
6. **OPERATE**
   Observe, support, detect, respond
7. **CHANGE**
   Reassess, compare, gate, reverse
8. **RETIRE**
   Revoke, export, dispose, preserve

LIFECYCLE INVARIANT

Every stage MUST leave sufficient evidence for the next authority to understand what was intended, what changed, what was tested, what failed, what was accepted, what remains uncertain, and how to recover or exit.

MYTHOS-CLD-0008 / CLOUD ARCHITECTURE

10. Adoption Roadmap

A sequenced path from uncontrolled capability to evidence-bearing operation

0-30 DAYS

Establish ownership and truth

Inventory systems and dependencies; classify risk; freeze unsupported claims; identify immediate privilege, data, and evidence gaps.

31-90 DAYS

Create enforceable boundaries

Define policy; isolate execution; implement identity and approval gates; establish representative evaluation and baseline telemetry.

3-6 MONTHS

Prove recovery and operating control

Exercise failure, rollback, revocation, incident, provider exit, and state-recovery scenarios; mature evidence packaging.

6-12 MONTHS

Scale assurance

Automate control checks; expand workload coverage; independent challenge; portfolio metrics; controlled exception expiry; continuous reassessment.

Adoption is complete only when operating evidence demonstrates the selected profile in the actual deployment context. Documentation alone is not conformance.

MYTHOS-CLD-0008 / CLOUD ARCHITECTURE

10.2 Conformance and Exception Statement

What an assurance claim must contain

SYSTEM / SERVICE

Named, versioned capability and deployment boundary.

PROFILE

Foundational, Advanced, or High Assurance with trigger rationale.

SCOPE

Workloads, users, data, tools, regions, providers, and exclusions.

CRITERIA

Pass, partial, fail, not applicable, and exception status for every criterion.

EVIDENCE

Artifact references, evidence grade, date, environment, and reproducibility.

LIMITATIONS

Known failure modes, unsupported workloads, uncertainty, and residual risk.

EXCEPTIONS

Owner, rationale, compensating control, expiration, and approval.

VALIDITY

Assessment date, change boundary, review date, and reassessment triggers.

Prohibited claim: “compliant” without the named scope, profile, evidence date, limitations, exceptions, and authority accepting residual risk.

MYTHOS-CLD-0008 / CLOUD ARCHITECTURE

11. Source Authority and Reference Register

Primary sources informing interpretation; current obligations and provider behavior must be verified at assessment t

NIST CSF 2.0	**Cybersecurity Framework 2.0**

https://doi.org/10.6028/NIST.CSWP.29

Enterprise governance and cybersecurity outcome framework.

NIST SP 800-53 Rev. 5	**Security and Privacy Controls for Information Systems and Organizations**

https://doi.org/10.6028/NIST.SP.800-53r5

Broad control baseline for organizational systems and services.

CSA CCM v4	**Cloud Controls Matrix**

https://cloudsecurityalliance.org/research/cloud-controls-matrix

Cloud control domains and shared-responsibility assessment structure.

FinOps Framework	**FinOps Framework**

https://www.finops.org/framework/

Cloud value, cost, usage, ownership, and operating-practice guidance.

NIST SP 800-204A	**Building Secure Microservices-based Applications Using Service-Mesh Architecture**

https://doi.org/10.6028/NIST.SP.800-204A

Microservice and service-mesh security architecture.

CNCF Cloud Native Security	**Cloud Native Security Whitepaper**

https://github.com/cncf/tag-security/tree/main/security-whitepaper

Cloud-native lifecycle, supply-chain, runtime, and operations guidance.

Source currency rule: authorities, specifications, legal obligations, provider services, firmware, browser behavior, carrier requirements, and operational evidence MUST be checked at assessment time. This publication records a 2026-07-24 source snapshot.

MYTHOS-CLD-0008 / CLOUD ARCHITECTURE

11.2 Revision History and Decision Register

Permanent identity, controlled change, and publication integrity

VERSION	DATE	STATE	SUMMARY
1.0.0-candidate	2026-07-24	Candidate	Permanent-publication edition; source refresh; expanded criteria, evidence, assurance, and lifecycle guidance.
Next review	2027-01-24	Scheduled	Review source currency, threat evolution, operating evidence, adoption feedback, and proposed revisions.

Publication decisions

PERMANENT IDENTITY The document ID remains stable across revisions; batch or production sequence metadata is not public identity.

CHANGE CONTROL Normative changes require rationale, impact analysis, affected-criterion mapping, and approval.

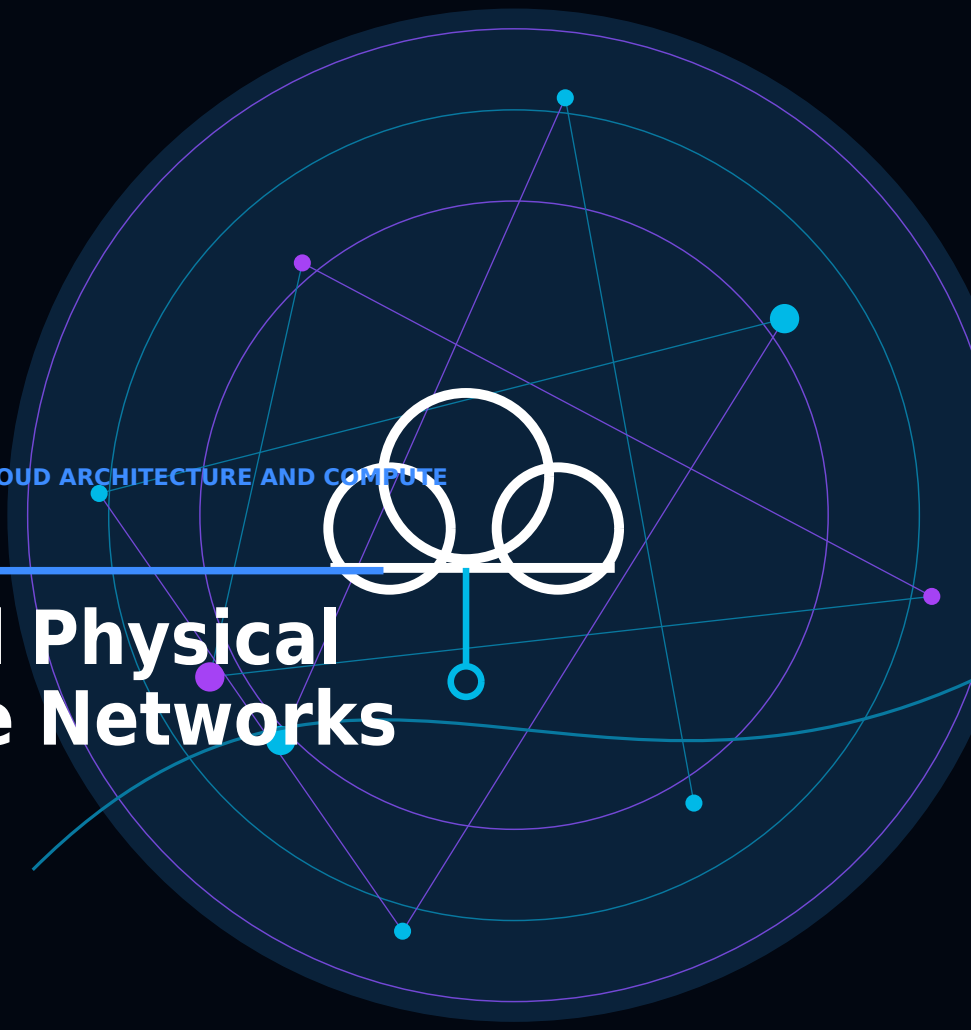
SUPERSESSION A superseded edition remains traceable; current routes and catalogs identify the active edition.

CORRECTION Material errors require a recorded correction, affected-evidence analysis, and notification appropriate to impact.

ARCHIVAL INTEGRITY Published artifacts receive hashes, metadata, and a release manifest sufficient for independent verification.

END OF PERMANENT PUBLICATION / MYTHOS-CLD-0008

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY



ENGINEERING STANDARDS LIBRARY / CLOUD ARCHITECTURE AND COMPUTE

Decentralized Physical Infrastructure Networks Standard

Distributed-node identity, attestation, measurement, incentives, governance, privacy, resilience, and service evidence.

PERMANENT DOCUMENT ID

MYTHOS-CLD-0009

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

Decentralized Physical Infrastructure Networks Standard

DOCUMENT ID
MYTHOS-CLD-0009

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

18	6	4	1
ATOMIC CRITERIA	ASSURANCE VIEWS	IMPLEMENTATION PROFILES	PERMANENT IDENTITY

Publication doctrine. This publication establishes durable requirements for its defined scope. Interpretation must preserve the permanent document identity, required evidence, controlled exceptions, source currency, and formal revision history.

INFRASTRUCTURE AND CONTROL TOPOLOGY

Cloud Architecture and Compute

A trustworthy decentralized physical infrastructure networks system is a governed chain of ownership, identity, desired state, enforcement, workload or device behavior, telemetry, recovery, and independently reviewable evidence.

OWNERSHIP

Purpose, service boundary, accountable owner, suppliers, users, and retained responsibility remain explicit.

ENFORCEMENT

Identity, policy, segmentation, configuration, and local safety mechanisms constrain every trust transition.

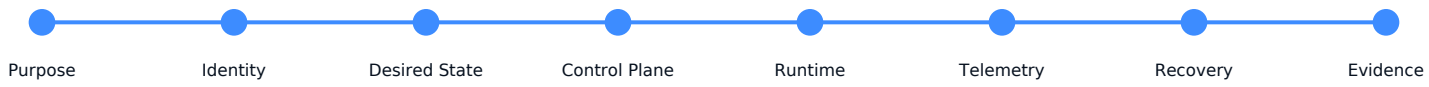
CONTINUITY

Capacity, dependency loss, safe degradation, backup, restoration, rollback, and exit are tested before reliance.

EVIDENCE

Inventory, desired state, runtime observation, tests, incidents, exceptions, and change records form the assurance chain.

GOVERNED INFRASTRUCTURE FLOW



Reconciliation, detection, response, restoration, and controlled evolution

INTERPRETATION AND ASSURANCE

How to apply this publication

Normative intent

Requirements define outcomes and constraints. Implementations may vary, but evidence must demonstrate equivalent control.

Evidence over assertion

Vendor documentation and design intent are not equivalent to reproduced behavior. Record evidence grade and uncertainty.

Risk-proportional depth

Higher consequence, autonomy, sensitivity, and irreversibility require stronger isolation, review, verification, and recovery.

Controlled exception

Exceptions require an owner, rationale, compensating control, residual-risk acceptance, expiration, and reevaluation trigger.

Normalized assessment scale

0	1	2	3	4	5
ABSENT	AD HOC	PARTIAL	OPERATIONAL	ADVANCED	LEADING

A numeric score must never hide a failed mandatory gate, missing evidence, untested workload, or unacceptable residual risk.

INTERACTIVE NAVIGATION

Contents

Navigation doctrine

Bookmarks and internal links are conveniences. Permanent document identity, criterion identifiers, evidence references, and revision history remain authoritative.

MYTHOS-CLD-0009 / CLOUD ARCHITECTURE

0. Executive Mandate

Purpose, strategic outcome, and non-negotiable operating doctrine

MANDATE

Organizations adopting decentralized physical infrastructure networks capabilities MUST define an owned service boundary, establish enforceable identity and configuration, protect data and safety, test failure and recovery, and preserve evidence sufficient for independent review.

Required outcomes

- Create a durable governance boundary for decentralized physical infrastructure networks across design, acquisition, deployment, operation, change, and retirement.
- Require risk-proportional segmentation, identity, configuration, telemetry, resilience, recovery, and supplier controls.
- Prevent central automation, administrative tooling, or provider services from becoming unbounded authority.
- Prove operation with representative workloads, devices, failure modes, and restoration exercises.
- Define measurable conformance, exceptions, reassessment triggers, and a viable exit path.

Decision boundary: conformance supports a documented assurance claim for the named scope. It does not prove universal availability, safety, regulatory acceptance, vendor fitness, or immunity from cyber-physical failure.

MYTHOS-CLD-0009 / CLOUD ARCHITECTURE

1. Scope and Applicability

Boundary definition, audience, exclusions, and triggering conditions

IN SCOPE

- Architecture, acquisition, configuration, integration, and operation of decentralized physical infrastructure networks capabilities.
- Human, workload, device, service, network, data, facility, provider, supplier, and evidence dependencies.
- Design, deployment, monitoring, support, incident response, recovery, migration, and retirement.
- On-premises, hosted, managed, carrier, manufacturer, integrator, and externally operated components.

OUT OF SCOPE

- Claims of legal, safety, carrier, or regulatory approval without the responsible authority.
- Vendor certification treated as proof of the adopter configuration or operating effectiveness.
- Theoretical or laboratory behavior presented as production evidence without a declared boundary.
- Inherited controls without documented scope, owner, period, limitations, and shared responsibility.

Applicability triggers

- The capability supports a business-critical, safety-relevant, regulated, public, or externally dependent service.
- The environment can expose sensitive data, identity, control, physical process, communications, or shared infrastructure.
- A management plane, automation system, carrier, provider, or supplier can affect broad operational scope.
- Failure, compromise, or change can be difficult to detect, reverse, isolate, or recover.
- The system depends on hardware, firmware, radio, browser, cloud, endpoint, IoT, OT, or real-time media behavior.

MYTHOS-CLD-0009 / CLOUD ARCHITECTURE

2. Definitions and Taxonomy

Controlled language for architecture, governance, and assessment

AUTHORITY

The right to approve, deny, constrain, or execute an action. Model output is not authority.

EVIDENCE

A reproducible source, trace, measurement, test, record, signed artifact, or documented observation supporting a claim.

ASSURANCE VIEW

A defined perspective such as governance, architecture, security, operations, privacy, or human oversight.

ATOMIC CRITERION

A uniquely identified requirement that can be assessed, evidenced, excepted, and revised independently.

IMPLEMENTATION PROFILE

A risk-proportional set of required depth, evidence, and gates for a class of use.

CONTROLLED EXCEPTION

A time-bounded deviation with owner, rationale, compensating control, residual risk, expiration, and review trigger.

DETERMINISTIC MECHANISM

A component whose inputs, policy, state transition, and side effects can be constrained and verified.

SOURCE AUTHORITY

The documented basis for treating a source as reliable for a defined claim, context, jurisdiction, and time.

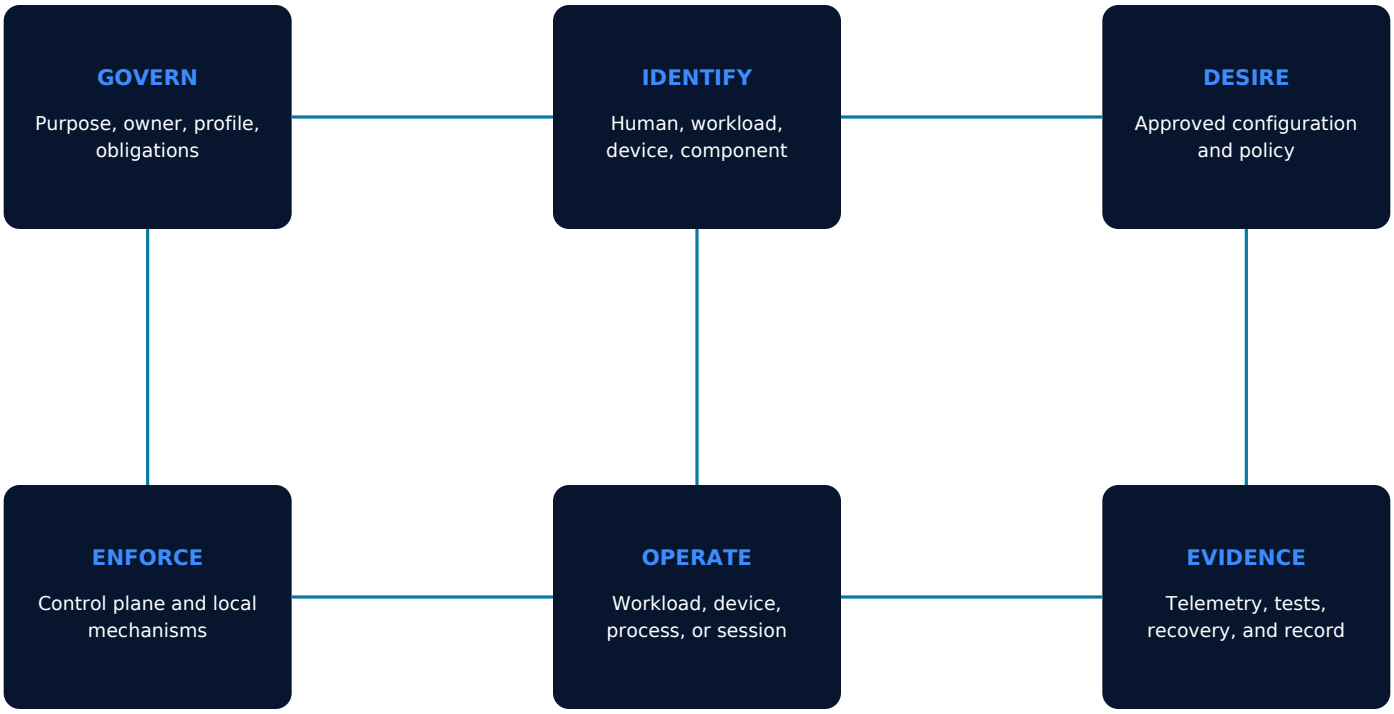
REASSESSMENT TRIGGER

A change, incident, drift signal, dependency revision, or time boundary requiring renewed evaluation.

MYTHOS-CLD-0009 / CLOUD ARCHITECTURE

3. Architecture and Trust Model

Separation of governance, management, enforcement, runtime, telemetry, and recovery



Mandatory trust boundaries

- Management planes MUST be isolated, strongly authenticated, monitored, and recoverable.
- Identity and policy MUST be evaluated at every provider, network, device, workload, and administrative transition.
- Runtime state MUST be compared with approved desired state and material drift MUST trigger response.
- Safety and continuity mechanisms MUST remain available when cloud, WAN, identity, DNS, time, carrier, or management dependencies fail.
- Evidence MUST be protected from the systems and administrators whose behavior it is intended to assess.

MYTHOS-CLD-0009 / CLOUD ARCHITECTURE

4. Governance and Accountability

Decision rights, separation of duties, and lifecycle ownership

ACCOUNTABLE OWNER

Accepts scope, risk tolerance, funding, and residual risk.

SYSTEM OWNER

Maintains architecture, inventory, operating boundaries, and change control.

SECURITY / PRIVACY

Defines threat, identity, data, isolation, monitoring, and incident requirements.

EVALUATION AUTHORITY

Designs representative tests, gates releases, and preserves reproducibility.

OPERATIONS

Maintains availability, rollback, recovery, evidence, and incident handling.

HUMAN OVERSIGHT

Reviews consequential decisions, exceptions, disputed outcomes, and harm signals.

DECISION PRINCIPLE

No single actor should be able to define the objective, grant authority, execute the consequential action, suppress evidence, and approve the result. Separation must be technical where consequence requires it.

MYTHOS-CLD-0009 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 01-04 of 18

MYTHOS-CLD-0009-C01 Service Purpose and Decentralization Rationale

The organization MUST define, implement, verify, and maintain service purpose and decentralization rationale for in-scope decentralized physical infrastructure networks capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0009-C02 Participant, Node, and Operator Identity

The organization MUST define, implement, verify, and maintain participant, node, and operator identity for in-scope decentralized physical infrastructure networks capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0009-C03 Node Enrollment, Attestation, and Revocation

The organization MUST define, implement, verify, and maintain node enrollment, attestation, and revocation for in-scope decentralized physical infrastructure networks capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-CLD-0009-C04 Sybil Resistance and Duplicate-Resource Detection

The organization MUST define, implement, verify, and maintain sybil resistance and duplicate-resource detection for in-scope decentralized physical infrastructure networks capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-CLD-0009 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 05-08 of 18

MYTHOS-CLD-0009-C05 Location, Capacity, and Service Measurement

The organization MUST define, implement, verify, and maintain location, capacity, and service measurement for in-scope decentralized physical infrastructure networks capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-CLD-0009-C06 Proof, Oracle, and Measurement Integrity

The organization MUST define, implement, verify, and maintain proof, oracle, and measurement integrity for in-scope decentralized physical infrastructure networks capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-CLD-0009-C07 Work Quality and Availability Verification

The organization MUST define, implement, verify, and maintain work quality and availability verification for in-scope decentralized physical infrastructure networks capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-CLD-0009-C08 Incentive, Reward, and Penalty Design

The organization MUST define, implement, verify, and maintain incentive, reward, and penalty design for in-scope decentralized physical infrastructure networks capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-CLD-0009 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 09-12 of 18

MYTHOS-CLD-0009-C09 Economic Attack and Collusion Resistance

The organization MUST define, implement, verify, and maintain economic attack and collusion resistance for in-scope decentralized physical infrastructure networks capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0009-C10 Software, Firmware, and Node Configuration

The organization MUST define, implement, verify, and maintain software, firmware, and node configuration for in-scope decentralized physical infrastructure networks capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0009-C11 Data Protection, Privacy, and Jurisdiction

The organization MUST define, implement, verify, and maintain data protection, privacy, and jurisdiction for in-scope decentralized physical infrastructure networks capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-CLD-0009-C12 Network Exposure and Administrative Access

The organization MUST define, implement, verify, and maintain network exposure and administrative access for in-scope decentralized physical infrastructure networks capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-CLD-0009 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 13-15 of 18

MYTHOS-CLD-0009-C13 Key Custody and Transaction Authorization

The organization MUST define, implement, verify, and maintain key custody and transaction authorization for in-scope decentralized physical infrastructure networks capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-CLD-0009-C14 Protocol Governance and Upgrade Authority

The organization MUST define, implement, verify, and maintain protocol governance and upgrade authority for in-scope decentralized physical infrastructure networks capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-CLD-0009-C15 Dispute, Fraud, and Appeals Process

The organization MUST define, implement, verify, and maintain dispute, fraud, and appeals process for in-scope decentralized physical infrastructure networks capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-CLD-0009 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 16-18 of 18

MYTHOS-CLD-0009-C16 Availability, Partition, and Recovery Behavior

The organization MUST define, implement, verify, and maintain availability, partition, and recovery behavior for in-scope decentralized physical infrastructure networks capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-CLD-0009-C17 Legal, Tax, Sanctions, and Supplier Boundaries

The organization MUST define, implement, verify, and maintain legal, tax, sanctions, and supplier boundaries for in-scope decentralized physical infrastructure networks capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0009-C18 Exit, Migration, and Evidence Preservation

The organization MUST define, implement, verify, and maintain exit, migration, and evidence preservation for in-scope decentralized physical infrastructure networks capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0009 / CLOUD ARCHITECTURE

6. Evidence and Assessment

Examine, interview, test, observe, restore, and trace

EXAMINE

Policies, inventories, diagrams, configuration, code, firmware, contracts, provider evidence, logs, exceptions, and lifecycle records.

INTERVIEW

Accountable owners, architects, engineers, operators, safety personnel, security teams, vendors, carriers, integrators, and responders.

TEST

Identity, segmentation, policy, negative paths, malformed input, capacity stress, dependency loss, failover, rollback, and revocation.

RESTORE

Independent restoration of configuration, data, service, device, controller, cluster, media, or browser state from protected evidence.

OBSERVE

Production-like performance, quality, drift, resource use, physical behavior, alarms, user impact, and incident execution.

TRACE

End-to-end linkage from approved intent and identity through change, runtime behavior, telemetry, outcome, exception, and review.

Evidence grades

A

Independently reproducible, complete, current, integrity-protected, and representative.

B

Reproduced by the implementing organization with strong traceability and controlled scope.

C

Partially reproduced or indirect; limitations, inherited responsibility, and uncertainty recorded.

D

Assertion, diagram, design intent, certificate, or vendor statement without reproduced behavior.

MYTHOS-CLD-0009 / CLOUD ARCHITECTURE

7. Assurance Views

Six perspectives required for a complete assurance claim

GOVERNANCE

Ownership, purpose, risk tolerance, policy, exception, and decision rights.

ARCHITECTURE

Boundaries, dependencies, failure modes, state, data flow, and portability.

SECURITY

Identity, authorization, isolation, secrets, abuse resistance, and incident response.

OPERATIONS

Reliability, performance, cost, observability, change, recovery, and support.

PRIVACY / RIGHTS

Purpose limitation, minimization, consent, access, correction, deletion, and egress.

HUMAN / SOCIETAL

Usability, accessibility, disclosure, contestability, impact, and human control.

Conformance requires a defensible position across every applicable view. A strong aggregate score cannot compensate for an unowned system, a failed mandatory gate, untested recovery, or evidence below the accepted grade.

MYTHOS-CLD-0009 / CLOUD ARCHITECTURE

7.2 Evaluation Metrics and Decision Gates

Measures must expose service behavior, control effectiveness, failure, uncertainty, and cost

SERVICE

Availability, correctness, coverage, quality, latency, throughput, and user or process outcome.

CONTROL

Unauthorized attempt, policy denial, drift, segmentation escape, privilege, and exception exposure.

RESILIENCE

Failover success, recovery time, data loss, safe degradation, restore validity, and dependency survival.

SECURITY

Exploitability, credential exposure, supply-chain integrity, attack detection, containment, and revocation.

CAPACITY

Utilization, saturation, queueing, radio or fabric headroom, thermal margin, quota, and forecast error.

OPERATIONS

Change failure, mean time to detect, repair, support burden, vendor escalation, and evidence completeness.

PRIVACY / SAFETY

Unauthorized egress, retention breach, consent coverage, unsafe action, physical consequence, and contestability.

LIFECYCLE

Patch debt, end-of-support exposure, portability, replacement time, retirement completion, and reassessment age.

Decision gate: thresholds MUST be declared before assessment, cover representative load and failure conditions, and identify mandatory safety, identity, recovery, and evidence failures that block promotion.

MYTHOS-CLD-0009 / CLOUD ARCHITECTURE

8. Failure Modes and Adversarial Scenarios

Challenge assumptions before the system is trusted with consequential work

CONTROL-PLANE COMPROMISE

An administrative or orchestration plane can alter broad production scope.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

PROVIDER OR REGION DEPENDENCY

A provider, region, identity, DNS, or service dependency fails without a tested alternative.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

CONFIGURATION DRIFT

Runtime state diverges from approved desired state without timely detection.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

SOVEREIGNTY BREACH

Data or keys move through an unapproved jurisdiction or subprocess.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

COST AND QUOTA RUNAWAY

Unbounded scaling, abuse, or configuration error exhausts budget or service limits.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

EXIT FAILURE

Workloads, data, identity, or evidence cannot be migrated within the required time.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

Adversarial testing MUST protect people and production systems. Use isolated environments, synthetic or minimized data, bounded authority, kill switches, explicit legal and ethical review, and documented stop conditions.

MYTHOS-CLD-0009 / CLOUD ARCHITECTURE

9. Implementation Profiles

Risk-proportional implementation without weakening mandatory boundaries

FOUNDATIONAL

TYPICAL SCOPE

Low consequence, limited autonomy, non-sensitive data, reversible outputs.

MINIMUM DEPTH

Named owner; inventory; basic evaluation; access control; logging; rollback; annual review.

ADVANCED

TYPICAL SCOPE

Business-critical workflow, sensitive data, multiple tools or providers, moderate autonomy.

MINIMUM DEPTH

Formal threat model; representative evaluation; approval gates; isolated execution; tested recovery; quarterly review.

HIGH ASSURANCE

TYPICAL SCOPE

Safety, security, regulated, financial, identity, or broadly consequential use.

MINIMUM DEPTH

Independent challenge; strongest identity; deterministic enforcement; comprehensive evidence; canary release; continuous monitoring.

Profile selection rule

Choose the highest profile triggered by consequence, autonomy, sensitivity, scale, irreversibility, external dependency, or inability to rapidly detect and recover. Tailoring may strengthen controls; it may not erase a mandatory gate without a controlled exception.

MYTHOS-CLD-0009 / CLOUD ARCHITECTURE

9.2 Operational Lifecycle

Evidence-bearing operation from concept through retirement

1. **CONCEPT**
   Purpose, owner, impact, boundaries
2. **DESIGN**
   Architecture, threat, data, identity
3. **BUILD**
   Controlled implementation and provenance
4. **EVALUATE**
   Representative, adversarial, recovery tests
5. **RELEASE**
   Approval, canary, rollback, evidence
6. **OPERATE**
   Observe, support, detect, respond
7. **CHANGE**
   Reassess, compare, gate, reverse
8. **RETIRE**
   Revoke, export, dispose, preserve

LIFECYCLE INVARIANT

Every stage MUST leave sufficient evidence for the next authority to understand what was intended, what changed, what was tested, what failed, what was accepted, what remains uncertain, and how to recover or exit.

MYTHOS-CLD-0009 / CLOUD ARCHITECTURE

10. Adoption Roadmap

A sequenced path from uncontrolled capability to evidence-bearing operation

0-30 DAYS

Establish ownership and truth

Inventory systems and dependencies; classify risk; freeze unsupported claims; identify immediate privilege, data, and evidence gaps.

31-90 DAYS

Create enforceable boundaries

Define policy; isolate execution; implement identity and approval gates; establish representative evaluation and baseline telemetry.

3-6 MONTHS

Prove recovery and operating control

Exercise failure, rollback, revocation, incident, provider exit, and state-recovery scenarios; mature evidence packaging.

6-12 MONTHS

Scale assurance

Automate control checks; expand workload coverage; independent challenge; portfolio metrics; controlled exception expiry; continuous reassessment.

Adoption is complete only when operating evidence demonstrates the selected profile in the actual deployment context. Documentation alone is not conformance.

MYTHOS-CLD-0009 / CLOUD ARCHITECTURE

10.2 Conformance and Exception Statement

What an assurance claim must contain

SYSTEM / SERVICE

Named, versioned capability and deployment boundary.

PROFILE

Foundational, Advanced, or High Assurance with trigger rationale.

SCOPE

Workloads, users, data, tools, regions, providers, and exclusions.

CRITERIA

Pass, partial, fail, not applicable, and exception status for every criterion.

EVIDENCE

Artifact references, evidence grade, date, environment, and reproducibility.

LIMITATIONS

Known failure modes, unsupported workloads, uncertainty, and residual risk.

EXCEPTIONS

Owner, rationale, compensating control, expiration, and approval.

VALIDITY

Assessment date, change boundary, review date, and reassessment triggers.

Prohibited claim: “compliant” without the named scope, profile, evidence date, limitations, exceptions, and authority accepting residual risk.

MYTHOS-CLD-0009 / CLOUD ARCHITECTURE

11. Source Authority and Reference Register

Primary sources informing interpretation; current obligations and provider behavior must be verified at assessment t

NIST CSF 2.0	**Cybersecurity Framework 2.0**

https://doi.org/10.6028/NIST.CSWP.29

Enterprise governance and cybersecurity outcome framework.

NIST SP 800-53 Rev. 5	**Security and Privacy Controls for Information Systems and Organizations**

https://doi.org/10.6028/NIST.SP.800-53r5

Broad control baseline for organizational systems and services.

CSA CCM v4	**Cloud Controls Matrix**

https://cloudsecurityalliance.org/research/cloud-controls-matrix

Cloud control domains and shared-responsibility assessment structure.

FinOps Framework	**FinOps Framework**

https://www.finops.org/framework/

Cloud value, cost, usage, ownership, and operating-practice guidance.

NIST SP 800-204A	**Building Secure Microservices-based Applications Using Service-Mesh Architecture**

https://doi.org/10.6028/NIST.SP.800-204A

Microservice and service-mesh security architecture.

CNCF Cloud Native Security	**Cloud Native Security Whitepaper**

https://github.com/cncf/tag-security/tree/main/security-whitepaper

Cloud-native lifecycle, supply-chain, runtime, and operations guidance.

Source currency rule: authorities, specifications, legal obligations, provider services, firmware, browser behavior, carrier requirements, and operational evidence MUST be checked at assessment time. This publication records a 2026-07-24 source snapshot.

MYTHOS-CLD-0009 / CLOUD ARCHITECTURE

11.2 Revision History and Decision Register

Permanent identity, controlled change, and publication integrity

VERSION	DATE	STATE	SUMMARY
1.0.0-candidate	2026-07-24	Candidate	Permanent-publication edition; source refresh; expanded criteria, evidence, assurance, and lifecycle guidance.
Next review	2027-01-24	Scheduled	Review source currency, threat evolution, operating evidence, adoption feedback, and proposed revisions.

Publication decisions

PERMANENT IDENTITY The document ID remains stable across revisions; batch or production sequence metadata is not public identity.

CHANGE CONTROL Normative changes require rationale, impact analysis, affected-criterion mapping, and approval.

SUPERSESSION A superseded edition remains traceable; current routes and catalogs identify the active edition.

CORRECTION Material errors require a recorded correction, affected-evidence analysis, and notification appropriate to impact.

ARCHIVAL INTEGRITY Published artifacts receive hashes, metadata, and a release manifest sufficient for independent verification.

END OF PERMANENT PUBLICATION / MYTHOS-CLD-0009

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY

ENGINEERING STANDARDS LIBRARY / CLOUD ARCHITECTURE AND COMPUTE

Private Cloud, Hypervisor, and Bare-Metal Standard

Private-cloud and virtualization governance, management-plane security, tenant isolation, hardware lifecycle, capacity, recovery, and evidence.

PERMANENT DOCUMENT ID

MYTHOS-CLD-0010

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

Private Cloud, Hypervisor, and Bare-Metal Standard

DOCUMENT ID
MYTHOS-CLD-0010

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

18	6	4	1
ATOMIC CRITERIA	ASSURANCE VIEWS	IMPLEMENTATION PROFILES	PERMANENT IDENTITY

Publication doctrine. This publication establishes durable requirements for its defined scope. Interpretation must preserve the permanent document identity, required evidence, controlled exceptions, source currency, and formal revision history.

INFRASTRUCTURE AND CONTROL TOPOLOGY

Cloud Architecture and Compute

A trustworthy private cloud, hypervisor, and bare metal system is a governed chain of ownership, identity, desired state, enforcement, workload or device behavior, telemetry, recovery, and independently reviewable evidence.

OWNERSHIP

Purpose, service boundary, accountable owner, suppliers, users, and retained responsibility remain explicit.

ENFORCEMENT

Identity, policy, segmentation, configuration, and local safety mechanisms constrain every trust transition.

CONTINUITY

Capacity, dependency loss, safe degradation, backup, restoration, rollback, and exit are tested before reliance.

EVIDENCE

Inventory, desired state, runtime observation, tests, incidents, exceptions, and change records form the assurance chain.

GOVERNED INFRASTRUCTURE FLOW

Reconciliation, detection, response, restoration, and controlled evolution

INTERPRETATION AND ASSURANCE

How to apply this publication

Normative intent

Requirements define outcomes and constraints. Implementations may vary, but evidence must demonstrate equivalent control.

Evidence over assertion

Vendor documentation and design intent are not equivalent to reproduced behavior. Record evidence grade and uncertainty.

Risk-proportional depth

Higher consequence, autonomy, sensitivity, and irreversibility require stronger isolation, review, verification, and recovery.

Controlled exception

Exceptions require an owner, rationale, compensating control, residual-risk acceptance, expiration, and reevaluation trigger.

Normalized assessment scale

0	1	2	3	4	5
ABSENT	AD HOC	PARTIAL	OPERATIONAL	ADVANCED	LEADING

A numeric score must never hide a failed mandatory gate, missing evidence, untested workload, or unacceptable residual risk.

INTERACTIVE NAVIGATION

Contents

Navigation doctrine

Bookmarks and internal links are conveniences. Permanent document identity, criterion identifiers, evidence references, and revision history remain authoritative.

MYTHOS-CLD-0010 / CLOUD ARCHITECTURE

0. Executive Mandate

Purpose, strategic outcome, and non-negotiable operating doctrine

MANDATE

Organizations adopting private cloud, hypervisor, and bare metal capabilities MUST define an owned service boundary, establish enforceable identity and configuration, protect data and safety, test failure and recovery, and preserve evidence sufficient for independent review.

Required outcomes

- Create a durable governance boundary for private cloud, hypervisor, and bare metal across design, acquisition, deployment, operation, change, and retirement.
- Require risk-proportional segmentation, identity, configuration, telemetry, resilience, recovery, and supplier controls.
- Prevent central automation, administrative tooling, or provider services from becoming unbounded authority.
- Prove operation with representative workloads, devices, failure modes, and restoration exercises.
- Define measurable conformance, exceptions, reassessment triggers, and a viable exit path.

Decision boundary: conformance supports a documented assurance claim for the named scope. It does not prove universal availability, safety, regulatory acceptance, vendor fitness, or immunity from cyber-physical failure.

MYTHOS-CLD-0010 / CLOUD ARCHITECTURE

1. Scope and Applicability

Boundary definition, audience, exclusions, and triggering conditions

IN SCOPE

- Architecture, acquisition, configuration, integration, and operation of private cloud, hypervisor, and bare metal capabilities.
- Human, workload, device, service, network, data, facility, provider, supplier, and evidence dependencies.
- Design, deployment, monitoring, support, incident response, recovery, migration, and retirement.
- On-premises, hosted, managed, carrier, manufacturer, integrator, and externally operated components.

OUT OF SCOPE

- Claims of legal, safety, carrier, or regulatory approval without the responsible authority.
- Vendor certification treated as proof of the adopter configuration or operating effectiveness.
- Theoretical or laboratory behavior presented as production evidence without a declared boundary.
- Inherited controls without documented scope, owner, period, limitations, and shared responsibility.

Applicability triggers

- The capability supports a business-critical, safety-relevant, regulated, public, or externally dependent service.
- The environment can expose sensitive data, identity, control, physical process, communications, or shared infrastructure.
- A management plane, automation system, carrier, provider, or supplier can affect broad operational scope.
- Failure, compromise, or change can be difficult to detect, reverse, isolate, or recover.
- The system depends on hardware, firmware, radio, browser, cloud, endpoint, IoT, OT, or real-time media behavior.

MYTHOS-CLD-0010 / CLOUD ARCHITECTURE

2. Definitions and Taxonomy

Controlled language for architecture, governance, and assessment

AUTHORITY

The right to approve, deny, constrain, or execute an action. Model output is not authority.

EVIDENCE

A reproducible source, trace, measurement, test, record, signed artifact, or documented observation supporting a claim.

ASSURANCE VIEW

A defined perspective such as governance, architecture, security, operations, privacy, or human oversight.

ATOMIC CRITERION

A uniquely identified requirement that can be assessed, evidenced, excepted, and revised independently.

IMPLEMENTATION PROFILE

A risk-proportional set of required depth, evidence, and gates for a class of use.

CONTROLLED EXCEPTION

A time-bounded deviation with owner, rationale, compensating control, residual risk, expiration, and review trigger.

DETERMINISTIC MECHANISM

A component whose inputs, policy, state transition, and side effects can be constrained and verified.

SOURCE AUTHORITY

The documented basis for treating a source as reliable for a defined claim, context, jurisdiction, and time.

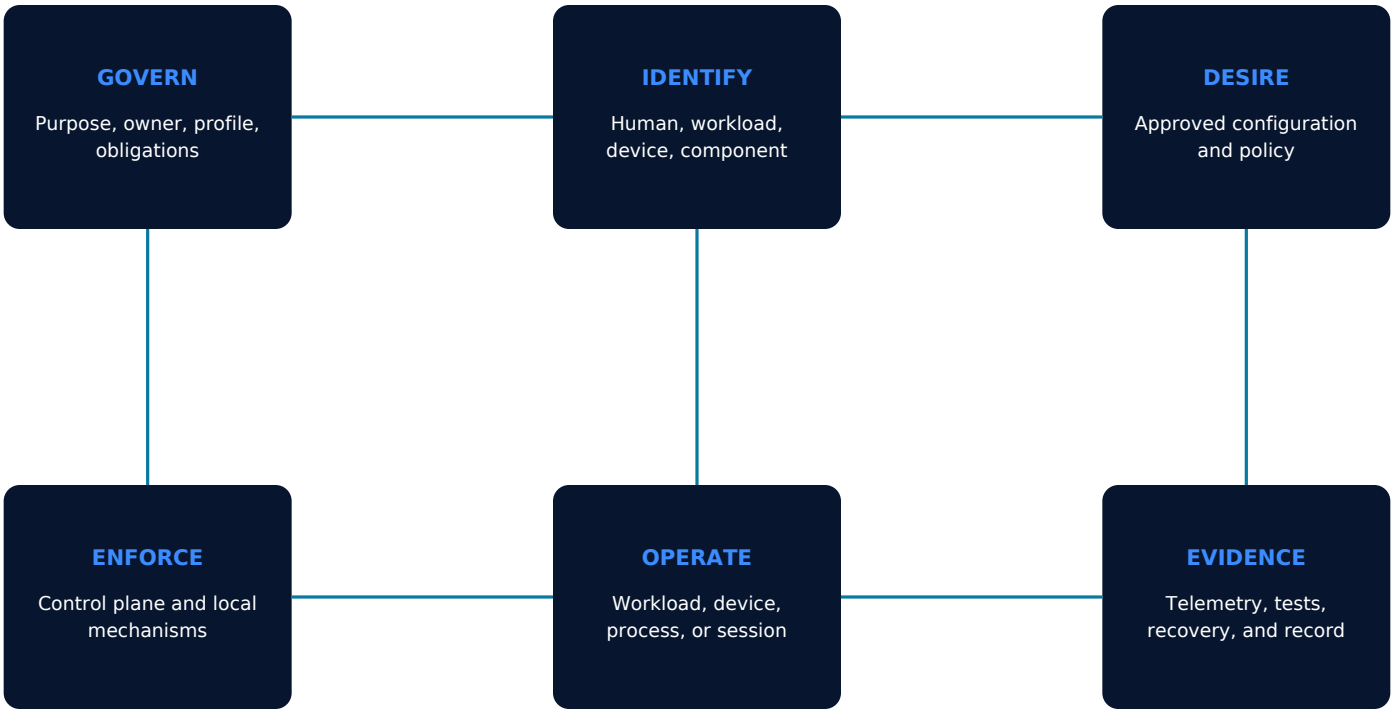
REASSESSMENT TRIGGER

A change, incident, drift signal, dependency revision, or time boundary requiring renewed evaluation.

MYTHOS-CLD-0010 / CLOUD ARCHITECTURE

3. Architecture and Trust Model

Separation of governance, management, enforcement, runtime, telemetry, and recovery



Mandatory trust boundaries

- Management planes MUST be isolated, strongly authenticated, monitored, and recoverable.
- Identity and policy MUST be evaluated at every provider, network, device, workload, and administrative transition.
- Runtime state MUST be compared with approved desired state and material drift MUST trigger response.
- Safety and continuity mechanisms MUST remain available when cloud, WAN, identity, DNS, time, carrier, or management dependencies fail.
- Evidence MUST be protected from the systems and administrators whose behavior it is intended to assess.

MYTHOS-CLD-0010 / CLOUD ARCHITECTURE

4. Governance and Accountability

Decision rights, separation of duties, and lifecycle ownership

ACCOUNTABLE OWNER

Accepts scope, risk tolerance, funding, and residual risk.

SYSTEM OWNER

Maintains architecture, inventory, operating boundaries, and change control.

SECURITY / PRIVACY

Defines threat, identity, data, isolation, monitoring, and incident requirements.

EVALUATION AUTHORITY

Designs representative tests, gates releases, and preserves reproducibility.

OPERATIONS

Maintains availability, rollback, recovery, evidence, and incident handling.

HUMAN OVERSIGHT

Reviews consequential decisions, exceptions, disputed outcomes, and harm signals.

DECISION PRINCIPLE

No single actor should be able to define the objective, grant authority, execute the consequential action, suppress evidence, and approve the result. Separation must be technical where consequence requires it.

MYTHOS-CLD-0010 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 01-04 of 18

MYTHOS-CLD-0010-C01 Private Cloud Service Boundary and Ownership

The organization MUST define, implement, verify, and maintain private cloud service boundary and ownership for in-scope private cloud, hypervisor, and bare metal capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0010-C02 Facility, Power, Cooling, and Physical Dependency

The organization MUST define, implement, verify, and maintain facility, power, cooling, and physical dependency for in-scope private cloud, hypervisor, and bare metal capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0010-C03 Hardware Inventory and Lifecycle Governance

The organization MUST define, implement, verify, and maintain hardware inventory and lifecycle governance for in-scope private cloud, hypervisor, and bare metal capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-CLD-0010-C04 BMC, Out-of-Band, and Management Network Isolation

The organization MUST define, implement, verify, and maintain bmc, out-of-band, and management network isolation for in-scope private cloud, hypervisor, and bare metal capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-CLD-0010 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 05-08 of 18

MYTHOS-CLD-0010-C05 Hypervisor and Management-Plane Hardening

The organization MUST define, implement, verify, and maintain hypervisor and management-plane hardening for in-scope private cloud, hypervisor, and bare metal capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-CLD-0010-C06 Administrative Identity and Privileged Access

The organization MUST define, implement, verify, and maintain administrative identity and privileged access for in-scope private cloud, hypervisor, and bare metal capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-CLD-0010-C07 Tenant, Project, and Resource Isolation

The organization MUST define, implement, verify, and maintain tenant, project, and resource isolation for in-scope private cloud, hypervisor, and bare metal capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-CLD-0010-C08 Virtual Network and Storage Segmentation

The organization MUST define, implement, verify, and maintain virtual network and storage segmentation for in-scope private cloud, hypervisor, and bare metal capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-CLD-0010 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 09-12 of 18

MYTHOS-CLD-0010-C09 Golden Images, Templates, and Artifact Provenance

The organization MUST define, implement, verify, and maintain golden images, templates, and artifact provenance for in-scope private cloud, hypervisor, and bare metal capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0010-C10 Bare-Metal Provisioning and Sanitization

The organization MUST define, implement, verify, and maintain bare-metal provisioning and sanitization for in-scope private cloud, hypervisor, and bare metal capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0010-C11 Patch, Firmware, and Compatibility Management

The organization MUST define, implement, verify, and maintain patch, firmware, and compatibility management for in-scope private cloud, hypervisor, and bare metal capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-CLD-0010-C12 Capacity, Overcommit, and Performance Governance

The organization MUST define, implement, verify, and maintain capacity, overcommit, and performance governance for in-scope private cloud, hypervisor, and bare metal capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-CLD-0010 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 13-15 of 18

MYTHOS-CLD-0010-C13 Telemetry, Configuration, and Drift Detection

The organization MUST define, implement, verify, and maintain telemetry, configuration, and drift detection for in-scope private cloud, hypervisor, and bare metal capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-CLD-0010-C14 Backup, Snapshot, and Restoration Assurance

The organization MUST define, implement, verify, and maintain backup, snapshot, and restoration assurance for in-scope private cloud, hypervisor, and bare metal capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-CLD-0010-C15 Cluster Failure and Management-Plane Recovery

The organization MUST define, implement, verify, and maintain cluster failure and management-plane recovery for in-scope private cloud, hypervisor, and bare metal capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-CLD-0010 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 16-18 of 18

MYTHOS-CLD-0010-C16 Incident Containment and Forensic Readiness

The organization MUST define, implement, verify, and maintain incident containment and forensic readiness for in-scope private cloud, hypervisor, and bare metal capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-CLD-0010-C17 Supplier Support, Spares, and End-of-Life Planning

The organization MUST define, implement, verify, and maintain supplier support, spares, and end-of-life planning for in-scope private cloud, hypervisor, and bare metal capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0010-C18 Reconstitution, Exit, and Evidence Package

The organization MUST define, implement, verify, and maintain reconstitution, exit, and evidence package for in-scope private cloud, hypervisor, and bare metal capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0010 / CLOUD ARCHITECTURE

6. Evidence and Assessment

Examine, interview, test, observe, restore, and trace

EXAMINE

Policies, inventories, diagrams, configuration, code, firmware, contracts, provider evidence, logs, exceptions, and lifecycle records.

INTERVIEW

Accountable owners, architects, engineers, operators, safety personnel, security teams, vendors, carriers, integrators, and responders.

TEST

Identity, segmentation, policy, negative paths, malformed input, capacity stress, dependency loss, failover, rollback, and revocation.

RESTORE

Independent restoration of configuration, data, service, device, controller, cluster, media, or browser state from protected evidence.

OBSERVE

Production-like performance, quality, drift, resource use, physical behavior, alarms, user impact, and incident execution.

TRACE

End-to-end linkage from approved intent and identity through change, runtime behavior, telemetry, outcome, exception, and review.

Evidence grades

A

Independently reproducible, complete, current, integrity-protected, and representative.

B

Reproduced by the implementing organization with strong traceability and controlled scope.

C

Partially reproduced or indirect; limitations, inherited responsibility, and uncertainty recorded.

D

Assertion, diagram, design intent, certificate, or vendor statement without reproduced behavior.

MYTHOS-CLD-0010 / CLOUD ARCHITECTURE

7. Assurance Views

Six perspectives required for a complete assurance claim

GOVERNANCE

Ownership, purpose, risk tolerance, policy, exception, and decision rights.

ARCHITECTURE

Boundaries, dependencies, failure modes, state, data flow, and portability.

SECURITY

Identity, authorization, isolation, secrets, abuse resistance, and incident response.

OPERATIONS

Reliability, performance, cost, observability, change, recovery, and support.

PRIVACY / RIGHTS

Purpose limitation, minimization, consent, access, correction, deletion, and egress.

HUMAN / SOCIETAL

Usability, accessibility, disclosure, contestability, impact, and human control.

Conformance requires a defensible position across every applicable view. A strong aggregate score cannot compensate for an unowned system, a failed mandatory gate, untested recovery, or evidence below the accepted grade.

MYTHOS-CLD-0010 / CLOUD ARCHITECTURE

7.2 Evaluation Metrics and Decision Gates

Measures must expose service behavior, control effectiveness, failure, uncertainty, and cost

SERVICE

Availability, correctness, coverage, quality, latency, throughput, and user or process outcome.

CONTROL

Unauthorized attempt, policy denial, drift, segmentation escape, privilege, and exception exposure.

RESILIENCE

Failover success, recovery time, data loss, safe degradation, restore validity, and dependency survival.

SECURITY

Exploitability, credential exposure, supply-chain integrity, attack detection, containment, and revocation.

CAPACITY

Utilization, saturation, queueing, radio or fabric headroom, thermal margin, quota, and forecast error.

OPERATIONS

Change failure, mean time to detect, repair, support burden, vendor escalation, and evidence completeness.

PRIVACY / SAFETY

Unauthorized egress, retention breach, consent coverage, unsafe action, physical consequence, and contestability.

LIFECYCLE

Patch debt, end-of-support exposure, portability, replacement time, retirement completion, and reassessment age.

Decision gate: thresholds MUST be declared before assessment, cover representative load and failure conditions, and identify mandatory safety, identity, recovery, and evidence failures that block promotion.

MYTHOS-CLD-0010 / CLOUD ARCHITECTURE

8. Failure Modes and Adversarial Scenarios

Challenge assumptions before the system is trusted with consequential work

CONTROL-PLANE COMPROMISE

An administrative or orchestration plane can alter broad production scope.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

PROVIDER OR REGION DEPENDENCY

A provider, region, identity, DNS, or service dependency fails without a tested alternative.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

CONFIGURATION DRIFT

Runtime state diverges from approved desired state without timely detection.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

SOVEREIGNTY BREACH

Data or keys move through an unapproved jurisdiction or subprocess.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

COST AND QUOTA RUNAWAY

Unbounded scaling, abuse, or configuration error exhausts budget or service limits.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

EXIT FAILURE

Workloads, data, identity, or evidence cannot be migrated within the required time.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

Adversarial testing MUST protect people and production systems. Use isolated environments, synthetic or minimized data, bounded authority, kill switches, explicit legal and ethical review, and documented stop conditions.

MYTHOS-CLD-0010 / CLOUD ARCHITECTURE

9. Implementation Profiles

Risk-proportional implementation without weakening mandatory boundaries

FOUNDATIONAL

TYPICAL SCOPE

Low consequence, limited autonomy, non-sensitive data, reversible outputs.

MINIMUM DEPTH

Named owner; inventory; basic evaluation; access control; logging; rollback; annual review.

ADVANCED

TYPICAL SCOPE

Business-critical workflow, sensitive data, multiple tools or providers, moderate autonomy.

MINIMUM DEPTH

Formal threat model; representative evaluation; approval gates; isolated execution; tested recovery; quarterly review.

HIGH ASSURANCE

TYPICAL SCOPE

Safety, security, regulated, financial, identity, or broadly consequential use.

MINIMUM DEPTH

Independent challenge; strongest identity; deterministic enforcement; comprehensive evidence; canary release; continuous monitoring.

Profile selection rule

Choose the highest profile triggered by consequence, autonomy, sensitivity, scale, irreversibility, external dependency, or inability to rapidly detect and recover. Tailoring may strengthen controls; it may not erase a mandatory gate without a controlled exception.

MYTHOS-CLD-0010 / CLOUD ARCHITECTURE

9.2 Operational Lifecycle

Evidence-bearing operation from concept through retirement

1. **CONCEPT**
   Purpose, owner, impact, boundaries
2. **DESIGN**
   Architecture, threat, data, identity
3. **BUILD**
   Controlled implementation and provenance
4. **EVALUATE**
   Representative, adversarial, recovery tests
5. **RELEASE**
   Approval, canary, rollback, evidence
6. **OPERATE**
   Observe, support, detect, respond
7. **CHANGE**
   Reassess, compare, gate, reverse
8. **RETIRE**
   Revoke, export, dispose, preserve

LIFECYCLE INVARIANT

Every stage MUST leave sufficient evidence for the next authority to understand what was intended, what changed, what was tested, what failed, what was accepted, what remains uncertain, and how to recover or exit.

MYTHOS-CLD-0010 / CLOUD ARCHITECTURE

10. Adoption Roadmap

A sequenced path from uncontrolled capability to evidence-bearing operation

0-30 DAYS

Establish ownership and truth

Inventory systems and dependencies; classify risk; freeze unsupported claims; identify immediate privilege, data, and evidence gaps.

31-90 DAYS

Create enforceable boundaries

Define policy; isolate execution; implement identity and approval gates; establish representative evaluation and baseline telemetry.

3-6 MONTHS

Prove recovery and operating control

Exercise failure, rollback, revocation, incident, provider exit, and state-recovery scenarios; mature evidence packaging.

6-12 MONTHS

Scale assurance

Automate control checks; expand workload coverage; independent challenge; portfolio metrics; controlled exception expiry; continuous reassessment.

Adoption is complete only when operating evidence demonstrates the selected profile in the actual deployment context. Documentation alone is not conformance.

MYTHOS-CLD-0010 / CLOUD ARCHITECTURE

10.2 Conformance and Exception Statement

What an assurance claim must contain

SYSTEM / SERVICE

Named, versioned capability and deployment boundary.

PROFILE

Foundational, Advanced, or High Assurance with trigger rationale.

SCOPE

Workloads, users, data, tools, regions, providers, and exclusions.

CRITERIA

Pass, partial, fail, not applicable, and exception status for every criterion.

EVIDENCE

Artifact references, evidence grade, date, environment, and reproducibility.

LIMITATIONS

Known failure modes, unsupported workloads, uncertainty, and residual risk.

EXCEPTIONS

Owner, rationale, compensating control, expiration, and approval.

VALIDITY

Assessment date, change boundary, review date, and reassessment triggers.

Prohibited claim: “compliant” without the named scope, profile, evidence date, limitations, exceptions, and authority accepting residual risk.

MYTHOS-CLD-0010 / CLOUD ARCHITECTURE

11. Source Authority and Reference Register

Primary sources informing interpretation; current obligations and provider behavior must be verified at assessment t

NIST CSF 2.0	**Cybersecurity Framework 2.0**

https://doi.org/10.6028/NIST.CSWP.29

Enterprise governance and cybersecurity outcome framework.

NIST SP 800-53 Rev. 5	**Security and Privacy Controls for Information Systems and Organizations**

https://doi.org/10.6028/NIST.SP.800-53r5

Broad control baseline for organizational systems and services.

CSA CCM v4	**Cloud Controls Matrix**

https://cloudsecurityalliance.org/research/cloud-controls-matrix

Cloud control domains and shared-responsibility assessment structure.

FinOps Framework	**FinOps Framework**

https://www.finops.org/framework/

Cloud value, cost, usage, ownership, and operating-practice guidance.

NIST SP 800-204A	**Building Secure Microservices-based Applications Using Service-Mesh Architecture**

https://doi.org/10.6028/NIST.SP.800-204A

Microservice and service-mesh security architecture.

CNCF Cloud Native Security	**Cloud Native Security Whitepaper**

https://github.com/cncf/tag-security/tree/main/security-whitepaper

Cloud-native lifecycle, supply-chain, runtime, and operations guidance.

Source currency rule: authorities, specifications, legal obligations, provider services, firmware, browser behavior, carrier requirements, and operational evidence MUST be checked at assessment time. This publication records a 2026-07-24 source snapshot.

MYTHOS-CLD-0010 / CLOUD ARCHITECTURE

11.2 Revision History and Decision Register

Permanent identity, controlled change, and publication integrity

VERSION	DATE	STATE	SUMMARY
1.0.0-candidate	2026-07-24	Candidate	Permanent-publication edition; source refresh; expanded criteria, evidence, assurance, and lifecycle guidance.
Next review	2027-01-24	Scheduled	Review source currency, threat evolution, operating evidence, adoption feedback, and proposed revisions.

Publication decisions

PERMANENT IDENTITY The document ID remains stable across revisions; batch or production sequence metadata is not public identity.

CHANGE CONTROL Normative changes require rationale, impact analysis, affected-criterion mapping, and approval.

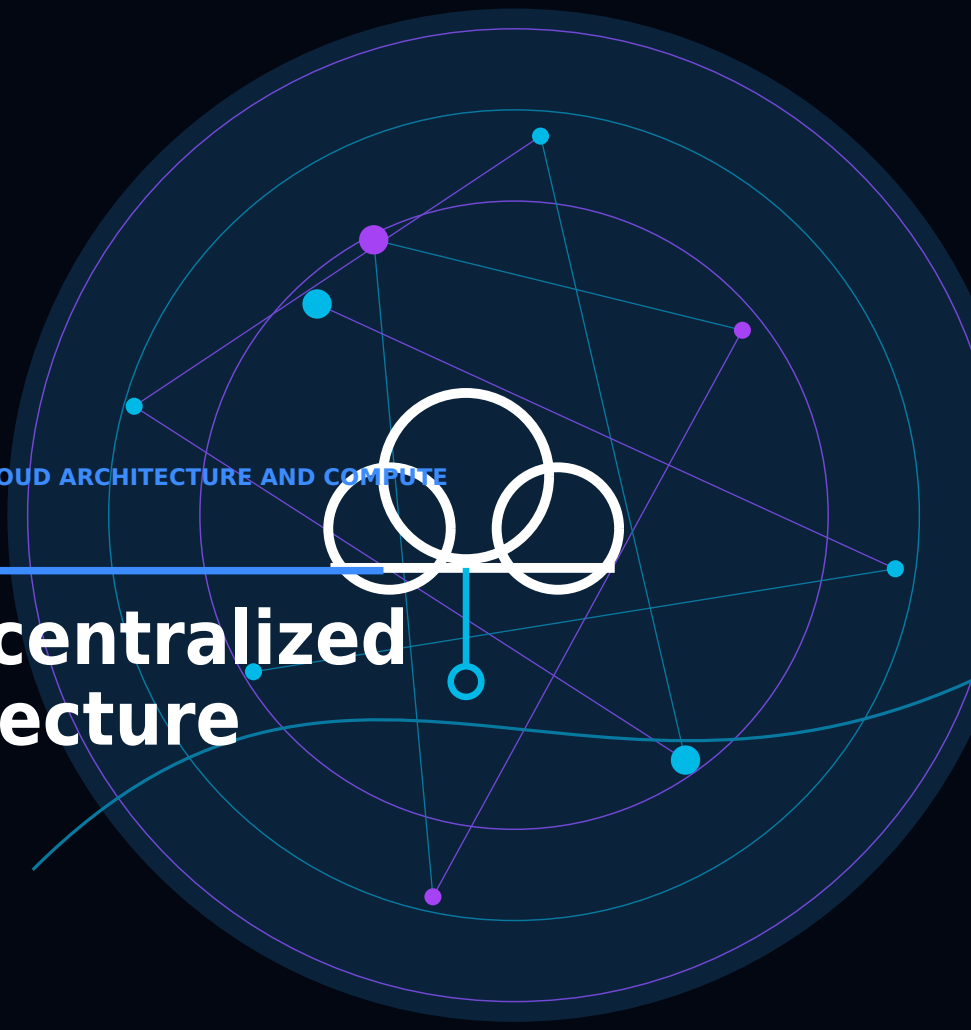
SUPERSESSION A superseded edition remains traceable; current routes and catalogs identify the active edition.

CORRECTION Material errors require a recorded correction, affected-evidence analysis, and notification appropriate to impact.

ARCHIVAL INTEGRITY Published artifacts receive hashes, metadata, and a release manifest sufficient for independent verification.

END OF PERMANENT PUBLICATION / MYTHOS-CLD-0010

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY



ENGINEERING STANDARDS LIBRARY / CLOUD ARCHITECTURE AND COMPUTE

Web3 and Decentralized Ledger Architecture Standard

Distributed-ledger architecture, smart contracts, custody, oracle, bridge, governance, monitoring, recovery, and retirement.

PERMANENT DOCUMENT ID

MYTHOS-CLD-0011

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

Web3 and Decentralized Ledger Architecture Standard

DOCUMENT ID
MYTHOS-CLD-0011

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

18	6	4	1
ATOMIC CRITERIA	ASSURANCE VIEWS	IMPLEMENTATION PROFILES	PERMANENT IDENTITY

Publication doctrine. This publication establishes durable requirements for its defined scope. Interpretation must preserve the permanent document identity, required evidence, controlled exceptions, source currency, and formal revision history.

INFRASTRUCTURE AND CONTROL TOPOLOGY

Cloud Architecture and Compute

A trustworthy Web3 and decentralized ledger architecture system is a governed chain of ownership, identity, desired state, enforcement, workload or device behavior, telemetry, recovery, and independently reviewable evidence.

OWNERSHIP

Purpose, service boundary, accountable owner, suppliers, users, and retained responsibility remain explicit.

ENFORCEMENT

Identity, policy, segmentation, configuration, and local safety mechanisms constrain every trust transition.

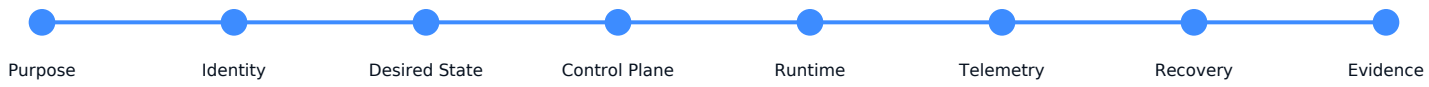
CONTINUITY

Capacity, dependency loss, safe degradation, backup, restoration, rollback, and exit are tested before reliance.

EVIDENCE

Inventory, desired state, runtime observation, tests, incidents, exceptions, and change records form the assurance chain.

GOVERNED INFRASTRUCTURE FLOW



Reconciliation, detection, response, restoration, and controlled evolution

INTERPRETATION AND ASSURANCE

How to apply this publication

Normative intent

Requirements define outcomes and constraints. Implementations may vary, but evidence must demonstrate equivalent control.

Evidence over assertion

Vendor documentation and design intent are not equivalent to reproduced behavior. Record evidence grade and uncertainty.

Risk-proportional depth

Higher consequence, autonomy, sensitivity, and irreversibility require stronger isolation, review, verification, and recovery.

Controlled exception

Exceptions require an owner, rationale, compensating control, residual-risk acceptance, expiration, and reevaluation trigger.

Normalized assessment scale

0	1	2	3	4	5
ABSENT	AD HOC	PARTIAL	OPERATIONAL	ADVANCED	LEADING

A numeric score must never hide a failed mandatory gate, missing evidence, untested workload, or unacceptable residual risk.

INTERACTIVE NAVIGATION

Contents

Navigation doctrine

Bookmarks and internal links are conveniences. Permanent document identity, criterion identifiers, evidence references, and revision history remain authoritative.

MYTHOS-CLD-0011 / CLOUD ARCHITECTURE

0. Executive Mandate

Purpose, strategic outcome, and non-negotiable operating doctrine

MANDATE

Organizations adopting Web3 and decentralized ledger architecture capabilities MUST define an owned service boundary, establish enforceable identity and configuration, protect data and safety, test failure and recovery, and preserve evidence sufficient for independent review.

Required outcomes

- Create a durable governance boundary for Web3 and decentralized ledger architecture across design, acquisition, deployment, operation, change, and retirement.
- Require risk-proportional segmentation, identity, configuration, telemetry, resilience, recovery, and supplier controls.
- Prevent central automation, administrative tooling, or provider services from becoming unbounded authority.
- Prove operation with representative workloads, devices, failure modes, and restoration exercises.
- Define measurable conformance, exceptions, reassessment triggers, and a viable exit path.

Decision boundary: conformance supports a documented assurance claim for the named scope. It does not prove universal availability, safety, regulatory acceptance, vendor fitness, or immunity from cyber-physical failure.

MYTHOS-CLD-0011 / CLOUD ARCHITECTURE

1. Scope and Applicability

Boundary definition, audience, exclusions, and triggering conditions

IN SCOPE

- Architecture, acquisition, configuration, integration, and operation of Web3 and decentralized ledger architecture capabilities.
- Human, workload, device, service, network, data, facility, provider, supplier, and evidence dependencies.
- Design, deployment, monitoring, support, incident response, recovery, migration, and retirement.
- On-premises, hosted, managed, carrier, manufacturer, integrator, and externally operated components.

OUT OF SCOPE

- Claims of legal, safety, carrier, or regulatory approval without the responsible authority.
- Vendor certification treated as proof of the adopter configuration or operating effectiveness.
- Theoretical or laboratory behavior presented as production evidence without a declared boundary.
- Inherited controls without documented scope, owner, period, limitations, and shared responsibility.

Applicability triggers

- The capability supports a business-critical, safety-relevant, regulated, public, or externally dependent service.
- The environment can expose sensitive data, identity, control, physical process, communications, or shared infrastructure.
- A management plane, automation system, carrier, provider, or supplier can affect broad operational scope.
- Failure, compromise, or change can be difficult to detect, reverse, isolate, or recover.
- The system depends on hardware, firmware, radio, browser, cloud, endpoint, IoT, OT, or real-time media behavior.

MYTHOS-CLD-0011 / CLOUD ARCHITECTURE

2. Definitions and Taxonomy

Controlled language for architecture, governance, and assessment

AUTHORITY

The right to approve, deny, constrain, or execute an action. Model output is not authority.

EVIDENCE

A reproducible source, trace, measurement, test, record, signed artifact, or documented observation supporting a claim.

ASSURANCE VIEW

A defined perspective such as governance, architecture, security, operations, privacy, or human oversight.

ATOMIC CRITERION

A uniquely identified requirement that can be assessed, evidenced, excepted, and revised independently.

IMPLEMENTATION PROFILE

A risk-proportional set of required depth, evidence, and gates for a class of use.

CONTROLLED EXCEPTION

A time-bounded deviation with owner, rationale, compensating control, residual risk, expiration, and review trigger.

DETERMINISTIC MECHANISM

A component whose inputs, policy, state transition, and side effects can be constrained and verified.

SOURCE AUTHORITY

The documented basis for treating a source as reliable for a defined claim, context, jurisdiction, and time.

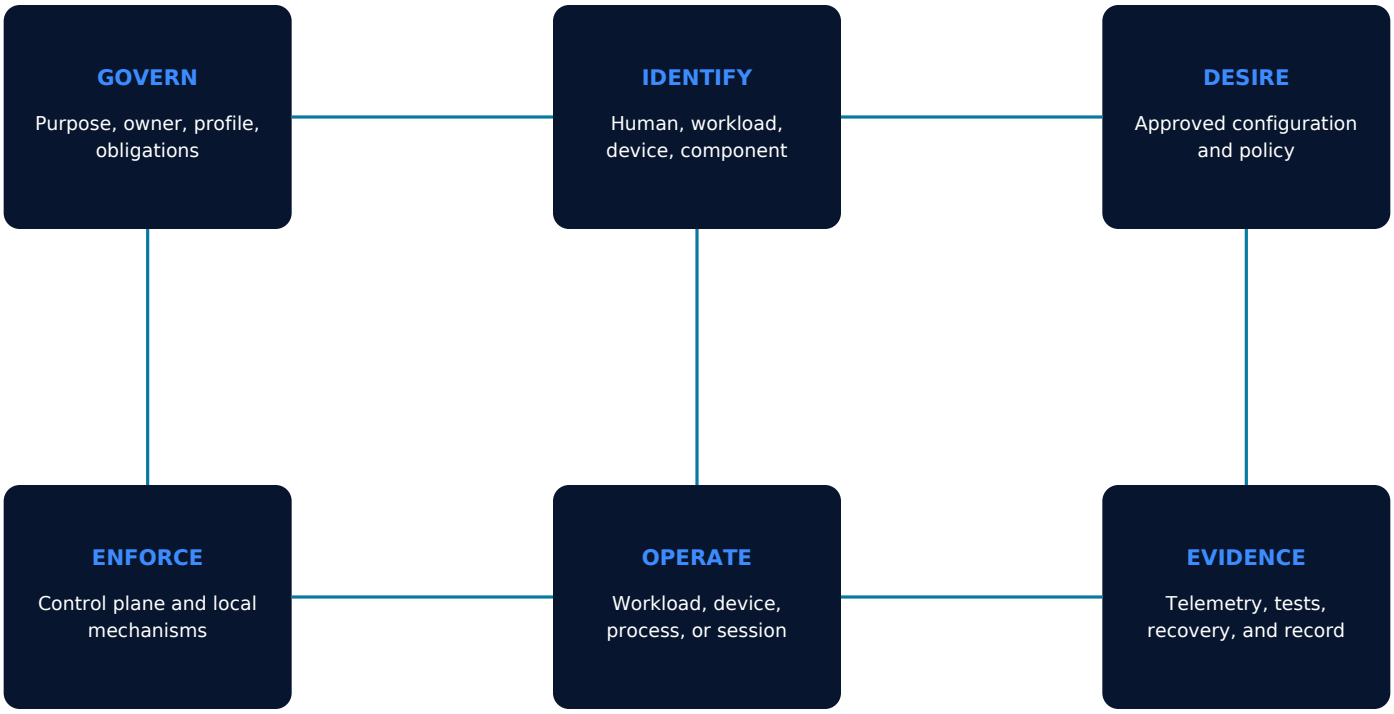
REASSESSMENT TRIGGER

A change, incident, drift signal, dependency revision, or time boundary requiring renewed evaluation.

MYTHOS-CLD-0011 / CLOUD ARCHITECTURE

3. Architecture and Trust Model

Separation of governance, management, enforcement, runtime, telemetry, and recovery



Mandatory trust boundaries

- Management planes MUST be isolated, strongly authenticated, monitored, and recoverable.
- Identity and policy MUST be evaluated at every provider, network, device, workload, and administrative transition.
- Runtime state MUST be compared with approved desired state and material drift MUST trigger response.
- Safety and continuity mechanisms MUST remain available when cloud, WAN, identity, DNS, time, carrier, or management dependencies fail.
- Evidence MUST be protected from the systems and administrators whose behavior it is intended to assess.

MYTHOS-CLD-0011 / CLOUD ARCHITECTURE

4. Governance and Accountability

Decision rights, separation of duties, and lifecycle ownership

ACCOUNTABLE OWNER

Accepts scope, risk tolerance, funding, and residual risk.

SYSTEM OWNER

Maintains architecture, inventory, operating boundaries, and change control.

SECURITY / PRIVACY

Defines threat, identity, data, isolation, monitoring, and incident requirements.

EVALUATION AUTHORITY

Designs representative tests, gates releases, and preserves reproducibility.

OPERATIONS

Maintains availability, rollback, recovery, evidence, and incident handling.

HUMAN OVERSIGHT

Reviews consequential decisions, exceptions, disputed outcomes, and harm signals.

DECISION PRINCIPLE

No single actor should be able to define the objective, grant authority, execute the consequential action, suppress evidence, and approve the result. Separation must be technical where consequence requires it.

MYTHOS-CLD-0011 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 01-04 of 18

MYTHOS-CLD-0011-C01 Use-Case Justification and Trust Assumptions

The organization MUST define, implement, verify, and maintain use-case justification and trust assumptions for in-scope Web3 and decentralized ledger architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0011-C02 Network, Consensus, and Finality Selection

The organization MUST define, implement, verify, and maintain network, consensus, and finality selection for in-scope Web3 and decentralized ledger architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0011-C03 Node, Validator, and Client Governance

The organization MUST define, implement, verify, and maintain node, validator, and client governance for in-scope Web3 and decentralized ledger architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-CLD-0011-C04 Key Custody, Wallet, and Transaction Authorization

The organization MUST define, implement, verify, and maintain key custody, wallet, and transaction authorization for in-scope Web3 and decentralized ledger architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-CLD-0011 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 05-08 of 18

MYTHOS-CLD-0011-C05 Smart-Contract Design and Upgrade Authority

The organization MUST define, implement, verify, and maintain smart-contract design and upgrade authority for in-scope Web3 and decentralized ledger architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-CLD-0011-C06 Secure Development, Review, and Formal Analysis

The organization MUST define, implement, verify, and maintain secure development, review, and formal analysis for in-scope Web3 and decentralized ledger architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-CLD-0011-C07 Oracle, Data Feed, and External-State Integrity

The organization MUST define, implement, verify, and maintain oracle, data feed, and external-state integrity for in-scope Web3 and decentralized ledger architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-CLD-0011-C08 Bridge, Interoperability, and Cross-Chain Risk

The organization MUST define, implement, verify, and maintain bridge, interoperability, and cross-chain risk for in-scope Web3 and decentralized ledger architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-CLD-0011 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 09-12 of 18

MYTHOS-CLD-0011-C09 Token, Incentive, and Economic-Security Design

The organization MUST define, implement, verify, and maintain token, incentive, and economic-security design for in-scope Web3 and decentralized ledger architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0011-C10 Privacy, Confidentiality, and Metadata Exposure

The organization MUST define, implement, verify, and maintain privacy, confidentiality, and metadata exposure for in-scope Web3 and decentralized ledger architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0011-C11 Identity, Compliance, and Jurisdictional Boundaries

The organization MUST define, implement, verify, and maintain identity, compliance, and jurisdictional boundaries for in-scope Web3 and decentralized ledger architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-CLD-0011-C12 Transaction Monitoring and Abuse Detection

The organization MUST define, implement, verify, and maintain transaction monitoring and abuse detection for in-scope Web3 and decentralized ledger architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-CLD-0011 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 13-15 of 18

MYTHOS-CLD-0011-C13 Availability, Congestion, and Fee Volatility

The organization MUST define, implement, verify, and maintain availability, congestion, and fee volatility for in-scope Web3 and decentralized ledger architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-CLD-0011-C14 Fork, Reorganization, and Governance Response

The organization MUST define, implement, verify, and maintain fork, reorganization, and governance response for in-scope Web3 and decentralized ledger architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-CLD-0011-C15 Incident Pause, Recovery, and Asset Protection

The organization MUST define, implement, verify, and maintain incident pause, recovery, and asset protection for in-scope Web3 and decentralized ledger architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-CLD-0011 / CLOUD ARCHITECTURE

5. Normative Control System

Atomic criteria 16-18 of 18

MYTHOS-CLD-0011-C16 Dependency, Library, and Client Supply Chain

The organization MUST define, implement, verify, and maintain dependency, library, and client supply chain for in-scope Web3 and decentralized ledger architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-CLD-0011-C17 Migration, Contract Retirement, and Key Destruction

The organization MUST define, implement, verify, and maintain migration, contract retirement, and key destruction for in-scope Web3 and decentralized ledger architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-CLD-0011-C18 Evidence, Disclosure, and Periodic Reassessment

The organization MUST define, implement, verify, and maintain evidence, disclosure, and periodic reassessment for in-scope Web3 and decentralized ledger architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-CLD-0011 / CLOUD ARCHITECTURE

6. Evidence and Assessment

Examine, interview, test, observe, restore, and trace

EXAMINE

Policies, inventories, diagrams, configuration, code, firmware, contracts, provider evidence, logs, exceptions, and lifecycle records.

INTERVIEW

Accountable owners, architects, engineers, operators, safety personnel, security teams, vendors, carriers, integrators, and responders.

TEST

Identity, segmentation, policy, negative paths, malformed input, capacity stress, dependency loss, failover, rollback, and revocation.

RESTORE

Independent restoration of configuration, data, service, device, controller, cluster, media, or browser state from protected evidence.

OBSERVE

Production-like performance, quality, drift, resource use, physical behavior, alarms, user impact, and incident execution.

TRACE

End-to-end linkage from approved intent and identity through change, runtime behavior, telemetry, outcome, exception, and review.

Evidence grades

A

Independently reproducible, complete, current, integrity-protected, and representative.

B

Reproduced by the implementing organization with strong traceability and controlled scope.

C

Partially reproduced or indirect; limitations, inherited responsibility, and uncertainty recorded.

D

Assertion, diagram, design intent, certificate, or vendor statement without reproduced behavior.

MYTHOS-CLD-0011 / CLOUD ARCHITECTURE

7. Assurance Views

Six perspectives required for a complete assurance claim

GOVERNANCE

Ownership, purpose, risk tolerance, policy, exception, and decision rights.

ARCHITECTURE

Boundaries, dependencies, failure modes, state, data flow, and portability.

SECURITY

Identity, authorization, isolation, secrets, abuse resistance, and incident response.

OPERATIONS

Reliability, performance, cost, observability, change, recovery, and support.

PRIVACY / RIGHTS

Purpose limitation, minimization, consent, access, correction, deletion, and egress.

HUMAN / SOCIETAL

Usability, accessibility, disclosure, contestability, impact, and human control.

Conformance requires a defensible position across every applicable view. A strong aggregate score cannot compensate for an unowned system, a failed mandatory gate, untested recovery, or evidence below the accepted grade.

MYTHOS-CLD-0011 / CLOUD ARCHITECTURE

7.2 Evaluation Metrics and Decision Gates

Measures must expose service behavior, control effectiveness, failure, uncertainty, and cost

SERVICE

Availability, correctness, coverage, quality, latency, throughput, and user or process outcome.

CONTROL

Unauthorized attempt, policy denial, drift, segmentation escape, privilege, and exception exposure.

RESILIENCE

Failover success, recovery time, data loss, safe degradation, restore validity, and dependency survival.

SECURITY

Exploitability, credential exposure, supply-chain integrity, attack detection, containment, and revocation.

CAPACITY

Utilization, saturation, queueing, radio or fabric headroom, thermal margin, quota, and forecast error.

OPERATIONS

Change failure, mean time to detect, repair, support burden, vendor escalation, and evidence completeness.

PRIVACY / SAFETY

Unauthorized egress, retention breach, consent coverage, unsafe action, physical consequence, and contestability.

LIFECYCLE

Patch debt, end-of-support exposure, portability, replacement time, retirement completion, and reassessment age.

Decision gate: thresholds MUST be declared before assessment, cover representative load and failure conditions, and identify mandatory safety, identity, recovery, and evidence failures that block promotion.

MYTHOS-CLD-0011 / CLOUD ARCHITECTURE

8. Failure Modes and Adversarial Scenarios

Challenge assumptions before the system is trusted with consequential work

CONTROL-PLANE COMPROMISE

An administrative or orchestration plane can alter broad production scope.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

PROVIDER OR REGION DEPENDENCY

A provider, region, identity, DNS, or service dependency fails without a tested alternative.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

CONFIGURATION DRIFT

Runtime state diverges from approved desired state without timely detection.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

SOVEREIGNTY BREACH

Data or keys move through an unapproved jurisdiction or subprocess.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

COST AND QUOTA RUNAWAY

Unbounded scaling, abuse, or configuration error exhausts budget or service limits.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

EXIT FAILURE

Workloads, data, identity, or evidence cannot be migrated within the required time.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

Adversarial testing MUST protect people and production systems. Use isolated environments, synthetic or minimized data, bounded authority, kill switches, explicit legal and ethical review, and documented stop conditions.

MYTHOS-CLD-0011 / CLOUD ARCHITECTURE

9. Implementation Profiles

Risk-proportional implementation without weakening mandatory boundaries

	FOUNDATIONAL	ADVANCED	HIGH ASSURANCE
TYPICAL SCOPE	Low consequence, limited autonomy, non-sensitive data, reversible outputs.	Business-critical workflow, sensitive data, multiple tools or providers, moderate autonomy.	Safety, security, regulated, financial, identity, or broadly consequential use.
MINIMUM DEPTH	Named owner; inventory; basic evaluation; access control; logging; rollback; annual review.	Formal threat model; representative evaluation; approval gates; isolated execution; tested recovery; quarterly review.	Independent challenge; strongest identity; deterministic enforcement; comprehensive evidence; canary release; continuous monitoring.

Profile selection rule

Choose the highest profile triggered by consequence, autonomy, sensitivity, scale, irreversibility, external dependency, or inability to rapidly detect and recover. Tailoring may strengthen controls; it may not erase a mandatory gate without a controlled exception.

MYTHOS-CLD-0011 / CLOUD ARCHITECTURE

9.2 Operational Lifecycle

Evidence-bearing operation from concept through retirement

1. **CONCEPT**
   Purpose, owner, impact, boundaries
2. **DESIGN**
   Architecture, threat, data, identity
3. **BUILD**
   Controlled implementation and provenance
4. **EVALUATE**
   Representative, adversarial, recovery tests
5. **RELEASE**
   Approval, canary, rollback, evidence
6. **OPERATE**
   Observe, support, detect, respond
7. **CHANGE**
   Reassess, compare, gate, reverse
8. **RETIRE**
   Revoke, export, dispose, preserve

LIFECYCLE INVARIANT

Every stage MUST leave sufficient evidence for the next authority to understand what was intended, what changed, what was tested, what failed, what was accepted, what remains uncertain, and how to recover or exit.

MYTHOS-CLD-0011 / CLOUD ARCHITECTURE

10. Adoption Roadmap

A sequenced path from uncontrolled capability to evidence-bearing operation

0-30 DAYS

Establish ownership and truth

Inventory systems and dependencies; classify risk; freeze unsupported claims; identify immediate privilege, data, and evidence gaps.

31-90 DAYS

Create enforceable boundaries

Define policy; isolate execution; implement identity and approval gates; establish representative evaluation and baseline telemetry.

3-6 MONTHS

Prove recovery and operating control

Exercise failure, rollback, revocation, incident, provider exit, and state-recovery scenarios; mature evidence packaging.

6-12 MONTHS

Scale assurance

Automate control checks; expand workload coverage; independent challenge; portfolio metrics; controlled exception expiry; continuous reassessment.

Adoption is complete only when operating evidence demonstrates the selected profile in the actual deployment context. Documentation alone is not conformance.

MYTHOS-CLD-0011 / CLOUD ARCHITECTURE

10.2 Conformance and Exception Statement

What an assurance claim must contain

SYSTEM / SERVICE

Named, versioned capability and deployment boundary.

PROFILE

Foundational, Advanced, or High Assurance with trigger rationale.

SCOPE

Workloads, users, data, tools, regions, providers, and exclusions.

CRITERIA

Pass, partial, fail, not applicable, and exception status for every criterion.

EVIDENCE

Artifact references, evidence grade, date, environment, and reproducibility.

LIMITATIONS

Known failure modes, unsupported workloads, uncertainty, and residual risk.

EXCEPTIONS

Owner, rationale, compensating control, expiration, and approval.

VALIDITY

Assessment date, change boundary, review date, and reassessment triggers.

Prohibited claim: “compliant” without the named scope, profile, evidence date, limitations, exceptions, and authority accepting residual risk.

MYTHOS-CLD-0011 / CLOUD ARCHITECTURE

11. Source Authority and Reference Register

Primary sources informing interpretation; current obligations and provider behavior must be verified at assessment t

NIST CSF 2.0	**Cybersecurity Framework 2.0**

https://doi.org/10.6028/NIST.CSWP.29

Enterprise governance and cybersecurity outcome framework.

NIST SP 800-53 Rev. 5	**Security and Privacy Controls for Information Systems and Organizations**

https://doi.org/10.6028/NIST.SP.800-53r5

Broad control baseline for organizational systems and services.

CSA CCM v4	**Cloud Controls Matrix**

https://cloudsecurityalliance.org/research/cloud-controls-matrix

Cloud control domains and shared-responsibility assessment structure.

FinOps Framework	**FinOps Framework**

https://www.finops.org/framework/

Cloud value, cost, usage, ownership, and operating-practice guidance.

NIST SP 800-204A	**Building Secure Microservices-based Applications Using Service-Mesh Architecture**

https://doi.org/10.6028/NIST.SP.800-204A

Microservice and service-mesh security architecture.

CNCF Cloud Native Security	**Cloud Native Security Whitepaper**

https://github.com/cncf/tag-security/tree/main/security-whitepaper

Cloud-native lifecycle, supply-chain, runtime, and operations guidance.

Source currency rule: authorities, specifications, legal obligations, provider services, firmware, browser behavior, carrier requirements, and operational evidence MUST be checked at assessment time. This publication records a 2026-07-24 source snapshot.

MYTHOS-CLD-0011 / CLOUD ARCHITECTURE

11.2 Revision History and Decision Register

Permanent identity, controlled change, and publication integrity

VERSION	DATE	STATE	SUMMARY
1.0.0-candidate	2026-07-24	Candidate	Permanent-publication edition; source refresh; expanded criteria, evidence, assurance, and lifecycle guidance.
Next review	2027-01-24	Scheduled	Review source currency, threat evolution, operating evidence, adoption feedback, and proposed revisions.

Publication decisions

PERMANENT IDENTITY The document ID remains stable across revisions; batch or production sequence metadata is not public identity.

CHANGE CONTROL Normative changes require rationale, impact analysis, affected-criterion mapping, and approval.

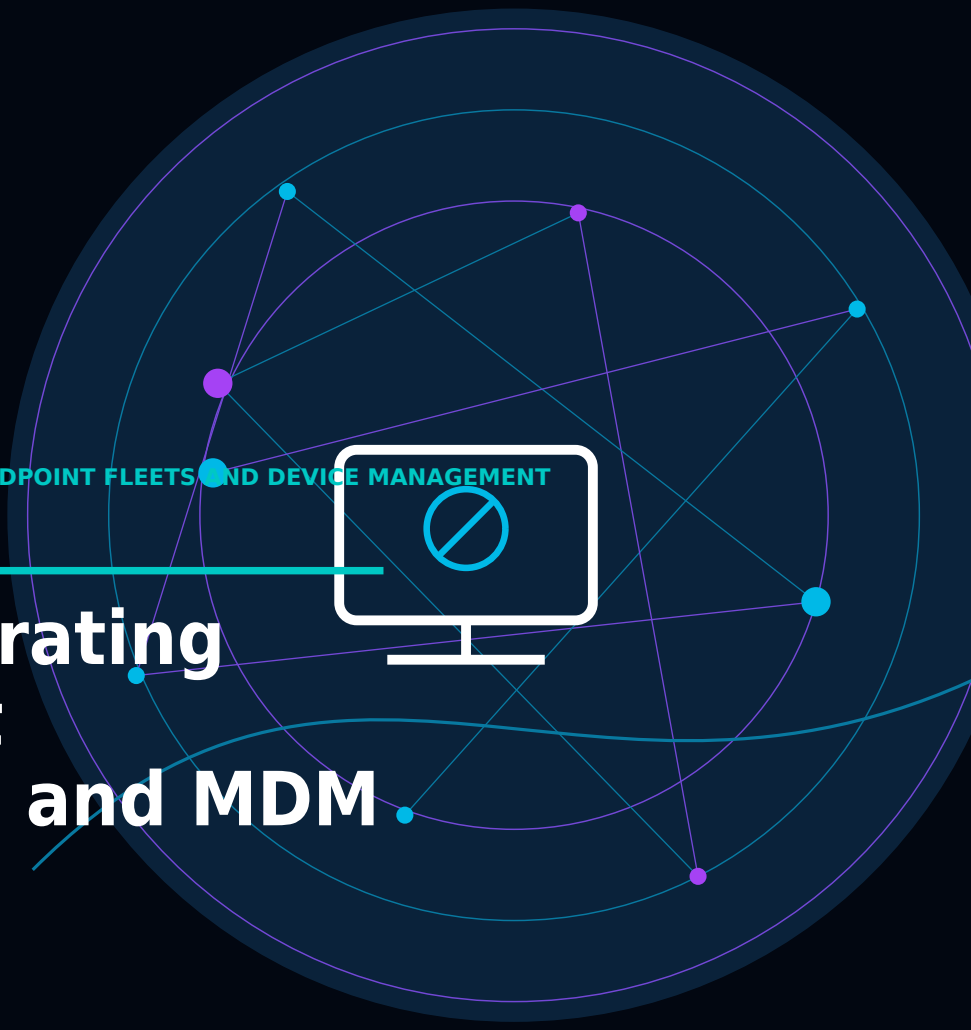
SUPERSESSION A superseded edition remains traceable; current routes and catalogs identify the active edition.

CORRECTION Material errors require a recorded correction, affected-evidence analysis, and notification appropriate to impact.

ARCHIVAL INTEGRITY Published artifacts receive hashes, metadata, and a release manifest sufficient for independent verification.

END OF PERMANENT PUBLICATION / MYTHOS-CLD-0011

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY



ENGINEERING STANDARDS LIBRARY / ENDPOINT FLEETS AND DEVICE MANAGEMENT

Endpoint Operating System, Fleet Management, and MDM Standard

Endpoint and mobile fleet identity, enrollment, configuration, application, data, detection, access, support, and retirement.

PERMANENT DOCUMENT ID

MYTHOS-END-0001

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

Endpoint Operating System, Fleet Management, and MDM Standard

DOCUMENT ID
MYTHOS-END-0001

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

18	6	4	1
ATOMIC CRITERIA	ASSURANCE VIEWS	IMPLEMENTATION PROFILES	PERMANENT IDENTITY

Publication doctrine. This publication establishes durable requirements for its defined scope. Interpretation must preserve the permanent document identity, required evidence, controlled exceptions, source currency, and formal revision history.

INFRASTRUCTURE AND CONTROL TOPOLOGY

Endpoint Fleets and Device Management

A trustworthy endpoint operating systems and fleet management system is a governed chain of ownership, identity, desired state, enforcement, workload or device behavior, telemetry, recovery, and independently reviewable evidence.

OWNERSHIP

Purpose, service boundary, accountable owner, suppliers, users, and retained responsibility remain explicit.

ENFORCEMENT

Identity, policy, segmentation, configuration, and local safety mechanisms constrain every trust transition.

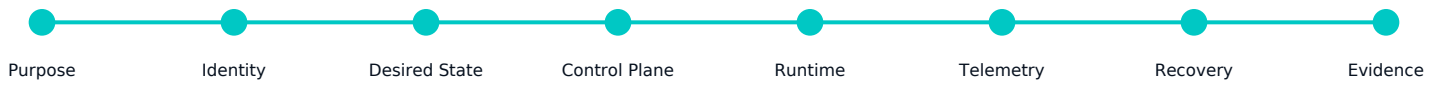
CONTINUITY

Capacity, dependency loss, safe degradation, backup, restoration, rollback, and exit are tested before reliance.

EVIDENCE

Inventory, desired state, runtime observation, tests, incidents, exceptions, and change records form the assurance chain.

GOVERNED INFRASTRUCTURE FLOW



Reconciliation, detection, response, restoration, and controlled evolution

INTERPRETATION AND ASSURANCE

How to apply this publication

Normative intent

Requirements define outcomes and constraints. Implementations may vary, but evidence must demonstrate equivalent control.

Evidence over assertion

Vendor documentation and design intent are not equivalent to reproduced behavior. Record evidence grade and uncertainty.

Risk-proportional depth

Higher consequence, autonomy, sensitivity, and irreversibility require stronger isolation, review, verification, and recovery.

Controlled exception

Exceptions require an owner, rationale, compensating control, residual-risk acceptance, expiration, and reevaluation trigger.

Normalized assessment scale

0	1	2	3	4	5
ABSENT	AD HOC	PARTIAL	OPERATIONAL	ADVANCED	LEADING

A numeric score must never hide a failed mandatory gate, missing evidence, untested workload, or unacceptable residual risk.

INTERACTIVE NAVIGATION

Contents

Navigation doctrine

Bookmarks and internal links are conveniences. Permanent document identity, criterion identifiers, evidence references, and revision history remain authoritative.

MYTHOS-END-0001 / ENDPOINT AND FLEET

0. Executive Mandate

Purpose, strategic outcome, and non-negotiable operating doctrine

MANDATE

Organizations adopting endpoint operating systems and fleet management capabilities MUST define an owned service boundary, establish enforceable identity and configuration, protect data and safety, test failure and recovery, and preserve evidence sufficient for independent review.

Required outcomes

- Create a durable governance boundary for endpoint operating systems and fleet management across design, acquisition, deployment, operation, change, and retirement.
- Require risk-proportional segmentation, identity, configuration, telemetry, resilience, recovery, and supplier controls.
- Prevent central automation, administrative tooling, or provider services from becoming unbounded authority.
- Prove operation with representative workloads, devices, failure modes, and restoration exercises.
- Define measurable conformance, exceptions, reassessment triggers, and a viable exit path.

Decision boundary: conformance supports a documented assurance claim for the named scope. It does not prove universal availability, safety, regulatory acceptance, vendor fitness, or immunity from cyber-physical failure.

MYTHOS-END-0001 / ENDPOINT AND FLEET

1. Scope and Applicability

Boundary definition, audience, exclusions, and triggering conditions

IN SCOPE

- Architecture, acquisition, configuration, integration, and operation of endpoint operating systems and fleet management capabilities.
- Human, workload, device, service, network, data, facility, provider, supplier, and evidence dependencies.
- Design, deployment, monitoring, support, incident response, recovery, migration, and retirement.
- On-premises, hosted, managed, carrier, manufacturer, integrator, and externally operated components.

OUT OF SCOPE

- Claims of legal, safety, carrier, or regulatory approval without the responsible authority.
- Vendor certification treated as proof of the adopter configuration or operating effectiveness.
- Theoretical or laboratory behavior presented as production evidence without a declared boundary.
- Inherited controls without documented scope, owner, period, limitations, and shared responsibility.

Applicability triggers

- The capability supports a business-critical, safety-relevant, regulated, public, or externally dependent service.
- The environment can expose sensitive data, identity, control, physical process, communications, or shared infrastructure.
- A management plane, automation system, carrier, provider, or supplier can affect broad operational scope.
- Failure, compromise, or change can be difficult to detect, reverse, isolate, or recover.
- The system depends on hardware, firmware, radio, browser, cloud, endpoint, IoT, OT, or real-time media behavior.

MYTHOS-END-0001 / ENDPOINT AND FLEET

2. Definitions and Taxonomy

Controlled language for architecture, governance, and assessment

AUTHORITY

The right to approve, deny, constrain, or execute an action. Model output is not authority.

EVIDENCE

A reproducible source, trace, measurement, test, record, signed artifact, or documented observation supporting a claim.

ASSURANCE VIEW

A defined perspective such as governance, architecture, security, operations, privacy, or human oversight.

ATOMIC CRITERION

A uniquely identified requirement that can be assessed, evidenced, excepted, and revised independently.

IMPLEMENTATION PROFILE

A risk-proportional set of required depth, evidence, and gates for a class of use.

CONTROLLED EXCEPTION

A time-bounded deviation with owner, rationale, compensating control, residual risk, expiration, and review trigger.

DETERMINISTIC MECHANISM

A component whose inputs, policy, state transition, and side effects can be constrained and verified.

SOURCE AUTHORITY

The documented basis for treating a source as reliable for a defined claim, context, jurisdiction, and time.

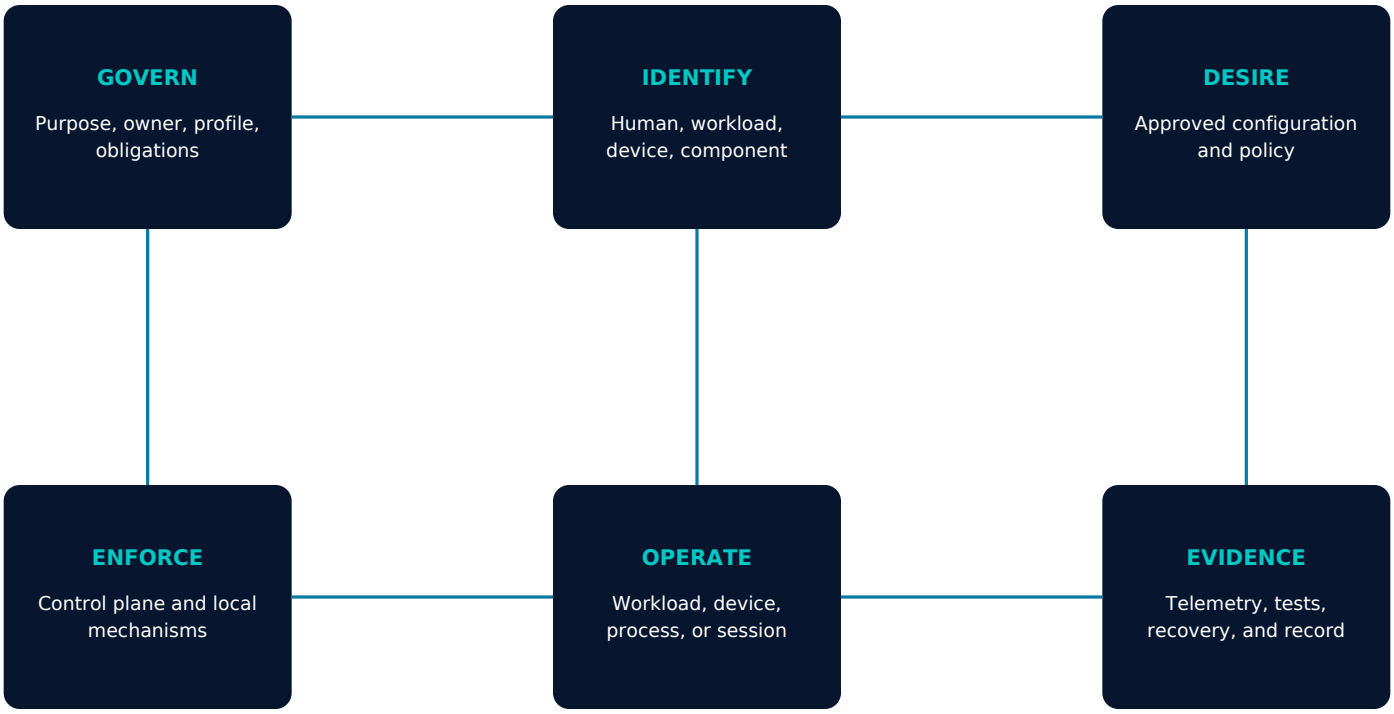
REASSESSMENT TRIGGER

A change, incident, drift signal, dependency revision, or time boundary requiring renewed evaluation.

MYTHOS-END-0001 / ENDPOINT AND FLEET

3. Architecture and Trust Model

Separation of governance, management, enforcement, runtime, telemetry, and recovery



Mandatory trust boundaries

- Management planes MUST be isolated, strongly authenticated, monitored, and recoverable.
- Identity and policy MUST be evaluated at every provider, network, device, workload, and administrative transition.
- Runtime state MUST be compared with approved desired state and material drift MUST trigger response.
- Safety and continuity mechanisms MUST remain available when cloud, WAN, identity, DNS, time, carrier, or management dependencies fail.
- Evidence MUST be protected from the systems and administrators whose behavior it is intended to assess.

MYTHOS-END-0001 / ENDPOINT AND FLEET

4. Governance and Accountability

Decision rights, separation of duties, and lifecycle ownership

ACCOUNTABLE OWNER

Accepts scope, risk tolerance, funding, and residual risk.

SYSTEM OWNER

Maintains architecture, inventory, operating boundaries, and change control.

SECURITY / PRIVACY

Defines threat, identity, data, isolation, monitoring, and incident requirements.

EVALUATION AUTHORITY

Designs representative tests, gates releases, and preserves reproducibility.

OPERATIONS

Maintains availability, rollback, recovery, evidence, and incident handling.

HUMAN OVERSIGHT

Reviews consequential decisions, exceptions, disputed outcomes, and harm signals.

DECISION PRINCIPLE

No single actor should be able to define the objective, grant authority, execute the consequential action, suppress evidence, and approve the result. Separation must be technical where consequence requires it.

MYTHOS-END-0001 / ENDPOINT AND FLEET

5. Normative Control System

Atomic criteria 01-04 of 18

MYTHOS-END-0001-C01 Fleet Inventory, Ownership, and Criticality

The organization MUST define, implement, verify, and maintain fleet inventory, ownership, and criticality for in-scope endpoint operating systems and fleet management capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-END-0001-C02 Procurement, Enrollment, and Unique Device Identity

The organization MUST define, implement, verify, and maintain procurement, enrollment, and unique device identity for in-scope endpoint operating systems and fleet management capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-END-0001-C03 Secure Boot, Platform Integrity, and Attestation

The organization MUST define, implement, verify, and maintain secure boot, platform integrity, and attestation for in-scope endpoint operating systems and fleet management capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-END-0001-C04 Operating System Baseline and Configuration Policy

The organization MUST define, implement, verify, and maintain operating system baseline and configuration policy for in-scope endpoint operating systems and fleet management capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-END-0001 / ENDPOINT AND FLEET

5. Normative Control System

Atomic criteria 05-08 of 18

MYTHOS-END-0001-C05 Disk Encryption and Recovery-Key Governance

The organization MUST define, implement, verify, and maintain disk encryption and recovery-key governance for in-scope endpoint operating systems and fleet management capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-END-0001-C06 Administrative Rights and Privilege Elevation

The organization MUST define, implement, verify, and maintain administrative rights and privilege elevation for in-scope endpoint operating systems and fleet management capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-END-0001-C07 Application Allowlisting and Software Distribution

The organization MUST define, implement, verify, and maintain application allowlisting and software distribution for in-scope endpoint operating systems and fleet management capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-END-0001-C08 Patch, Vulnerability, and End-of-Support Management

The organization MUST define, implement, verify, and maintain patch, vulnerability, and end-of-support management for in-scope endpoint operating systems and fleet management capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-END-0001 / ENDPOINT AND FLEET

5. Normative Control System

Atomic criteria 09-12 of 18

MYTHOS-END-0001-C09 EDR, Telemetry, and Threat Detection

The organization MUST define, implement, verify, and maintain edr, telemetry, and threat detection for in-scope endpoint operating systems and fleet management capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-END-0001-C10 Network Access, Posture, and Certificate Binding

The organization MUST define, implement, verify, and maintain network access, posture, and certificate binding for in-scope endpoint operating systems and fleet management capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-END-0001-C11 Browser, Email, and Collaboration Protection

The organization MUST define, implement, verify, and maintain browser, email, and collaboration protection for in-scope endpoint operating systems and fleet management capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-END-0001-C12 Local Data, Removable Media, and DLP

The organization MUST define, implement, verify, and maintain local data, removable media, and dlp for in-scope endpoint operating systems and fleet management capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-END-0001 / ENDPOINT AND FLEET

5. Normative Control System

Atomic criteria 13-15 of 18

MYTHOS-END-0001-C13 Mobile, BYOD, and Personally Owned Device Boundaries

The organization MUST define, implement, verify, and maintain mobile, byod, and personally owned device boundaries for in-scope endpoint operating systems and fleet management capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-END-0001-C14 Remote Support and Administrative Session Control

The organization MUST define, implement, verify, and maintain remote support and administrative session control for in-scope endpoint operating systems and fleet management capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-END-0001-C15 Lost, Stolen, and Compromised Device Response

The organization MUST define, implement, verify, and maintain lost, stolen, and compromised device response for in-scope endpoint operating systems and fleet management capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-END-0001 / ENDPOINT AND FLEET

5. Normative Control System

Atomic criteria 16-18 of 18

MYTHOS-END-0001-C16 Backup, Restore, and User-State Recovery

The organization MUST define, implement, verify, and maintain backup, restore, and user-state recovery for in-scope endpoint operating systems and fleet management capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-END-0001-C17 Transfer, Reassignment, Wipe, and Retirement

The organization MUST define, implement, verify, and maintain transfer, reassignment, wipe, and retirement for in-scope endpoint operating systems and fleet management capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-END-0001-C18 Fleet Evidence, Exceptions, and Reassessment

The organization MUST define, implement, verify, and maintain fleet evidence, exceptions, and reassessment for in-scope endpoint operating systems and fleet management capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-END-0001 / ENDPOINT AND FLEET

6. Evidence and Assessment

Examine, interview, test, observe, restore, and trace

EXAMINE

Policies, inventories, diagrams, configuration, code, firmware, contracts, provider evidence, logs, exceptions, and lifecycle records.

INTERVIEW

Accountable owners, architects, engineers, operators, safety personnel, security teams, vendors, carriers, integrators, and responders.

TEST

Identity, segmentation, policy, negative paths, malformed input, capacity stress, dependency loss, failover, rollback, and revocation.

RESTORE

Independent restoration of configuration, data, service, device, controller, cluster, media, or browser state from protected evidence.

OBSERVE

Production-like performance, quality, drift, resource use, physical behavior, alarms, user impact, and incident execution.

TRACE

End-to-end linkage from approved intent and identity through change, runtime behavior, telemetry, outcome, exception, and review.

Evidence grades

A

Independently reproducible, complete, current, integrity-protected, and representative.

B

Reproduced by the implementing organization with strong traceability and controlled scope.

C

Partially reproduced or indirect; limitations, inherited responsibility, and uncertainty recorded.

D

Assertion, diagram, design intent, certificate, or vendor statement without reproduced behavior.

MYTHOS-END-0001 / ENDPOINT AND FLEET

7. Assurance Views

Six perspectives required for a complete assurance claim

GOVERNANCE

Ownership, purpose, risk tolerance, policy, exception, and decision rights.

ARCHITECTURE

Boundaries, dependencies, failure modes, state, data flow, and portability.

SECURITY

Identity, authorization, isolation, secrets, abuse resistance, and incident response.

OPERATIONS

Reliability, performance, cost, observability, change, recovery, and support.

PRIVACY / RIGHTS

Purpose limitation, minimization, consent, access, correction, deletion, and egress.

HUMAN / SOCIETAL

Usability, accessibility, disclosure, contestability, impact, and human control.

Conformance requires a defensible position across every applicable view. A strong aggregate score cannot compensate for an unowned system, a failed mandatory gate, untested recovery, or evidence below the accepted grade.

MYTHOS-END-0001 / ENDPOINT AND FLEET

7.2 Evaluation Metrics and Decision Gates

Measures must expose service behavior, control effectiveness, failure, uncertainty, and cost

SERVICE

Availability, correctness, coverage, quality, latency, throughput, and user or process outcome.

CONTROL

Unauthorized attempt, policy denial, drift, segmentation escape, privilege, and exception exposure.

RESILIENCE

Failover success, recovery time, data loss, safe degradation, restore validity, and dependency survival.

SECURITY

Exploitability, credential exposure, supply-chain integrity, attack detection, containment, and revocation.

CAPACITY

Utilization, saturation, queueing, radio or fabric headroom, thermal margin, quota, and forecast error.

OPERATIONS

Change failure, mean time to detect, repair, support burden, vendor escalation, and evidence completeness.

PRIVACY / SAFETY

Unauthorized egress, retention breach, consent coverage, unsafe action, physical consequence, and contestability.

LIFECYCLE

Patch debt, end-of-support exposure, portability, replacement time, retirement completion, and reassessment age.

Decision gate: thresholds MUST be declared before assessment, cover representative load and failure conditions, and identify mandatory safety, identity, recovery, and evidence failures that block promotion.

MYTHOS-END-0001 / ENDPOINT AND FLEET

8. Failure Modes and Adversarial Scenarios

Challenge assumptions before the system is trusted with consequential work

UNMANAGED ENDPOINT

A device reaches sensitive services without enrollment or current posture.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

CREDENTIAL AND TOKEN THEFT

Endpoint compromise exposes reusable user or device authority.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

PATCH DEBT

Unsupported or vulnerable systems remain operational beyond accepted risk.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

LOCAL DATA LEAKAGE

Sensitive information escapes through storage, browser, removable media, or sync.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

MANAGEMENT OUTAGE

MDM, EDR, identity, or update infrastructure fails during an incident.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

RETIREMENT FAILURE

A transferred or disposed device retains data, keys, or active identity.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

Adversarial testing MUST protect people and production systems. Use isolated environments, synthetic or minimized data, bounded authority, kill switches, explicit legal and ethical review, and documented stop conditions.

MYTHOS-END-0001 / ENDPOINT AND FLEET

9. Implementation Profiles

Risk-proportional implementation without weakening mandatory boundaries

FOUNDATIONAL

TYPICAL SCOPE

Low consequence, limited autonomy, non-sensitive data, reversible outputs.

MINIMUM DEPTH

Named owner; inventory; basic evaluation; access control; logging; rollback; annual review.

ADVANCED

TYPICAL SCOPE

Business-critical workflow, sensitive data, multiple tools or providers, moderate autonomy.

MINIMUM DEPTH

Formal threat model; representative evaluation; approval gates; isolated execution; tested recovery; quarterly review.

HIGH ASSURANCE

TYPICAL SCOPE

Safety, security, regulated, financial, identity, or broadly consequential use.

MINIMUM DEPTH

Independent challenge; strongest identity; deterministic enforcement; comprehensive evidence; canary release; continuous monitoring.

Profile selection rule

Choose the highest profile triggered by consequence, autonomy, sensitivity, scale, irreversibility, external dependency, or inability to rapidly detect and recover. Tailoring may strengthen controls; it may not erase a mandatory gate without a controlled exception.

MYTHOS-END-0001 / ENDPOINT AND FLEET

9.2 Operational Lifecycle

Evidence-bearing operation from concept through retirement

1. **CONCEPT**
   Purpose, owner, impact, boundaries
2. **DESIGN**
   Architecture, threat, data, identity
3. **BUILD**
   Controlled implementation and provenance
4. **EVALUATE**
   Representative, adversarial, recovery tests
5. **RELEASE**
   Approval, canary, rollback, evidence
6. **OPERATE**
   Observe, support, detect, respond
7. **CHANGE**
   Reassess, compare, gate, reverse
8. **RETIRE**
   Revoke, export, dispose, preserve

LIFECYCLE INVARIANT

Every stage MUST leave sufficient evidence for the next authority to understand what was intended, what changed, what was tested, what failed, what was accepted, what remains uncertain, and how to recover or exit.

MYTHOS-END-0001 / ENDPOINT AND FLEET

10. Adoption Roadmap

A sequenced path from uncontrolled capability to evidence-bearing operation

0-30 DAYS

Establish ownership and truth

Inventory systems and dependencies; classify risk; freeze unsupported claims; identify immediate privilege, data, and evidence gaps.

31-90 DAYS

Create enforceable boundaries

Define policy; isolate execution; implement identity and approval gates; establish representative evaluation and baseline telemetry.

3-6 MONTHS

Prove recovery and operating control

Exercise failure, rollback, revocation, incident, provider exit, and state-recovery scenarios; mature evidence packaging.

6-12 MONTHS

Scale assurance

Automate control checks; expand workload coverage; independent challenge; portfolio metrics; controlled exception expiry; continuous reassessment.

Adoption is complete only when operating evidence demonstrates the selected profile in the actual deployment context. Documentation alone is not conformance.

MYTHOS-END-0001 / ENDPOINT AND FLEET

10.2 Conformance and Exception Statement

What an assurance claim must contain

SYSTEM / SERVICE

Named, versioned capability and deployment boundary.

PROFILE

Foundational, Advanced, or High Assurance with trigger rationale.

SCOPE

Workloads, users, data, tools, regions, providers, and exclusions.

CRITERIA

Pass, partial, fail, not applicable, and exception status for every criterion.

EVIDENCE

Artifact references, evidence grade, date, environment, and reproducibility.

LIMITATIONS

Known failure modes, unsupported workloads, uncertainty, and residual risk.

EXCEPTIONS

Owner, rationale, compensating control, expiration, and approval.

VALIDITY

Assessment date, change boundary, review date, and reassessment triggers.

Prohibited claim: “compliant” without the named scope, profile, evidence date, limitations, exceptions, and authority accepting residual risk.

MYTHOS-END-0001 / ENDPOINT AND FLEET

11. Source Authority and Reference Register

Primary sources informing interpretation; current obligations and provider behavior must be verified at assessment t

NIST CSF 2.0 — **Cybersecurity Framework 2.0**

https://doi.org/10.6028/NIST.CSWP.29

Enterprise governance and cybersecurity outcome framework.

NIST SP 800-53 Rev. 5 — **Security and Privacy Controls for Information Systems and Organizations**

https://doi.org/10.6028/NIST.SP.800-53r5

Broad control baseline for organizational systems and services.

NIST SP 800-124 Rev. 2 — **Guidelines for Managing the Security of Mobile Devices in the Enterprise**

https://doi.org/10.6028/NIST.SP.800-124r2

Mobile-device lifecycle and enterprise management guidance.

NIST SP 800-193 — **Platform Firmware Resiliency Guidelines**

https://doi.org/10.6028/NIST.SP.800-193

Protection, detection, and recovery for platform firmware.

CIS Controls v8 — **CIS Critical Security Controls v8**

https://www.cisecurity.org/controls/v8

Inventory, configuration, vulnerability, access, logging, and response practices.

NIST SP 800-207 — **Zero Trust Architecture**

https://doi.org/10.6028/NIST.SP.800-207

Resource-centric access decisions and continuous verification.

Source currency rule: authorities, specifications, legal obligations, provider services, firmware, browser behavior, carrier requirements, and operational evidence MUST be checked at assessment time. This publication records a 2026-07-24 source snapshot.

MYTHOS-END-0001 / ENDPOINT AND FLEET

11.2 Revision History and Decision Register

Permanent identity, controlled change, and publication integrity

VERSION	DATE	STATE	SUMMARY
1.0.0-candidate	2026-07-24	Candidate	Permanent-publication edition; source refresh; expanded criteria, evidence, assurance, and lifecycle guidance.
Next review	2027-01-24	Scheduled	Review source currency, threat evolution, operating evidence, adoption feedback, and proposed revisions.

Publication decisions

PERMANENT IDENTITY The document ID remains stable across revisions; batch or production sequence metadata is not public identity.

CHANGE CONTROL Normative changes require rationale, impact analysis, affected-criterion mapping, and approval.

SUPERSESSION A superseded edition remains traceable; current routes and catalogs identify the active edition.

CORRECTION Material errors require a recorded correction, affected-evidence analysis, and notification appropriate to impact.

ARCHIVAL INTEGRITY Published artifacts receive hashes, metadata, and a release manifest sufficient for independent verification.

END OF PERMANENT PUBLICATION / MYTHOS-END-0001

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY

ENGINEERING STANDARDS LIBRARY / OPERATIONAL TECHNOLOGY AND CRITICAL INFRASTRUCTURE

Industrial Control Systems, Critical Infrastructure, and OT Segmentation Standard

Cyber-physical safety, asset and flow inventory, zones and conduits, remote access, change, monitoring, recovery, and evidence.

PERMANENT DOCUMENT ID

MYTHOS-OT-0001

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

Industrial Control Systems, Critical Infrastructure, and OT Segmentation Standard

DOCUMENT ID
MYTHOS-OT-0001

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

18	6	4	1
ATOMIC CRITERIA	ASSURANCE VIEWS	IMPLEMENTATION PROFILES	PERMANENT IDENTITY

Publication doctrine. This publication establishes durable requirements for its defined scope. Interpretation must preserve the permanent document identity, required evidence, controlled exceptions, source currency, and formal revision history.

INFRASTRUCTURE AND CONTROL TOPOLOGY

Operational Technology and Critical Infrastructure

A trustworthy industrial control systems and operational technology system is a governed chain of ownership, identity, desired state, enforcement, workload or device behavior, telemetry, recovery, and independently reviewable evidence.

OWNERSHIP

Purpose, service boundary, accountable owner, suppliers, users, and retained responsibility remain explicit.

ENFORCEMENT

Identity, policy, segmentation, configuration, and local safety mechanisms constrain every trust transition.

CONTINUITY

Capacity, dependency loss, safe degradation, backup, restoration, rollback, and exit are tested before reliance.

EVIDENCE

Inventory, desired state, runtime observation, tests, incidents, exceptions, and change records form the assurance chain.

GOVERNED INFRASTRUCTURE FLOW

Reconciliation, detection, response, restoration, and controlled evolution

INTERPRETATION AND ASSURANCE

How to apply this publication

Normative intent

Requirements define outcomes and constraints. Implementations may vary, but evidence must demonstrate equivalent control.

Evidence over assertion

Vendor documentation and design intent are not equivalent to reproduced behavior. Record evidence grade and uncertainty.

Risk-proportional depth

Higher consequence, autonomy, sensitivity, and irreversibility require stronger isolation, review, verification, and recovery.

Controlled exception

Exceptions require an owner, rationale, compensating control, residual-risk acceptance, expiration, and reevaluation trigger.

Normalized assessment scale

0	1	2	3	4	5
ABSENT	AD HOC	PARTIAL	OPERATIONAL	ADVANCED	LEADING

A numeric score must never hide a failed mandatory gate, missing evidence, untested workload, or unacceptable residual risk.

INTERACTIVE NAVIGATION

Contents

Navigation doctrine

Bookmarks and internal links are conveniences. Permanent document identity, criterion identifiers, evidence references, and revision history remain authoritative.

MYTHOS-OT-0001 / OPERATIONAL TECHNOLOGY

0. Executive Mandate

Purpose, strategic outcome, and non-negotiable operating doctrine

MANDATE

Organizations adopting industrial control systems and operational technology capabilities MUST define an owned service boundary, establish enforceable identity and configuration, protect data and safety, test failure and recovery, and preserve evidence sufficient for independent review.

Required outcomes

- Create a durable governance boundary for industrial control systems and operational technology across design, acquisition, deployment, operation, change, and retirement.
- Require risk-proportional segmentation, identity, configuration, telemetry, resilience, recovery, and supplier controls.
- Prevent central automation, administrative tooling, or provider services from becoming unbounded authority.
- Prove operation with representative workloads, devices, failure modes, and restoration exercises.
- Define measurable conformance, exceptions, reassessment triggers, and a viable exit path.

Decision boundary: conformance supports a documented assurance claim for the named scope. It does not prove universal availability, safety, regulatory acceptance, vendor fitness, or immunity from cyber-physical failure.

MYTHOS-OT-0001 / OPERATIONAL TECHNOLOGY

1. Scope and Applicability

Boundary definition, audience, exclusions, and triggering conditions

IN SCOPE

- Architecture, acquisition, configuration, integration, and operation of industrial control systems and operational technology capabilities.
- Human, workload, device, service, network, data, facility, provider, supplier, and evidence dependencies.
- Design, deployment, monitoring, support, incident response, recovery, migration, and retirement.
- On-premises, hosted, managed, carrier, manufacturer, integrator, and externally operated components.

OUT OF SCOPE

- Claims of legal, safety, carrier, or regulatory approval without the responsible authority.
- Vendor certification treated as proof of the adopter configuration or operating effectiveness.
- Theoretical or laboratory behavior presented as production evidence without a declared boundary.
- Inherited controls without documented scope, owner, period, limitations, and shared responsibility.

Applicability triggers

- The capability supports a business-critical, safety-relevant, regulated, public, or externally dependent service.
- The environment can expose sensitive data, identity, control, physical process, communications, or shared infrastructure.
- A management plane, automation system, carrier, provider, or supplier can affect broad operational scope.
- Failure, compromise, or change can be difficult to detect, reverse, isolate, or recover.
- The system depends on hardware, firmware, radio, browser, cloud, endpoint, IoT, OT, or real-time media behavior.

MYTHOS-OT-0001 / OPERATIONAL TECHNOLOGY

2. Definitions and Taxonomy

Controlled language for architecture, governance, and assessment

AUTHORITY

The right to approve, deny, constrain, or execute an action. Model output is not authority.

EVIDENCE

A reproducible source, trace, measurement, test, record, signed artifact, or documented observation supporting a claim.

ASSURANCE VIEW

A defined perspective such as governance, architecture, security, operations, privacy, or human oversight.

ATOMIC CRITERION

A uniquely identified requirement that can be assessed, evidenced, excepted, and revised independently.

IMPLEMENTATION PROFILE

A risk-proportional set of required depth, evidence, and gates for a class of use.

CONTROLLED EXCEPTION

A time-bounded deviation with owner, rationale, compensating control, residual risk, expiration, and review trigger.

DETERMINISTIC MECHANISM

A component whose inputs, policy, state transition, and side effects can be constrained and verified.

SOURCE AUTHORITY

The documented basis for treating a source as reliable for a defined claim, context, jurisdiction, and time.

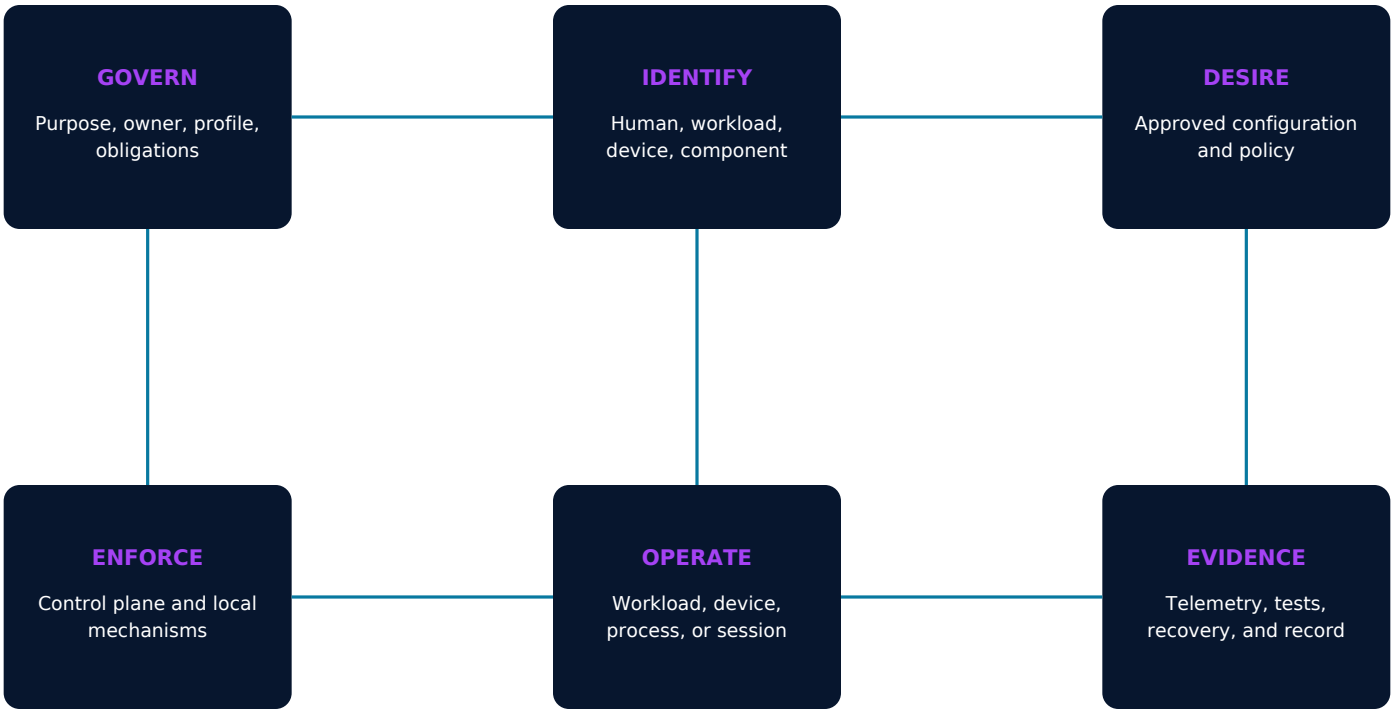
REASSESSMENT TRIGGER

A change, incident, drift signal, dependency revision, or time boundary requiring renewed evaluation.

MYTHOS-OT-0001 / OPERATIONAL TECHNOLOGY

3. Architecture and Trust Model

Separation of governance, management, enforcement, runtime, telemetry, and recovery



Mandatory trust boundaries

- Management planes MUST be isolated, strongly authenticated, monitored, and recoverable.
- Identity and policy MUST be evaluated at every provider, network, device, workload, and administrative transition.
- Runtime state MUST be compared with approved desired state and material drift MUST trigger response.
- Safety and continuity mechanisms MUST remain available when cloud, WAN, identity, DNS, time, carrier, or management dependencies fail.
- Evidence MUST be protected from the systems and administrators whose behavior it is intended to assess.

MYTHOS-OT-0001 / OPERATIONAL TECHNOLOGY

4. Governance and Accountability

Decision rights, separation of duties, and lifecycle ownership

ACCOUNTABLE OWNER

Accepts scope, risk tolerance, funding, and residual risk.

SYSTEM OWNER

Maintains architecture, inventory, operating boundaries, and change control.

SECURITY / PRIVACY

Defines threat, identity, data, isolation, monitoring, and incident requirements.

EVALUATION AUTHORITY

Designs representative tests, gates releases, and preserves reproducibility.

OPERATIONS

Maintains availability, rollback, recovery, evidence, and incident handling.

HUMAN OVERSIGHT

Reviews consequential decisions, exceptions, disputed outcomes, and harm signals.

DECISION PRINCIPLE

No single actor should be able to define the objective, grant authority, execute the consequential action, suppress evidence, and approve the result. Separation must be technical where consequence requires it.

MYTHOS-OT-0001 / OPERATIONAL TECHNOLOGY

5. Normative Control System

Atomic criteria 01-04 of 18

MYTHOS-OT-0001-C01 Process, Mission, and Safety Consequence Definition

The organization MUST define, implement, verify, and maintain process, mission, and safety consequence definition for in-scope industrial control systems and operational technology capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-OT-0001-C02 OT Asset, Firmware, Logic, and Dependency Inventory

The organization MUST define, implement, verify, and maintain ot asset, firmware, logic, and dependency inventory for in-scope industrial control systems and operational technology capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-OT-0001-C03 Industrial Zones, Conduits, and Security Levels

The organization MUST define, implement, verify, and maintain industrial zones, conduits, and security levels for in-scope industrial control systems and operational technology capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-OT-0001-C04 Industrial DMZ and IT-OT Boundary Enforcement

The organization MUST define, implement, verify, and maintain industrial dmz and it-ot boundary enforcement for in-scope industrial control systems and operational technology capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-OT-0001 / OPERATIONAL TECHNOLOGY

5. Normative Control System

Atomic criteria 05-08 of 18

MYTHOS-OT-0001-C05 Remote Access, Vendor Support, and Session Recording

The organization MUST define, implement, verify, and maintain remote access, vendor support, and session recording for in-scope industrial control systems and operational technology capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-OT-0001-C06 Engineering Workstation and Portable-Media Control

The organization MUST define, implement, verify, and maintain engineering workstation and portable-media control for in-scope industrial control systems and operational technology capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-OT-0001-C07 Controller, HMI, Historian, and Safety-System Separation

The organization MUST define, implement, verify, and maintain controller, hmi, historian, and safety-system separation for in-scope industrial control systems and operational technology capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-OT-0001-C08 Industrial Protocol and Command Authorization

The organization MUST define, implement, verify, and maintain industrial protocol and command authorization for in-scope industrial control systems and operational technology capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-OT-0001 / OPERATIONAL TECHNOLOGY

5. Normative Control System

Atomic criteria 09-12 of 18

MYTHOS-OT-0001-C09 Passive Monitoring and Safe Detection Methods

The organization MUST define, implement, verify, and maintain passive monitoring and safe detection methods for in-scope industrial control systems and operational technology capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-OT-0001-C10 Vulnerability Assessment and Compensating Controls

The organization MUST define, implement, verify, and maintain vulnerability assessment and compensating controls for in-scope industrial control systems and operational technology capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-OT-0001-C11 Change Windows, Hazard Review, and Rollback

The organization MUST define, implement, verify, and maintain change windows, hazard review, and rollback for in-scope industrial control systems and operational technology capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-OT-0001-C12 Logic, Configuration, Recipe, and Golden Backup

The organization MUST define, implement, verify, and maintain logic, configuration, recipe, and golden backup for in-scope industrial control systems and operational technology capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-OT-0001 / OPERATIONAL TECHNOLOGY

5. Normative Control System

Atomic criteria 13-15 of 18

MYTHOS-OT-0001-C13 Time, Naming, and Infrastructure Dependency Resilience

The organization MUST define, implement, verify, and maintain time, naming, and infrastructure dependency resilience for in-scope industrial control systems and operational technology capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-OT-0001-C14 Local Manual Operation and Safe Degradation

The organization MUST define, implement, verify, and maintain local manual operation and safe degradation for in-scope industrial control systems and operational technology capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-OT-0001-C15 Cyber-Physical Incident Command and Coordination

The organization MUST define, implement, verify, and maintain cyber-physical incident command and coordination for in-scope industrial control systems and operational technology capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-OT-0001 / OPERATIONAL TECHNOLOGY

5. Normative Control System

Atomic criteria 16-18 of 18

MYTHOS-OT-0001-C16 Recovery Order, Spare Strategy, and Restoration Exercise

The organization MUST define, implement, verify, and maintain recovery order, spare strategy, and restoration exercise for in-scope industrial control systems and operational technology capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-OT-0001-C17 Supplier, Integrator, and End-of-Life Governance

The organization MUST define, implement, verify, and maintain supplier, integrator, and end-of-life governance for in-scope industrial control systems and operational technology capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-OT-0001-C18 Safety Evidence, Exceptions, and Periodic Reassessment

The organization MUST define, implement, verify, and maintain safety evidence, exceptions, and periodic reassessment for in-scope industrial control systems and operational technology capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-OT-0001 / OPERATIONAL TECHNOLOGY

6. Evidence and Assessment

Examine, interview, test, observe, restore, and trace

EXAMINE

Policies, inventories, diagrams, configuration, code, firmware, contracts, provider evidence, logs, exceptions, and lifecycle records.

INTERVIEW

Accountable owners, architects, engineers, operators, safety personnel, security teams, vendors, carriers, integrators, and responders.

TEST

Identity, segmentation, policy, negative paths, malformed input, capacity stress, dependency loss, failover, rollback, and revocation.

RESTORE

Independent restoration of configuration, data, service, device, controller, cluster, media, or browser state from protected evidence.

OBSERVE

Production-like performance, quality, drift, resource use, physical behavior, alarms, user impact, and incident execution.

TRACE

End-to-end linkage from approved intent and identity through change, runtime behavior, telemetry, outcome, exception, and review.

Evidence grades

A

Independently reproducible, complete, current, integrity-protected, and representative.

B

Reproduced by the implementing organization with strong traceability and controlled scope.

C

Partially reproduced or indirect; limitations, inherited responsibility, and uncertainty recorded.

D

Assertion, diagram, design intent, certificate, or vendor statement without reproduced behavior.

MYTHOS-OT-0001 / OPERATIONAL TECHNOLOGY

7. Assurance Views

Six perspectives required for a complete assurance claim

GOVERNANCE

Ownership, purpose, risk tolerance, policy, exception, and decision rights.

ARCHITECTURE

Boundaries, dependencies, failure modes, state, data flow, and portability.

SECURITY

Identity, authorization, isolation, secrets, abuse resistance, and incident response.

OPERATIONS

Reliability, performance, cost, observability, change, recovery, and support.

PRIVACY / RIGHTS

Purpose limitation, minimization, consent, access, correction, deletion, and egress.

HUMAN / SOCIETAL

Usability, accessibility, disclosure, contestability, impact, and human control.

Conformance requires a defensible position across every applicable view. A strong aggregate score cannot compensate for an unowned system, a failed mandatory gate, untested recovery, or evidence below the accepted grade.

MYTHOS-OT-0001 / OPERATIONAL TECHNOLOGY

7.2 Evaluation Metrics and Decision Gates

Measures must expose service behavior, control effectiveness, failure, uncertainty, and cost

SERVICE

Availability, correctness, coverage, quality, latency, throughput, and user or process outcome.

CONTROL

Unauthorized attempt, policy denial, drift, segmentation escape, privilege, and exception exposure.

RESILIENCE

Failover success, recovery time, data loss, safe degradation, restore validity, and dependency survival.

SECURITY

Exploitability, credential exposure, supply-chain integrity, attack detection, containment, and revocation.

CAPACITY

Utilization, saturation, queueing, radio or fabric headroom, thermal margin, quota, and forecast error.

OPERATIONS

Change failure, mean time to detect, repair, support burden, vendor escalation, and evidence completeness.

PRIVACY / SAFETY

Unauthorized egress, retention breach, consent coverage, unsafe action, physical consequence, and contestability.

LIFECYCLE

Patch debt, end-of-support exposure, portability, replacement time, retirement completion, and reassessment age.

Decision gate: thresholds MUST be declared before assessment, cover representative load and failure conditions, and identify mandatory safety, identity, recovery, and evidence failures that block promotion.

MYTHOS-OT-0001 / OPERATIONAL TECHNOLOGY

8. Failure Modes and Adversarial Scenarios

Challenge assumptions before the system is trusted with consequential work

UNSAFE ASSESSMENT

Scanning or remediation disrupts a physical process or safety function.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

REMOTE ACCESS COMPROMISE

Vendor or engineering access bypasses zones, approval, or session oversight.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

SAFETY COUPLING

Cybersecurity failure propagates into a safety system assumed to be independent.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

LEGACY PROTOCOL ABUSE

Unauthenticated or unsafe commands cross an industrial conduit.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

LOGIC LOSS

Controller programs, recipes, or configurations cannot be restored correctly.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

RECOVERY SEQUENCE FAILURE

Systems restart in an order that damages process stability or equipment.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

Adversarial testing MUST protect people and production systems. Use isolated environments, synthetic or minimized data, bounded authority, kill switches, explicit legal and ethical review, and documented stop conditions.

MYTHOS-OT-0001 / OPERATIONAL TECHNOLOGY

9. Implementation Profiles

Risk-proportional implementation without weakening mandatory boundaries

FOUNDATIONAL

TYPICAL SCOPE

Low consequence, limited autonomy, non-sensitive data, reversible outputs.

MINIMUM DEPTH

Named owner; inventory; basic evaluation; access control; logging; rollback; annual review.

ADVANCED

TYPICAL SCOPE

Business-critical workflow, sensitive data, multiple tools or providers, moderate autonomy.

MINIMUM DEPTH

Formal threat model; representative evaluation; approval gates; isolated execution; tested recovery; quarterly review.

HIGH ASSURANCE

TYPICAL SCOPE

Safety, security, regulated, financial, identity, or broadly consequential use.

MINIMUM DEPTH

Independent challenge; strongest identity; deterministic enforcement; comprehensive evidence; canary release; continuous monitoring.

Profile selection rule

Choose the highest profile triggered by consequence, autonomy, sensitivity, scale, irreversibility, external dependency, or inability to rapidly detect and recover. Tailoring may strengthen controls; it may not erase a mandatory gate without a controlled exception.

MYTHOS-OT-0001 / OPERATIONAL TECHNOLOGY

9.2 Operational Lifecycle

Evidence-bearing operation from concept through retirement

1. **CONCEPT**
   Purpose, owner, impact, boundaries
2. **DESIGN**
   Architecture, threat, data, identity
3. **BUILD**
   Controlled implementation and provenance
4. **EVALUATE**
   Representative, adversarial, recovery tests
5. **RELEASE**
   Approval, canary, rollback, evidence
6. **OPERATE**
   Observe, support, detect, respond
7. **CHANGE**
   Reassess, compare, gate, reverse
8. **RETIRE**
   Revoke, export, dispose, preserve

LIFECYCLE INVARIANT

Every stage MUST leave sufficient evidence for the next authority to understand what was intended, what changed, what was tested, what failed, what was accepted, what remains uncertain, and how to recover or exit.

MYTHOS-OT-0001 / OPERATIONAL TECHNOLOGY

10. Adoption Roadmap

A sequenced path from uncontrolled capability to evidence-bearing operation

0-30 DAYS

Establish ownership and truth

Inventory systems and dependencies; classify risk; freeze unsupported claims; identify immediate privilege, data, and evidence gaps.

31-90 DAYS

Create enforceable boundaries

Define policy; isolate execution; implement identity and approval gates; establish representative evaluation and baseline telemetry.

3-6 MONTHS

Prove recovery and operating control

Exercise failure, rollback, revocation, incident, provider exit, and state-recovery scenarios; mature evidence packaging.

6-12 MONTHS

Scale assurance

Automate control checks; expand workload coverage; independent challenge; portfolio metrics; controlled exception expiry; continuous reassessment.

Adoption is complete only when operating evidence demonstrates the selected profile in the actual deployment context. Documentation alone is not conformance.

MYTHOS-OT-0001 / OPERATIONAL TECHNOLOGY

10.2 Conformance and Exception Statement

What an assurance claim must contain

SYSTEM / SERVICE

Named, versioned capability and deployment boundary.

PROFILE

Foundational, Advanced, or High Assurance with trigger rationale.

SCOPE

Workloads, users, data, tools, regions, providers, and exclusions.

CRITERIA

Pass, partial, fail, not applicable, and exception status for every criterion.

EVIDENCE

Artifact references, evidence grade, date, environment, and reproducibility.

LIMITATIONS

Known failure modes, unsupported workloads, uncertainty, and residual risk.

EXCEPTIONS

Owner, rationale, compensating control, expiration, and approval.

VALIDITY

Assessment date, change boundary, review date, and reassessment triggers.

Prohibited claim: “compliant” without the named scope, profile, evidence date, limitations, exceptions, and authority accepting residual risk.

MYTHOS-OT-0001 / OPERATIONAL TECHNOLOGY

11. Source Authority and Reference Register

Primary sources informing interpretation; current obligations and provider behavior must be verified at assessment t

NIST CSF 2.0 — **Cybersecurity Framework 2.0**

https://doi.org/10.6028/NIST.CSWP.29

Enterprise governance and cybersecurity outcome framework.

NIST SP 800-53 Rev. 5 — **Security and Privacy Controls for Information Systems and Organizations**

https://doi.org/10.6028/NIST.SP.800-53r5

Broad control baseline for organizational systems and services.

NIST SP 800-82 Rev. 3 — **Guide to Operational Technology Security**

https://doi.org/10.6028/NIST.SP.800-82r3

OT architecture, threats, controls, safety context, and lifecycle guidance.

IEC 62443 Series — **Security for industrial automation and control systems**

https://www.iec.ch/industrial-cyber-security

Industrial security lifecycle, zones, conduits, and component requirements.

MITRE ATT&CK for ICS — **ATT&CK for Industrial Control Systems**

https://attack.mitre.org/matrices/ics/

Adversary behavior and technique knowledge base for ICS.

CISA ICS — **Industrial Control Systems Cybersecurity Guidance**

https://www.cisa.gov/topics/industrial-control-systems

Operational advisories, practices, and incident resources.

Source currency rule: authorities, specifications, legal obligations, provider services, firmware, browser behavior, carrier requirements, and operational evidence MUST be checked at assessment time. This publication records a 2026-07-24 source snapshot.

MYTHOS-OT-0001 / OPERATIONAL TECHNOLOGY

11.2 Revision History and Decision Register

Permanent identity, controlled change, and publication integrity

VERSION	DATE	STATE	SUMMARY
1.0.0-candidate	2026-07-24	Candidate	Permanent-publication edition; source refresh; expanded criteria, evidence, assurance, and lifecycle guidance.
Next review	2027-01-24	Scheduled	Review source currency, threat evolution, operating evidence, adoption feedback, and proposed revisions.

Publication decisions

PERMANENT IDENTITY The document ID remains stable across revisions; batch or production sequence metadata is not public identity.

CHANGE CONTROL Normative changes require rationale, impact analysis, affected-criterion mapping, and approval.

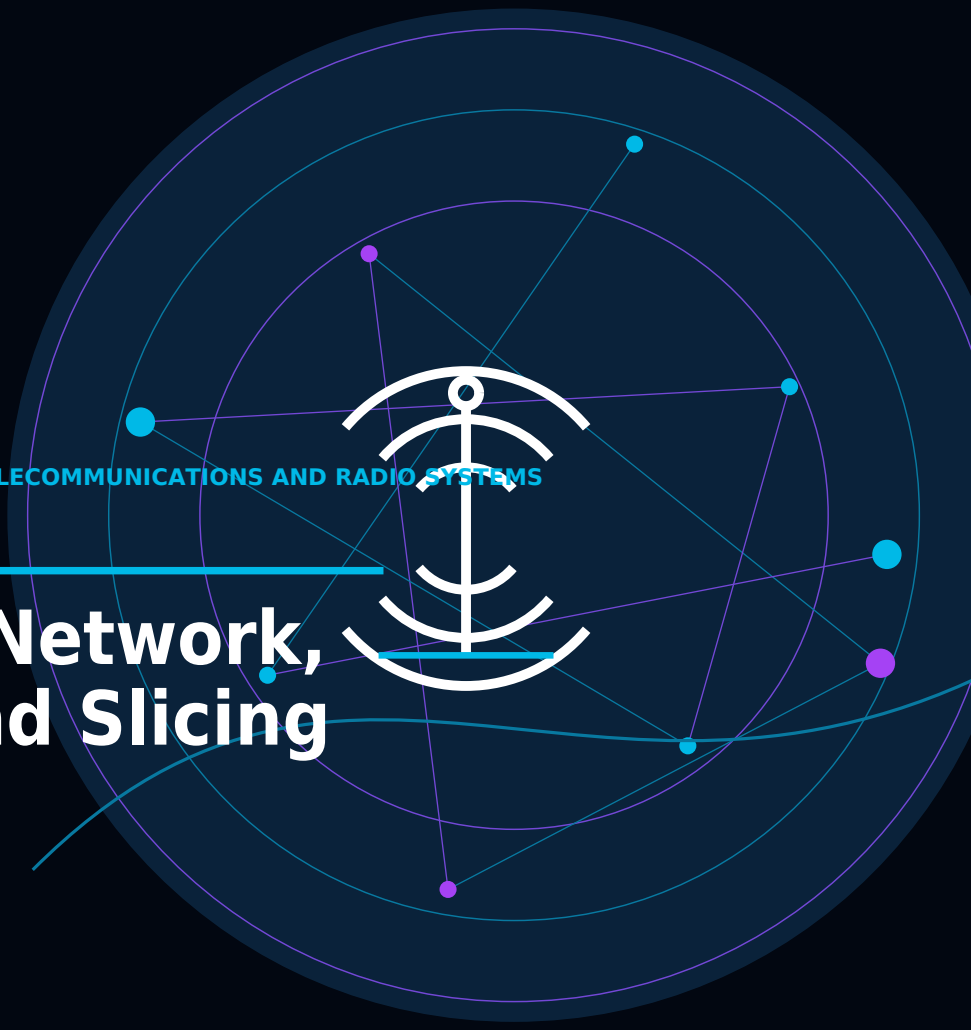
SUPERSESSION A superseded edition remains traceable; current routes and catalogs identify the active edition.

CORRECTION Material errors require a recorded correction, affected-evidence analysis, and notification appropriate to impact.

ARCHIVAL INTEGRITY Published artifacts receive hashes, metadata, and a release manifest sufficient for independent verification.

END OF PERMANENT PUBLICATION / MYTHOS-OT-0001

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY



ENGINEERING STANDARDS LIBRARY / TELECOMMUNICATIONS AND RADIO SYSTEMS

5G, 6G, Core Network, Open RAN, and Slicing Standard

Mobile-core and RAN architecture, identity, slicing, timing, MEC, cloud-native functions, assurance, and lifecycle governance.

PERMANENT DOCUMENT ID

MYTHOS-TEL-0001

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

5G, 6G, Core Network, Open RAN, and Slicing Standard

DOCUMENT ID
MYTHOS-TEL-0001

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

18	6	4	1
ATOMIC CRITERIA	ASSURANCE VIEWS	IMPLEMENTATION PROFILES	PERMANENT IDENTITY

Publication doctrine. This publication establishes durable requirements for its defined scope. Interpretation must preserve the permanent document identity, required evidence, controlled exceptions, source currency, and formal revision history.

INFRASTRUCTURE AND CONTROL TOPOLOGY

Telecommunications and Radio Systems

A trustworthy 5G, 6G, Open RAN, and network slicing system is a governed chain of ownership, identity, desired state, enforcement, workload or device behavior, telemetry, recovery, and independently reviewable evidence.

OWNERSHIP

Purpose, service boundary, accountable owner, suppliers, users, and retained responsibility remain explicit.

ENFORCEMENT

Identity, policy, segmentation, configuration, and local safety mechanisms constrain every trust transition.

CONTINUITY

Capacity, dependency loss, safe degradation, backup, restoration, rollback, and exit are tested before reliance.

EVIDENCE

Inventory, desired state, runtime observation, tests, incidents, exceptions, and change records form the assurance chain.

GOVERNED INFRASTRUCTURE FLOW

Reconciliation, detection, response, restoration, and controlled evolution

INTERPRETATION AND ASSURANCE

How to apply this publication

Normative intent

Requirements define outcomes and constraints. Implementations may vary, but evidence must demonstrate equivalent control.

Evidence over assertion

Vendor documentation and design intent are not equivalent to reproduced behavior. Record evidence grade and uncertainty.

Risk-proportional depth

Higher consequence, autonomy, sensitivity, and irreversibility require stronger isolation, review, verification, and recovery.

Controlled exception

Exceptions require an owner, rationale, compensating control, residual-risk acceptance, expiration, and reevaluation trigger.

Normalized assessment scale

0	1	2	3	4	5
ABSENT	AD HOC	PARTIAL	OPERATIONAL	ADVANCED	LEADING

A numeric score must never hide a failed mandatory gate, missing evidence, untested workload, or unacceptable residual risk.

INTERACTIVE NAVIGATION

Contents

Navigation doctrine

Bookmarks and internal links are conveniences. Permanent document identity, criterion identifiers, evidence references, and revision history remain authoritative.

MYTHOS-TEL-0001 / TELECOMMUNICATIONS

0. Executive Mandate

Purpose, strategic outcome, and non-negotiable operating doctrine

MANDATE

Organizations adopting 5G, 6G, Open RAN, and network slicing capabilities MUST define an owned service boundary, establish enforceable identity and configuration, protect data and safety, test failure and recovery, and preserve evidence sufficient for independent review.

Required outcomes

- Create a durable governance boundary for 5G, 6G, Open RAN, and network slicing across design, acquisition, deployment, operation, change, and retirement.
- Require risk-proportional segmentation, identity, configuration, telemetry, resilience, recovery, and supplier controls.
- Prevent central automation, administrative tooling, or provider services from becoming unbounded authority.
- Prove operation with representative workloads, devices, failure modes, and restoration exercises.
- Define measurable conformance, exceptions, reassessment triggers, and a viable exit path.

Decision boundary: conformance supports a documented assurance claim for the named scope. It does not prove universal availability, safety, regulatory acceptance, vendor fitness, or immunity from cyber-physical failure.

MYTHOS-TEL-0001 / TELECOMMUNICATIONS

1. Scope and Applicability

Boundary definition, audience, exclusions, and triggering conditions

IN SCOPE

- Architecture, acquisition, configuration, integration, and operation of 5G, 6G, Open RAN, and network slicing capabilities.
- Human, workload, device, service, network, data, facility, provider, supplier, and evidence dependencies.
- Design, deployment, monitoring, support, incident response, recovery, migration, and retirement.
- On-premises, hosted, managed, carrier, manufacturer, integrator, and externally operated components.

OUT OF SCOPE

- Claims of legal, safety, carrier, or regulatory approval without the responsible authority.
- Vendor certification treated as proof of the adopter configuration or operating effectiveness.
- Theoretical or laboratory behavior presented as production evidence without a declared boundary.
- Inherited controls without documented scope, owner, period, limitations, and shared responsibility.

Applicability triggers

- The capability supports a business-critical, safety-relevant, regulated, public, or externally dependent service.
- The environment can expose sensitive data, identity, control, physical process, communications, or shared infrastructure.
- A management plane, automation system, carrier, provider, or supplier can affect broad operational scope.
- Failure, compromise, or change can be difficult to detect, reverse, isolate, or recover.
- The system depends on hardware, firmware, radio, browser, cloud, endpoint, IoT, OT, or real-time media behavior.

MYTHOS-TEL-0001 / TELECOMMUNICATIONS

2. Definitions and Taxonomy

Controlled language for architecture, governance, and assessment

AUTHORITY

The right to approve, deny, constrain, or execute an action. Model output is not authority.

EVIDENCE

A reproducible source, trace, measurement, test, record, signed artifact, or documented observation supporting a claim.

ASSURANCE VIEW

A defined perspective such as governance, architecture, security, operations, privacy, or human oversight.

ATOMIC CRITERION

A uniquely identified requirement that can be assessed, evidenced, excepted, and revised independently.

IMPLEMENTATION PROFILE

A risk-proportional set of required depth, evidence, and gates for a class of use.

CONTROLLED EXCEPTION

A time-bounded deviation with owner, rationale, compensating control, residual risk, expiration, and review trigger.

DETERMINISTIC MECHANISM

A component whose inputs, policy, state transition, and side effects can be constrained and verified.

SOURCE AUTHORITY

The documented basis for treating a source as reliable for a defined claim, context, jurisdiction, and time.

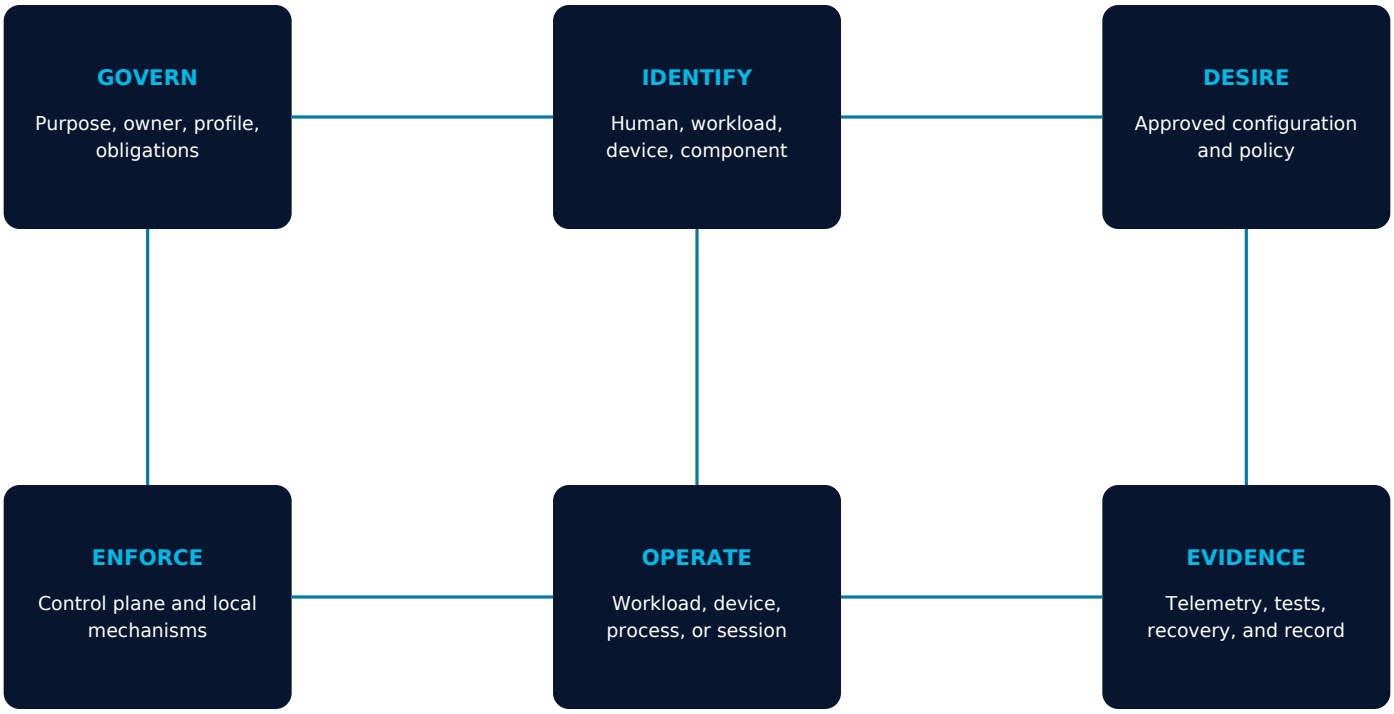
REASSESSMENT TRIGGER

A change, incident, drift signal, dependency revision, or time boundary requiring renewed evaluation.

MYTHOS-TEL-0001 / TELECOMMUNICATIONS

3. Architecture and Trust Model

Separation of governance, management, enforcement, runtime, telemetry, and recovery



Mandatory trust boundaries

- Management planes MUST be isolated, strongly authenticated, monitored, and recoverable.
- Identity and policy MUST be evaluated at every provider, network, device, workload, and administrative transition.
- Runtime state MUST be compared with approved desired state and material drift MUST trigger response.
- Safety and continuity mechanisms MUST remain available when cloud, WAN, identity, DNS, time, carrier, or management dependencies fail.
- Evidence MUST be protected from the systems and administrators whose behavior it is intended to assess.

MYTHOS-TEL-0001 / TELECOMMUNICATIONS

4. Governance and Accountability

Decision rights, separation of duties, and lifecycle ownership

ACCOUNTABLE OWNER

Accepts scope, risk tolerance, funding, and residual risk.

SYSTEM OWNER

Maintains architecture, inventory, operating boundaries, and change control.

SECURITY / PRIVACY

Defines threat, identity, data, isolation, monitoring, and incident requirements.

EVALUATION AUTHORITY

Designs representative tests, gates releases, and preserves reproducibility.

OPERATIONS

Maintains availability, rollback, recovery, evidence, and incident handling.

HUMAN OVERSIGHT

Reviews consequential decisions, exceptions, disputed outcomes, and harm signals.

DECISION PRINCIPLE

No single actor should be able to define the objective, grant authority, execute the consequential action, suppress evidence, and approve the result. Separation must be technical where consequence requires it.

MYTHOS-TEL-0001 / TELECOMMUNICATIONS

5. Normative Control System

Atomic criteria 01-04 of 18

MYTHOS-TEL-0001-C01 Release Profile, Service Scope, and Regulatory Boundary

The organization MUST define, implement, verify, and maintain release profile, service scope, and regulatory boundary for in-scope 5G, 6G, Open RAN, and network slicing capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-TEL-0001-C02 Subscriber, SIM, eSIM, and Network Identity

The organization MUST define, implement, verify, and maintain subscriber, sim, esim, and network identity for in-scope 5G, 6G, Open RAN, and network slicing capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-TEL-0001-C03 RAN, Transport, Core, and Edge Trust Domains

The organization MUST define, implement, verify, and maintain ran, transport, core, and edge trust domains for in-scope 5G, 6G, Open RAN, and network slicing capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-TEL-0001-C04 Service-Based Interface and API Protection

The organization MUST define, implement, verify, and maintain service-based interface and api protection for in-scope 5G, 6G, Open RAN, and network slicing capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-TEL-0001 / TELECOMMUNICATIONS

5. Normative Control System

Atomic criteria 05-08 of 18

MYTHOS-TEL-0001-C05 Control-Plane and User-Plane Separation

The organization MUST define, implement, verify, and maintain control-plane and user-plane separation for in-scope 5G, 6G, Open RAN, and network slicing capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-TEL-0001-C06 Slice Definition, Admission, and Isolation Evidence

The organization MUST define, implement, verify, and maintain slice definition, admission, and isolation evidence for in-scope 5G, 6G, Open RAN, and network slicing capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-TEL-0001-C07 QoS, Policy, Charging, and Resource Enforcement

The organization MUST define, implement, verify, and maintain qos, policy, charging, and resource enforcement for in-scope 5G, 6G, Open RAN, and network slicing capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-TEL-0001-C08 Open RAN SMO, RIC, xApp, and rApp Governance

The organization MUST define, implement, verify, and maintain open ran smo, ric, xapp, and rapp governance for in-scope 5G, 6G, Open RAN, and network slicing capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-TEL-0001 / TELECOMMUNICATIONS

5. Normative Control System

Atomic criteria 09-12 of 18

MYTHOS-TEL-0001-C09 Virtualized and Cloud-Native Network Function Security

The organization MUST define, implement, verify, and maintain virtualized and cloud-native network function security for in-scope 5G, 6G, Open RAN, and network slicing capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-TEL-0001-C10 MEC Workload, Data, and Local-Breakout Control

The organization MUST define, implement, verify, and maintain mec workload, data, and local-breakout control for in-scope 5G, 6G, Open RAN, and network slicing capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-TEL-0001-C11 Timing, Synchronization, and Positioning Resilience

The organization MUST define, implement, verify, and maintain timing, synchronization, and positioning resilience for in-scope 5G, 6G, Open RAN, and network slicing capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-TEL-0001-C12 Roaming, Interconnect, and Partner Trust

The organization MUST define, implement, verify, and maintain roaming, interconnect, and partner trust for in-scope 5G, 6G, Open RAN, and network slicing capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-TEL-0001 / TELECOMMUNICATIONS

5. Normative Control System

Atomic criteria 13-15 of 18

MYTHOS-TEL-0001-C13 Lawful, Emergency, and Priority-Service Obligations

The organization MUST define, implement, verify, and maintain lawful, emergency, and priority-service obligations for in-scope 5G, 6G, Open RAN, and network slicing capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-TEL-0001-C14 Signaling, Fraud, and Abuse Detection

The organization MUST define, implement, verify, and maintain signaling, fraud, and abuse detection for in-scope 5G, 6G, Open RAN, and network slicing capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-TEL-0001-C15 Capacity, Mobility, and Service Quality Assurance

The organization MUST define, implement, verify, and maintain capacity, mobility, and service quality assurance for in-scope 5G, 6G, Open RAN, and network slicing capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-TEL-0001 / TELECOMMUNICATIONS

5. Normative Control System

Atomic criteria 16-18 of 18

MYTHOS-TEL-0001-C16 Software, Model, and Supply-Chain Governance

The organization MUST define, implement, verify, and maintain software, model, and supply-chain governance for in-scope 5G, 6G, Open RAN, and network slicing capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-TEL-0001-C17 Outage, Disaster, and Regional Recovery

The organization MUST define, implement, verify, and maintain outage, disaster, and regional recovery for in-scope 5G, 6G, Open RAN, and network slicing capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-TEL-0001-C18 Lifecycle Evidence and Technology Reassessment

The organization MUST define, implement, verify, and maintain lifecycle evidence and technology reassessment for in-scope 5G, 6G, Open RAN, and network slicing capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-TEL-0001 / TELECOMMUNICATIONS

6. Evidence and Assessment

Examine, interview, test, observe, restore, and trace

EXAMINE

Policies, inventories, diagrams, configuration, code, firmware, contracts, provider evidence, logs, exceptions, and lifecycle records.

INTERVIEW

Accountable owners, architects, engineers, operators, safety personnel, security teams, vendors, carriers, integrators, and responders.

TEST

Identity, segmentation, policy, negative paths, malformed input, capacity stress, dependency loss, failover, rollback, and revocation.

RESTORE

Independent restoration of configuration, data, service, device, controller, cluster, media, or browser state from protected evidence.

OBSERVE

Production-like performance, quality, drift, resource use, physical behavior, alarms, user impact, and incident execution.

TRACE

End-to-end linkage from approved intent and identity through change, runtime behavior, telemetry, outcome, exception, and review.

Evidence grades

A

Independently reproducible, complete, current, integrity-protected, and representative.

B

Reproduced by the implementing organization with strong traceability and controlled scope.

C

Partially reproduced or indirect; limitations, inherited responsibility, and uncertainty recorded.

D

Assertion, diagram, design intent, certificate, or vendor statement without reproduced behavior.

MYTHOS-TEL-0001 / TELECOMMUNICATIONS

7. Assurance Views

Six perspectives required for a complete assurance claim

GOVERNANCE

Ownership, purpose, risk tolerance, policy, exception, and decision rights.

ARCHITECTURE

Boundaries, dependencies, failure modes, state, data flow, and portability.

SECURITY

Identity, authorization, isolation, secrets, abuse resistance, and incident response.

OPERATIONS

Reliability, performance, cost, observability, change, recovery, and support.

PRIVACY / RIGHTS

Purpose limitation, minimization, consent, access, correction, deletion, and egress.

HUMAN / SOCIETAL

Usability, accessibility, disclosure, contestability, impact, and human control.

Conformance requires a defensible position across every applicable view. A strong aggregate score cannot compensate for an unowned system, a failed mandatory gate, untested recovery, or evidence below the accepted grade.

MYTHOS-TEL-0001 / TELECOMMUNICATIONS

7.2 Evaluation Metrics and Decision Gates

Measures must expose service behavior, control effectiveness, failure, uncertainty, and cost

SERVICE

Availability, correctness, coverage, quality, latency, throughput, and user or process outcome.

CONTROL

Unauthorized attempt, policy denial, drift, segmentation escape, privilege, and exception exposure.

RESILIENCE

Failover success, recovery time, data loss, safe degradation, restore validity, and dependency survival.

SECURITY

Exploitability, credential exposure, supply-chain integrity, attack detection, containment, and revocation.

CAPACITY

Utilization, saturation, queueing, radio or fabric headroom, thermal margin, quota, and forecast error.

OPERATIONS

Change failure, mean time to detect, repair, support burden, vendor escalation, and evidence completeness.

PRIVACY / SAFETY

Unauthorized egress, retention breach, consent coverage, unsafe action, physical consequence, and contestability.

LIFECYCLE

Patch debt, end-of-support exposure, portability, replacement time, retirement completion, and reassessment age.

Decision gate: thresholds MUST be declared before assessment, cover representative load and failure conditions, and identify mandatory safety, identity, recovery, and evidence failures that block promotion.

MYTHOS-TEL-0001 / TELECOMMUNICATIONS

8. Failure Modes and Adversarial Scenarios

Challenge assumptions before the system is trusted with consequential work

SUBSCRIBER IDENTITY COMPROMISE

SIM, eSIM, credential, or provisioning weakness enables impersonation.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

SLICE ISOLATION FAILURE

A label implies isolation not enforced across radio, transport, core, edge, and cloud.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

SIGNALING ABUSE

Protocol manipulation causes fraud, denial, interception, or unauthorized routing.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

TIMING LOSS

Synchronization failure degrades radio, handoff, positioning, or service quality.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

RF INTERFERENCE

Coverage, PIM, noise, or external interference defeats expected service.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

EMERGENCY-SERVICE FAILURE

Calling, location, priority, public-safety, or survivability obligations are not met.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

Adversarial testing MUST protect people and production systems. Use isolated environments, synthetic or minimized data, bounded authority, kill switches, explicit legal and ethical review, and documented stop conditions.

MYTHOS-TEL-0001 / TELECOMMUNICATIONS

9. Implementation Profiles

Risk-proportional implementation without weakening mandatory boundaries

FOUNDATIONAL

TYPICAL SCOPE

Low consequence, limited autonomy, non-sensitive data, reversible outputs.

MINIMUM DEPTH

Named owner; inventory; basic evaluation; access control; logging; rollback; annual review.

ADVANCED

TYPICAL SCOPE

Business-critical workflow, sensitive data, multiple tools or providers, moderate autonomy.

MINIMUM DEPTH

Formal threat model; representative evaluation; approval gates; isolated execution; tested recovery; quarterly review.

HIGH ASSURANCE

TYPICAL SCOPE

Safety, security, regulated, financial, identity, or broadly consequential use.

MINIMUM DEPTH

Independent challenge; strongest identity; deterministic enforcement; comprehensive evidence; canary release; continuous monitoring.

Profile selection rule

Choose the highest profile triggered by consequence, autonomy, sensitivity, scale, irreversibility, external dependency, or inability to rapidly detect and recover. Tailoring may strengthen controls; it may not erase a mandatory gate without a controlled exception.

MYTHOS-TEL-0001 / TELECOMMUNICATIONS

9.2 Operational Lifecycle

Evidence-bearing operation from concept through retirement

1 **CONCEPT**
Purpose, owner, impact, boundaries

2 **DESIGN**
Architecture, threat, data, identity

3 **BUILD**
Controlled implementation and provenance

4 **EVALUATE**
Representative, adversarial, recovery tests

5 **RELEASE**
Approval, canary, rollback, evidence

6 **OPERATE**
Observe, support, detect, respond

7 **CHANGE**
Reassess, compare, gate, reverse

8 **RETIRE**
Revoke, export, dispose, preserve

LIFECYCLE INVARIANT

Every stage MUST leave sufficient evidence for the next authority to understand what was intended, what changed, what was tested, what failed, what was accepted, what remains uncertain, and how to recover or exit.

MYTHOS-TEL-0001 / TELECOMMUNICATIONS

10. Adoption Roadmap

A sequenced path from uncontrolled capability to evidence-bearing operation

0-30 DAYS

Establish ownership and truth

Inventory systems and dependencies; classify risk; freeze unsupported claims; identify immediate privilege, data, and evidence gaps.

31-90 DAYS

Create enforceable boundaries

Define policy; isolate execution; implement identity and approval gates; establish representative evaluation and baseline telemetry.

3-6 MONTHS

Prove recovery and operating control

Exercise failure, rollback, revocation, incident, provider exit, and state-recovery scenarios; mature evidence packaging.

6-12 MONTHS

Scale assurance

Automate control checks; expand workload coverage; independent challenge; portfolio metrics; controlled exception expiry; continuous reassessment.

Adoption is complete only when operating evidence demonstrates the selected profile in the actual deployment context. Documentation alone is not conformance.

MYTHOS-TEL-0001 / TELECOMMUNICATIONS

10.2 Conformance and Exception Statement

What an assurance claim must contain

SYSTEM / SERVICE

Named, versioned capability and deployment boundary.

PROFILE

Foundational, Advanced, or High Assurance with trigger rationale.

SCOPE

Workloads, users, data, tools, regions, providers, and exclusions.

CRITERIA

Pass, partial, fail, not applicable, and exception status for every criterion.

EVIDENCE

Artifact references, evidence grade, date, environment, and reproducibility.

LIMITATIONS

Known failure modes, unsupported workloads, uncertainty, and residual risk.

EXCEPTIONS

Owner, rationale, compensating control, expiration, and approval.

VALIDITY

Assessment date, change boundary, review date, and reassessment triggers.

Prohibited claim: “compliant” without the named scope, profile, evidence date, limitations, exceptions, and authority accepting residual risk.

MYTHOS-TEL-0001 / TELECOMMUNICATIONS

11. Source Authority and Reference Register

Primary sources informing interpretation; current obligations and provider behavior must be verified at assessment t

NIST CSF 2.0 — **Cybersecurity Framework 2.0**

https://doi.org/10.6028/NIST.CSWP.29

Enterprise governance and cybersecurity outcome framework.

NIST SP 800-53 Rev. 5 — **Security and Privacy Controls for Information Systems and Organizations**

https://doi.org/10.6028/NIST.SP.800-53r5

Broad control baseline for organizational systems and services.

3GPP Specifications — **3GPP technical specifications and reports**

https://www.3gpp.org/specifications-technologies/specifications-by-series

Mobile-system architecture, security, radio, core, and service specifications.

O-RAN Alliance — **O-RAN specifications and security work**

https://www.o-ran.org/specifications

Open RAN architecture, interfaces, management, and security.

GSMA NESAS — **Network Equipment Security Assurance Scheme**

https://www.gsma.com/solutions-and-impact/technologies/security/network-equipment-security-assurance-scheme/

Telecom equipment security development and assessment scheme.

IETF SIP — **Session Initiation Protocol - RFC 3261**

https://www.rfc-editor.org/rfc/rfc3261

Core SIP signaling model and protocol semantics.

Source currency rule: authorities, specifications, legal obligations, provider services, firmware, browser behavior, carrier requirements, and operational evidence MUST be checked at assessment time. This publication records a 2026-07-24 source snapshot.

MYTHOS-TEL-0001 / TELECOMMUNICATIONS

11.2 Revision History and Decision Register

Permanent identity, controlled change, and publication integrity

VERSION	DATE	STATE	SUMMARY
1.0.0-candidate	2026-07-24	Candidate	Permanent-publication edition; source refresh; expanded criteria, evidence, assurance, and lifecycle guidance.
Next review	2027-01-24	Scheduled	Review source currency, threat evolution, operating evidence, adoption feedback, and proposed revisions.

Publication decisions

PERMANENT IDENTITY The document ID remains stable across revisions; batch or production sequence metadata is not public identity.

CHANGE CONTROL Normative changes require rationale, impact analysis, affected-criterion mapping, and approval.

SUPERSESSION A superseded edition remains traceable; current routes and catalogs identify the active edition.

CORRECTION Material errors require a recorded correction, affected-evidence analysis, and notification appropriate to impact.

ARCHIVAL INTEGRITY Published artifacts receive hashes, metadata, and a release manifest sufficient for independent verification.

END OF PERMANENT PUBLICATION / MYTHOS-TEL-0001

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY

ENGINEERING STANDARDS LIBRARY / TELECOMMUNICATIONS AND RADIO SYSTEMS

Distributed Antenna Systems and Indoor RF Standard

Indoor RF, DAS, neutral-host, public-safety coverage, interference, power, survivability, acceptance, and operational evidence.

PERMANENT DOCUMENT ID

MYTHOS-TEL-0002

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

Distributed Antenna Systems and Indoor RF Standard

DOCUMENT ID
MYTHOS-TEL-0002

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

18	6	4	1
ATOMIC CRITERIA	ASSURANCE VIEWS	IMPLEMENTATION PROFILES	PERMANENT IDENTITY

Publication doctrine. This publication establishes durable requirements for its defined scope. Interpretation must preserve the permanent document identity, required evidence, controlled exceptions, source currency, and formal revision history.

INFRASTRUCTURE AND CONTROL TOPOLOGY

Telecommunications and Radio Systems

A trustworthy distributed antenna systems and indoor RF system is a governed chain of ownership, identity, desired state, enforcement, workload or device behavior, telemetry, recovery, and independently reviewable evidence.

OWNERSHIP

Purpose, service boundary, accountable owner, suppliers, users, and retained responsibility remain explicit.

ENFORCEMENT

Identity, policy, segmentation, configuration, and local safety mechanisms constrain every trust transition.

CONTINUITY

Capacity, dependency loss, safe degradation, backup, restoration, rollback, and exit are tested before reliance.

EVIDENCE

Inventory, desired state, runtime observation, tests, incidents, exceptions, and change records form the assurance chain.

GOVERNED INFRASTRUCTURE FLOW

Reconciliation, detection, response, restoration, and controlled evolution

INTERPRETATION AND ASSURANCE

How to apply this publication

Normative intent

Requirements define outcomes and constraints. Implementations may vary, but evidence must demonstrate equivalent control.

Evidence over assertion

Vendor documentation and design intent are not equivalent to reproduced behavior. Record evidence grade and uncertainty.

Risk-proportional depth

Higher consequence, autonomy, sensitivity, and irreversibility require stronger isolation, review, verification, and recovery.

Controlled exception

Exceptions require an owner, rationale, compensating control, residual-risk acceptance, expiration, and reevaluation trigger.

Normalized assessment scale

0	1	2	3	4	5
ABSENT	AD HOC	PARTIAL	OPERATIONAL	ADVANCED	LEADING

A numeric score must never hide a failed mandatory gate, missing evidence, untested workload, or unacceptable residual risk.

INTERACTIVE NAVIGATION

Contents

Navigation doctrine

Bookmarks and internal links are conveniences. Permanent document identity, criterion identifiers, evidence references, and revision history remain authoritative.

MYTHOS-TEL-0002 / TELECOMMUNICATIONS

0. Executive Mandate

Purpose, strategic outcome, and non-negotiable operating doctrine

MANDATE

Organizations adopting distributed antenna systems and indoor RF capabilities MUST define an owned service boundary, establish enforceable identity and configuration, protect data and safety, test failure and recovery, and preserve evidence sufficient for independent review.

Required outcomes

- Create a durable governance boundary for distributed antenna systems and indoor RF across design, acquisition, deployment, operation, change, and retirement.
- Require risk-proportional segmentation, identity, configuration, telemetry, resilience, recovery, and supplier controls.
- Prevent central automation, administrative tooling, or provider services from becoming unbounded authority.
- Prove operation with representative workloads, devices, failure modes, and restoration exercises.
- Define measurable conformance, exceptions, reassessment triggers, and a viable exit path.

Decision boundary: conformance supports a documented assurance claim for the named scope. It does not prove universal availability, safety, regulatory acceptance, vendor fitness, or immunity from cyber-physical failure.

MYTHOS-TEL-0002 / TELECOMMUNICATIONS

1. Scope and Applicability

Boundary definition, audience, exclusions, and triggering conditions

IN SCOPE

- Architecture, acquisition, configuration, integration, and operation of distributed antenna systems and indoor RF capabilities.
- Human, workload, device, service, network, data, facility, provider, supplier, and evidence dependencies.
- Design, deployment, monitoring, support, incident response, recovery, migration, and retirement.
- On-premises, hosted, managed, carrier, manufacturer, integrator, and externally operated components.

OUT OF SCOPE

- Claims of legal, safety, carrier, or regulatory approval without the responsible authority.
- Vendor certification treated as proof of the adopter configuration or operating effectiveness.
- Theoretical or laboratory behavior presented as production evidence without a declared boundary.
- Inherited controls without documented scope, owner, period, limitations, and shared responsibility.

Applicability triggers

- The capability supports a business-critical, safety-relevant, regulated, public, or externally dependent service.
- The environment can expose sensitive data, identity, control, physical process, communications, or shared infrastructure.
- A management plane, automation system, carrier, provider, or supplier can affect broad operational scope.
- Failure, compromise, or change can be difficult to detect, reverse, isolate, or recover.
- The system depends on hardware, firmware, radio, browser, cloud, endpoint, IoT, OT, or real-time media behavior.

MYTHOS-TEL-0002 / TELECOMMUNICATIONS

2. Definitions and Taxonomy

Controlled language for architecture, governance, and assessment

AUTHORITY

The right to approve, deny, constrain, or execute an action. Model output is not authority.

EVIDENCE

A reproducible source, trace, measurement, test, record, signed artifact, or documented observation supporting a claim.

ASSURANCE VIEW

A defined perspective such as governance, architecture, security, operations, privacy, or human oversight.

ATOMIC CRITERION

A uniquely identified requirement that can be assessed, evidenced, excepted, and revised independently.

IMPLEMENTATION PROFILE

A risk-proportional set of required depth, evidence, and gates for a class of use.

CONTROLLED EXCEPTION

A time-bounded deviation with owner, rationale, compensating control, residual risk, expiration, and review trigger.

DETERMINISTIC MECHANISM

A component whose inputs, policy, state transition, and side effects can be constrained and verified.

SOURCE AUTHORITY

The documented basis for treating a source as reliable for a defined claim, context, jurisdiction, and time.

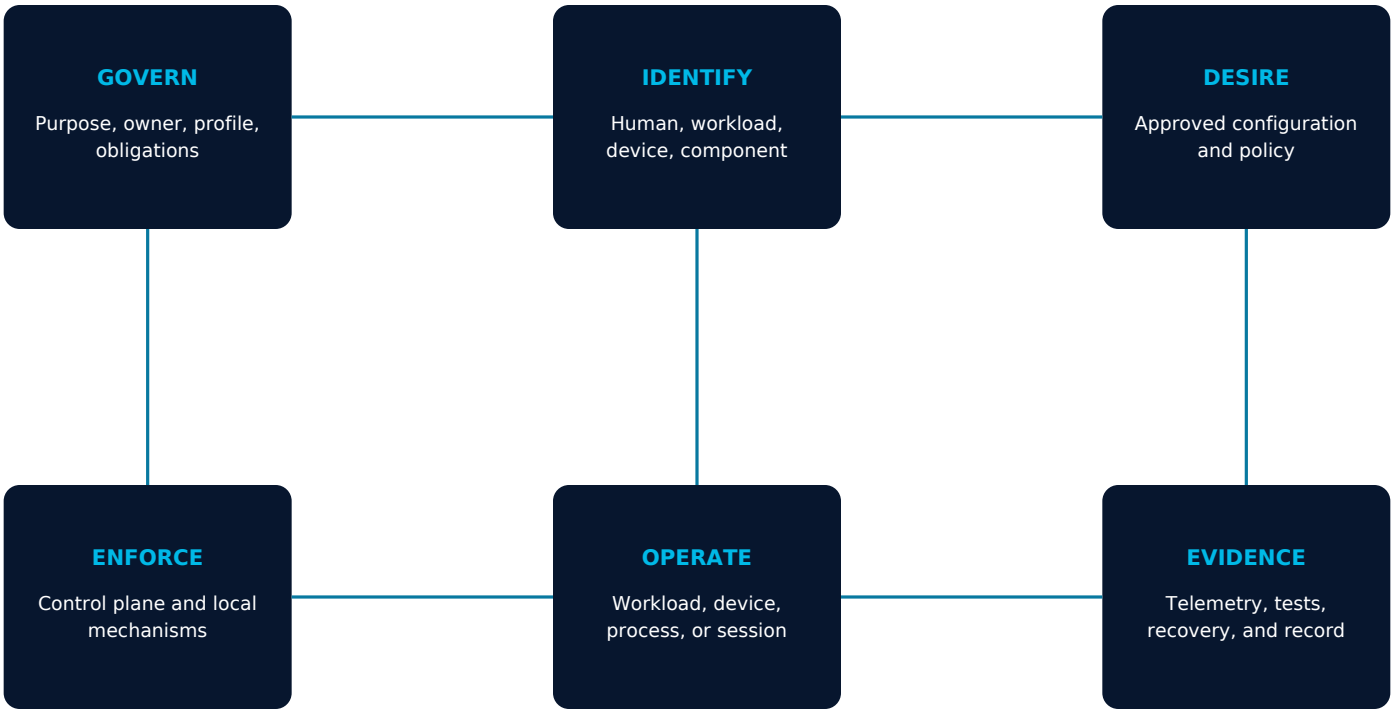
REASSESSMENT TRIGGER

A change, incident, drift signal, dependency revision, or time boundary requiring renewed evaluation.

MYTHOS-TEL-0002 / TELECOMMUNICATIONS

3. Architecture and Trust Model

Separation of governance, management, enforcement, runtime, telemetry, and recovery



Mandatory trust boundaries

- Management planes MUST be isolated, strongly authenticated, monitored, and recoverable.
- Identity and policy MUST be evaluated at every provider, network, device, workload, and administrative transition.
- Runtime state MUST be compared with approved desired state and material drift MUST trigger response.
- Safety and continuity mechanisms MUST remain available when cloud, WAN, identity, DNS, time, carrier, or management dependencies fail.
- Evidence MUST be protected from the systems and administrators whose behavior it is intended to assess.

MYTHOS-TEL-0002 / TELECOMMUNICATIONS

4. Governance and Accountability

Decision rights, separation of duties, and lifecycle ownership

ACCOUNTABLE OWNER

Accepts scope, risk tolerance, funding, and residual risk.

SYSTEM OWNER

Maintains architecture, inventory, operating boundaries, and change control.

SECURITY / PRIVACY

Defines threat, identity, data, isolation, monitoring, and incident requirements.

EVALUATION AUTHORITY

Designs representative tests, gates releases, and preserves reproducibility.

OPERATIONS

Maintains availability, rollback, recovery, evidence, and incident handling.

HUMAN OVERSIGHT

Reviews consequential decisions, exceptions, disputed outcomes, and harm signals.

DECISION PRINCIPLE

No single actor should be able to define the objective, grant authority, execute the consequential action, suppress evidence, and approve the result. Separation must be technical where consequence requires it.

MYTHOS-TEL-0002 / TELECOMMUNICATIONS

5. Normative Control System

Atomic criteria 01-04 of 18

MYTHOS-TEL-0002-C01 Coverage Purpose, Occupancy, and Authority Boundary

The organization MUST define, implement, verify, and maintain coverage purpose, occupancy, and authority boundary for in-scope distributed antenna systems and indoor RF capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-TEL-0002-C02 RF Survey, Prediction, and Design Assumptions

The organization MUST define, implement, verify, and maintain rf survey, prediction, and design assumptions for in-scope distributed antenna systems and indoor RF capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-TEL-0002-C03 Carrier, Band, Technology, and Capacity Plan

The organization MUST define, implement, verify, and maintain carrier, band, technology, and capacity plan for in-scope distributed antenna systems and indoor RF capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-TEL-0002-C04 Public-Safety Radio and AHJ Requirements

The organization MUST define, implement, verify, and maintain public-safety radio and ahj requirements for in-scope distributed antenna systems and indoor RF capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-TEL-0002 / TELECOMMUNICATIONS

5. Normative Control System

Atomic criteria 05-08 of 18

MYTHOS-TEL-0002-C05 Head-End, Source, Fiber, and Remote-Unit Architecture

The organization MUST define, implement, verify, and maintain head-end, source, fiber, and remote-unit architecture for in-scope distributed antenna systems and indoor RF capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-TEL-0002-C06 Neutral-Host and Multi-Operator Governance

The organization MUST define, implement, verify, and maintain neutral-host and multi-operator governance for in-scope distributed antenna systems and indoor RF capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-TEL-0002-C07 Link Budget, Noise, Interference, and PIM Control

The organization MUST define, implement, verify, and maintain link budget, noise, interference, and pim control for in-scope distributed antenna systems and indoor RF capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-TEL-0002-C08 Antenna Placement, Isolation, and Exposure Safety

The organization MUST define, implement, verify, and maintain antenna placement, isolation, and exposure safety for in-scope distributed antenna systems and indoor RF capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-TEL-0002 / TELECOMMUNICATIONS

5. Normative Control System

Atomic criteria 09-12 of 18

MYTHOS-TEL-0002-C09 Backhaul, Timing, Synchronization, and Monitoring

The organization MUST define, implement, verify, and maintain backhaul, timing, synchronization, and monitoring for in-scope distributed antenna systems and indoor RF capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-TEL-0002-C10 Power, Battery, Generator, and Survivability

The organization MUST define, implement, verify, and maintain power, battery, generator, and survivability for in-scope distributed antenna systems and indoor RF capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-TEL-0002-C11 Physical Security, Firestopping, and Pathway Integrity

The organization MUST define, implement, verify, and maintain physical security, firestopping, and pathway integrity for in-scope distributed antenna systems and indoor RF capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-TEL-0002-C12 Configuration, Firmware, and Change Control

The organization MUST define, implement, verify, and maintain configuration, firmware, and change control for in-scope distributed antenna systems and indoor RF capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-TEL-0002 / TELECOMMUNICATIONS

5. Normative Control System

Atomic criteria 13-15 of 18

MYTHOS-TEL-0002-C13 Acceptance Testing and Grid-Based Validation

The organization MUST define, implement, verify, and maintain acceptance testing and grid-based validation for in-scope distributed antenna systems and indoor RF capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-TEL-0002-C14 Voice, Data, Emergency, and Handoff Performance

The organization MUST define, implement, verify, and maintain voice, data, emergency, and handoff performance for in-scope distributed antenna systems and indoor RF capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-TEL-0002-C15 Fault Localization, Alarming, and Maintenance

The organization MUST define, implement, verify, and maintain fault localization, alarming, and maintenance for in-scope distributed antenna systems and indoor RF capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-TEL-0002 / TELECOMMUNICATIONS

5. Normative Control System

Atomic criteria 16-18 of 18

MYTHOS-TEL-0002-C16 Capacity Growth and Technology Refresh

The organization MUST define, implement, verify, and maintain capacity growth and technology refresh for in-scope distributed antenna systems and indoor RF capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-TEL-0002-C17 Incident, Outage, and Emergency Coordination

The organization MUST define, implement, verify, and maintain incident, outage, and emergency coordination for in-scope distributed antenna systems and indoor RF capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-TEL-0002-C18 As-Built Evidence and Periodic Recertification

The organization MUST define, implement, verify, and maintain as-built evidence and periodic recertification for in-scope distributed antenna systems and indoor RF capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-TEL-0002 / TELECOMMUNICATIONS

6. Evidence and Assessment

Examine, interview, test, observe, restore, and trace

EXAMINE

Policies, inventories, diagrams, configuration, code, firmware, contracts, provider evidence, logs, exceptions, and lifecycle records.

INTERVIEW

Accountable owners, architects, engineers, operators, safety personnel, security teams, vendors, carriers, integrators, and responders.

TEST

Identity, segmentation, policy, negative paths, malformed input, capacity stress, dependency loss, failover, rollback, and revocation.

RESTORE

Independent restoration of configuration, data, service, device, controller, cluster, media, or browser state from protected evidence.

OBSERVE

Production-like performance, quality, drift, resource use, physical behavior, alarms, user impact, and incident execution.

TRACE

End-to-end linkage from approved intent and identity through change, runtime behavior, telemetry, outcome, exception, and review.

Evidence grades

A

Independently reproducible, complete, current, integrity-protected, and representative.

B

Reproduced by the implementing organization with strong traceability and controlled scope.

C

Partially reproduced or indirect; limitations, inherited responsibility, and uncertainty recorded.

D

Assertion, diagram, design intent, certificate, or vendor statement without reproduced behavior.

MYTHOS-TEL-0002 / TELECOMMUNICATIONS

7. Assurance Views

Six perspectives required for a complete assurance claim

GOVERNANCE

Ownership, purpose, risk tolerance, policy, exception, and decision rights.

ARCHITECTURE

Boundaries, dependencies, failure modes, state, data flow, and portability.

SECURITY

Identity, authorization, isolation, secrets, abuse resistance, and incident response.

OPERATIONS

Reliability, performance, cost, observability, change, recovery, and support.

PRIVACY / RIGHTS

Purpose limitation, minimization, consent, access, correction, deletion, and egress.

HUMAN / SOCIETAL

Usability, accessibility, disclosure, contestability, impact, and human control.

Conformance requires a defensible position across every applicable view. A strong aggregate score cannot compensate for an unowned system, a failed mandatory gate, untested recovery, or evidence below the accepted grade.

MYTHOS-TEL-0002 / TELECOMMUNICATIONS

7.2 Evaluation Metrics and Decision Gates

Measures must expose service behavior, control effectiveness, failure, uncertainty, and cost

SERVICE

Availability, correctness, coverage, quality, latency, throughput, and user or process outcome.

CONTROL

Unauthorized attempt, policy denial, drift, segmentation escape, privilege, and exception exposure.

RESILIENCE

Failover success, recovery time, data loss, safe degradation, restore validity, and dependency survival.

SECURITY

Exploitability, credential exposure, supply-chain integrity, attack detection, containment, and revocation.

CAPACITY

Utilization, saturation, queueing, radio or fabric headroom, thermal margin, quota, and forecast error.

OPERATIONS

Change failure, mean time to detect, repair, support burden, vendor escalation, and evidence completeness.

PRIVACY / SAFETY

Unauthorized egress, retention breach, consent coverage, unsafe action, physical consequence, and contestability.

LIFECYCLE

Patch debt, end-of-support exposure, portability, replacement time, retirement completion, and reassessment age.

Decision gate: thresholds MUST be declared before assessment, cover representative load and failure conditions, and identify mandatory safety, identity, recovery, and evidence failures that block promotion.

MYTHOS-TEL-0002 / TELECOMMUNICATIONS

8. Failure Modes and Adversarial Scenarios

Challenge assumptions before the system is trusted with consequential work

SUBSCRIBER IDENTITY COMPROMISE

SIM, eSIM, credential, or provisioning weakness enables impersonation.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

SLICE ISOLATION FAILURE

A label implies isolation not enforced across radio, transport, core, edge, and cloud.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

SIGNALING ABUSE

Protocol manipulation causes fraud, denial, interception, or unauthorized routing.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

TIMING LOSS

Synchronization failure degrades radio, handoff, positioning, or service quality.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

RF INTERFERENCE

Coverage, PIM, noise, or external interference defeats expected service.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

EMERGENCY-SERVICE FAILURE

Calling, location, priority, public-safety, or survivability obligations are not met.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

Adversarial testing MUST protect people and production systems. Use isolated environments, synthetic or minimized data, bounded authority, kill switches, explicit legal and ethical review, and documented stop conditions.

MYTHOS-TEL-0002 / TELECOMMUNICATIONS

9. Implementation Profiles

Risk-proportional implementation without weakening mandatory boundaries

FOUNDATIONAL

TYPICAL SCOPE

Low consequence, limited autonomy, non-sensitive data, reversible outputs.

MINIMUM DEPTH

Named owner; inventory; basic evaluation; access control; logging; rollback; annual review.

ADVANCED

TYPICAL SCOPE

Business-critical workflow, sensitive data, multiple tools or providers, moderate autonomy.

MINIMUM DEPTH

Formal threat model; representative evaluation; approval gates; isolated execution; tested recovery; quarterly review.

HIGH ASSURANCE

TYPICAL SCOPE

Safety, security, regulated, financial, identity, or broadly consequential use.

MINIMUM DEPTH

Independent challenge; strongest identity; deterministic enforcement; comprehensive evidence; canary release; continuous monitoring.

Profile selection rule

Choose the highest profile triggered by consequence, autonomy, sensitivity, scale, irreversibility, external dependency, or inability to rapidly detect and recover. Tailoring may strengthen controls; it may not erase a mandatory gate without a controlled exception.

MYTHOS-TEL-0002 / TELECOMMUNICATIONS

9.2 Operational Lifecycle

Evidence-bearing operation from concept through retirement

1 **CONCEPT**
Purpose, owner, impact, boundaries

2 **DESIGN**
Architecture, threat, data, identity

3 **BUILD**
Controlled implementation and provenance

4 **EVALUATE**
Representative, adversarial, recovery tests

5 **RELEASE**
Approval, canary, rollback, evidence

6 **OPERATE**
Observe, support, detect, respond

7 **CHANGE**
Reassess, compare, gate, reverse

8 **RETIRE**
Revoke, export, dispose, preserve

LIFECYCLE INVARIANT

Every stage MUST leave sufficient evidence for the next authority to understand what was intended, what changed, what was tested, what failed, what was accepted, what remains uncertain, and how to recover or exit.

MYTHOS-TEL-0002 / TELECOMMUNICATIONS

10. Adoption Roadmap

A sequenced path from uncontrolled capability to evidence-bearing operation

0-30 DAYS

Establish ownership and truth

Inventory systems and dependencies; classify risk; freeze unsupported claims; identify immediate privilege, data, and evidence gaps.

31-90 DAYS

Create enforceable boundaries

Define policy; isolate execution; implement identity and approval gates; establish representative evaluation and baseline telemetry.

3-6 MONTHS

Prove recovery and operating control

Exercise failure, rollback, revocation, incident, provider exit, and state-recovery scenarios; mature evidence packaging.

6-12 MONTHS

Scale assurance

Automate control checks; expand workload coverage; independent challenge; portfolio metrics; controlled exception expiry; continuous reassessment.

Adoption is complete only when operating evidence demonstrates the selected profile in the actual deployment context. Documentation alone is not conformance.

MYTHOS-TEL-0002 / TELECOMMUNICATIONS

10.2 Conformance and Exception Statement

What an assurance claim must contain

SYSTEM / SERVICE

Named, versioned capability and deployment boundary.

PROFILE

Foundational, Advanced, or High Assurance with trigger rationale.

SCOPE

Workloads, users, data, tools, regions, providers, and exclusions.

CRITERIA

Pass, partial, fail, not applicable, and exception status for every criterion.

EVIDENCE

Artifact references, evidence grade, date, environment, and reproducibility.

LIMITATIONS

Known failure modes, unsupported workloads, uncertainty, and residual risk.

EXCEPTIONS

Owner, rationale, compensating control, expiration, and approval.

VALIDITY

Assessment date, change boundary, review date, and reassessment triggers.

Prohibited claim: “compliant” without the named scope, profile, evidence date, limitations, exceptions, and authority accepting residual risk.

MYTHOS-TEL-0002 / TELECOMMUNICATIONS

11. Source Authority and Reference Register

Primary sources informing interpretation; current obligations and provider behavior must be verified at assessment t

NIST CSF 2.0 — **Cybersecurity Framework 2.0**

https://doi.org/10.6028/NIST.CSWP.29

Enterprise governance and cybersecurity outcome framework.

NIST SP 800-53 Rev. 5 — **Security and Privacy Controls for Information Systems and Organizations**

https://doi.org/10.6028/NIST.SP.800-53r5

Broad control baseline for organizational systems and services.

3GPP Specifications — **3GPP technical specifications and reports**

https://www.3gpp.org/specifications-technologies/specifications-by-series

Mobile-system architecture, security, radio, core, and service specifications.

O-RAN Alliance — **O-RAN specifications and security work**

https://www.o-ran.org/specifications

Open RAN architecture, interfaces, management, and security.

GSMA NESAS — **Network Equipment Security Assurance Scheme**

https://www.gsma.com/solutions-and-impact/technologies/security/network-equipment-security-assurance-scheme/

Telecom equipment security development and assessment scheme.

IETF SIP — **Session Initiation Protocol - RFC 3261**

https://www.rfc-editor.org/rfc/rfc3261

Core SIP signaling model and protocol semantics.

Source currency rule: authorities, specifications, legal obligations, provider services, firmware, browser behavior, carrier requirements, and operational evidence MUST be checked at assessment time. This publication records a 2026-07-24 source snapshot.

MYTHOS-TEL-0002 / TELECOMMUNICATIONS

11.2 Revision History and Decision Register

Permanent identity, controlled change, and publication integrity

VERSION	DATE	STATE	SUMMARY
1.0.0-candidate	2026-07-24	Candidate	Permanent-publication edition; source refresh; expanded criteria, evidence, assurance, and lifecycle guidance.
Next review	2027-01-24	Scheduled	Review source currency, threat evolution, operating evidence, adoption feedback, and proposed revisions.

Publication decisions

PERMANENT IDENTITY The document ID remains stable across revisions; batch or production sequence metadata is not public identity.

CHANGE CONTROL Normative changes require rationale, impact analysis, affected-criterion mapping, and approval.

SUPERSESSION A superseded edition remains traceable; current routes and catalogs identify the active edition.

CORRECTION Material errors require a recorded correction, affected-evidence analysis, and notification appropriate to impact.

ARCHIVAL INTEGRITY Published artifacts receive hashes, metadata, and a release manifest sufficient for independent verification.

END OF PERMANENT PUBLICATION / MYTHOS-TEL-0002

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY

MYTHOS SYSTEMS

ENGINEERING STANDARDS LIBRARY / TELECOMMUNICATIONS AND RADIO SYSTEMS

Enterprise Telephony, VoIP, and Call-Control Standard

SIP and call-control architecture, identity, SBCs, emergency calling, fraud, recording, quality, recovery, and evidence.

PERMANENT DOCUMENT ID

MYTHOS-TEL-0003

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

Enterprise Telephony, VoIP, and Call-Control Standard

DOCUMENT ID
MYTHOS-TEL-0003

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

18	6	4	1
ATOMIC CRITERIA	ASSURANCE VIEWS	IMPLEMENTATION PROFILES	PERMANENT IDENTITY

Publication doctrine. This publication establishes durable requirements for its defined scope. Interpretation must preserve the permanent document identity, required evidence, controlled exceptions, source currency, and formal revision history.

INFRASTRUCTURE AND CONTROL TOPOLOGY

Telecommunications and Radio Systems

A trustworthy enterprise telephony and call control system is a governed chain of ownership, identity, desired state, enforcement, workload or device behavior, telemetry, recovery, and independently reviewable evidence.

OWNERSHIP

Purpose, service boundary, accountable owner, suppliers, users, and retained responsibility remain explicit.

ENFORCEMENT

Identity, policy, segmentation, configuration, and local safety mechanisms constrain every trust transition.

CONTINUITY

Capacity, dependency loss, safe degradation, backup, restoration, rollback, and exit are tested before reliance.

EVIDENCE

Inventory, desired state, runtime observation, tests, incidents, exceptions, and change records form the assurance chain.

GOVERNED INFRASTRUCTURE FLOW

Reconciliation, detection, response, restoration, and controlled evolution

INTERPRETATION AND ASSURANCE

How to apply this publication

Normative intent

Requirements define outcomes and constraints. Implementations may vary, but evidence must demonstrate equivalent control.

Evidence over assertion

Vendor documentation and design intent are not equivalent to reproduced behavior. Record evidence grade and uncertainty.

Risk-proportional depth

Higher consequence, autonomy, sensitivity, and irreversibility require stronger isolation, review, verification, and recovery.

Controlled exception

Exceptions require an owner, rationale, compensating control, residual-risk acceptance, expiration, and reevaluation trigger.

Normalized assessment scale

0	1	2	3	4	5
ABSENT	AD HOC	PARTIAL	OPERATIONAL	ADVANCED	LEADING

A numeric score must never hide a failed mandatory gate, missing evidence, untested workload, or unacceptable residual risk.

INTERACTIVE NAVIGATION

Contents

Navigation doctrine

Bookmarks and internal links are conveniences. Permanent document identity, criterion identifiers, evidence references, and revision history remain authoritative.

MYTHOS-TEL-0003 / TELECOMMUNICATIONS

0. Executive Mandate

Purpose, strategic outcome, and non-negotiable operating doctrine

MANDATE

Organizations adopting enterprise telephony and call control capabilities MUST define an owned service boundary, establish enforceable identity and configuration, protect data and safety, test failure and recovery, and preserve evidence sufficient for independent review.

Required outcomes

- Create a durable governance boundary for enterprise telephony and call control across design, acquisition, deployment, operation, change, and retirement.
- Require risk-proportional segmentation, identity, configuration, telemetry, resilience, recovery, and supplier controls.
- Prevent central automation, administrative tooling, or provider services from becoming unbounded authority.
- Prove operation with representative workloads, devices, failure modes, and restoration exercises.
- Define measurable conformance, exceptions, reassessment triggers, and a viable exit path.

Decision boundary: conformance supports a documented assurance claim for the named scope. It does not prove universal availability, safety, regulatory acceptance, vendor fitness, or immunity from cyber-physical failure.

MYTHOS-TEL-0003 / TELECOMMUNICATIONS

1. Scope and Applicability

Boundary definition, audience, exclusions, and triggering conditions

IN SCOPE

- Architecture, acquisition, configuration, integration, and operation of enterprise telephony and call control capabilities.
- Human, workload, device, service, network, data, facility, provider, supplier, and evidence dependencies.
- Design, deployment, monitoring, support, incident response, recovery, migration, and retirement.
- On-premises, hosted, managed, carrier, manufacturer, integrator, and externally operated components.

OUT OF SCOPE

- Claims of legal, safety, carrier, or regulatory approval without the responsible authority.
- Vendor certification treated as proof of the adopter configuration or operating effectiveness.
- Theoretical or laboratory behavior presented as production evidence without a declared boundary.
- Inherited controls without documented scope, owner, period, limitations, and shared responsibility.

Applicability triggers

- The capability supports a business-critical, safety-relevant, regulated, public, or externally dependent service.
- The environment can expose sensitive data, identity, control, physical process, communications, or shared infrastructure.
- A management plane, automation system, carrier, provider, or supplier can affect broad operational scope.
- Failure, compromise, or change can be difficult to detect, reverse, isolate, or recover.
- The system depends on hardware, firmware, radio, browser, cloud, endpoint, IoT, OT, or real-time media behavior.

MYTHOS-TEL-0003 / TELECOMMUNICATIONS

2. Definitions and Taxonomy

Controlled language for architecture, governance, and assessment

AUTHORITY

The right to approve, deny, constrain, or execute an action. Model output is not authority.

EVIDENCE

A reproducible source, trace, measurement, test, record, signed artifact, or documented observation supporting a claim.

ASSURANCE VIEW

A defined perspective such as governance, architecture, security, operations, privacy, or human oversight.

ATOMIC CRITERION

A uniquely identified requirement that can be assessed, evidenced, excepted, and revised independently.

IMPLEMENTATION PROFILE

A risk-proportional set of required depth, evidence, and gates for a class of use.

CONTROLLED EXCEPTION

A time-bounded deviation with owner, rationale, compensating control, residual risk, expiration, and review trigger.

DETERMINISTIC MECHANISM

A component whose inputs, policy, state transition, and side effects can be constrained and verified.

SOURCE AUTHORITY

The documented basis for treating a source as reliable for a defined claim, context, jurisdiction, and time.

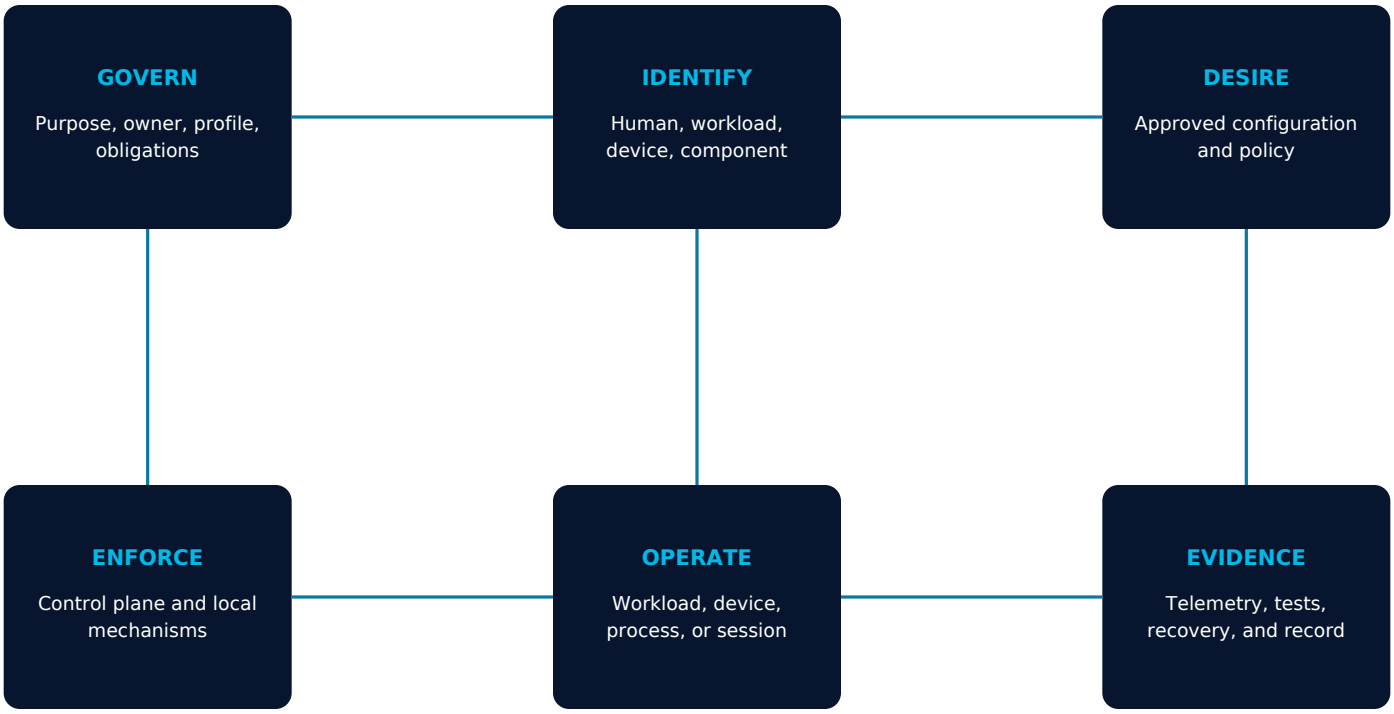
REASSESSMENT TRIGGER

A change, incident, drift signal, dependency revision, or time boundary requiring renewed evaluation.

MYTHOS-TEL-0003 / TELECOMMUNICATIONS

3. Architecture and Trust Model

Separation of governance, management, enforcement, runtime, telemetry, and recovery



Mandatory trust boundaries

- Management planes MUST be isolated, strongly authenticated, monitored, and recoverable.
- Identity and policy MUST be evaluated at every provider, network, device, workload, and administrative transition.
- Runtime state MUST be compared with approved desired state and material drift MUST trigger response.
- Safety and continuity mechanisms MUST remain available when cloud, WAN, identity, DNS, time, carrier, or management dependencies fail.
- Evidence MUST be protected from the systems and administrators whose behavior it is intended to assess.

MYTHOS-TEL-0003 / TELECOMMUNICATIONS

4. Governance and Accountability

Decision rights, separation of duties, and lifecycle ownership

ACCOUNTABLE OWNER

Accepts scope, risk tolerance, funding, and residual risk.

SYSTEM OWNER

Maintains architecture, inventory, operating boundaries, and change control.

SECURITY / PRIVACY

Defines threat, identity, data, isolation, monitoring, and incident requirements.

EVALUATION AUTHORITY

Designs representative tests, gates releases, and preserves reproducibility.

OPERATIONS

Maintains availability, rollback, recovery, evidence, and incident handling.

HUMAN OVERSIGHT

Reviews consequential decisions, exceptions, disputed outcomes, and harm signals.

DECISION PRINCIPLE

No single actor should be able to define the objective, grant authority, execute the consequential action, suppress evidence, and approve the result. Separation must be technical where consequence requires it.

MYTHOS-TEL-0003 / TELECOMMUNICATIONS

5. Normative Control System

Atomic criteria 01-04 of 18

MYTHOS-TEL-0003-C01 Telephony Service Boundary and Numbering Plan

The organization MUST define, implement, verify, and maintain telephony service boundary and numbering plan for in-scope enterprise telephony and call control capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-TEL-0003-C02 User, Device, Trunk, and Application Identity

The organization MUST define, implement, verify, and maintain user, device, trunk, and application identity for in-scope enterprise telephony and call control capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-TEL-0003-C03 Call-Control Cluster and Administrative Access

The organization MUST define, implement, verify, and maintain call-control cluster and administrative access for in-scope enterprise telephony and call control capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-TEL-0003-C04 SIP Trunk, SBC, and Carrier Interconnection

The organization MUST define, implement, verify, and maintain sip trunk, sbc, and carrier interconnection for in-scope enterprise telephony and call control capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-TEL-0003 / TELECOMMUNICATIONS

5. Normative Control System

Atomic criteria 05-08 of 18

MYTHOS-TEL-0003-C05 Signaling Authentication, Encryption, and Policy

The organization MUST define, implement, verify, and maintain signaling authentication, encryption, and policy for in-scope enterprise telephony and call control capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-TEL-0003-C06 Media Security, NAT Traversal, and Relay

The organization MUST define, implement, verify, and maintain media security, nat traversal, and relay for in-scope enterprise telephony and call control capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-TEL-0003-C07 Dial Plan, Route Pattern, and Toll Restriction

The organization MUST define, implement, verify, and maintain dial plan, route pattern, and toll restriction for in-scope enterprise telephony and call control capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-TEL-0003-C08 Emergency Calling, Location, and Notification

The organization MUST define, implement, verify, and maintain emergency calling, location, and notification for in-scope enterprise telephony and call control capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-TEL-0003 / TELECOMMUNICATIONS

5. Normative Control System

Atomic criteria 09-12 of 18

MYTHOS-TEL-0003-C09 Fraud, Robocall, and Abuse Prevention

The organization MUST define, implement, verify, and maintain fraud, robocall, and abuse prevention for in-scope enterprise telephony and call control capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-TEL-0003-C10 Voice Endpoint and Softphone Governance

The organization MUST define, implement, verify, and maintain voice endpoint and softphone governance for in-scope enterprise telephony and call control capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-TEL-0003-C11 Voicemail, Recording, Consent, and Retention

The organization MUST define, implement, verify, and maintain voicemail, recording, consent, and retention for in-scope enterprise telephony and call control capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-TEL-0003-C12 Contact Center, Bot, and Application Integration

The organization MUST define, implement, verify, and maintain contact center, bot, and application integration for in-scope enterprise telephony and call control capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-TEL-0003 / TELECOMMUNICATIONS

5. Normative Control System

Atomic criteria 13-15 of 18

MYTHOS-TEL-0003-C13 QoS, Jitter, Loss, Latency, and MOS Evidence

The organization MUST define, implement, verify, and maintain qos, jitter, loss, latency, and mos evidence for in-scope enterprise telephony and call control capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-TEL-0003-C14 Monitoring, CDR, Trace, and Troubleshooting

The organization MUST define, implement, verify, and maintain monitoring, cdr, trace, and troubleshooting for in-scope enterprise telephony and call control capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-TEL-0003-C15 High Availability, Survivability, and Carrier Failure

The organization MUST define, implement, verify, and maintain high availability, survivability, and carrier failure for in-scope enterprise telephony and call control capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-TEL-0003 / TELECOMMUNICATIONS

5. Normative Control System

Atomic criteria 16-18 of 18

MYTHOS-TEL-0003-C16 Backup, Restore, and Disaster-Recovery Exercise

The organization MUST define, implement, verify, and maintain backup, restore, and disaster-recovery exercise for in-scope enterprise telephony and call control capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-TEL-0003-C17 Change, Upgrade, Certificate, and Codec Lifecycle

The organization MUST define, implement, verify, and maintain change, upgrade, certificate, and codec lifecycle for in-scope enterprise telephony and call control capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-TEL-0003-C18 Telephony Evidence and Periodic Reassessment

The organization MUST define, implement, verify, and maintain telephony evidence and periodic reassessment for in-scope enterprise telephony and call control capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-TEL-0003 / TELECOMMUNICATIONS

6. Evidence and Assessment

Examine, interview, test, observe, restore, and trace

EXAMINE

Policies, inventories, diagrams, configuration, code, firmware, contracts, provider evidence, logs, exceptions, and lifecycle records.

INTERVIEW

Accountable owners, architects, engineers, operators, safety personnel, security teams, vendors, carriers, integrators, and responders.

TEST

Identity, segmentation, policy, negative paths, malformed input, capacity stress, dependency loss, failover, rollback, and revocation.

RESTORE

Independent restoration of configuration, data, service, device, controller, cluster, media, or browser state from protected evidence.

OBSERVE

Production-like performance, quality, drift, resource use, physical behavior, alarms, user impact, and incident execution.

TRACE

End-to-end linkage from approved intent and identity through change, runtime behavior, telemetry, outcome, exception, and review.

Evidence grades

A	B	C	D
Independently reproducible, complete, current, integrity-protected, and representative.	Reproduced by the implementing organization with strong traceability and controlled scope.	Partially reproduced or indirect; limitations, inherited responsibility, and uncertainty recorded.	Assertion, diagram, design intent, certificate, or vendor statement without reproduced behavior.

MYTHOS-TEL-0003 / TELECOMMUNICATIONS

7. Assurance Views

Six perspectives required for a complete assurance claim

GOVERNANCE

Ownership, purpose, risk tolerance, policy, exception, and decision rights.

ARCHITECTURE

Boundaries, dependencies, failure modes, state, data flow, and portability.

SECURITY

Identity, authorization, isolation, secrets, abuse resistance, and incident response.

OPERATIONS

Reliability, performance, cost, observability, change, recovery, and support.

PRIVACY / RIGHTS

Purpose limitation, minimization, consent, access, correction, deletion, and egress.

HUMAN / SOCIETAL

Usability, accessibility, disclosure, contestability, impact, and human control.

Conformance requires a defensible position across every applicable view. A strong aggregate score cannot compensate for an unowned system, a failed mandatory gate, untested recovery, or evidence below the accepted grade.

MYTHOS-TEL-0003 / TELECOMMUNICATIONS

7.2 Evaluation Metrics and Decision Gates

Measures must expose service behavior, control effectiveness, failure, uncertainty, and cost

SERVICE

Availability, correctness, coverage, quality, latency, throughput, and user or process outcome.

CONTROL

Unauthorized attempt, policy denial, drift, segmentation escape, privilege, and exception exposure.

RESILIENCE

Failover success, recovery time, data loss, safe degradation, restore validity, and dependency survival.

SECURITY

Exploitability, credential exposure, supply-chain integrity, attack detection, containment, and revocation.

CAPACITY

Utilization, saturation, queueing, radio or fabric headroom, thermal margin, quota, and forecast error.

OPERATIONS

Change failure, mean time to detect, repair, support burden, vendor escalation, and evidence completeness.

PRIVACY / SAFETY

Unauthorized egress, retention breach, consent coverage, unsafe action, physical consequence, and contestability.

LIFECYCLE

Patch debt, end-of-support exposure, portability, replacement time, retirement completion, and reassessment age.

Decision gate: thresholds MUST be declared before assessment, cover representative load and failure conditions, and identify mandatory safety, identity, recovery, and evidence failures that block promotion.

MYTHOS-TEL-0003 / TELECOMMUNICATIONS

8. Failure Modes and Adversarial Scenarios

Challenge assumptions before the system is trusted with consequential work

SUBSCRIBER IDENTITY COMPROMISE

SIM, eSIM, credential, or provisioning weakness enables impersonation.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

SLICE ISOLATION FAILURE

A label implies isolation not enforced across radio, transport, core, edge, and cloud.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

SIGNALING ABUSE

Protocol manipulation causes fraud, denial, interception, or unauthorized routing.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

TIMING LOSS

Synchronization failure degrades radio, handoff, positioning, or service quality.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

RF INTERFERENCE

Coverage, PIM, noise, or external interference defeats expected service.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

EMERGENCY-SERVICE FAILURE

Calling, location, priority, public-safety, or survivability obligations are not met.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

Adversarial testing MUST protect people and production systems. Use isolated environments, synthetic or minimized data, bounded authority, kill switches, explicit legal and ethical review, and documented stop conditions.

MYTHOS-TEL-0003 / TELECOMMUNICATIONS

9. Implementation Profiles

Risk-proportional implementation without weakening mandatory boundaries

FOUNDATIONAL

TYPICAL SCOPE

Low consequence, limited autonomy, non-sensitive data, reversible outputs.

MINIMUM DEPTH

Named owner; inventory; basic evaluation; access control; logging; rollback; annual review.

ADVANCED

TYPICAL SCOPE

Business-critical workflow, sensitive data, multiple tools or providers, moderate autonomy.

MINIMUM DEPTH

Formal threat model; representative evaluation; approval gates; isolated execution; tested recovery; quarterly review.

HIGH ASSURANCE

TYPICAL SCOPE

Safety, security, regulated, financial, identity, or broadly consequential use.

MINIMUM DEPTH

Independent challenge; strongest identity; deterministic enforcement; comprehensive evidence; canary release; continuous monitoring.

Profile selection rule

Choose the highest profile triggered by consequence, autonomy, sensitivity, scale, irreversibility, external dependency, or inability to rapidly detect and recover. Tailoring may strengthen controls; it may not erase a mandatory gate without a controlled exception.

MYTHOS-TEL-0003 / TELECOMMUNICATIONS

9.2 Operational Lifecycle

Evidence-bearing operation from concept through retirement

1 **CONCEPT**
Purpose, owner, impact, boundaries

2 **DESIGN**
Architecture, threat, data, identity

3 **BUILD**
Controlled implementation and provenance

4 **EVALUATE**
Representative, adversarial, recovery tests

5 **RELEASE**
Approval, canary, rollback, evidence

6 **OPERATE**
Observe, support, detect, respond

7 **CHANGE**
Reassess, compare, gate, reverse

8 **RETIRE**
Revoke, export, dispose, preserve

LIFECYCLE INVARIANT

Every stage MUST leave sufficient evidence for the next authority to understand what was intended, what changed, what was tested, what failed, what was accepted, what remains uncertain, and how to recover or exit.

MYTHOS-TEL-0003 / TELECOMMUNICATIONS

10. Adoption Roadmap

A sequenced path from uncontrolled capability to evidence-bearing operation

0-30 DAYS

Establish ownership and truth

Inventory systems and dependencies; classify risk; freeze unsupported claims; identify immediate privilege, data, and evidence gaps.

31-90 DAYS

Create enforceable boundaries

Define policy; isolate execution; implement identity and approval gates; establish representative evaluation and baseline telemetry.

3-6 MONTHS

Prove recovery and operating control

Exercise failure, rollback, revocation, incident, provider exit, and state-recovery scenarios; mature evidence packaging.

6-12 MONTHS

Scale assurance

Automate control checks; expand workload coverage; independent challenge; portfolio metrics; controlled exception expiry; continuous reassessment.

Adoption is complete only when operating evidence demonstrates the selected profile in the actual deployment context. Documentation alone is not conformance.

MYTHOS-TEL-0003 / TELECOMMUNICATIONS

10.2 Conformance and Exception Statement

What an assurance claim must contain

SYSTEM / SERVICE

Named, versioned capability and deployment boundary.

PROFILE

Foundational, Advanced, or High Assurance with trigger rationale.

SCOPE

Workloads, users, data, tools, regions, providers, and exclusions.

CRITERIA

Pass, partial, fail, not applicable, and exception status for every criterion.

EVIDENCE

Artifact references, evidence grade, date, environment, and reproducibility.

LIMITATIONS

Known failure modes, unsupported workloads, uncertainty, and residual risk.

EXCEPTIONS

Owner, rationale, compensating control, expiration, and approval.

VALIDITY

Assessment date, change boundary, review date, and reassessment triggers.

Prohibited claim: “compliant” without the named scope, profile, evidence date, limitations, exceptions, and authority accepting residual risk.

MYTHOS-TEL-0003 / TELECOMMUNICATIONS

11. Source Authority and Reference Register

Primary sources informing interpretation; current obligations and provider behavior must be verified at assessment t

NIST CSF 2.0 — **Cybersecurity Framework 2.0**

https://doi.org/10.6028/NIST.CSWP.29

Enterprise governance and cybersecurity outcome framework.

NIST SP 800-53 Rev. 5 — **Security and Privacy Controls for Information Systems and Organizations**

https://doi.org/10.6028/NIST.SP.800-53r5

Broad control baseline for organizational systems and services.

3GPP Specifications — **3GPP technical specifications and reports**

https://www.3gpp.org/specifications-technologies/specifications-by-series

Mobile-system architecture, security, radio, core, and service specifications.

O-RAN Alliance — **O-RAN specifications and security work**

https://www.o-ran.org/specifications

Open RAN architecture, interfaces, management, and security.

GSMA NESAS — **Network Equipment Security Assurance Scheme**

https://www.gsma.com/solutions-and-impact/technologies/security/network-equipment-security-assurance-scheme/

Telecom equipment security development and assessment scheme.

IETF SIP — **Session Initiation Protocol - RFC 3261**

https://www.rfc-editor.org/rfc/rfc3261

Core SIP signaling model and protocol semantics.

Source currency rule: authorities, specifications, legal obligations, provider services, firmware, browser behavior, carrier requirements, and operational evidence MUST be checked at assessment time. This publication records a 2026-07-24 source snapshot.

MYTHOS-TEL-0003 / TELECOMMUNICATIONS

11.2 Revision History and Decision Register

Permanent identity, controlled change, and publication integrity

VERSION	DATE	STATE	SUMMARY
1.0.0-candidate	2026-07-24	Candidate	Permanent-publication edition; source refresh; expanded criteria, evidence, assurance, and lifecycle guidance.
Next review	2027-01-24	Scheduled	Review source currency, threat evolution, operating evidence, adoption feedback, and proposed revisions.

Publication decisions

PERMANENT IDENTITY The document ID remains stable across revisions; batch or production sequence metadata is not public identity.

CHANGE CONTROL Normative changes require rationale, impact analysis, affected-criterion mapping, and approval.

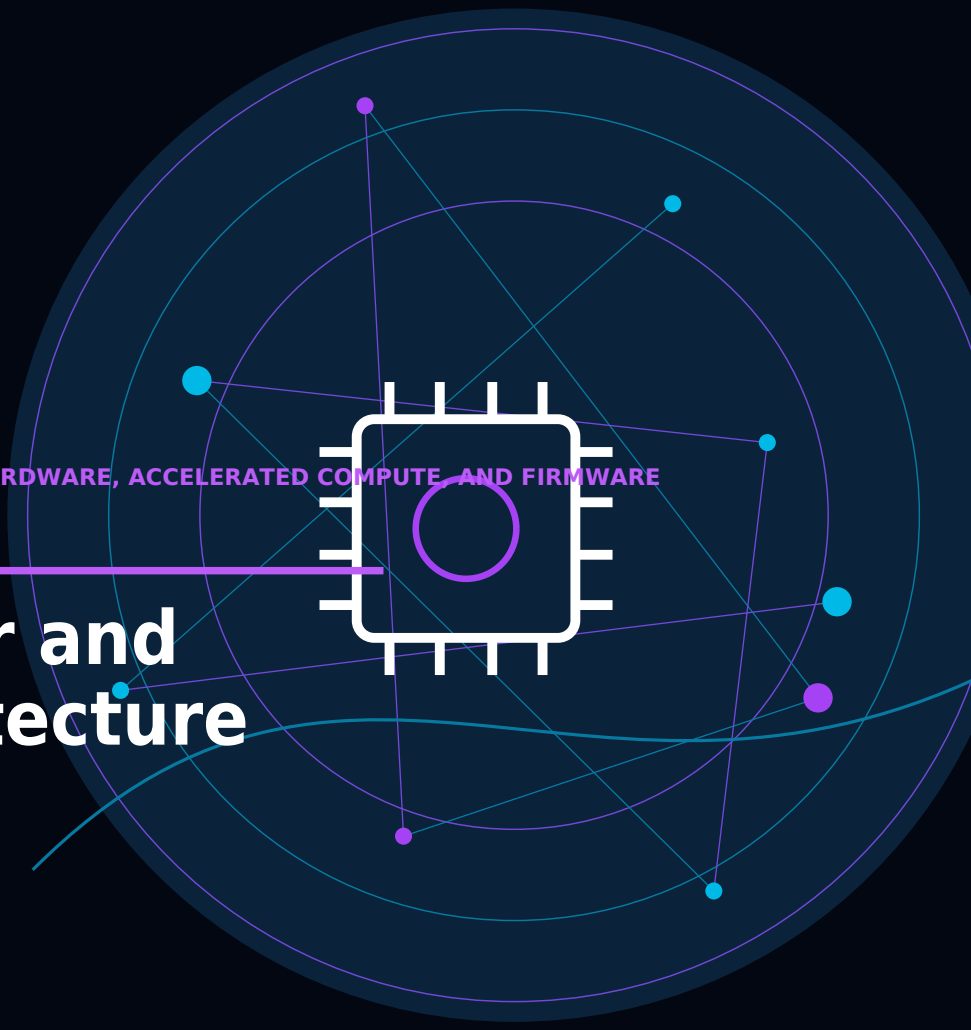
SUPERSESSION A superseded edition remains traceable; current routes and catalogs identify the active edition.

CORRECTION Material errors require a recorded correction, affected-evidence analysis, and notification appropriate to impact.

ARCHIVAL INTEGRITY Published artifacts receive hashes, metadata, and a release manifest sufficient for independent verification.

END OF PERMANENT PUBLICATION / MYTHOS-TEL-0003

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY



ENGINEERING STANDARDS LIBRARY / HARDWARE, ACCELERATED COMPUTE, AND FIRMWARE

AI GPU Server and Cluster Architecture Standard

Accelerated-compute topology, fabric, storage, scheduling, isolation, power, cooling, benchmarks, resilience, and lifecycle evidence.

PERMANENT DOCUMENT ID

MYTHOS-HW-0001

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

AI GPU Server and Cluster Architecture Standard

DOCUMENT ID
MYTHOS-HW-0001

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

18	6	4	1
ATOMIC CRITERIA	ASSURANCE VIEWS	IMPLEMENTATION PROFILES	PERMANENT IDENTITY

Publication doctrine. This publication establishes durable requirements for its defined scope. Interpretation must preserve the permanent document identity, required evidence, controlled exceptions, source currency, and formal revision history.

INFRASTRUCTURE AND CONTROL TOPOLOGY

Hardware, Accelerated Compute, and Firmware

A trustworthy AI GPU server and cluster architecture system is a governed chain of ownership, identity, desired state, enforcement, workload or device behavior, telemetry, recovery, and independently reviewable evidence.

OWNERSHIP

Purpose, service boundary, accountable owner, suppliers, users, and retained responsibility remain explicit.

ENFORCEMENT

Identity, policy, segmentation, configuration, and local safety mechanisms constrain every trust transition.

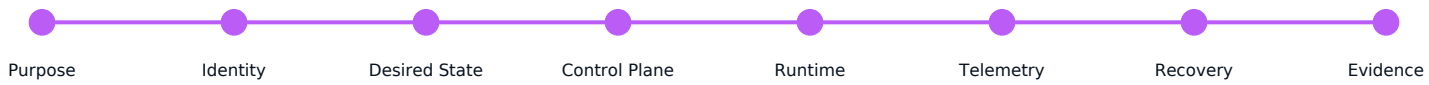
CONTINUITY

Capacity, dependency loss, safe degradation, backup, restoration, rollback, and exit are tested before reliance.

EVIDENCE

Inventory, desired state, runtime observation, tests, incidents, exceptions, and change records form the assurance chain.

GOVERNED INFRASTRUCTURE FLOW



Reconciliation, detection, response, restoration, and controlled evolution

INTERPRETATION AND ASSURANCE

How to apply this publication

Normative intent

Requirements define outcomes and constraints. Implementations may vary, but evidence must demonstrate equivalent control.

Evidence over assertion

Vendor documentation and design intent are not equivalent to reproduced behavior. Record evidence grade and uncertainty.

Risk-proportional depth

Higher consequence, autonomy, sensitivity, and irreversibility require stronger isolation, review, verification, and recovery.

Controlled exception

Exceptions require an owner, rationale, compensating control, residual-risk acceptance, expiration, and reevaluation trigger.

Normalized assessment scale

0	1	2	3	4	5
ABSENT	AD HOC	PARTIAL	OPERATIONAL	ADVANCED	LEADING

A numeric score must never hide a failed mandatory gate, missing evidence, untested workload, or unacceptable residual risk.

INTERACTIVE NAVIGATION

Contents

Navigation doctrine

Bookmarks and internal links are conveniences. Permanent document identity, criterion identifiers, evidence references, and revision history remain authoritative.

MYTHOS-HW-0001 / HARDWARE AND COMPUTE

0. Executive Mandate

Purpose, strategic outcome, and non-negotiable operating doctrine

MANDATE

Organizations adopting AI GPU server and cluster architecture capabilities MUST define an owned service boundary, establish enforceable identity and configuration, protect data and safety, test failure and recovery, and preserve evidence sufficient for independent review.

Required outcomes

- Create a durable governance boundary for AI GPU server and cluster architecture across design, acquisition, deployment, operation, change, and retirement.
- Require risk-proportional segmentation, identity, configuration, telemetry, resilience, recovery, and supplier controls.
- Prevent central automation, administrative tooling, or provider services from becoming unbounded authority.
- Prove operation with representative workloads, devices, failure modes, and restoration exercises.
- Define measurable conformance, exceptions, reassessment triggers, and a viable exit path.

Decision boundary: conformance supports a documented assurance claim for the named scope. It does not prove universal availability, safety, regulatory acceptance, vendor fitness, or immunity from cyber-physical failure.

MYTHOS-HW-0001 / HARDWARE AND COMPUTE

1. Scope and Applicability

Boundary definition, audience, exclusions, and triggering conditions

IN SCOPE

- Architecture, acquisition, configuration, integration, and operation of AI GPU server and cluster architecture capabilities.
- Human, workload, device, service, network, data, facility, provider, supplier, and evidence dependencies.
- Design, deployment, monitoring, support, incident response, recovery, migration, and retirement.
- On-premises, hosted, managed, carrier, manufacturer, integrator, and externally operated components.

OUT OF SCOPE

- Claims of legal, safety, carrier, or regulatory approval without the responsible authority.
- Vendor certification treated as proof of the adopter configuration or operating effectiveness.
- Theoretical or laboratory behavior presented as production evidence without a declared boundary.
- Inherited controls without documented scope, owner, period, limitations, and shared responsibility.

Applicability triggers

- The capability supports a business-critical, safety-relevant, regulated, public, or externally dependent service.
- The environment can expose sensitive data, identity, control, physical process, communications, or shared infrastructure.
- A management plane, automation system, carrier, provider, or supplier can affect broad operational scope.
- Failure, compromise, or change can be difficult to detect, reverse, isolate, or recover.
- The system depends on hardware, firmware, radio, browser, cloud, endpoint, IoT, OT, or real-time media behavior.

MYTHOS-HW-0001 / HARDWARE AND COMPUTE

2. Definitions and Taxonomy

Controlled language for architecture, governance, and assessment

AUTHORITY

The right to approve, deny, constrain, or execute an action. Model output is not authority.

EVIDENCE

A reproducible source, trace, measurement, test, record, signed artifact, or documented observation supporting a claim.

ASSURANCE VIEW

A defined perspective such as governance, architecture, security, operations, privacy, or human oversight.

ATOMIC CRITERION

A uniquely identified requirement that can be assessed, evidenced, excepted, and revised independently.

IMPLEMENTATION PROFILE

A risk-proportional set of required depth, evidence, and gates for a class of use.

CONTROLLED EXCEPTION

A time-bounded deviation with owner, rationale, compensating control, residual risk, expiration, and review trigger.

DETERMINISTIC MECHANISM

A component whose inputs, policy, state transition, and side effects can be constrained and verified.

SOURCE AUTHORITY

The documented basis for treating a source as reliable for a defined claim, context, jurisdiction, and time.

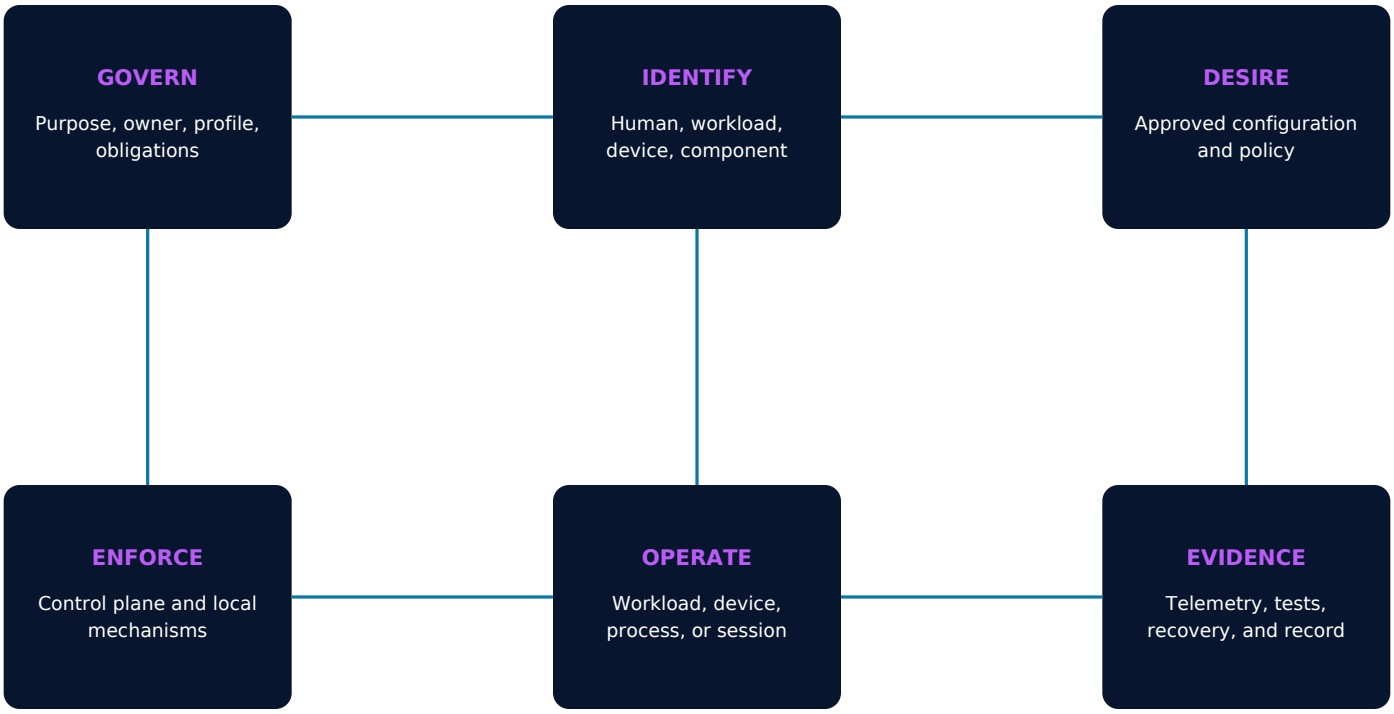
REASSESSMENT TRIGGER

A change, incident, drift signal, dependency revision, or time boundary requiring renewed evaluation.

MYTHOS-HW-0001 / HARDWARE AND COMPUTE

3. Architecture and Trust Model

Separation of governance, management, enforcement, runtime, telemetry, and recovery



Mandatory trust boundaries

- Management planes MUST be isolated, strongly authenticated, monitored, and recoverable.
- Identity and policy MUST be evaluated at every provider, network, device, workload, and administrative transition.
- Runtime state MUST be compared with approved desired state and material drift MUST trigger response.
- Safety and continuity mechanisms MUST remain available when cloud, WAN, identity, DNS, time, carrier, or management dependencies fail.
- Evidence MUST be protected from the systems and administrators whose behavior it is intended to assess.

MYTHOS-HW-0001 / HARDWARE AND COMPUTE

4. Governance and Accountability

Decision rights, separation of duties, and lifecycle ownership

ACCOUNTABLE OWNER

Accepts scope, risk tolerance, funding, and residual risk.

SYSTEM OWNER

Maintains architecture, inventory, operating boundaries, and change control.

SECURITY / PRIVACY

Defines threat, identity, data, isolation, monitoring, and incident requirements.

EVALUATION AUTHORITY

Designs representative tests, gates releases, and preserves reproducibility.

OPERATIONS

Maintains availability, rollback, recovery, evidence, and incident handling.

HUMAN OVERSIGHT

Reviews consequential decisions, exceptions, disputed outcomes, and harm signals.

DECISION PRINCIPLE

No single actor should be able to define the objective, grant authority, execute the consequential action, suppress evidence, and approve the result. Separation must be technical where consequence requires it.

MYTHOS-HW-0001 / HARDWARE AND COMPUTE

5. Normative Control System

Atomic criteria 01-04 of 18

MYTHOS-HW-0001-C01 Workload, Model, and Service-Level Requirements

The organization MUST define, implement, verify, and maintain workload, model, and service-level requirements for in-scope AI GPU server and cluster architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-HW-0001-C02 Accelerator, CPU, Memory, and Platform Selection

The organization MUST define, implement, verify, and maintain accelerator, cpu, memory, and platform selection for in-scope AI GPU server and cluster architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-HW-0001-C03 Node Topology, NUMA, PCIe, and CXL Design

The organization MUST define, implement, verify, and maintain node topology, numa, pcie, and cxl design for in-scope AI GPU server and cluster architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-HW-0001-C04 Scale-Up and Scale-Out Fabric Architecture

The organization MUST define, implement, verify, and maintain scale-up and scale-out fabric architecture for in-scope AI GPU server and cluster architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-HW-0001 / HARDWARE AND COMPUTE

5. Normative Control System

Atomic criteria 05-08 of 18

MYTHOS-HW-0001-C05 RDMA, Congestion, and Lossless-Network Control

The organization MUST define, implement, verify, and maintain rdma, congestion, and lossless-network control for in-scope AI GPU server and cluster architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-HW-0001-C06 Storage, Dataset, Checkpoint, and Cache Architecture

The organization MUST define, implement, verify, and maintain storage, dataset, checkpoint, and cache architecture for in-scope AI GPU server and cluster architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-HW-0001-C07 Scheduler, Queue, Quota, and Admission Policy

The organization MUST define, implement, verify, and maintain scheduler, queue, quota, and admission policy for in-scope AI GPU server and cluster architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-HW-0001-C08 Tenant, Job, Model, and Data Isolation

The organization MUST define, implement, verify, and maintain tenant, job, model, and data isolation for in-scope AI GPU server and cluster architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-HW-0001 / HARDWARE AND COMPUTE

5. Normative Control System

Atomic criteria 09-12 of 18

MYTHOS-HW-0001-C09 Driver, Runtime, Library, and Compatibility Matrix

The organization MUST define, implement, verify, and maintain driver, runtime, library, and compatibility matrix for in-scope AI GPU server and cluster architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-HW-0001-C10 Image, Container, and Artifact Provenance

The organization MUST define, implement, verify, and maintain image, container, and artifact provenance for in-scope AI GPU server and cluster architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-HW-0001-C11 Power, Cooling, Rack, and Facility Capacity

The organization MUST define, implement, verify, and maintain power, cooling, rack, and facility capacity for in-scope AI GPU server and cluster architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-HW-0001-C12 Thermal, ECC, Link, and Hardware Health Monitoring

The organization MUST define, implement, verify, and maintain thermal, ecc, link, and hardware health monitoring for in-scope AI GPU server and cluster architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-HW-0001 / HARDWARE AND COMPUTE

5. Normative Control System

Atomic criteria 13-15 of 18

MYTHOS-HW-0001-C13 Representative Benchmark and Scaling Efficiency

The organization MUST define, implement, verify, and maintain representative benchmark and scaling efficiency for in-scope AI GPU server and cluster architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-HW-0001-C14 Failure Injection, Job Recovery, and Checkpoint Restore

The organization MUST define, implement, verify, and maintain failure injection, job recovery, and checkpoint restore for in-scope AI GPU server and cluster architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-HW-0001-C15 Capacity Planning, Energy, and Cost per Useful Result

The organization MUST define, implement, verify, and maintain capacity planning, energy, and cost per useful result for in-scope AI GPU server and cluster architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-HW-0001 / HARDWARE AND COMPUTE

5. Normative Control System

Atomic criteria 16-18 of 18

MYTHOS-HW-0001-C16 Administrative Access and Out-of-Band Control

The organization MUST define, implement, verify, and maintain administrative access and out-of-band control for in-scope AI GPU server and cluster architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-HW-0001-C17 Spares, Firmware, Repair, and End-of-Life

The organization MUST define, implement, verify, and maintain spares, firmware, repair, and end-of-life for in-scope AI GPU server and cluster architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-HW-0001-C18 Cluster Evidence and Periodic Requalification

The organization MUST define, implement, verify, and maintain cluster evidence and periodic requalification for in-scope AI GPU server and cluster architecture capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-HW-0001 / HARDWARE AND COMPUTE

6. Evidence and Assessment

Examine, interview, test, observe, restore, and trace

EXAMINE

Policies, inventories, diagrams, configuration, code, firmware, contracts, provider evidence, logs, exceptions, and lifecycle records.

INTERVIEW

Accountable owners, architects, engineers, operators, safety personnel, security teams, vendors, carriers, integrators, and responders.

TEST

Identity, segmentation, policy, negative paths, malformed input, capacity stress, dependency loss, failover, rollback, and revocation.

RESTORE

Independent restoration of configuration, data, service, device, controller, cluster, media, or browser state from protected evidence.

OBSERVE

Production-like performance, quality, drift, resource use, physical behavior, alarms, user impact, and incident execution.

TRACE

End-to-end linkage from approved intent and identity through change, runtime behavior, telemetry, outcome, exception, and review.

Evidence grades

A

Independently reproducible, complete, current, integrity-protected, and representative.

B

Reproduced by the implementing organization with strong traceability and controlled scope.

C

Partially reproduced or indirect; limitations, inherited responsibility, and uncertainty recorded.

D

Assertion, diagram, design intent, certificate, or vendor statement without reproduced behavior.

MYTHOS-HW-0001 / HARDWARE AND COMPUTE

7. Assurance Views

Six perspectives required for a complete assurance claim

GOVERNANCE

Ownership, purpose, risk tolerance, policy, exception, and decision rights.

ARCHITECTURE

Boundaries, dependencies, failure modes, state, data flow, and portability.

SECURITY

Identity, authorization, isolation, secrets, abuse resistance, and incident response.

OPERATIONS

Reliability, performance, cost, observability, change, recovery, and support.

PRIVACY / RIGHTS

Purpose limitation, minimization, consent, access, correction, deletion, and egress.

HUMAN / SOCIETAL

Usability, accessibility, disclosure, contestability, impact, and human control.

Conformance requires a defensible position across every applicable view. A strong aggregate score cannot compensate for an unowned system, a failed mandatory gate, untested recovery, or evidence below the accepted grade.

MYTHOS-HW-0001 / HARDWARE AND COMPUTE

7.2 Evaluation Metrics and Decision Gates

Measures must expose service behavior, control effectiveness, failure, uncertainty, and cost

SERVICE

Availability, correctness, coverage, quality, latency, throughput, and user or process outcome.

CONTROL

Unauthorized attempt, policy denial, drift, segmentation escape, privilege, and exception exposure.

RESILIENCE

Failover success, recovery time, data loss, safe degradation, restore validity, and dependency survival.

SECURITY

Exploitability, credential exposure, supply-chain integrity, attack detection, containment, and revocation.

CAPACITY

Utilization, saturation, queueing, radio or fabric headroom, thermal margin, quota, and forecast error.

OPERATIONS

Change failure, mean time to detect, repair, support burden, vendor escalation, and evidence completeness.

PRIVACY / SAFETY

Unauthorized egress, retention breach, consent coverage, unsafe action, physical consequence, and contestability.

LIFECYCLE

Patch debt, end-of-support exposure, portability, replacement time, retirement completion, and reassessment age.

Decision gate: thresholds MUST be declared before assessment, cover representative load and failure conditions, and identify mandatory safety, identity, recovery, and evidence failures that block promotion.

MYTHOS-HW-0001 / HARDWARE AND COMPUTE

8. Failure Modes and Adversarial Scenarios

Challenge assumptions before the system is trusted with consequential work

BMC COMPROMISE

Out-of-band management provides persistent control beneath the operating system.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

FIRMWARE SUPPLY-CHAIN ATTACK

Signed or trusted firmware is malicious, vulnerable, or incorrectly sourced.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

DMA BOUNDARY BYPASS

A device accesses memory outside its assigned trust domain.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

ATTESTATION REPLAY

Stale or forged evidence causes keys or workloads to be released to an untrusted platform.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

POWER OR THERMAL COLLAPSE

Facility or rack constraints create correlated hardware failure.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

CROSS-TENANT LEAKAGE

Scheduler, cache, fabric, storage, or diagnostics expose another workload.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

Adversarial testing MUST protect people and production systems. Use isolated environments, synthetic or minimized data, bounded authority, kill switches, explicit legal and ethical review, and documented stop conditions.

MYTHOS-HW-0001 / HARDWARE AND COMPUTE

9. Implementation Profiles

Risk-proportional implementation without weakening mandatory boundaries

FOUNDATIONAL

TYPICAL SCOPE

Low consequence, limited autonomy, non-sensitive data, reversible outputs.

MINIMUM DEPTH

Named owner; inventory; basic evaluation; access control; logging; rollback; annual review.

ADVANCED

TYPICAL SCOPE

Business-critical workflow, sensitive data, multiple tools or providers, moderate autonomy.

MINIMUM DEPTH

Formal threat model; representative evaluation; approval gates; isolated execution; tested recovery; quarterly review.

HIGH ASSURANCE

TYPICAL SCOPE

Safety, security, regulated, financial, identity, or broadly consequential use.

MINIMUM DEPTH

Independent challenge; strongest identity; deterministic enforcement; comprehensive evidence; canary release; continuous monitoring.

Profile selection rule

Choose the highest profile triggered by consequence, autonomy, sensitivity, scale, irreversibility, external dependency, or inability to rapidly detect and recover. Tailoring may strengthen controls; it may not erase a mandatory gate without a controlled exception.

MYTHOS-HW-0001 / HARDWARE AND COMPUTE

9.2 Operational Lifecycle

Evidence-bearing operation from concept through retirement

1. **CONCEPT**
   Purpose, owner, impact, boundaries
2. **DESIGN**
   Architecture, threat, data, identity
3. **BUILD**
   Controlled implementation and provenance
4. **EVALUATE**
   Representative, adversarial, recovery tests
5. **RELEASE**
   Approval, canary, rollback, evidence
6. **OPERATE**
   Observe, support, detect, respond
7. **CHANGE**
   Reassess, compare, gate, reverse
8. **RETIRE**
   Revoke, export, dispose, preserve

LIFECYCLE INVARIANT

Every stage MUST leave sufficient evidence for the next authority to understand what was intended, what changed, what was tested, what failed, what was accepted, what remains uncertain, and how to recover or exit.

MYTHOS-HW-0001 / HARDWARE AND COMPUTE

10. Adoption Roadmap

A sequenced path from uncontrolled capability to evidence-bearing operation

0-30 DAYS

Establish ownership and truth

Inventory systems and dependencies; classify risk; freeze unsupported claims; identify immediate privilege, data, and evidence gaps.

31-90 DAYS

Create enforceable boundaries

Define policy; isolate execution; implement identity and approval gates; establish representative evaluation and baseline telemetry.

3-6 MONTHS

Prove recovery and operating control

Exercise failure, rollback, revocation, incident, provider exit, and state-recovery scenarios; mature evidence packaging.

6-12 MONTHS

Scale assurance

Automate control checks; expand workload coverage; independent challenge; portfolio metrics; controlled exception expiry; continuous reassessment.

Adoption is complete only when operating evidence demonstrates the selected profile in the actual deployment context. Documentation alone is not conformance.

MYTHOS-HW-0001 / HARDWARE AND COMPUTE

10.2 Conformance and Exception Statement

What an assurance claim must contain

SYSTEM / SERVICE

Named, versioned capability and deployment boundary.

PROFILE

Foundational, Advanced, or High Assurance with trigger rationale.

SCOPE

Workloads, users, data, tools, regions, providers, and exclusions.

CRITERIA

Pass, partial, fail, not applicable, and exception status for every criterion.

EVIDENCE

Artifact references, evidence grade, date, environment, and reproducibility.

LIMITATIONS

Known failure modes, unsupported workloads, uncertainty, and residual risk.

EXCEPTIONS

Owner, rationale, compensating control, expiration, and approval.

VALIDITY

Assessment date, change boundary, review date, and reassessment triggers.

Prohibited claim: “compliant” without the named scope, profile, evidence date, limitations, exceptions, and authority accepting residual risk.

MYTHOS-HW-0001 / HARDWARE AND COMPUTE

11. Source Authority and Reference Register

Primary sources informing interpretation; current obligations and provider behavior must be verified at assessment t

NIST CSF 2.0 — **Cybersecurity Framework 2.0**

https://doi.org/10.6028/NIST.CSWP.29

Enterprise governance and cybersecurity outcome framework.

NIST SP 800-53 Rev. 5 — **Security and Privacy Controls for Information Systems and Organizations**

https://doi.org/10.6028/NIST.SP.800-53r5

Broad control baseline for organizational systems and services.

NIST SP 800-193 — **Platform Firmware Resiliency Guidelines**

https://doi.org/10.6028/NIST.SP.800-193

Platform firmware protection, detection, and recovery.

TCG TPM 2.0 — **Trusted Platform Module Library Specification**

https://trustedcomputinggroup.org/resource/tpm-library-specification/

TPM commands, objects, measurement, keys, and attestation foundations.

DMTF Redfish — **Redfish Standard**

https://www.dmtf.org/standards/redfish

Secure and interoperable hardware-management interfaces.

NIST SP 800-155 — **BIOS Integrity Measurement Guidelines**

https://doi.org/10.6028/NIST.SP.800-155

Firmware measurement and integrity-verification principles.

Source currency rule: authorities, specifications, legal obligations, provider services, firmware, browser behavior, carrier requirements, and operational evidence MUST be checked at assessment time. This publication records a 2026-07-24 source snapshot.

MYTHOS-HW-0001 / HARDWARE AND COMPUTE

11.2 Revision History and Decision Register

Permanent identity, controlled change, and publication integrity

VERSION	DATE	STATE	SUMMARY
1.0.0-candidate	2026-07-24	Candidate	Permanent-publication edition; source refresh; expanded criteria, evidence, assurance, and lifecycle guidance.
Next review	2027-01-24	Scheduled	Review source currency, threat evolution, operating evidence, adoption feedback, and proposed revisions.

Publication decisions

PERMANENT IDENTITY The document ID remains stable across revisions; batch or production sequence metadata is not public identity.

CHANGE CONTROL Normative changes require rationale, impact analysis, affected-criterion mapping, and approval.

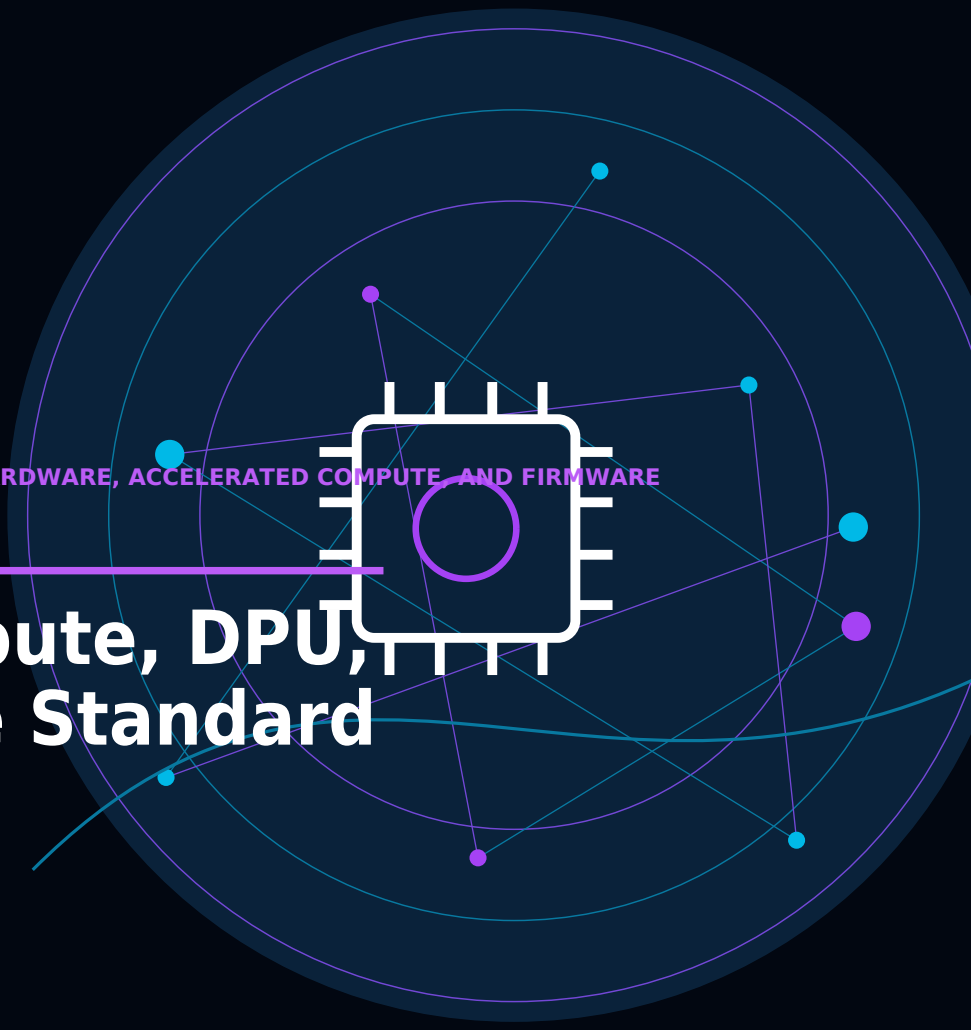
SUPERSESSION A superseded edition remains traceable; current routes and catalogs identify the active edition.

CORRECTION Material errors require a recorded correction, affected-evidence analysis, and notification appropriate to impact.

ARCHIVAL INTEGRITY Published artifacts receive hashes, metadata, and a release manifest sufficient for independent verification.

END OF PERMANENT PUBLICATION / MYTHOS-HW-0001

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY



ENGINEERING STANDARDS LIBRARY / HARDWARE, ACCELERATED COMPUTE, AND FIRMWARE

Trusted Compute, DPU, and Firmware Standard

Roots of trust, secure and measured boot, attestation, BMC, DPU, DMA isolation, confidential computing, firmware, and lifecycle assurance.

PERMANENT DOCUMENT ID

MYTHOS-HW-0002

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

Trusted Compute, DPU, and Firmware Standard

DOCUMENT ID
MYTHOS-HW-0002

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

18	6	4	1
ATOMIC CRITERIA	ASSURANCE VIEWS	IMPLEMENTATION PROFILES	PERMANENT IDENTITY

Publication doctrine. This publication establishes durable requirements for its defined scope. Interpretation must preserve the permanent document identity, required evidence, controlled exceptions, source currency, and formal revision history.

INFRASTRUCTURE AND CONTROL TOPOLOGY

Hardware, Accelerated Compute, and Firmware

A trustworthy trusted compute, DPU, and firmware system is a governed chain of ownership, identity, desired state, enforcement, workload or device behavior, telemetry, recovery, and independently reviewable evidence.

OWNERSHIP

Purpose, service boundary, accountable owner, suppliers, users, and retained responsibility remain explicit.

ENFORCEMENT

Identity, policy, segmentation, configuration, and local safety mechanisms constrain every trust transition.

CONTINUITY

Capacity, dependency loss, safe degradation, backup, restoration, rollback, and exit are tested before reliance.

EVIDENCE

Inventory, desired state, runtime observation, tests, incidents, exceptions, and change records form the assurance chain.

GOVERNED INFRASTRUCTURE FLOW

Reconciliation, detection, response, restoration, and controlled evolution

INTERPRETATION AND ASSURANCE

How to apply this publication

Normative intent

Requirements define outcomes and constraints. Implementations may vary, but evidence must demonstrate equivalent control.

Evidence over assertion

Vendor documentation and design intent are not equivalent to reproduced behavior. Record evidence grade and uncertainty.

Risk-proportional depth

Higher consequence, autonomy, sensitivity, and irreversibility require stronger isolation, review, verification, and recovery.

Controlled exception

Exceptions require an owner, rationale, compensating control, residual-risk acceptance, expiration, and reevaluation trigger.

Normalized assessment scale

0	1	2	3	4	5
ABSENT	AD HOC	PARTIAL	OPERATIONAL	ADVANCED	LEADING

A numeric score must never hide a failed mandatory gate, missing evidence, untested workload, or unacceptable residual risk.

INTERACTIVE NAVIGATION

Contents

Navigation doctrine

Bookmarks and internal links are conveniences. Permanent document identity, criterion identifiers, evidence references, and revision history remain authoritative.

MYTHOS-HW-0002 / HARDWARE AND COMPUTE

0. Executive Mandate

Purpose, strategic outcome, and non-negotiable operating doctrine

MANDATE

Organizations adopting trusted compute, DPU, and firmware capabilities MUST define an owned service boundary, establish enforceable identity and configuration, protect data and safety, test failure and recovery, and preserve evidence sufficient for independent review.

Required outcomes

- Create a durable governance boundary for trusted compute, DPU, and firmware across design, acquisition, deployment, operation, change, and retirement.
- Require risk-proportional segmentation, identity, configuration, telemetry, resilience, recovery, and supplier controls.
- Prevent central automation, administrative tooling, or provider services from becoming unbounded authority.
- Prove operation with representative workloads, devices, failure modes, and restoration exercises.
- Define measurable conformance, exceptions, reassessment triggers, and a viable exit path.

Decision boundary: conformance supports a documented assurance claim for the named scope. It does not prove universal availability, safety, regulatory acceptance, vendor fitness, or immunity from cyber-physical failure.

MYTHOS-HW-0002 / HARDWARE AND COMPUTE

1. Scope and Applicability

Boundary definition, audience, exclusions, and triggering conditions

IN SCOPE

- Architecture, acquisition, configuration, integration, and operation of trusted compute, DPU, and firmware capabilities.
- Human, workload, device, service, network, data, facility, provider, supplier, and evidence dependencies.
- Design, deployment, monitoring, support, incident response, recovery, migration, and retirement.
- On-premises, hosted, managed, carrier, manufacturer, integrator, and externally operated components.

OUT OF SCOPE

- Claims of legal, safety, carrier, or regulatory approval without the responsible authority.
- Vendor certification treated as proof of the adopter configuration or operating effectiveness.
- Theoretical or laboratory behavior presented as production evidence without a declared boundary.
- Inherited controls without documented scope, owner, period, limitations, and shared responsibility.

Applicability triggers

- The capability supports a business-critical, safety-relevant, regulated, public, or externally dependent service.
- The environment can expose sensitive data, identity, control, physical process, communications, or shared infrastructure.
- A management plane, automation system, carrier, provider, or supplier can affect broad operational scope.
- Failure, compromise, or change can be difficult to detect, reverse, isolate, or recover.
- The system depends on hardware, firmware, radio, browser, cloud, endpoint, IoT, OT, or real-time media behavior.

MYTHOS-HW-0002 / HARDWARE AND COMPUTE

2. Definitions and Taxonomy

Controlled language for architecture, governance, and assessment

AUTHORITY

The right to approve, deny, constrain, or execute an action. Model output is not authority.

EVIDENCE

A reproducible source, trace, measurement, test, record, signed artifact, or documented observation supporting a claim.

ASSURANCE VIEW

A defined perspective such as governance, architecture, security, operations, privacy, or human oversight.

ATOMIC CRITERION

A uniquely identified requirement that can be assessed, evidenced, excepted, and revised independently.

IMPLEMENTATION PROFILE

A risk-proportional set of required depth, evidence, and gates for a class of use.

CONTROLLED EXCEPTION

A time-bounded deviation with owner, rationale, compensating control, residual risk, expiration, and review trigger.

DETERMINISTIC MECHANISM

A component whose inputs, policy, state transition, and side effects can be constrained and verified.

SOURCE AUTHORITY

The documented basis for treating a source as reliable for a defined claim, context, jurisdiction, and time.

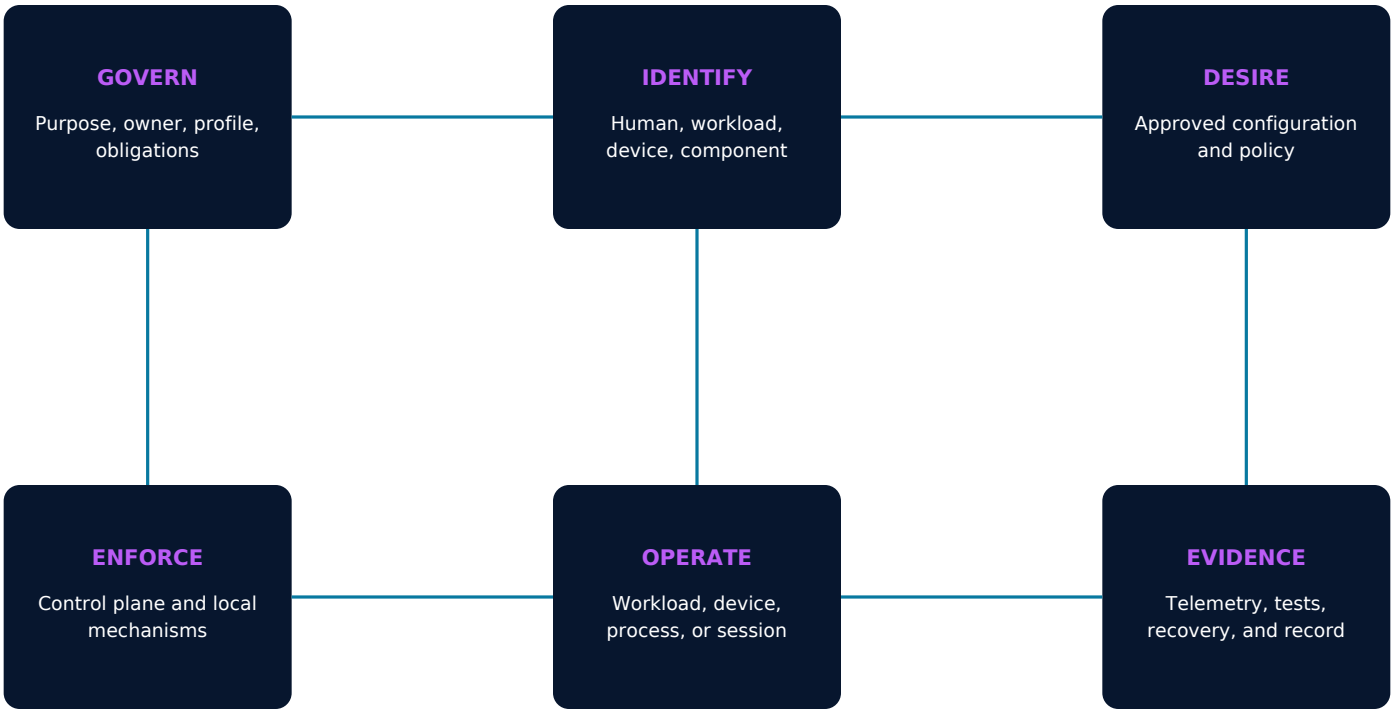
REASSESSMENT TRIGGER

A change, incident, drift signal, dependency revision, or time boundary requiring renewed evaluation.

MYTHOS-HW-0002 / HARDWARE AND COMPUTE

3. Architecture and Trust Model

Separation of governance, management, enforcement, runtime, telemetry, and recovery



Mandatory trust boundaries

- Management planes MUST be isolated, strongly authenticated, monitored, and recoverable.
- Identity and policy MUST be evaluated at every provider, network, device, workload, and administrative transition.
- Runtime state MUST be compared with approved desired state and material drift MUST trigger response.
- Safety and continuity mechanisms MUST remain available when cloud, WAN, identity, DNS, time, carrier, or management dependencies fail.
- Evidence MUST be protected from the systems and administrators whose behavior it is intended to assess.

MYTHOS-HW-0002 / HARDWARE AND COMPUTE

4. Governance and Accountability

Decision rights, separation of duties, and lifecycle ownership

ACCOUNTABLE OWNER

Accepts scope, risk tolerance, funding, and residual risk.

SYSTEM OWNER

Maintains architecture, inventory, operating boundaries, and change control.

SECURITY / PRIVACY

Defines threat, identity, data, isolation, monitoring, and incident requirements.

EVALUATION AUTHORITY

Designs representative tests, gates releases, and preserves reproducibility.

OPERATIONS

Maintains availability, rollback, recovery, evidence, and incident handling.

HUMAN OVERSIGHT

Reviews consequential decisions, exceptions, disputed outcomes, and harm signals.

DECISION PRINCIPLE

No single actor should be able to define the objective, grant authority, execute the consequential action, suppress evidence, and approve the result. Separation must be technical where consequence requires it.

MYTHOS-HW-0002 / HARDWARE AND COMPUTE

5. Normative Control System

Atomic criteria 01-04 of 18

MYTHOS-HW-0002-C01 Hardware Trust Model and Component Inventory

The organization MUST define, implement, verify, and maintain hardware trust model and component inventory for in-scope trusted compute, DPU, and firmware capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-HW-0002-C02 Root of Trust, TPM, DICE, and Endorsement Identity

The organization MUST define, implement, verify, and maintain root of trust, tpm, dice, and endorsement identity for in-scope trusted compute, DPU, and firmware capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-HW-0002-C03 Secure Boot, Measured Boot, and Reference Values

The organization MUST define, implement, verify, and maintain secure boot, measured boot, and reference values for in-scope trusted compute, DPU, and firmware capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-HW-0002-C04 Remote Attestation and Verifier Policy

The organization MUST define, implement, verify, and maintain remote attestation and verifier policy for in-scope trusted compute, DPU, and firmware capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-HW-0002 / HARDWARE AND COMPUTE

5. Normative Control System

Atomic criteria 05-08 of 18

MYTHOS-HW-0002-C05 BMC, Service Processor, and Out-of-Band Isolation

The organization MUST define, implement, verify, and maintain bmc, service processor, and out-of-band isolation for in-scope trusted compute, DPU, and firmware capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-HW-0002-C06 Firmware Source, Signing, and Update Authority

The organization MUST define, implement, verify, and maintain firmware source, signing, and update authority for in-scope trusted compute, DPU, and firmware capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-HW-0002-C07 DPU, SmartNIC, and Infrastructure Processing Boundary

The organization MUST define, implement, verify, and maintain dpu, smartnic, and infrastructure processing boundary for in-scope trusted compute, DPU, and firmware capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-HW-0002-C08 IOMMU, DMA, PCIe, and Device Assignment Isolation

The organization MUST define, implement, verify, and maintain iommu, dma, pcie, and device assignment isolation for in-scope trusted compute, DPU, and firmware capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-HW-0002 / HARDWARE AND COMPUTE

5. Normative Control System

Atomic criteria 09-12 of 18

MYTHOS-HW-0002-C09 Confidential Computing and Key-Release Policy

The organization MUST define, implement, verify, and maintain confidential computing and key-release policy for in-scope trusted compute, DPU, and firmware capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-HW-0002-C10 Supply-Chain Provenance and Counterfeit Detection

The organization MUST define, implement, verify, and maintain supply-chain provenance and counterfeit detection for in-scope trusted compute, DPU, and firmware capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-HW-0002-C11 Debug, Recovery, Manufacturing, and Service Modes

The organization MUST define, implement, verify, and maintain debug, recovery, manufacturing, and service modes for in-scope trusted compute, DPU, and firmware capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-HW-0002-C12 Secrets, Certificates, and Hardware-Bound Keys

The organization MUST define, implement, verify, and maintain secrets, certificates, and hardware-bound keys for in-scope trusted compute, DPU, and firmware capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-HW-0002 / HARDWARE AND COMPUTE

5. Normative Control System

Atomic criteria 13-15 of 18

MYTHOS-HW-0002-C13 Vulnerability, Revocation, and Emergency Update

The organization MUST define, implement, verify, and maintain vulnerability, revocation, and emergency update for in-scope trusted compute, DPU, and firmware capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-HW-0002-C14 Attestation Replay, Freshness, and Evidence Integrity

The organization MUST define, implement, verify, and maintain attestation replay, freshness, and evidence integrity for in-scope trusted compute, DPU, and firmware capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-HW-0002-C15 Tenant Placement and Trusted-Scheduling Decisions

The organization MUST define, implement, verify, and maintain tenant placement and trusted-scheduling decisions for in-scope trusted compute, DPU, and firmware capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-HW-0002 / HARDWARE AND COMPUTE

5. Normative Control System

Atomic criteria 16-18 of 18

MYTHOS-HW-0002-C16 Repair, Replacement, RMA, and Ownership Transfer

The organization MUST define, implement, verify, and maintain repair, replacement, rma, and ownership transfer for in-scope trusted compute, DPU, and firmware capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-HW-0002-C17 Sanitization, Decommissioning, and Key Destruction

The organization MUST define, implement, verify, and maintain sanitization, decommissioning, and key destruction for in-scope trusted compute, DPU, and firmware capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-HW-0002-C18 Trust Evidence and Periodic Reference-Value Review

The organization MUST define, implement, verify, and maintain trust evidence and periodic reference-value review for in-scope trusted compute, DPU, and firmware capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-HW-0002 / HARDWARE AND COMPUTE

6. Evidence and Assessment

Examine, interview, test, observe, restore, and trace

EXAMINE

Policies, inventories, diagrams, configuration, code, firmware, contracts, provider evidence, logs, exceptions, and lifecycle records.

INTERVIEW

Accountable owners, architects, engineers, operators, safety personnel, security teams, vendors, carriers, integrators, and responders.

TEST

Identity, segmentation, policy, negative paths, malformed input, capacity stress, dependency loss, failover, rollback, and revocation.

RESTORE

Independent restoration of configuration, data, service, device, controller, cluster, media, or browser state from protected evidence.

OBSERVE

Production-like performance, quality, drift, resource use, physical behavior, alarms, user impact, and incident execution.

TRACE

End-to-end linkage from approved intent and identity through change, runtime behavior, telemetry, outcome, exception, and review.

Evidence grades

A

Independently reproducible, complete, current, integrity-protected, and representative.

B

Reproduced by the implementing organization with strong traceability and controlled scope.

C

Partially reproduced or indirect; limitations, inherited responsibility, and uncertainty recorded.

D

Assertion, diagram, design intent, certificate, or vendor statement without reproduced behavior.

MYTHOS-HW-0002 / HARDWARE AND COMPUTE

7. Assurance Views

Six perspectives required for a complete assurance claim

GOVERNANCE

Ownership, purpose, risk tolerance, policy, exception, and decision rights.

ARCHITECTURE

Boundaries, dependencies, failure modes, state, data flow, and portability.

SECURITY

Identity, authorization, isolation, secrets, abuse resistance, and incident response.

OPERATIONS

Reliability, performance, cost, observability, change, recovery, and support.

PRIVACY / RIGHTS

Purpose limitation, minimization, consent, access, correction, deletion, and egress.

HUMAN / SOCIETAL

Usability, accessibility, disclosure, contestability, impact, and human control.

Conformance requires a defensible position across every applicable view. A strong aggregate score cannot compensate for an unowned system, a failed mandatory gate, untested recovery, or evidence below the accepted grade.

MYTHOS-HW-0002 / HARDWARE AND COMPUTE

7.2 Evaluation Metrics and Decision Gates

Measures must expose service behavior, control effectiveness, failure, uncertainty, and cost

SERVICE

Availability, correctness, coverage, quality, latency, throughput, and user or process outcome.

CONTROL

Unauthorized attempt, policy denial, drift, segmentation escape, privilege, and exception exposure.

RESILIENCE

Failover success, recovery time, data loss, safe degradation, restore validity, and dependency survival.

SECURITY

Exploitability, credential exposure, supply-chain integrity, attack detection, containment, and revocation.

CAPACITY

Utilization, saturation, queueing, radio or fabric headroom, thermal margin, quota, and forecast error.

OPERATIONS

Change failure, mean time to detect, repair, support burden, vendor escalation, and evidence completeness.

PRIVACY / SAFETY

Unauthorized egress, retention breach, consent coverage, unsafe action, physical consequence, and contestability.

LIFECYCLE

Patch debt, end-of-support exposure, portability, replacement time, retirement completion, and reassessment age.

Decision gate: thresholds MUST be declared before assessment, cover representative load and failure conditions, and identify mandatory safety, identity, recovery, and evidence failures that block promotion.

MYTHOS-HW-0002 / HARDWARE AND COMPUTE

8. Failure Modes and Adversarial Scenarios

Challenge assumptions before the system is trusted with consequential work

BMC COMPROMISE

Out-of-band management provides persistent control beneath the operating system.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

FIRMWARE SUPPLY-CHAIN ATTACK

Signed or trusted firmware is malicious, vulnerable, or incorrectly sourced.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

DMA BOUNDARY BYPASS

A device accesses memory outside its assigned trust domain.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

ATTESTATION REPLAY

Stale or forged evidence causes keys or workloads to be released to an untrusted platform.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

POWER OR THERMAL COLLAPSE

Facility or rack constraints create correlated hardware failure.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

CROSS-TENANT LEAKAGE

Scheduler, cache, fabric, storage, or diagnostics expose another workload.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

Adversarial testing MUST protect people and production systems. Use isolated environments, synthetic or minimized data, bounded authority, kill switches, explicit legal and ethical review, and documented stop conditions.

MYTHOS-HW-0002 / HARDWARE AND COMPUTE

9. Implementation Profiles

Risk-proportional implementation without weakening mandatory boundaries

FOUNDATIONAL

TYPICAL SCOPE

Low consequence, limited autonomy, non-sensitive data, reversible outputs.

MINIMUM DEPTH

Named owner; inventory; basic evaluation; access control; logging; rollback; annual review.

ADVANCED

TYPICAL SCOPE

Business-critical workflow, sensitive data, multiple tools or providers, moderate autonomy.

MINIMUM DEPTH

Formal threat model; representative evaluation; approval gates; isolated execution; tested recovery; quarterly review.

HIGH ASSURANCE

TYPICAL SCOPE

Safety, security, regulated, financial, identity, or broadly consequential use.

MINIMUM DEPTH

Independent challenge; strongest identity; deterministic enforcement; comprehensive evidence; canary release; continuous monitoring.

Profile selection rule

Choose the highest profile triggered by consequence, autonomy, sensitivity, scale, irreversibility, external dependency, or inability to rapidly detect and recover. Tailoring may strengthen controls; it may not erase a mandatory gate without a controlled exception.

MYTHOS-HW-0002 / HARDWARE AND COMPUTE

9.2 Operational Lifecycle

Evidence-bearing operation from concept through retirement

1. **CONCEPT**
   Purpose, owner, impact, boundaries
2. **DESIGN**
   Architecture, threat, data, identity
3. **BUILD**
   Controlled implementation and provenance
4. **EVALUATE**
   Representative, adversarial, recovery tests
5. **RELEASE**
   Approval, canary, rollback, evidence
6. **OPERATE**
   Observe, support, detect, respond
7. **CHANGE**
   Reassess, compare, gate, reverse
8. **RETIRE**
   Revoke, export, dispose, preserve

LIFECYCLE INVARIANT

Every stage MUST leave sufficient evidence for the next authority to understand what was intended, what changed, what was tested, what failed, what was accepted, what remains uncertain, and how to recover or exit.

MYTHOS-HW-0002 / HARDWARE AND COMPUTE

10. Adoption Roadmap

A sequenced path from uncontrolled capability to evidence-bearing operation

0-30 DAYS

Establish ownership and truth

Inventory systems and dependencies; classify risk; freeze unsupported claims; identify immediate privilege, data, and evidence gaps.

31-90 DAYS

Create enforceable boundaries

Define policy; isolate execution; implement identity and approval gates; establish representative evaluation and baseline telemetry.

3-6 MONTHS

Prove recovery and operating control

Exercise failure, rollback, revocation, incident, provider exit, and state-recovery scenarios; mature evidence packaging.

6-12 MONTHS

Scale assurance

Automate control checks; expand workload coverage; independent challenge; portfolio metrics; controlled exception expiry; continuous reassessment.

Adoption is complete only when operating evidence demonstrates the selected profile in the actual deployment context. Documentation alone is not conformance.

MYTHOS-HW-0002 / HARDWARE AND COMPUTE

10.2 Conformance and Exception Statement

What an assurance claim must contain

SYSTEM / SERVICE

Named, versioned capability and deployment boundary.

PROFILE

Foundational, Advanced, or High Assurance with trigger rationale.

SCOPE

Workloads, users, data, tools, regions, providers, and exclusions.

CRITERIA

Pass, partial, fail, not applicable, and exception status for every criterion.

EVIDENCE

Artifact references, evidence grade, date, environment, and reproducibility.

LIMITATIONS

Known failure modes, unsupported workloads, uncertainty, and residual risk.

EXCEPTIONS

Owner, rationale, compensating control, expiration, and approval.

VALIDITY

Assessment date, change boundary, review date, and reassessment triggers.

Prohibited claim: “compliant” without the named scope, profile, evidence date, limitations, exceptions, and authority accepting residual risk.

MYTHOS-HW-0002 / HARDWARE AND COMPUTE

11. Source Authority and Reference Register

Primary sources informing interpretation; current obligations and provider behavior must be verified at assessment t

NIST CSF 2.0 — **Cybersecurity Framework 2.0**

https://doi.org/10.6028/NIST.CSWP.29

Enterprise governance and cybersecurity outcome framework.

NIST SP 800-53 Rev. 5 — **Security and Privacy Controls for Information Systems and Organizations**

https://doi.org/10.6028/NIST.SP.800-53r5

Broad control baseline for organizational systems and services.

NIST SP 800-193 — **Platform Firmware Resiliency Guidelines**

https://doi.org/10.6028/NIST.SP.800-193

Platform firmware protection, detection, and recovery.

TCG TPM 2.0 — **Trusted Platform Module Library Specification**

https://trustedcomputinggroup.org/resource/tpm-library-specification/

TPM commands, objects, measurement, keys, and attestation foundations.

DMTF Redfish — **Redfish Standard**

https://www.dmtf.org/standards/redfish

Secure and interoperable hardware-management interfaces.

NIST SP 800-155 — **BIOS Integrity Measurement Guidelines**

https://doi.org/10.6028/NIST.SP.800-155

Firmware measurement and integrity-verification principles.

Source currency rule: authorities, specifications, legal obligations, provider services, firmware, browser behavior, carrier requirements, and operational evidence MUST be checked at assessment time. This publication records a 2026-07-24 source snapshot.

MYTHOS-HW-0002 / HARDWARE AND COMPUTE

11.2 Revision History and Decision Register

Permanent identity, controlled change, and publication integrity

VERSION	DATE	STATE	SUMMARY
1.0.0-candidate	2026-07-24	Candidate	Permanent-publication edition; source refresh; expanded criteria, evidence, assurance, and lifecycle guidance.
Next review	2027-01-24	Scheduled	Review source currency, threat evolution, operating evidence, adoption feedback, and proposed revisions.

Publication decisions

PERMANENT IDENTITY The document ID remains stable across revisions; batch or production sequence metadata is not public identity.

CHANGE CONTROL Normative changes require rationale, impact analysis, affected-criterion mapping, and approval.

SUPERSESSION A superseded edition remains traceable; current routes and catalogs identify the active edition.

CORRECTION Material errors require a recorded correction, affected-evidence analysis, and notification appropriate to impact.

ARCHIVAL INTEGRITY Published artifacts receive hashes, metadata, and a release manifest sufficient for independent verification.

END OF PERMANENT PUBLICATION / MYTHOS-HW-0002

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY

ENGINEERING STANDARDS LIBRARY / IOT, MATTER, AND EDGE SENSORS

IoT, Matter, and Edge Sensor Standard

Device identity, commissioning, fabrics, gateways, data minimization, updates, offline safety, transfer, and lifecycle evidence.

PERMANENT DOCUMENT ID

MYTHOS-IOT-0001

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

IoT, Matter, and Edge Sensor Standard

DOCUMENT ID
MYTHOS-IOT-0001

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

18	6	4	1
ATOMIC CRITERIA	ASSURANCE VIEWS	IMPLEMENTATION PROFILES	PERMANENT IDENTITY

Publication doctrine. This publication establishes durable requirements for its defined scope. Interpretation must preserve the permanent document identity, required evidence, controlled exceptions, source currency, and formal revision history.

INFRASTRUCTURE AND CONTROL TOPOLOGY

IoT, Matter, and Edge Sensors

A trustworthy IoT, Matter, and edge sensors system is a governed chain of ownership, identity, desired state, enforcement, workload or device behavior, telemetry, recovery, and independently reviewable evidence.

OWNERSHIP

Purpose, service boundary, accountable owner, suppliers, users, and retained responsibility remain explicit.

ENFORCEMENT

Identity, policy, segmentation, configuration, and local safety mechanisms constrain every trust transition.

CONTINUITY

Capacity, dependency loss, safe degradation, backup, restoration, rollback, and exit are tested before reliance.

EVIDENCE

Inventory, desired state, runtime observation, tests, incidents, exceptions, and change records form the assurance chain.

GOVERNED INFRASTRUCTURE FLOW

Reconciliation, detection, response, restoration, and controlled evolution

INTERPRETATION AND ASSURANCE

How to apply this publication

Normative intent

Requirements define outcomes and constraints. Implementations may vary, but evidence must demonstrate equivalent control.

Evidence over assertion

Vendor documentation and design intent are not equivalent to reproduced behavior. Record evidence grade and uncertainty.

Risk-proportional depth

Higher consequence, autonomy, sensitivity, and irreversibility require stronger isolation, review, verification, and recovery.

Controlled exception

Exceptions require an owner, rationale, compensating control, residual-risk acceptance, expiration, and reevaluation trigger.

Normalized assessment scale

0	1	2	3	4	5
ABSENT	AD HOC	PARTIAL	OPERATIONAL	ADVANCED	LEADING

A numeric score must never hide a failed mandatory gate, missing evidence, untested workload, or unacceptable residual risk.

INTERACTIVE NAVIGATION

Contents

Navigation doctrine

Bookmarks and internal links are conveniences. Permanent document identity, criterion identifiers, evidence references, and revision history remain authoritative.

MYTHOS-IOT-0001 / IOT AND EDGE SENSORS

0. Executive Mandate

Purpose, strategic outcome, and non-negotiable operating doctrine

MANDATE

Organizations adopting IoT, Matter, and edge sensors capabilities MUST define an owned service boundary, establish enforceable identity and configuration, protect data and safety, test failure and recovery, and preserve evidence sufficient for independent review.

Required outcomes

- Create a durable governance boundary for IoT, Matter, and edge sensors across design, acquisition, deployment, operation, change, and retirement.
- Require risk-proportional segmentation, identity, configuration, telemetry, resilience, recovery, and supplier controls.
- Prevent central automation, administrative tooling, or provider services from becoming unbounded authority.
- Prove operation with representative workloads, devices, failure modes, and restoration exercises.
- Define measurable conformance, exceptions, reassessment triggers, and a viable exit path.

Decision boundary: conformance supports a documented assurance claim for the named scope. It does not prove universal availability, safety, regulatory acceptance, vendor fitness, or immunity from cyber-physical failure.

MYTHOS-IOT-0001 / IOT AND EDGE SENSORS

1. Scope and Applicability

Boundary definition, audience, exclusions, and triggering conditions

IN SCOPE

- Architecture, acquisition, configuration, integration, and operation of IoT, Matter, and edge sensors capabilities.
- Human, workload, device, service, network, data, facility, provider, supplier, and evidence dependencies.
- Design, deployment, monitoring, support, incident response, recovery, migration, and retirement.
- On-premises, hosted, managed, carrier, manufacturer, integrator, and externally operated components.

OUT OF SCOPE

- Claims of legal, safety, carrier, or regulatory approval without the responsible authority.
- Vendor certification treated as proof of the adopter configuration or operating effectiveness.
- Theoretical or laboratory behavior presented as production evidence without a declared boundary.
- Inherited controls without documented scope, owner, period, limitations, and shared responsibility.

Applicability triggers

- The capability supports a business-critical, safety-relevant, regulated, public, or externally dependent service.
- The environment can expose sensitive data, identity, control, physical process, communications, or shared infrastructure.
- A management plane, automation system, carrier, provider, or supplier can affect broad operational scope.
- Failure, compromise, or change can be difficult to detect, reverse, isolate, or recover.
- The system depends on hardware, firmware, radio, browser, cloud, endpoint, IoT, OT, or real-time media behavior.

MYTHOS-IOT-0001 / IOT AND EDGE SENSORS

2. Definitions and Taxonomy

Controlled language for architecture, governance, and assessment

AUTHORITY

The right to approve, deny, constrain, or execute an action. Model output is not authority.

EVIDENCE

A reproducible source, trace, measurement, test, record, signed artifact, or documented observation supporting a claim.

ASSURANCE VIEW

A defined perspective such as governance, architecture, security, operations, privacy, or human oversight.

ATOMIC CRITERION

A uniquely identified requirement that can be assessed, evidenced, excepted, and revised independently.

IMPLEMENTATION PROFILE

A risk-proportional set of required depth, evidence, and gates for a class of use.

CONTROLLED EXCEPTION

A time-bounded deviation with owner, rationale, compensating control, residual risk, expiration, and review trigger.

DETERMINISTIC MECHANISM

A component whose inputs, policy, state transition, and side effects can be constrained and verified.

SOURCE AUTHORITY

The documented basis for treating a source as reliable for a defined claim, context, jurisdiction, and time.

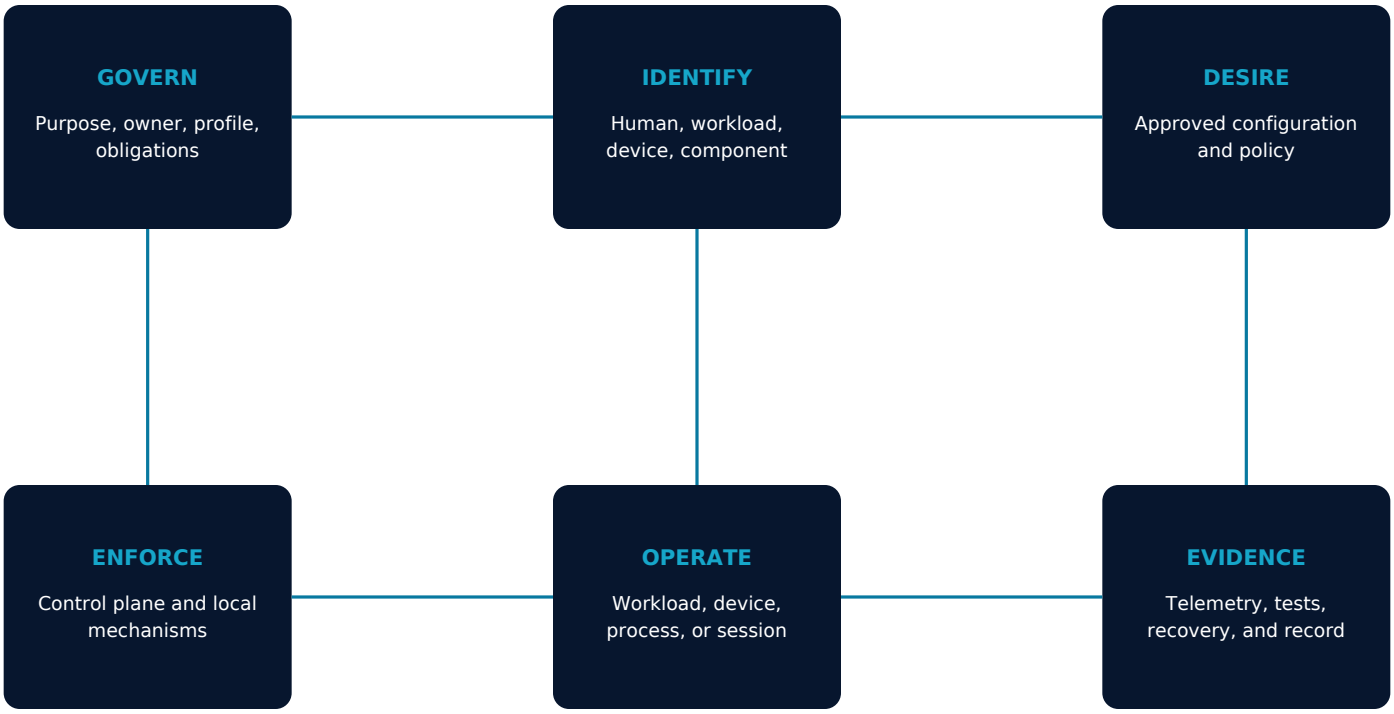
REASSESSMENT TRIGGER

A change, incident, drift signal, dependency revision, or time boundary requiring renewed evaluation.

MYTHOS-IOT-0001 / IOT AND EDGE SENSORS

3. Architecture and Trust Model

Separation of governance, management, enforcement, runtime, telemetry, and recovery



Mandatory trust boundaries

- Management planes MUST be isolated, strongly authenticated, monitored, and recoverable.
- Identity and policy MUST be evaluated at every provider, network, device, workload, and administrative transition.
- Runtime state MUST be compared with approved desired state and material drift MUST trigger response.
- Safety and continuity mechanisms MUST remain available when cloud, WAN, identity, DNS, time, carrier, or management dependencies fail.
- Evidence MUST be protected from the systems and administrators whose behavior it is intended to assess.

MYTHOS-IOT-0001 / IOT AND EDGE SENSORS

4. Governance and Accountability

Decision rights, separation of duties, and lifecycle ownership

ACCOUNTABLE OWNER

Accepts scope, risk tolerance, funding, and residual risk.

SYSTEM OWNER

Maintains architecture, inventory, operating boundaries, and change control.

SECURITY / PRIVACY

Defines threat, identity, data, isolation, monitoring, and incident requirements.

EVALUATION AUTHORITY

Designs representative tests, gates releases, and preserves reproducibility.

OPERATIONS

Maintains availability, rollback, recovery, evidence, and incident handling.

HUMAN OVERSIGHT

Reviews consequential decisions, exceptions, disputed outcomes, and harm signals.

DECISION PRINCIPLE

No single actor should be able to define the objective, grant authority, execute the consequential action, suppress evidence, and approve the result. Separation must be technical where consequence requires it.

MYTHOS-IOT-0001 / IOT AND EDGE SENSORS

5. Normative Control System

Atomic criteria 01-04 of 18

MYTHOS-IOT-0001-C01 Device Purpose, Capability, and Harm Classification

The organization MUST define, implement, verify, and maintain device purpose, capability, and harm classification for in-scope IoT, Matter, and edge sensors capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-IOT-0001-C02 Manufacturer, Model, Firmware, and Component Inventory

The organization MUST define, implement, verify, and maintain manufacturer, model, firmware, and component inventory for in-scope IoT, Matter, and edge sensors capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-IOT-0001-C03 Unique Device Identity and Credential Provisioning

The organization MUST define, implement, verify, and maintain unique device identity and credential provisioning for in-scope IoT, Matter, and edge sensors capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-IOT-0001-C04 Secure Commissioning and Ownership Establishment

The organization MUST define, implement, verify, and maintain secure commissioning and ownership establishment for in-scope IoT, Matter, and edge sensors capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-IOT-0001 / IOT AND EDGE SENSORS

5. Normative Control System

Atomic criteria 05-08 of 18

MYTHOS-IOT-0001-C05 Matter Fabric, Administrator, and Access Governance

The organization MUST define, implement, verify, and maintain matter fabric, administrator, and access governance for in-scope IoT, Matter, and edge sensors capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-IOT-0001-C06 Gateway, Hub, Broker, and Cloud Trust Boundary

The organization MUST define, implement, verify, and maintain gateway, hub, broker, and cloud trust boundary for in-scope IoT, Matter, and edge sensors capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-IOT-0001-C07 Network Segmentation and Constrained Egress

The organization MUST define, implement, verify, and maintain network segmentation and constrained egress for in-scope IoT, Matter, and edge sensors capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-IOT-0001-C08 Sensor Integrity, Calibration, and Spoofing Resistance

The organization MUST define, implement, verify, and maintain sensor integrity, calibration, and spoofing resistance for in-scope IoT, Matter, and edge sensors capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-IOT-0001 / IOT AND EDGE SENSORS

5. Normative Control System

Atomic criteria 09-12 of 18

MYTHOS-IOT-0001-C09 Data Minimization, Consent, and Retention

The organization MUST define, implement, verify, and maintain data minimization, consent, and retention for in-scope IoT, Matter, and edge sensors capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-IOT-0001-C10 Local Processing, Buffering, and Offline Behavior

The organization MUST define, implement, verify, and maintain local processing, buffering, and offline behavior for in-scope IoT, Matter, and edge sensors capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-IOT-0001-C11 Signed Update, Rollback, and Vulnerability Response

The organization MUST define, implement, verify, and maintain signed update, rollback, and vulnerability response for in-scope IoT, Matter, and edge sensors capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-IOT-0001-C12 Physical Tamper, Debug, and Service Interface

The organization MUST define, implement, verify, and maintain physical tamper, debug, and service interface for in-scope IoT, Matter, and edge sensors capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-IOT-0001 / IOT AND EDGE SENSORS

5. Normative Control System

Atomic criteria 13-15 of 18

MYTHOS-IOT-0001-C13 Power, Battery, Connectivity, and Safe Degradation

The organization MUST define, implement, verify, and maintain power, battery, connectivity, and safe degradation for in-scope IoT, Matter, and edge sensors capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-IOT-0001-C14 Telemetry, Health, and Fleet Observability

The organization MUST define, implement, verify, and maintain telemetry, health, and fleet observability for in-scope IoT, Matter, and edge sensors capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-IOT-0001-C15 Compromise Containment and Credential Revocation

The organization MUST define, implement, verify, and maintain compromise containment and credential revocation for in-scope IoT, Matter, and edge sensors capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-IOT-0001 / IOT AND EDGE SENSORS

5. Normative Control System

Atomic criteria 16-18 of 18

MYTHOS-IOT-0001-C16 Factory Reset, Ownership Transfer, and Recommissioning

The organization MUST define, implement, verify, and maintain factory reset, ownership transfer, and recommissioning for in-scope IoT, Matter, and edge sensors capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-IOT-0001-C17 Support Period, End-of-Life, and Replacement

The organization MUST define, implement, verify, and maintain support period, end-of-life, and replacement for in-scope IoT, Matter, and edge sensors capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-IOT-0001-C18 Device Evidence and Periodic Reassessment

The organization MUST define, implement, verify, and maintain device evidence and periodic reassessment for in-scope IoT, Matter, and edge sensors capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-IOT-0001 / IOT AND EDGE SENSORS

6. Evidence and Assessment

Examine, interview, test, observe, restore, and trace

EXAMINE

Policies, inventories, diagrams, configuration, code, firmware, contracts, provider evidence, logs, exceptions, and lifecycle records.

INTERVIEW

Accountable owners, architects, engineers, operators, safety personnel, security teams, vendors, carriers, integrators, and responders.

TEST

Identity, segmentation, policy, negative paths, malformed input, capacity stress, dependency loss, failover, rollback, and revocation.

RESTORE

Independent restoration of configuration, data, service, device, controller, cluster, media, or browser state from protected evidence.

OBSERVE

Production-like performance, quality, drift, resource use, physical behavior, alarms, user impact, and incident execution.

TRACE

End-to-end linkage from approved intent and identity through change, runtime behavior, telemetry, outcome, exception, and review.

Evidence grades

A

Independently reproducible, complete, current, integrity-protected, and representative.

B

Reproduced by the implementing organization with strong traceability and controlled scope.

C

Partially reproduced or indirect; limitations, inherited responsibility, and uncertainty recorded.

D

Assertion, diagram, design intent, certificate, or vendor statement without reproduced behavior.

MYTHOS-IOT-0001 / IOT AND EDGE SENSORS

7. Assurance Views

Six perspectives required for a complete assurance claim

GOVERNANCE

Ownership, purpose, risk tolerance, policy, exception, and decision rights.

ARCHITECTURE

Boundaries, dependencies, failure modes, state, data flow, and portability.

SECURITY

Identity, authorization, isolation, secrets, abuse resistance, and incident response.

OPERATIONS

Reliability, performance, cost, observability, change, recovery, and support.

PRIVACY / RIGHTS

Purpose limitation, minimization, consent, access, correction, deletion, and egress.

HUMAN / SOCIETAL

Usability, accessibility, disclosure, contestability, impact, and human control.

Conformance requires a defensible position across every applicable view. A strong aggregate score cannot compensate for an unowned system, a failed mandatory gate, untested recovery, or evidence below the accepted grade.

MYTHOS-IOT-0001 / IOT AND EDGE SENSORS

7.2 Evaluation Metrics and Decision Gates

Measures must expose service behavior, control effectiveness, failure, uncertainty, and cost

SERVICE

Availability, correctness, coverage, quality, latency, throughput, and user or process outcome.

CONTROL

Unauthorized attempt, policy denial, drift, segmentation escape, privilege, and exception exposure.

RESILIENCE

Failover success, recovery time, data loss, safe degradation, restore validity, and dependency survival.

SECURITY

Exploitability, credential exposure, supply-chain integrity, attack detection, containment, and revocation.

CAPACITY

Utilization, saturation, queueing, radio or fabric headroom, thermal margin, quota, and forecast error.

OPERATIONS

Change failure, mean time to detect, repair, support burden, vendor escalation, and evidence completeness.

PRIVACY / SAFETY

Unauthorized egress, retention breach, consent coverage, unsafe action, physical consequence, and contestability.

LIFECYCLE

Patch debt, end-of-support exposure, portability, replacement time, retirement completion, and reassessment age.

Decision gate: thresholds MUST be declared before assessment, cover representative load and failure conditions, and identify mandatory safety, identity, recovery, and evidence failures that block promotion.

MYTHOS-IOT-0001 / IOT AND EDGE SENSORS

8. Failure Modes and Adversarial Scenarios

Challenge assumptions before the system is trusted with consequential work

INSECURE COMMISSIONING

A rogue party claims or joins a device during ownership establishment.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

SHARED OR STATIC KEYS

Compromise of one device enables broader fleet access.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

SENSOR SPOOFING

False physical measurements drive unsafe or incorrect automation.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

UPDATE FAILURE

A signed update bricks devices or cannot be rolled back.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

GATEWAY COMPROMISE

A hub or broker becomes a bridge across segmented device groups.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

UNSAFE OFFLINE BEHAVIOR

Connectivity loss causes uncontrolled operation or silent data loss.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

Adversarial testing MUST protect people and production systems. Use isolated environments, synthetic or minimized data, bounded authority, kill switches, explicit legal and ethical review, and documented stop conditions.

MYTHOS-IOT-0001 / IOT AND EDGE SENSORS

9. Implementation Profiles

Risk-proportional implementation without weakening mandatory boundaries

	FOUNDATIONAL	ADVANCED	HIGH ASSURANCE
TYPICAL SCOPE	Low consequence, limited autonomy, non-sensitive data, reversible outputs.	Business-critical workflow, sensitive data, multiple tools or providers, moderate autonomy.	Safety, security, regulated, financial, identity, or broadly consequential use.
MINIMUM DEPTH	Named owner; inventory; basic evaluation; access control; logging; rollback; annual review.	Formal threat model; representative evaluation; approval gates; isolated execution; tested recovery; quarterly review.	Independent challenge; strongest identity; deterministic enforcement; comprehensive evidence; canary release; continuous monitoring.

Profile selection rule

Choose the highest profile triggered by consequence, autonomy, sensitivity, scale, irreversibility, external dependency, or inability to rapidly detect and recover. Tailoring may strengthen controls; it may not erase a mandatory gate without a controlled exception.

MYTHOS-IOT-0001 / IOT AND EDGE SENSORS

9.2 Operational Lifecycle

Evidence-bearing operation from concept through retirement

1. **CONCEPT**
   Purpose, owner, impact, boundaries
2. **DESIGN**
   Architecture, threat, data, identity
3. **BUILD**
   Controlled implementation and provenance
4. **EVALUATE**
   Representative, adversarial, recovery tests
5. **RELEASE**
   Approval, canary, rollback, evidence
6. **OPERATE**
   Observe, support, detect, respond
7. **CHANGE**
   Reassess, compare, gate, reverse
8. **RETIRE**
   Revoke, export, dispose, preserve

LIFECYCLE INVARIANT

Every stage MUST leave sufficient evidence for the next authority to understand what was intended, what changed, what was tested, what failed, what was accepted, what remains uncertain, and how to recover or exit.

MYTHOS-IOT-0001 / IOT AND EDGE SENSORS

10. Adoption Roadmap

A sequenced path from uncontrolled capability to evidence-bearing operation

0-30 DAYS

Establish ownership and truth

Inventory systems and dependencies; classify risk; freeze unsupported claims; identify immediate privilege, data, and evidence gaps.

31-90 DAYS

Create enforceable boundaries

Define policy; isolate execution; implement identity and approval gates; establish representative evaluation and baseline telemetry.

3-6 MONTHS

Prove recovery and operating control

Exercise failure, rollback, revocation, incident, provider exit, and state-recovery scenarios; mature evidence packaging.

6-12 MONTHS

Scale assurance

Automate control checks; expand workload coverage; independent challenge; portfolio metrics; controlled exception expiry; continuous reassessment.

Adoption is complete only when operating evidence demonstrates the selected profile in the actual deployment context. Documentation alone is not conformance.

MYTHOS-IOT-0001 / IOT AND EDGE SENSORS

10.2 Conformance and Exception Statement

What an assurance claim must contain

SYSTEM / SERVICE

Named, versioned capability and deployment boundary.

PROFILE

Foundational, Advanced, or High Assurance with trigger rationale.

SCOPE

Workloads, users, data, tools, regions, providers, and exclusions.

CRITERIA

Pass, partial, fail, not applicable, and exception status for every criterion.

EVIDENCE

Artifact references, evidence grade, date, environment, and reproducibility.

LIMITATIONS

Known failure modes, unsupported workloads, uncertainty, and residual risk.

EXCEPTIONS

Owner, rationale, compensating control, expiration, and approval.

VALIDITY

Assessment date, change boundary, review date, and reassessment triggers.

Prohibited claim: “compliant” without the named scope, profile, evidence date, limitations, exceptions, and authority accepting residual risk.

MYTHOS-IOT-0001 / IOT AND EDGE SENSORS

11. Source Authority and Reference Register

Primary sources informing interpretation; current obligations and provider behavior must be verified at assessment t

NIST CSF 2.0 — **Cybersecurity Framework 2.0**

https://doi.org/10.6028/NIST.CSWP.29

Enterprise governance and cybersecurity outcome framework.

NIST SP 800-53 Rev. 5 — **Security and Privacy Controls for Information Systems and Organizations**

https://doi.org/10.6028/NIST.SP.800-53r5

Broad control baseline for organizational systems and services.

NISTIR 8259A — **IoT Device Cybersecurity Capability Core Baseline**

https://doi.org/10.6028/NIST.IR.8259A

Core device capabilities for identification, configuration, data, interfaces, update, and state awareness.

NIST SP 800-213 — **IoT Device Cybersecurity Guidance for the Federal Government**

https://doi.org/10.6028/NIST.SP.800-213

Risk-based IoT device requirements and supporting capabilities.

ETSI EN 303 645 — **Cyber Security for Consumer Internet of Things**

https://www.etsi.org/deliver/etsi_en/303600_303699/303645/

Consumer-IoT security provisions and implementation guidance.

Matter Specification — **Connectivity Standards Alliance Matter specifications**

https://csa-iot.org/developer-resource/specifications-download-request/

Commissioning, fabrics, device identity, access, and interoperability.

Source currency rule: authorities, specifications, legal obligations, provider services, firmware, browser behavior, carrier requirements, and operational evidence MUST be checked at assessment time. This publication records a 2026-07-24 source snapshot.

MYTHOS-IOT-0001 / IOT AND EDGE SENSORS

11.2 Revision History and Decision Register

Permanent identity, controlled change, and publication integrity

VERSION	DATE	STATE	SUMMARY
1.0.0-candidate	2026-07-24	Candidate	Permanent-publication edition; source refresh; expanded criteria, evidence, assurance, and lifecycle guidance.
Next review	2027-01-24	Scheduled	Review source currency, threat evolution, operating evidence, adoption feedback, and proposed revisions.

Publication decisions

PERMANENT IDENTITY The document ID remains stable across revisions; batch or production sequence metadata is not public identity.

CHANGE CONTROL Normative changes require rationale, impact analysis, affected-criterion mapping, and approval.

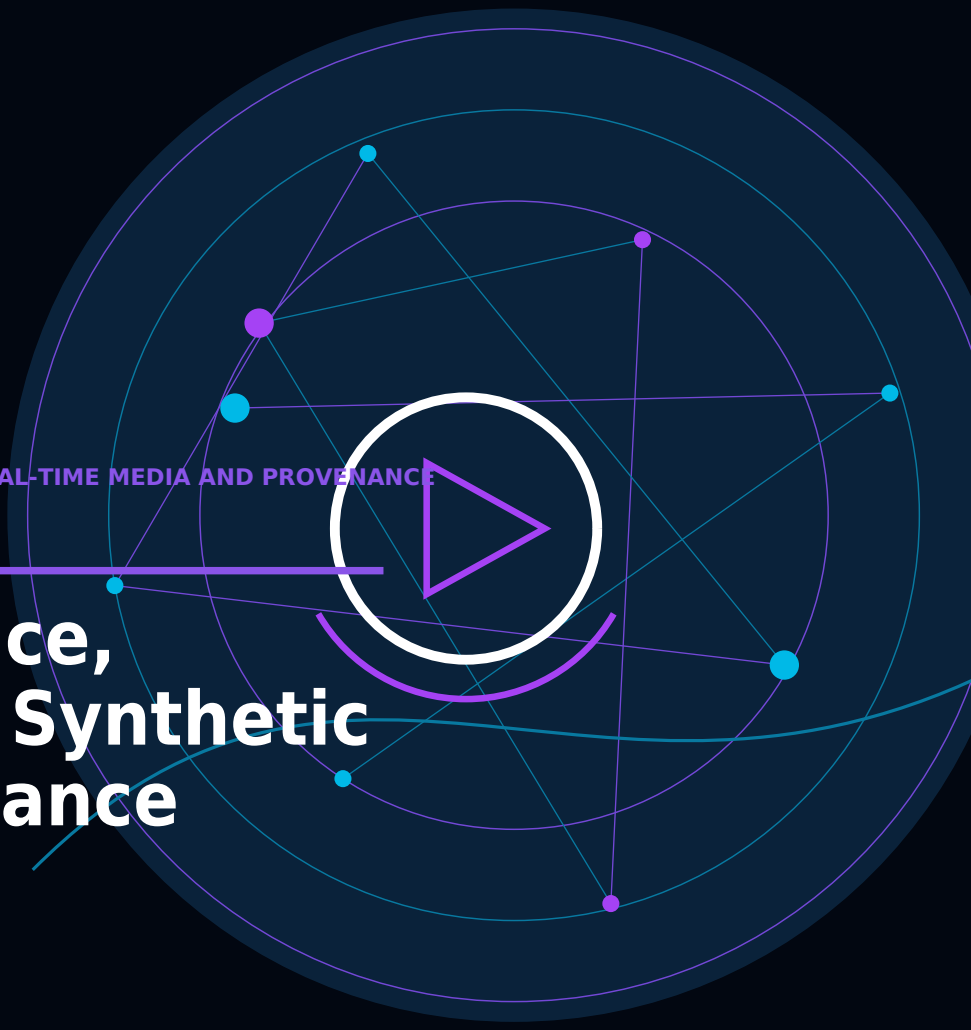
SUPERSESSION A superseded edition remains traceable; current routes and catalogs identify the active edition.

CORRECTION Material errors require a recorded correction, affected-evidence analysis, and notification appropriate to impact.

ARCHIVAL INTEGRITY Published artifacts receive hashes, metadata, and a release manifest sufficient for independent verification.

END OF PERMANENT PUBLICATION / MYTHOS-IOT-0001

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY



ENGINEERING STANDARDS LIBRARY / REAL-TIME MEDIA AND PROVENANCE

Real-Time Voice, WebRTC, and Synthetic Media Provenance Standard

Real-time media security, identity, consent, relay, recording, synthetic participants, provenance, quality, abuse response, and evidence.

PERMANENT DOCUMENT ID

MYTHOS-MED-0001

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

Real-Time Voice, WebRTC, and Synthetic Media Provenance Standard

DOCUMENT ID
MYTHOS-MED-0001

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

18	6	4	1
ATOMIC CRITERIA	ASSURANCE VIEWS	IMPLEMENTATION PROFILES	PERMANENT IDENTITY

Publication doctrine. This publication establishes durable requirements for its defined scope. Interpretation must preserve the permanent document identity, required evidence, controlled exceptions, source currency, and formal revision history.

INFRASTRUCTURE AND CONTROL TOPOLOGY

Real-Time Media and Provenance

A trustworthy real-time voice, WebRTC, and synthetic media system is a governed chain of ownership, identity, desired state, enforcement, workload or device behavior, telemetry, recovery, and independently reviewable evidence.

OWNERSHIP

Purpose, service boundary, accountable owner, suppliers, users, and retained responsibility remain explicit.

ENFORCEMENT

Identity, policy, segmentation, configuration, and local safety mechanisms constrain every trust transition.

CONTINUITY

Capacity, dependency loss, safe degradation, backup, restoration, rollback, and exit are tested before reliance.

EVIDENCE

Inventory, desired state, runtime observation, tests, incidents, exceptions, and change records form the assurance chain.

GOVERNED INFRASTRUCTURE FLOW

Reconciliation, detection, response, restoration, and controlled evolution

INTERPRETATION AND ASSURANCE

How to apply this publication

Normative intent

Requirements define outcomes and constraints. Implementations may vary, but evidence must demonstrate equivalent control.

Evidence over assertion

Vendor documentation and design intent are not equivalent to reproduced behavior. Record evidence grade and uncertainty.

Risk-proportional depth

Higher consequence, autonomy, sensitivity, and irreversibility require stronger isolation, review, verification, and recovery.

Controlled exception

Exceptions require an owner, rationale, compensating control, residual-risk acceptance, expiration, and reevaluation trigger.

Normalized assessment scale

0	1	2	3	4	5
ABSENT	AD HOC	PARTIAL	OPERATIONAL	ADVANCED	LEADING

A numeric score must never hide a failed mandatory gate, missing evidence, untested workload, or unacceptable residual risk.

INTERACTIVE NAVIGATION

Contents

Navigation doctrine

Bookmarks and internal links are conveniences. Permanent document identity, criterion identifiers, evidence references, and revision history remain authoritative.

MYTHOS-MED-0001 / REAL-TIME MEDIA

0. Executive Mandate

Purpose, strategic outcome, and non-negotiable operating doctrine

MANDATE

Organizations adopting real-time voice, WebRTC, and synthetic media capabilities MUST define an owned service boundary, establish enforceable identity and configuration, protect data and safety, test failure and recovery, and preserve evidence sufficient for independent review.

Required outcomes

- Create a durable governance boundary for real-time voice, WebRTC, and synthetic media across design, acquisition, deployment, operation, change, and retirement.
- Require risk-proportional segmentation, identity, configuration, telemetry, resilience, recovery, and supplier controls.
- Prevent central automation, administrative tooling, or provider services from becoming unbounded authority.
- Prove operation with representative workloads, devices, failure modes, and restoration exercises.
- Define measurable conformance, exceptions, reassessment triggers, and a viable exit path.

Decision boundary: conformance supports a documented assurance claim for the named scope. It does not prove universal availability, safety, regulatory acceptance, vendor fitness, or immunity from cyber-physical failure.

MYTHOS-MED-0001 / REAL-TIME MEDIA

1. Scope and Applicability

Boundary definition, audience, exclusions, and triggering conditions

IN SCOPE

- Architecture, acquisition, configuration, integration, and operation of real-time voice, WebRTC, and synthetic media capabilities.
- Human, workload, device, service, network, data, facility, provider, supplier, and evidence dependencies.
- Design, deployment, monitoring, support, incident response, recovery, migration, and retirement.
- On-premises, hosted, managed, carrier, manufacturer, integrator, and externally operated components.

OUT OF SCOPE

- Claims of legal, safety, carrier, or regulatory approval without the responsible authority.
- Vendor certification treated as proof of the adopter configuration or operating effectiveness.
- Theoretical or laboratory behavior presented as production evidence without a declared boundary.
- Inherited controls without documented scope, owner, period, limitations, and shared responsibility.

Applicability triggers

- The capability supports a business-critical, safety-relevant, regulated, public, or externally dependent service.
- The environment can expose sensitive data, identity, control, physical process, communications, or shared infrastructure.
- A management plane, automation system, carrier, provider, or supplier can affect broad operational scope.
- Failure, compromise, or change can be difficult to detect, reverse, isolate, or recover.
- The system depends on hardware, firmware, radio, browser, cloud, endpoint, IoT, OT, or real-time media behavior.

MYTHOS-MED-0001 / REAL-TIME MEDIA

2. Definitions and Taxonomy

Controlled language for architecture, governance, and assessment

AUTHORITY

The right to approve, deny, constrain, or execute an action. Model output is not authority.

EVIDENCE

A reproducible source, trace, measurement, test, record, signed artifact, or documented observation supporting a claim.

ASSURANCE VIEW

A defined perspective such as governance, architecture, security, operations, privacy, or human oversight.

ATOMIC CRITERION

A uniquely identified requirement that can be assessed, evidenced, excepted, and revised independently.

IMPLEMENTATION PROFILE

A risk-proportional set of required depth, evidence, and gates for a class of use.

CONTROLLED EXCEPTION

A time-bounded deviation with owner, rationale, compensating control, residual risk, expiration, and review trigger.

DETERMINISTIC MECHANISM

A component whose inputs, policy, state transition, and side effects can be constrained and verified.

SOURCE AUTHORITY

The documented basis for treating a source as reliable for a defined claim, context, jurisdiction, and time.

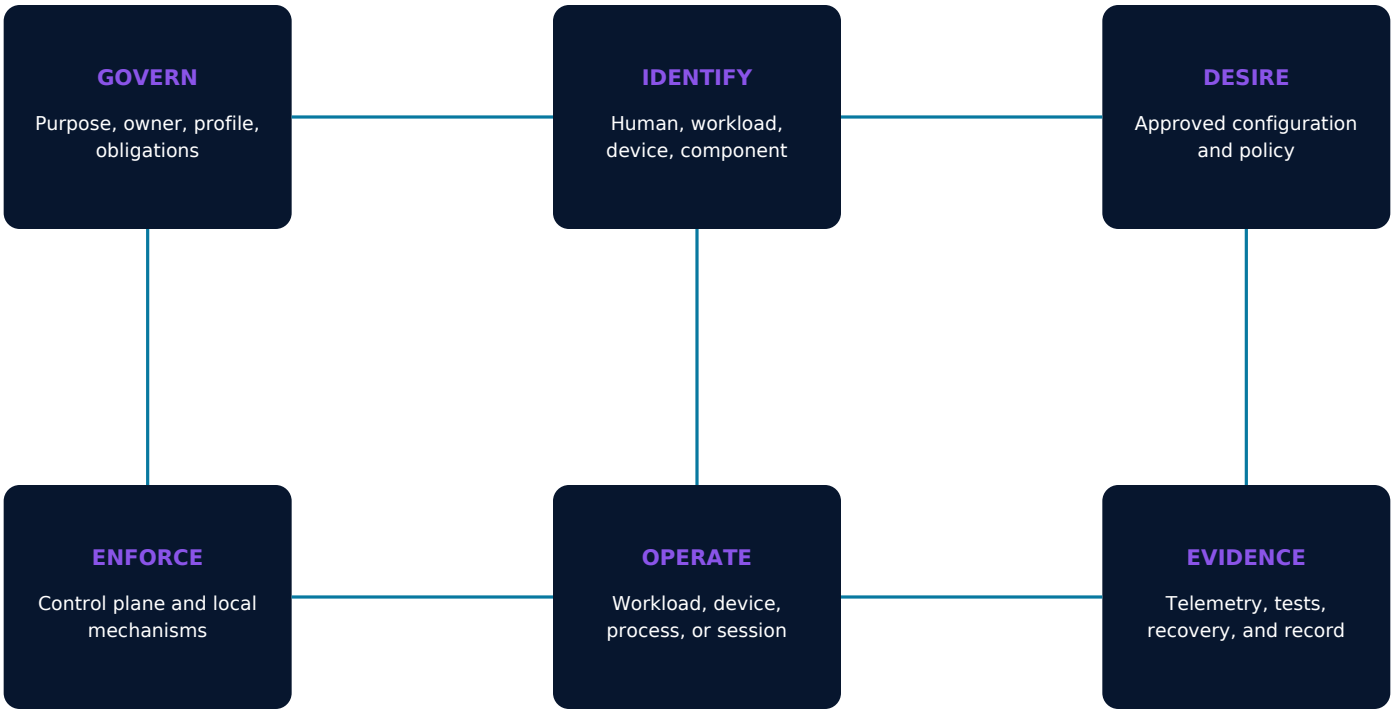
REASSESSMENT TRIGGER

A change, incident, drift signal, dependency revision, or time boundary requiring renewed evaluation.

MYTHOS-MED-0001 / REAL-TIME MEDIA

3. Architecture and Trust Model

Separation of governance, management, enforcement, runtime, telemetry, and recovery



Mandatory trust boundaries

- Management planes MUST be isolated, strongly authenticated, monitored, and recoverable.
- Identity and policy MUST be evaluated at every provider, network, device, workload, and administrative transition.
- Runtime state MUST be compared with approved desired state and material drift MUST trigger response.
- Safety and continuity mechanisms MUST remain available when cloud, WAN, identity, DNS, time, carrier, or management dependencies fail.
- Evidence MUST be protected from the systems and administrators whose behavior it is intended to assess.

MYTHOS-MED-0001 / REAL-TIME MEDIA

4. Governance and Accountability

Decision rights, separation of duties, and lifecycle ownership

ACCOUNTABLE OWNER

Accepts scope, risk tolerance, funding, and residual risk.

SYSTEM OWNER

Maintains architecture, inventory, operating boundaries, and change control.

SECURITY / PRIVACY

Defines threat, identity, data, isolation, monitoring, and incident requirements.

EVALUATION AUTHORITY

Designs representative tests, gates releases, and preserves reproducibility.

OPERATIONS

Maintains availability, rollback, recovery, evidence, and incident handling.

HUMAN OVERSIGHT

Reviews consequential decisions, exceptions, disputed outcomes, and harm signals.

DECISION PRINCIPLE

No single actor should be able to define the objective, grant authority, execute the consequential action, suppress evidence, and approve the result. Separation must be technical where consequence requires it.

MYTHOS-MED-0001 / REAL-TIME MEDIA

5. Normative Control System

Atomic criteria 01-04 of 18

MYTHOS-MED-0001-C01 Session Purpose, Participant, and Trust Disclosure

The organization MUST define, implement, verify, and maintain session purpose, participant, and trust disclosure for in-scope real-time voice, WebRTC, and synthetic media capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-MED-0001-C02 User, Device, Service, Bot, and Persona Identity

The organization MUST define, implement, verify, and maintain user, device, service, bot, and persona identity for in-scope real-time voice, WebRTC, and synthetic media capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-MED-0001-C03 Signaling Authentication and Session Authorization

The organization MUST define, implement, verify, and maintain signaling authentication and session authorization for in-scope real-time voice, WebRTC, and synthetic media capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-MED-0001-C04 DTLS-SRTP, Keying, and Media Confidentiality

The organization MUST define, implement, verify, and maintain dtls-srtp, keying, and media confidentiality for in-scope real-time voice, WebRTC, and synthetic media capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-MED-0001 / REAL-TIME MEDIA

5. Normative Control System

Atomic criteria 05-08 of 18

MYTHOS-MED-0001-C05 ICE, STUN, TURN, and Relay Trust Boundary

The organization MUST define, implement, verify, and maintain ice, stun, turn, and relay trust boundary for in-scope real-time voice, WebRTC, and synthetic media capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-MED-0001-C06 Browser, Application, and Media Permission Control

The organization MUST define, implement, verify, and maintain browser, application, and media permission control for in-scope real-time voice, WebRTC, and synthetic media capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-MED-0001-C07 Recording, Transcription, Consent, and Retention

The organization MUST define, implement, verify, and maintain recording, transcription, consent, and retention for in-scope real-time voice, WebRTC, and synthetic media capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-MED-0001-C08 Synthetic Voice, Avatar, and Transformation Disclosure

The organization MUST define, implement, verify, and maintain synthetic voice, avatar, and transformation disclosure for in-scope real-time voice, WebRTC, and synthetic media capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-MED-0001 / REAL-TIME MEDIA

5. Normative Control System

Atomic criteria 09-12 of 18

MYTHOS-MED-0001-C09 Model, Voice, Likeness, and Rights Governance

The organization MUST define, implement, verify, and maintain model, voice, likeness, and rights governance for in-scope real-time voice, WebRTC, and synthetic media capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-MED-0001-C10 Content Credentials, C2PA, and Provenance Binding

The organization MUST define, implement, verify, and maintain content credentials, c2pa, and provenance binding for in-scope real-time voice, WebRTC, and synthetic media capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-MED-0001-C11 Deepfake, Impersonation, and Social-Engineering Defense

The organization MUST define, implement, verify, and maintain deepfake, impersonation, and social-engineering defense for in-scope real-time voice, WebRTC, and synthetic media capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-MED-0001-C12 Moderation, Abuse Reporting, and Rapid Revocation

The organization MUST define, implement, verify, and maintain moderation, abuse reporting, and rapid revocation for in-scope real-time voice, WebRTC, and synthetic media capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-MED-0001 / REAL-TIME MEDIA

5. Normative Control System

Atomic criteria 13-15 of 18

MYTHOS-MED-0001-C13 Latency, Jitter, Loss, Synchronization, and Quality

The organization MUST define, implement, verify, and maintain latency, jitter, loss, synchronization, and quality for in-scope real-time voice, WebRTC, and synthetic media capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-MED-0001-C14 Accessibility, Captioning, and Assistive Interaction

The organization MUST define, implement, verify, and maintain accessibility, captioning, and assistive interaction for in-scope real-time voice, WebRTC, and synthetic media capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-MED-0001-C15 Metadata, Biometric, and Sensitive Content Protection

The organization MUST define, implement, verify, and maintain metadata, biometric, and sensitive content protection for in-scope real-time voice, WebRTC, and synthetic media capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-MED-0001 / REAL-TIME MEDIA

5. Normative Control System

Atomic criteria 16-18 of 18

MYTHOS-MED-0001-C16 Service, Relay, Provider, and Network Failure

The organization MUST define, implement, verify, and maintain service, relay, provider, and network failure for in-scope real-time voice, WebRTC, and synthetic media capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-MED-0001-C17 Evidence Capture, Dispute, and Forensic Preservation

The organization MUST define, implement, verify, and maintain evidence capture, dispute, and forensic preservation for in-scope real-time voice, WebRTC, and synthetic media capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-MED-0001-C18 Media Lifecycle and Periodic Reassessment

The organization MUST define, implement, verify, and maintain media lifecycle and periodic reassessment for in-scope real-time voice, WebRTC, and synthetic media capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-MED-0001 / REAL-TIME MEDIA

6. Evidence and Assessment

Examine, interview, test, observe, restore, and trace

EXAMINE

Policies, inventories, diagrams, configuration, code, firmware, contracts, provider evidence, logs, exceptions, and lifecycle records.

INTERVIEW

Accountable owners, architects, engineers, operators, safety personnel, security teams, vendors, carriers, integrators, and responders.

TEST

Identity, segmentation, policy, negative paths, malformed input, capacity stress, dependency loss, failover, rollback, and revocation.

RESTORE

Independent restoration of configuration, data, service, device, controller, cluster, media, or browser state from protected evidence.

OBSERVE

Production-like performance, quality, drift, resource use, physical behavior, alarms, user impact, and incident execution.

TRACE

End-to-end linkage from approved intent and identity through change, runtime behavior, telemetry, outcome, exception, and review.

Evidence grades

A	B	C	D
Independently reproducible, complete, current, integrity-protected, and representative.	Reproduced by the implementing organization with strong traceability and controlled scope.	Partially reproduced or indirect; limitations, inherited responsibility, and uncertainty recorded.	Assertion, diagram, design intent, certificate, or vendor statement without reproduced behavior.

MYTHOS-MED-0001 / REAL-TIME MEDIA

7. Assurance Views

Six perspectives required for a complete assurance claim

GOVERNANCE

Ownership, purpose, risk tolerance, policy, exception, and decision rights.

ARCHITECTURE

Boundaries, dependencies, failure modes, state, data flow, and portability.

SECURITY

Identity, authorization, isolation, secrets, abuse resistance, and incident response.

OPERATIONS

Reliability, performance, cost, observability, change, recovery, and support.

PRIVACY / RIGHTS

Purpose limitation, minimization, consent, access, correction, deletion, and egress.

HUMAN / SOCIETAL

Usability, accessibility, disclosure, contestability, impact, and human control.

Conformance requires a defensible position across every applicable view. A strong aggregate score cannot compensate for an unowned system, a failed mandatory gate, untested recovery, or evidence below the accepted grade.

MYTHOS-MED-0001 / REAL-TIME MEDIA

7.2 Evaluation Metrics and Decision Gates

Measures must expose service behavior, control effectiveness, failure, uncertainty, and cost

SERVICE

Availability, correctness, coverage, quality, latency, throughput, and user or process outcome.

CONTROL

Unauthorized attempt, policy denial, drift, segmentation escape, privilege, and exception exposure.

RESILIENCE

Failover success, recovery time, data loss, safe degradation, restore validity, and dependency survival.

SECURITY

Exploitability, credential exposure, supply-chain integrity, attack detection, containment, and revocation.

CAPACITY

Utilization, saturation, queueing, radio or fabric headroom, thermal margin, quota, and forecast error.

OPERATIONS

Change failure, mean time to detect, repair, support burden, vendor escalation, and evidence completeness.

PRIVACY / SAFETY

Unauthorized egress, retention breach, consent coverage, unsafe action, physical consequence, and contestability.

LIFECYCLE

Patch debt, end-of-support exposure, portability, replacement time, retirement completion, and reassessment age.

Decision gate: thresholds MUST be declared before assessment, cover representative load and failure conditions, and identify mandatory safety, identity, recovery, and evidence failures that block promotion.

MYTHOS-MED-0001 / REAL-TIME MEDIA

8. Failure Modes and Adversarial Scenarios

Challenge assumptions before the system is trusted with consequential work

CONSENT FAILURE

Recording, transcription, synthesis, or training occurs without valid disclosure or permission.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

RELAY EXPOSURE

A media relay or provider can access content or metadata beyond the stated boundary.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

SYNTHETIC IMPERSONATION

Generated voice or likeness deceives participants or authorizes action.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

PROVENANCE STRIPPING

Content credentials are removed, invalid, or misunderstood as truth verification.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

RECORDING MISUSE

Captured media is retained, copied, or repurposed outside the declared purpose.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

QUALITY OR ACCESSIBILITY FAILURE

Latency, loss, captioning, or assistive interaction makes service unusable or unsafe.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

Adversarial testing MUST protect people and production systems. Use isolated environments, synthetic or minimized data, bounded authority, kill switches, explicit legal and ethical review, and documented stop conditions.

MYTHOS-MED-0001 / REAL-TIME MEDIA

9. Implementation Profiles

Risk-proportional implementation without weakening mandatory boundaries

FOUNDATIONAL

TYPICAL SCOPE

Low consequence, limited autonomy, non-sensitive data, reversible outputs.

MINIMUM DEPTH

Named owner; inventory; basic evaluation; access control; logging; rollback; annual review.

ADVANCED

TYPICAL SCOPE

Business-critical workflow, sensitive data, multiple tools or providers, moderate autonomy.

MINIMUM DEPTH

Formal threat model; representative evaluation; approval gates; isolated execution; tested recovery; quarterly review.

HIGH ASSURANCE

TYPICAL SCOPE

Safety, security, regulated, financial, identity, or broadly consequential use.

MINIMUM DEPTH

Independent challenge; strongest identity; deterministic enforcement; comprehensive evidence; canary release; continuous monitoring.

Profile selection rule

Choose the highest profile triggered by consequence, autonomy, sensitivity, scale, irreversibility, external dependency, or inability to rapidly detect and recover. Tailoring may strengthen controls; it may not erase a mandatory gate without a controlled exception.

MYTHOS-MED-0001 / REAL-TIME MEDIA

9.2 Operational Lifecycle

Evidence-bearing operation from concept through retirement

1. **CONCEPT**
   Purpose, owner, impact, boundaries
2. **DESIGN**
   Architecture, threat, data, identity
3. **BUILD**
   Controlled implementation and provenance
4. **EVALUATE**
   Representative, adversarial, recovery tests
5. **RELEASE**
   Approval, canary, rollback, evidence
6. **OPERATE**
   Observe, support, detect, respond
7. **CHANGE**
   Reassess, compare, gate, reverse
8. **RETIRE**
   Revoke, export, dispose, preserve

LIFECYCLE INVARIANT

Every stage MUST leave sufficient evidence for the next authority to understand what was intended, what changed, what was tested, what failed, what was accepted, what remains uncertain, and how to recover or exit.

MYTHOS-MED-0001 / REAL-TIME MEDIA

10. Adoption Roadmap

A sequenced path from uncontrolled capability to evidence-bearing operation

0-30 DAYS

Establish ownership and truth

Inventory systems and dependencies; classify risk; freeze unsupported claims; identify immediate privilege, data, and evidence gaps.

31-90 DAYS

Create enforceable boundaries

Define policy; isolate execution; implement identity and approval gates; establish representative evaluation and baseline telemetry.

3-6 MONTHS

Prove recovery and operating control

Exercise failure, rollback, revocation, incident, provider exit, and state-recovery scenarios; mature evidence packaging.

6-12 MONTHS

Scale assurance

Automate control checks; expand workload coverage; independent challenge; portfolio metrics; controlled exception expiry; continuous reassessment.

Adoption is complete only when operating evidence demonstrates the selected profile in the actual deployment context. Documentation alone is not conformance.

MYTHOS-MED-0001 / REAL-TIME MEDIA

10.2 Conformance and Exception Statement

What an assurance claim must contain

SYSTEM / SERVICE

Named, versioned capability and deployment boundary.

PROFILE

Foundational, Advanced, or High Assurance with trigger rationale.

SCOPE

Workloads, users, data, tools, regions, providers, and exclusions.

CRITERIA

Pass, partial, fail, not applicable, and exception status for every criterion.

EVIDENCE

Artifact references, evidence grade, date, environment, and reproducibility.

LIMITATIONS

Known failure modes, unsupported workloads, uncertainty, and residual risk.

EXCEPTIONS

Owner, rationale, compensating control, expiration, and approval.

VALIDITY

Assessment date, change boundary, review date, and reassessment triggers.

Prohibited claim: “compliant” without the named scope, profile, evidence date, limitations, exceptions, and authority accepting residual risk.

MYTHOS-MED-0001 / REAL-TIME MEDIA

11. Source Authority and Reference Register

Primary sources informing interpretation; current obligations and provider behavior must be verified at assessment t

NIST CSF 2.0 — **Cybersecurity Framework 2.0**

https://doi.org/10.6028/NIST.CSWP.29

Enterprise governance and cybersecurity outcome framework.

NIST SP 800-53 Rev. 5 — **Security and Privacy Controls for Information Systems and Organizations**

https://doi.org/10.6028/NIST.SP.800-53r5

Broad control baseline for organizational systems and services.

RFC 8826 — **Security Considerations for WebRTC**

https://www.rfc-editor.org/rfc/rfc8826

WebRTC threat model, identity, signaling, media, and browser security considerations.

W3C WebRTC — **WebRTC 1.0: Real-Time Communication Between Browsers**

https://www.w3.org/TR/webrtc/

Browser media and peer-connection API specification.

C2PA — **Coalition for Content Provenance and Authenticity Specification**

https://c2pa.org/specifications/specifications/

Content credentials, manifests, signing, validation, and provenance.

WCAG 2.2 — **Web Content Accessibility Guidelines 2.2**

https://www.w3.org/TR/WCAG22/

Accessibility requirements relevant to media, captions, input, and interaction.

Source currency rule: authorities, specifications, legal obligations, provider services, firmware, browser behavior, carrier requirements, and operational evidence MUST be checked at assessment time. This publication records a 2026-07-24 source snapshot.

MYTHOS-MED-0001 / REAL-TIME MEDIA

11.2 Revision History and Decision Register

Permanent identity, controlled change, and publication integrity

VERSION	DATE	STATE	SUMMARY
1.0.0-candidate	2026-07-24	Candidate	Permanent-publication edition; source refresh; expanded criteria, evidence, assurance, and lifecycle guidance.
Next review	2027-01-24	Scheduled	Review source currency, threat evolution, operating evidence, adoption feedback, and proposed revisions.

Publication decisions

PERMANENT IDENTITY The document ID remains stable across revisions; batch or production sequence metadata is not public identity.

CHANGE CONTROL Normative changes require rationale, impact analysis, affected-criterion mapping, and approval.

SUPERSESSION A superseded edition remains traceable; current routes and catalogs identify the active edition.

CORRECTION Material errors require a recorded correction, affected-evidence analysis, and notification appropriate to impact.

ARCHIVAL INTEGRITY Published artifacts receive hashes, metadata, and a release manifest sufficient for independent verification.

END OF PERMANENT PUBLICATION / MYTHOS-MED-0001

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY

ENGINEERING STANDARDS LIBRARY / WEB PLATFORMS AND BROWSER-LOCAL COMPUTE

WebGPU, WebLLM, and Browser Isolation Standard

Browser origin and dependency security, local models, WebGPU, WebAssembly, service workers, storage, permissions, fallback, and evidence.

PERMANENT DOCUMENT ID

MYTHOS-WEB-0001

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

WebGPU, WebLLM, and Browser Isolation Standard

DOCUMENT ID
MYTHOS-WEB-0001

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

18	6	4	1
ATOMIC CRITERIA	ASSURANCE VIEWS	IMPLEMENTATION PROFILES	PERMANENT IDENTITY

Publication doctrine. This publication establishes durable requirements for its defined scope. Interpretation must preserve the permanent document identity, required evidence, controlled exceptions, source currency, and formal revision history.

INFRASTRUCTURE AND CONTROL TOPOLOGY

Web Platforms and Browser-Local Compute

A trustworthy WebGPU, WebLLM, and browser-local compute system is a governed chain of ownership, identity, desired state, enforcement, workload or device behavior, telemetry, recovery, and independently reviewable evidence.

OWNERSHIP

Purpose, service boundary, accountable owner, suppliers, users, and retained responsibility remain explicit.

ENFORCEMENT

Identity, policy, segmentation, configuration, and local safety mechanisms constrain every trust transition.

CONTINUITY

Capacity, dependency loss, safe degradation, backup, restoration, rollback, and exit are tested before reliance.

EVIDENCE

Inventory, desired state, runtime observation, tests, incidents, exceptions, and change records form the assurance chain.

GOVERNED INFRASTRUCTURE FLOW

Reconciliation, detection, response, restoration, and controlled evolution

INTERPRETATION AND ASSURANCE

How to apply this publication

Normative intent

Requirements define outcomes and constraints. Implementations may vary, but evidence must demonstrate equivalent control.

Evidence over assertion

Vendor documentation and design intent are not equivalent to reproduced behavior. Record evidence grade and uncertainty.

Risk-proportional depth

Higher consequence, autonomy, sensitivity, and irreversibility require stronger isolation, review, verification, and recovery.

Controlled exception

Exceptions require an owner, rationale, compensating control, residual-risk acceptance, expiration, and reevaluation trigger.

Normalized assessment scale

0	1	2	3	4	5
ABSENT	AD HOC	PARTIAL	OPERATIONAL	ADVANCED	LEADING

A numeric score must never hide a failed mandatory gate, missing evidence, untested workload, or unacceptable residual risk.

INTERACTIVE NAVIGATION

Contents

Navigation doctrine

Bookmarks and internal links are conveniences. Permanent document identity, criterion identifiers, evidence references, and revision history remain authoritative.

MYTHOS-WEB-0001 / WEB AND BROWSER COMPUTE

0. Executive Mandate

Purpose, strategic outcome, and non-negotiable operating doctrine

MANDATE

Organizations adopting WebGPU, WebLLM, and browser-local compute capabilities MUST define an owned service boundary, establish enforceable identity and configuration, protect data and safety, test failure and recovery, and preserve evidence sufficient for independent review.

Required outcomes

- Create a durable governance boundary for WebGPU, WebLLM, and browser-local compute across design, acquisition, deployment, operation, change, and retirement.
- Require risk-proportional segmentation, identity, configuration, telemetry, resilience, recovery, and supplier controls.
- Prevent central automation, administrative tooling, or provider services from becoming unbounded authority.
- Prove operation with representative workloads, devices, failure modes, and restoration exercises.
- Define measurable conformance, exceptions, reassessment triggers, and a viable exit path.

Decision boundary: conformance supports a documented assurance claim for the named scope. It does not prove universal availability, safety, regulatory acceptance, vendor fitness, or immunity from cyber-physical failure.

MYTHOS-WEB-0001 / WEB AND BROWSER COMPUTE

1. Scope and Applicability

Boundary definition, audience, exclusions, and triggering conditions

IN SCOPE

- Architecture, acquisition, configuration, integration, and operation of WebGPU, WebLLM, and browser-local compute capabilities.
- Human, workload, device, service, network, data, facility, provider, supplier, and evidence dependencies.
- Design, deployment, monitoring, support, incident response, recovery, migration, and retirement.
- On-premises, hosted, managed, carrier, manufacturer, integrator, and externally operated components.

OUT OF SCOPE

- Claims of legal, safety, carrier, or regulatory approval without the responsible authority.
- Vendor certification treated as proof of the adopter configuration or operating effectiveness.
- Theoretical or laboratory behavior presented as production evidence without a declared boundary.
- Inherited controls without documented scope, owner, period, limitations, and shared responsibility.

Applicability triggers

- The capability supports a business-critical, safety-relevant, regulated, public, or externally dependent service.
- The environment can expose sensitive data, identity, control, physical process, communications, or shared infrastructure.
- A management plane, automation system, carrier, provider, or supplier can affect broad operational scope.
- Failure, compromise, or change can be difficult to detect, reverse, isolate, or recover.
- The system depends on hardware, firmware, radio, browser, cloud, endpoint, IoT, OT, or real-time media behavior.

MYTHOS-WEB-0001 / WEB AND BROWSER COMPUTE

2. Definitions and Taxonomy

Controlled language for architecture, governance, and assessment

AUTHORITY

The right to approve, deny, constrain, or execute an action. Model output is not authority.

EVIDENCE

A reproducible source, trace, measurement, test, record, signed artifact, or documented observation supporting a claim.

ASSURANCE VIEW

A defined perspective such as governance, architecture, security, operations, privacy, or human oversight.

ATOMIC CRITERION

A uniquely identified requirement that can be assessed, evidenced, excepted, and revised independently.

IMPLEMENTATION PROFILE

A risk-proportional set of required depth, evidence, and gates for a class of use.

CONTROLLED EXCEPTION

A time-bounded deviation with owner, rationale, compensating control, residual risk, expiration, and review trigger.

DETERMINISTIC MECHANISM

A component whose inputs, policy, state transition, and side effects can be constrained and verified.

SOURCE AUTHORITY

The documented basis for treating a source as reliable for a defined claim, context, jurisdiction, and time.

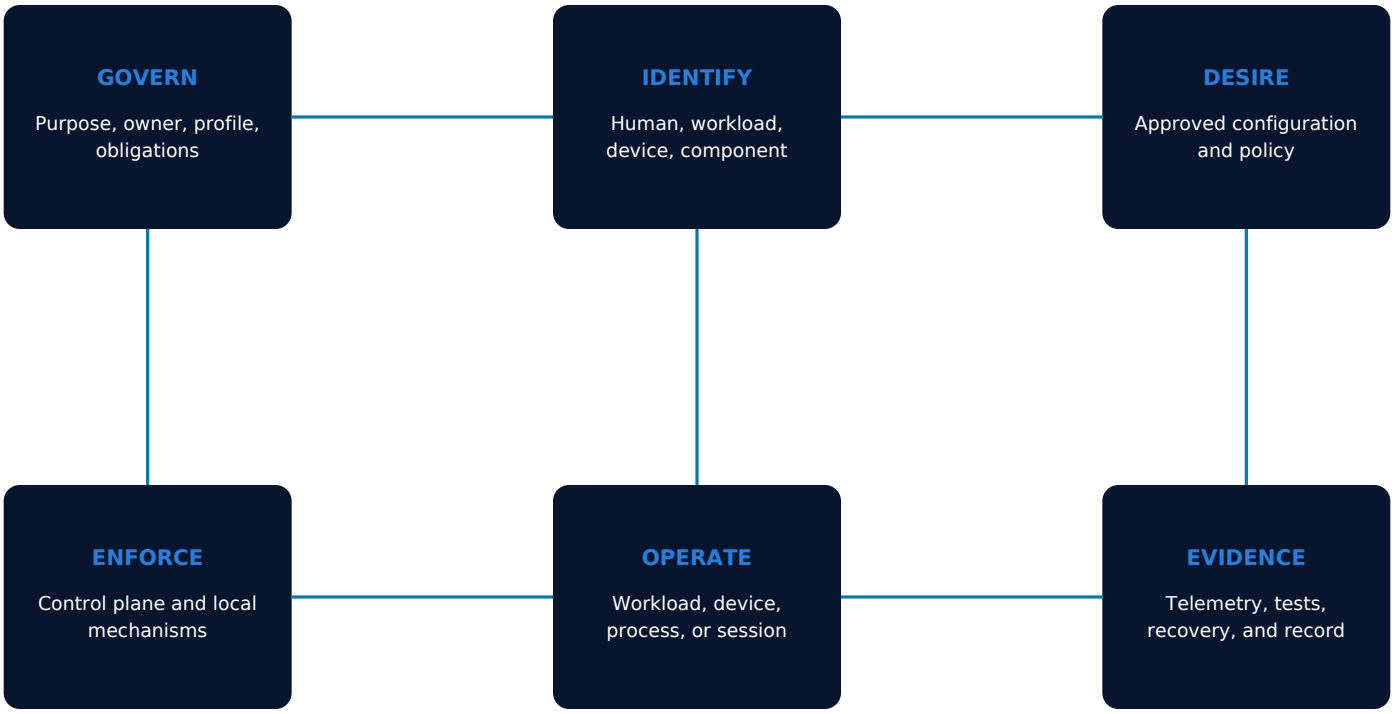
REASSESSMENT TRIGGER

A change, incident, drift signal, dependency revision, or time boundary requiring renewed evaluation.

MYTHOS-WEB-0001 / WEB AND BROWSER COMPUTE

3. Architecture and Trust Model

Separation of governance, management, enforcement, runtime, telemetry, and recovery



Mandatory trust boundaries

- Management planes MUST be isolated, strongly authenticated, monitored, and recoverable.
- Identity and policy MUST be evaluated at every provider, network, device, workload, and administrative transition.
- Runtime state MUST be compared with approved desired state and material drift MUST trigger response.
- Safety and continuity mechanisms MUST remain available when cloud, WAN, identity, DNS, time, carrier, or management dependencies fail.
- Evidence MUST be protected from the systems and administrators whose behavior it is intended to assess.

MYTHOS-WEB-0001 / WEB AND BROWSER COMPUTE

4. Governance and Accountability

Decision rights, separation of duties, and lifecycle ownership

ACCOUNTABLE OWNER

Accepts scope, risk tolerance, funding, and residual risk.

SYSTEM OWNER

Maintains architecture, inventory, operating boundaries, and change control.

SECURITY / PRIVACY

Defines threat, identity, data, isolation, monitoring, and incident requirements.

EVALUATION AUTHORITY

Designs representative tests, gates releases, and preserves reproducibility.

OPERATIONS

Maintains availability, rollback, recovery, evidence, and incident handling.

HUMAN OVERSIGHT

Reviews consequential decisions, exceptions, disputed outcomes, and harm signals.

DECISION PRINCIPLE

No single actor should be able to define the objective, grant authority, execute the consequential action, suppress evidence, and approve the result. Separation must be technical where consequence requires it.

MYTHOS-WEB-0001 / WEB AND BROWSER COMPUTE

5. Normative Control System

Atomic criteria 01-04 of 18

MYTHOS-WEB-0001-C01 Application Origin, Scope, and Trust Boundary

The organization MUST define, implement, verify, and maintain application origin, scope, and trust boundary for in-scope WebGPU, WebLLM, and browser-local compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-WEB-0001-C02 Content Security Policy and Script Execution Control

The organization MUST define, implement, verify, and maintain content security policy and script execution control for in-scope WebGPU, WebLLM, and browser-local compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-WEB-0001-C03 Dependency, Package, and Build Provenance

The organization MUST define, implement, verify, and maintain dependency, package, and build provenance for in-scope WebGPU, WebLLM, and browser-local compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-WEB-0001-C04 Subresource Integrity and Artifact Verification

The organization MUST define, implement, verify, and maintain subresource integrity and artifact verification for in-scope WebGPU, WebLLM, and browser-local compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-WEB-0001 / WEB AND BROWSER COMPUTE

5. Normative Control System

Atomic criteria 05-08 of 18

MYTHOS-WEB-0001-C05 Cross-Origin Isolation and Shared-Memory Preconditions

The organization MUST define, implement, verify, and maintain cross-origin isolation and shared-memory preconditions for in-scope WebGPU, WebLLM, and browser-local compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-WEB-0001-C06 WebAssembly Module and Native Interface Governance

The organization MUST define, implement, verify, and maintain webassembly module and native interface governance for in-scope WebGPU, WebLLM, and browser-local compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-WEB-0001-C07 WebGPU Adapter, Device, and Resource Authorization

The organization MUST define, implement, verify, and maintain webgpu adapter, device, and resource authorization for in-scope WebGPU, WebLLM, and browser-local compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-WEB-0001-C08 Model Artifact Integrity, Origin, and Version Control

The organization MUST define, implement, verify, and maintain model artifact integrity, origin, and version control for in-scope WebGPU, WebLLM, and browser-local compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-WEB-0001 / WEB AND BROWSER COMPUTE

5. Normative Control System

Atomic criteria 09-12 of 18

MYTHOS-WEB-0001-C09 Local Data, Cache, IndexedDB, and Retention

The organization MUST define, implement, verify, and maintain local data, cache, indexeddb, and retention for in-scope WebGPU, WebLLM, and browser-local compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-WEB-0001-C10 Service Worker Installation, Update, and Revocation

The organization MUST define, implement, verify, and maintain service worker installation, update, and revocation for in-scope WebGPU, WebLLM, and browser-local compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-WEB-0001-C11 Permission, Device, Media, and File Access

The organization MUST define, implement, verify, and maintain permission, device, media, and file access for in-scope WebGPU, WebLLM, and browser-local compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE desired-state configuration; policy evidence; privileged-access and negative tests

MYTHOS-WEB-0001-C12 Prompt, Retrieval, and Untrusted-Content Isolation

The organization MUST define, implement, verify, and maintain prompt, retrieval, and untrusted-content isolation for in-scope WebGPU, WebLLM, and browser-local compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE network, data-flow, zone, conduit, radio, device, or trust-boundary diagrams

MYTHOS-WEB-0001 / WEB AND BROWSER COMPUTE

5. Normative Control System

Atomic criteria 13-15 of 18

MYTHOS-WEB-0001-C13 GPU, CPU, Memory, Battery, and Thermal Budgets

The organization MUST define, implement, verify, and maintain gpu, cpu, memory, battery, and thermal budgets for in-scope WebGPU, WebLLM, and browser-local compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE runtime logs, traces, metrics, alarms, quality, capacity, and drift evidence

MYTHOS-WEB-0001-C14 Privacy, Telemetry, and Network-Egress Disclosure

The organization MUST define, implement, verify, and maintain privacy, telemetry, and network-egress disclosure for in-scope WebGPU, WebLLM, and browser-local compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE failure, failover, rollback, restoration, revocation, and recovery exercise

MYTHOS-WEB-0001-C15 Browser Compatibility and Security-Change Testing

The organization MUST define, implement, verify, and maintain browser compatibility and security-change testing for in-scope WebGPU, WebLLM, and browser-local compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE safety, privacy, residency, consent, or physical-impact assessment

MYTHOS-WEB-0001 / WEB AND BROWSER COMPUTE

5. Normative Control System

Atomic criteria 16-18 of 18

MYTHOS-WEB-0001-C16 Fallback, Degraded Mode, and Accessibility

The organization MUST define, implement, verify, and maintain fallback, degraded mode, and accessibility for in-scope WebGPU, WebLLM, and browser-local compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE supplier, carrier, firmware, license, support, portability, and exit evidence

MYTHOS-WEB-0001-C17 Incident Response, Cache Purge, and Key Revocation

The organization MUST define, implement, verify, and maintain incident response, cache purge, and key revocation for in-scope WebGPU, WebLLM, and browser-local compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE approved service contract; architecture decision; owner and risk sign-off

MYTHOS-WEB-0001-C18 Browser-Local Evidence and Periodic Reassessment

The organization MUST define, implement, verify, and maintain browser-local evidence and periodic reassessment for in-scope WebGPU, WebLLM, and browser-local compute capabilities. The control boundary MUST identify accountable ownership, affected services and assets, authoritative desired state, trust transitions, failure and recovery behavior, minimum evidence, inherited responsibility, and reassessment triggers.

MINIMUM EVIDENCE asset, service, dependency, supplier, region, and lifecycle inventory

MYTHOS-WEB-0001 / WEB AND BROWSER COMPUTE

6. Evidence and Assessment

Examine, interview, test, observe, restore, and trace

EXAMINE

Policies, inventories, diagrams, configuration, code, firmware, contracts, provider evidence, logs, exceptions, and lifecycle records.

INTERVIEW

Accountable owners, architects, engineers, operators, safety personnel, security teams, vendors, carriers, integrators, and responders.

TEST

Identity, segmentation, policy, negative paths, malformed input, capacity stress, dependency loss, failover, rollback, and revocation.

RESTORE

Independent restoration of configuration, data, service, device, controller, cluster, media, or browser state from protected evidence.

OBSERVE

Production-like performance, quality, drift, resource use, physical behavior, alarms, user impact, and incident execution.

TRACE

End-to-end linkage from approved intent and identity through change, runtime behavior, telemetry, outcome, exception, and review.

Evidence grades

A

Independently reproducible, complete, current, integrity-protected, and representative.

B

Reproduced by the implementing organization with strong traceability and controlled scope.

C

Partially reproduced or indirect; limitations, inherited responsibility, and uncertainty recorded.

D

Assertion, diagram, design intent, certificate, or vendor statement without reproduced behavior.

MYTHOS-WEB-0001 / WEB AND BROWSER COMPUTE

7. Assurance Views

Six perspectives required for a complete assurance claim

GOVERNANCE

Ownership, purpose, risk tolerance, policy, exception, and decision rights.

ARCHITECTURE

Boundaries, dependencies, failure modes, state, data flow, and portability.

SECURITY

Identity, authorization, isolation, secrets, abuse resistance, and incident response.

OPERATIONS

Reliability, performance, cost, observability, change, recovery, and support.

PRIVACY / RIGHTS

Purpose limitation, minimization, consent, access, correction, deletion, and egress.

HUMAN / SOCIETAL

Usability, accessibility, disclosure, contestability, impact, and human control.

Conformance requires a defensible position across every applicable view. A strong aggregate score cannot compensate for an unowned system, a failed mandatory gate, untested recovery, or evidence below the accepted grade.

MYTHOS-WEB-0001 / WEB AND BROWSER COMPUTE

7.2 Evaluation Metrics and Decision Gates

Measures must expose service behavior, control effectiveness, failure, uncertainty, and cost

SERVICE

Availability, correctness, coverage, quality, latency, throughput, and user or process outcome.

CONTROL

Unauthorized attempt, policy denial, drift, segmentation escape, privilege, and exception exposure.

RESILIENCE

Failover success, recovery time, data loss, safe degradation, restore validity, and dependency survival.

SECURITY

Exploitability, credential exposure, supply-chain integrity, attack detection, containment, and revocation.

CAPACITY

Utilization, saturation, queueing, radio or fabric headroom, thermal margin, quota, and forecast error.

OPERATIONS

Change failure, mean time to detect, repair, support burden, vendor escalation, and evidence completeness.

PRIVACY / SAFETY

Unauthorized egress, retention breach, consent coverage, unsafe action, physical consequence, and contestability.

LIFECYCLE

Patch debt, end-of-support exposure, portability, replacement time, retirement completion, and reassessment age.

Decision gate: thresholds MUST be declared before assessment, cover representative load and failure conditions, and identify mandatory safety, identity, recovery, and evidence failures that block promotion.

MYTHOS-WEB-0001 / WEB AND BROWSER COMPUTE

8. Failure Modes and Adversarial Scenarios

Challenge assumptions before the system is trusted with consequential work

ORIGIN ESCAPE

Untrusted content crosses the browser origin, worker, frame, or isolation boundary.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

DEPENDENCY COMPROMISE

A package, script, model, or CDN artifact is replaced or poisoned.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

SERVICE-WORKER PERSISTENCE

A malicious or stale worker continues controlling requests after remediation.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

MODEL TAMPERING

A local model or tokenizer changes without verified provenance.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

RESOURCE EXHAUSTION

WebGPU or local inference consumes excessive memory, power, thermals, or responsiveness.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

PRIVACY LEAKAGE

Prompts, files, telemetry, cache, or browser storage leave the declared local boundary.

REQUIRED: PREVENT | DETECT | CONTAIN | RECOVER | EVIDENCE

Adversarial testing MUST protect people and production systems. Use isolated environments, synthetic or minimized data, bounded authority, kill switches, explicit legal and ethical review, and documented stop conditions.

MYTHOS-WEB-0001 / WEB AND BROWSER COMPUTE

9. Implementation Profiles

Risk-proportional implementation without weakening mandatory boundaries

FOUNDATIONAL

TYPICAL SCOPE

Low consequence, limited autonomy, non-sensitive data, reversible outputs.

MINIMUM DEPTH

Named owner; inventory; basic evaluation; access control; logging; rollback; annual review.

ADVANCED

TYPICAL SCOPE

Business-critical workflow, sensitive data, multiple tools or providers, moderate autonomy.

MINIMUM DEPTH

Formal threat model; representative evaluation; approval gates; isolated execution; tested recovery; quarterly review.

HIGH ASSURANCE

TYPICAL SCOPE

Safety, security, regulated, financial, identity, or broadly consequential use.

MINIMUM DEPTH

Independent challenge; strongest identity; deterministic enforcement; comprehensive evidence; canary release; continuous monitoring.

Profile selection rule

Choose the highest profile triggered by consequence, autonomy, sensitivity, scale, irreversibility, external dependency, or inability to rapidly detect and recover. Tailoring may strengthen controls; it may not erase a mandatory gate without a controlled exception.

MYTHOS-WEB-0001 / WEB AND BROWSER COMPUTE

9.2 Operational Lifecycle

Evidence-bearing operation from concept through retirement

1. **CONCEPT**
   Purpose, owner, impact, boundaries
2. **DESIGN**
   Architecture, threat, data, identity
3. **BUILD**
   Controlled implementation and provenance
4. **EVALUATE**
   Representative, adversarial, recovery tests
5. **RELEASE**
   Approval, canary, rollback, evidence
6. **OPERATE**
   Observe, support, detect, respond
7. **CHANGE**
   Reassess, compare, gate, reverse
8. **RETIRE**
   Revoke, export, dispose, preserve

LIFECYCLE INVARIANT

Every stage MUST leave sufficient evidence for the next authority to understand what was intended, what changed, what was tested, what failed, what was accepted, what remains uncertain, and how to recover or exit.

MYTHOS-WEB-0001 / WEB AND BROWSER COMPUTE

10. Adoption Roadmap

A sequenced path from uncontrolled capability to evidence-bearing operation

0-30 DAYS

Establish ownership and truth

Inventory systems and dependencies; classify risk; freeze unsupported claims; identify immediate privilege, data, and evidence gaps.

31-90 DAYS

Create enforceable boundaries

Define policy; isolate execution; implement identity and approval gates; establish representative evaluation and baseline telemetry.

3-6 MONTHS

Prove recovery and operating control

Exercise failure, rollback, revocation, incident, provider exit, and state-recovery scenarios; mature evidence packaging.

6-12 MONTHS

Scale assurance

Automate control checks; expand workload coverage; independent challenge; portfolio metrics; controlled exception expiry; continuous reassessment.

Adoption is complete only when operating evidence demonstrates the selected profile in the actual deployment context. Documentation alone is not conformance.

MYTHOS-WEB-0001 / WEB AND BROWSER COMPUTE

10.2 Conformance and Exception Statement

What an assurance claim must contain

SYSTEM / SERVICE

Named, versioned capability and deployment boundary.

PROFILE

Foundational, Advanced, or High Assurance with trigger rationale.

SCOPE

Workloads, users, data, tools, regions, providers, and exclusions.

CRITERIA

Pass, partial, fail, not applicable, and exception status for every criterion.

EVIDENCE

Artifact references, evidence grade, date, environment, and reproducibility.

LIMITATIONS

Known failure modes, unsupported workloads, uncertainty, and residual risk.

EXCEPTIONS

Owner, rationale, compensating control, expiration, and approval.

VALIDITY

Assessment date, change boundary, review date, and reassessment triggers.

Prohibited claim: “compliant” without the named scope, profile, evidence date, limitations, exceptions, and authority accepting residual risk.

MYTHOS-WEB-0001 / WEB AND BROWSER COMPUTE

11. Source Authority and Reference Register

Primary sources informing interpretation; current obligations and provider behavior must be verified at assessment t

NIST CSF 2.0 — **Cybersecurity Framework 2.0**

https://doi.org/10.6028/NIST.CSWP.29

Enterprise governance and cybersecurity outcome framework.

NIST SP 800-53 Rev. 5 — **Security and Privacy Controls for Information Systems and Organizations**

https://doi.org/10.6028/NIST.SP.800-53r5

Broad control baseline for organizational systems and services.

W3C WebGPU — **WebGPU Specification**

https://www.w3.org/TR/webgpu/

Browser GPU programming, resource, security, and privacy semantics.

W3C WebAssembly — **WebAssembly Core Specification**

https://www.w3.org/TR/wasm-core-2/

Portable low-level execution model and validation semantics.

W3C CSP3 — **Content Security Policy Level 3**

https://www.w3.org/TR/CSP3/

Browser content-loading and script-execution policy.

OWASP ASVS — **Application Security Verification Standard**

https://owasp.org/www-project-application-security-verification-standard/

Web application security verification requirements and test structure.

Source currency rule: authorities, specifications, legal obligations, provider services, firmware, browser behavior, carrier requirements, and operational evidence MUST be checked at assessment time. This publication records a 2026-07-24 source snapshot.

MYTHOS-WEB-0001 / WEB AND BROWSER COMPUTE

11.2 Revision History and Decision Register

Permanent identity, controlled change, and publication integrity

VERSION	DATE	STATE	SUMMARY
1.0.0-candidate	2026-07-24	Candidate	Permanent-publication edition; source refresh; expanded criteria, evidence, assurance, and lifecycle guidance.
Next review	2027-01-24	Scheduled	Review source currency, threat evolution, operating evidence, adoption feedback, and proposed revisions.

Publication decisions

PERMANENT IDENTITY The document ID remains stable across revisions; batch or production sequence metadata is not public identity.

CHANGE CONTROL Normative changes require rationale, impact analysis, affected-criterion mapping, and approval.

SUPERSESSION A superseded edition remains traceable; current routes and catalogs identify the active edition.

CORRECTION Material errors require a recorded correction, affected-evidence analysis, and notification appropriate to impact.

ARCHIVAL INTEGRITY Published artifacts receive hashes, metadata, and a release manifest sufficient for independent verification.

END OF PERMANENT PUBLICATION / MYTHOS-WEB-0001

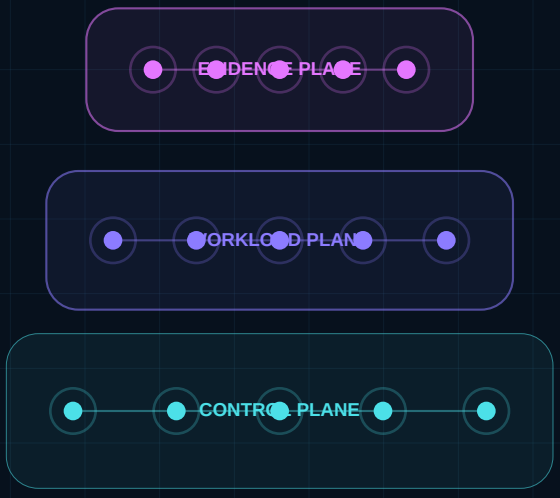
MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY

MYTHOS SYSTEMS

ENGINEERING GUIDE

Enterprise Cloud Compute and Hardware Engineering Guide

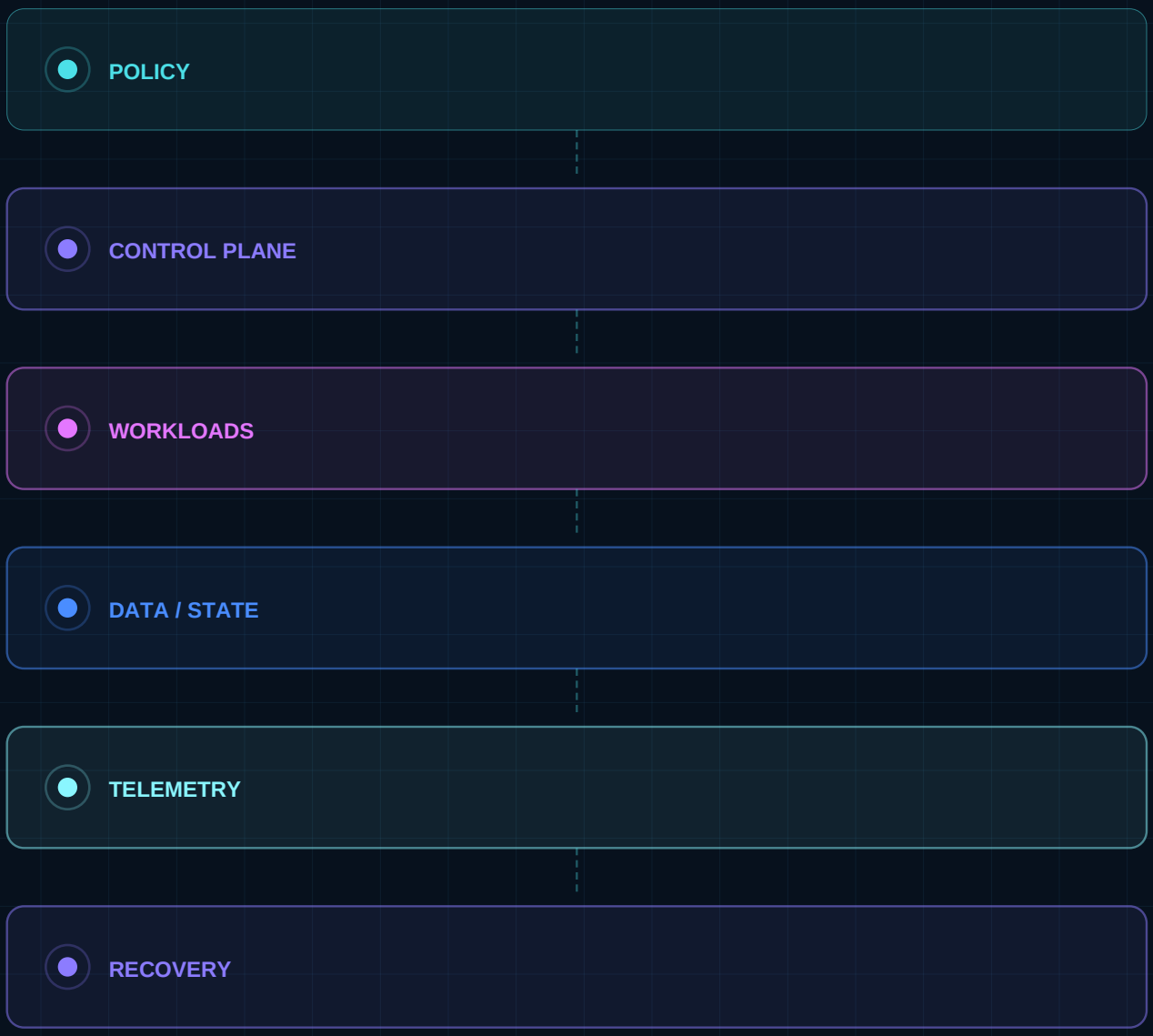
Cloud control planes, provider boundaries, compute fabrics, orchestration, trusted hardware, and evidence-bearing operations.



MYTHOS-GUIDE-CLD-0001

PERMANENT PUBLICATION IDENTITY / CANDIDATE PUBLICATION / PUBLIC

Operating architecture



Evidence-bearing implementation path

01 DEFINE SCOPE & AUTHORITY

02 MODEL STATE & BOUNDARIES

03 IMPLEMENT POLICY & AUTOMATION

04 OBSERVE REAL EFFECTS

05 TEST FAILURE & RECOVERY

06 RECONCILE EVIDENCE

The guide remains informative; conformance is established only by the applicable permanent standards and evidence.

INFRASTRUCTURE AND CONTROL TOPOLOGY

Cloud Architecture and Compute

A trustworthy cloud and compute engineering guide system is a governed chain of ownership, identity, desired state, enforcement, workload or device behavior, telemetry, recovery, and independently reviewable evidence.

OWNERSHIP

Purpose, service boundary, accountable owner, suppliers, users, and retained responsibility remain explicit.

ENFORCEMENT

Identity, policy, segmentation, configuration, and local safety mechanisms constrain every trust transition.

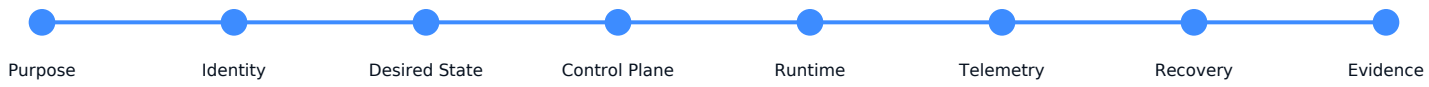
CONTINUITY

Capacity, dependency loss, safe degradation, backup, restoration, rollback, and exit are tested before reliance.

EVIDENCE

Inventory, desired state, runtime observation, tests, incidents, exceptions, and change records form the assurance chain.

GOVERNED INFRASTRUCTURE FLOW



Reconciliation, detection, response, restoration, and controlled evolution

INTERPRETATION AND ASSURANCE

How to apply this publication

Normative intent

Requirements define outcomes and constraints. Implementations may vary, but evidence must demonstrate equivalent control.

Evidence over assertion

Vendor documentation and design intent are not equivalent to reproduced behavior. Record evidence grade and uncertainty.

Risk-proportional depth

Higher consequence, autonomy, sensitivity, and irreversibility require stronger isolation, review, verification, and recovery.

Controlled exception

Exceptions require an owner, rationale, compensating control, residual-risk acceptance, expiration, and reevaluation trigger.

Normalized assessment scale

0	1	2	3	4	5
ABSENT	AD HOC	PARTIAL	OPERATIONAL	ADVANCED	LEADING

A numeric score must never hide a failed mandatory gate, missing evidence, untested workload, or unacceptable residual risk.

MYTHOS-GUIDE-CLD-0001 / CLOUD ARCHITECTURE

Engineering Doctrine

Engineering implementation guide for multi-cloud, provider-native, Kubernetes, serverless, private-cloud, DePIN, and

OPERATING POSITION

Define portfolio, provider, region, identity, data, network, logging, cost, recovery, and exit before onboarding workloads.

- Use declarative desired state, preventive policy, drift reconciliation, representative testing, and evidence-bearing operations.

MYTHOS-GUIDE-CLD-0001 / CLOUD ARCHITECTURE

Cloud Onboarding

OPERATING POSITION

Classify service and data.

- Select approved provider and region.
- Instantiate landing zone and federated identity.
- Deploy through controlled automation.
- Verify telemetry, recovery, cost, and exit assumptions.

MYTHOS-GUIDE-CLD-0001 / CLOUD ARCHITECTURE

Cloud-Native Operations

OPERATING POSITION

Govern APIs, admission, RBAC, image provenance, service identity, event semantics, retries, replay, quotas, and fleet upgrades.

MYTHOS-GUIDE-CLD-0001 / CLOUD ARCHITECTURE

Private and Accelerated Compute

OPERATING POSITION

Protect management planes, BMCs, fabrics, storage, schedulers, power, cooling, firmware, and recovery dependencies.

MYTHOS-GUIDE-CLD-0001 / CLOUD ARCHITECTURE

Assurance and Exit

OPERATING POSITION

Reproduce failure, restoration, region loss, provider substitution, data export, identity revocation, and evidence preservation.

MYTHOS-GUIDE-CLD-0001 / CLOUD ARCHITECTURE

Publication Controls and Decision Record

Permanent identity, evidence, exceptions, and revision discipline

- Document identity remains stable across revision and packaging.
- Evidence grades and uncertainty accompany consequential claims.
- Exceptions are time-bounded and independently visible.
- Source currency is reviewed at assessment and scheduled publication review.
- Release artifacts are hashed and listed in the public manifest.

MYTHOS-GUIDE-CLD-0001

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY

MYTHOS-GUIDE-CLD-0001 / CLOUD ARCHITECTURE

Publication Controls and Decision Record

Permanent identity, evidence, exceptions, and revision discipline

- Document identity remains stable across revision and packaging.
- Evidence grades and uncertainty accompany consequential claims.
- Exceptions are time-bounded and independently visible.
- Source currency is reviewed at assessment and scheduled publication review.
- Release artifacts are hashed and listed in the public manifest.

MYTHOS-GUIDE-CLD-0001

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY

MYTHOS-GUIDE-CLD-0001 / CLOUD ARCHITECTURE

Publication Controls and Decision Record

Permanent identity, evidence, exceptions, and revision discipline

- Document identity remains stable across revision and packaging.
- Evidence grades and uncertainty accompany consequential claims.
- Exceptions are time-bounded and independently visible.
- Source currency is reviewed at assessment and scheduled publication review.
- Release artifacts are hashed and listed in the public manifest.

MYTHOS-GUIDE-CLD-0001

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY

MYTHOS-GUIDE-CLD-0001 / CLOUD ARCHITECTURE

Publication Controls and Decision Record

Permanent identity, evidence, exceptions, and revision discipline

- Document identity remains stable across revision and packaging.
- Evidence grades and uncertainty accompany consequential claims.
- Exceptions are time-bounded and independently visible.
- Source currency is reviewed at assessment and scheduled publication review.
- Release artifacts are hashed and listed in the public manifest.

MYTHOS-GUIDE-CLD-0001

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY

MYTHOS-GUIDE-CLD-0001 / CLOUD ARCHITECTURE

11. Source Authority and Reference Register

Primary sources informing interpretation; current obligations and provider behavior must be verified at assessment t

NIST CSF 2.0	**Cybersecurity Framework 2.0**

https://doi.org/10.6028/NIST.CSWP.29

Enterprise governance and cybersecurity outcome framework.

NIST SP 800-53 Rev. 5	**Security and Privacy Controls for Information Systems and Organizations**

https://doi.org/10.6028/NIST.SP.800-53r5

Broad control baseline for organizational systems and services.

CSA CCM v4	**Cloud Controls Matrix**

https://cloudsecurityalliance.org/research/cloud-controls-matrix

Cloud control domains and shared-responsibility assessment structure.

FinOps Framework	**FinOps Framework**

https://www.finops.org/framework/

Cloud value, cost, usage, ownership, and operating-practice guidance.

NIST SP 800-204A	**Building Secure Microservices-based Applications Using Service-Mesh Architecture**

https://doi.org/10.6028/NIST.SP.800-204A

Microservice and service-mesh security architecture.

CNCF Cloud Native Security	**Cloud Native Security Whitepaper**

https://github.com/cncf/tag-security/tree/main/security-whitepaper

Cloud-native lifecycle, supply-chain, runtime, and operations guidance.

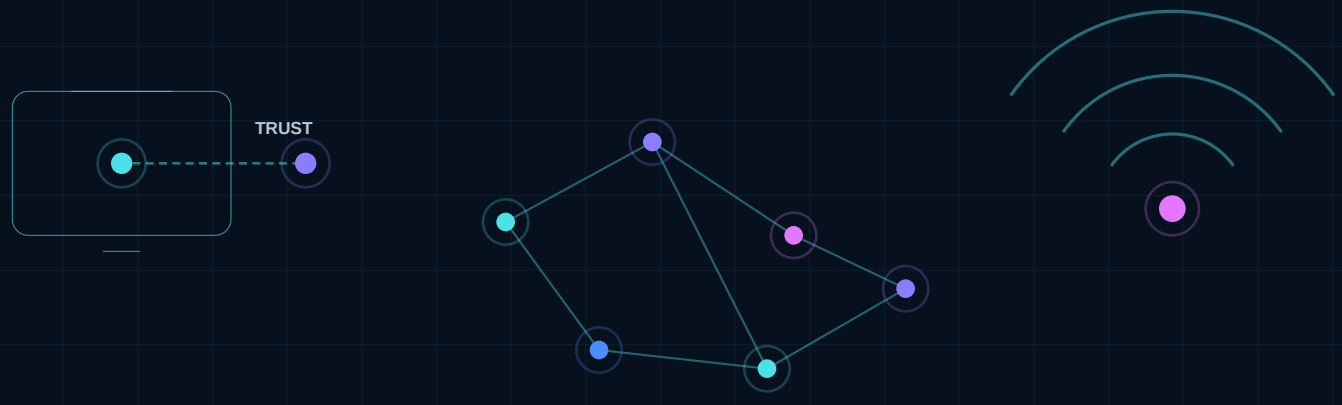
Source currency rule: authorities, specifications, legal obligations, provider services, firmware, browser behavior, carrier requirements, and operational evidence MUST be checked at assessment time. This publication records a 2026-07-24 source snapshot.

MYTHOS SYSTEMS

ENGINEERING GUIDE

Endpoint, Edge, OT, IoT, and Telecommunications Operations Guide

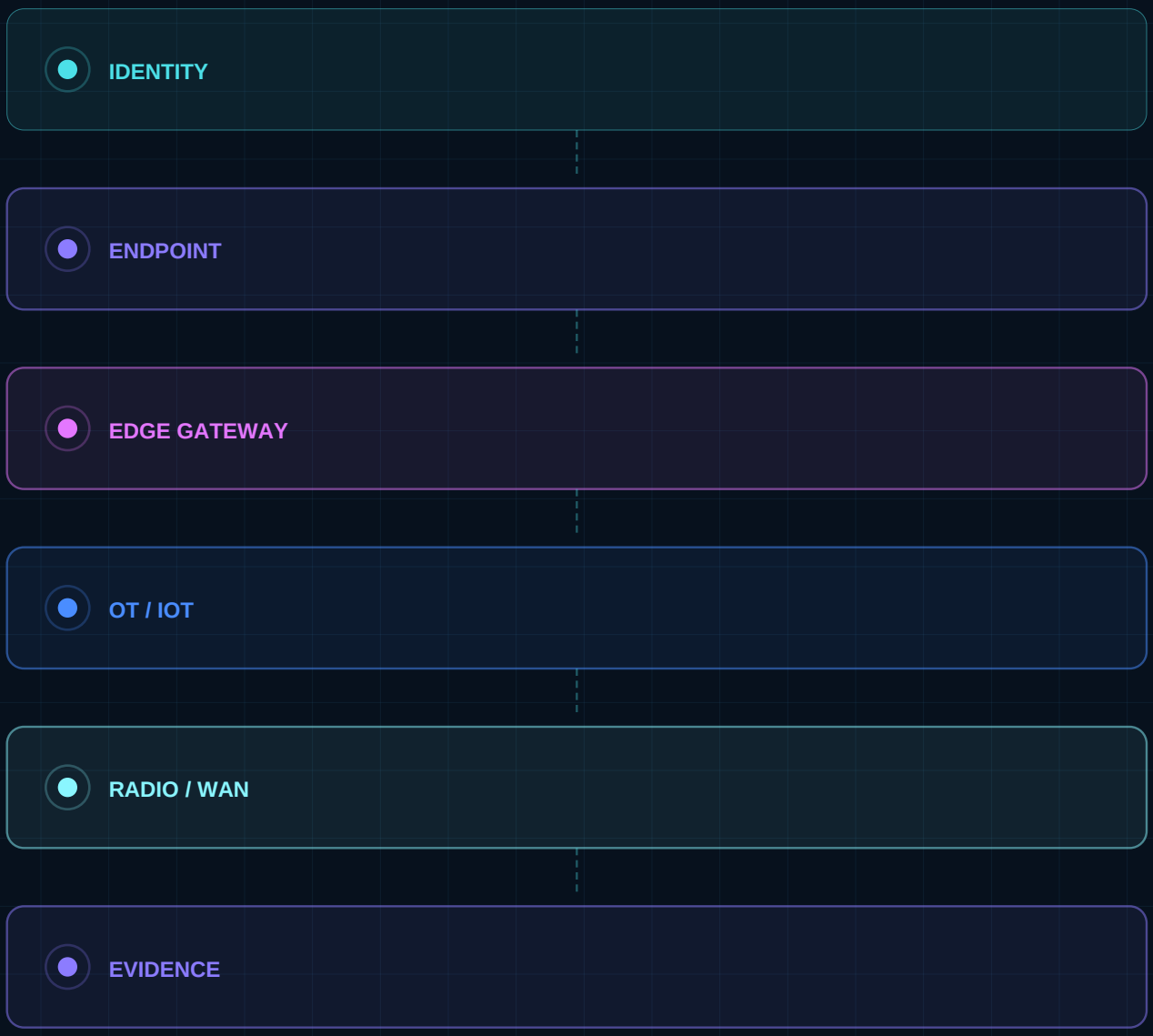
Operational integration across endpoint trust, industrial zones, edge fleets, sensors, radio systems, remote access, and lifecycle assurance.



MYTHOS-GUIDE-EDGE-0001

PERMANENT PUBLICATION IDENTITY / CANDIDATE PUBLICATION / PUBLIC

Operating architecture



Evidence-bearing implementation path

01 DEFINE SCOPE & AUTHORITY

02 MODEL STATE & BOUNDARIES

03 IMPLEMENT POLICY & AUTOMATION

04 OBSERVE REAL EFFECTS

05 TEST FAILURE & RECOVERY

06 RECONCILE EVIDENCE

The guide remains informative; conformance is established only by the applicable permanent standards and evidence.

INFRASTRUCTURE AND CONTROL TOPOLOGY

Operational Technology and Critical Infrastructure

A trustworthy endpoint, ot, telecom, and iot engineering guide system is a governed chain of ownership, identity, desired state, enforcement, workload or device behavior, telemetry, recovery, and independently reviewable evidence.

OWNERSHIP

Purpose, service boundary, accountable owner, suppliers, users, and retained responsibility remain explicit.

ENFORCEMENT

Identity, policy, segmentation, configuration, and local safety mechanisms constrain every trust transition.

CONTINUITY

Capacity, dependency loss, safe degradation, backup, restoration, rollback, and exit are tested before reliance.

EVIDENCE

Inventory, desired state, runtime observation, tests, incidents, exceptions, and change records form the assurance chain.

GOVERNED INFRASTRUCTURE FLOW

Reconciliation, detection, response, restoration, and controlled evolution

INTERPRETATION AND ASSURANCE

How to apply this publication

Normative intent

Requirements define outcomes and constraints. Implementations may vary, but evidence must demonstrate equivalent control.

Evidence over assertion

Vendor documentation and design intent are not equivalent to reproduced behavior. Record evidence grade and uncertainty.

Risk-proportional depth

Higher consequence, autonomy, sensitivity, and irreversibility require stronger isolation, review, verification, and recovery.

Controlled exception

Exceptions require an owner, rationale, compensating control, residual-risk acceptance, expiration, and reevaluation trigger.

Normalized assessment scale

0	1	2	3	4	5
ABSENT	AD HOC	PARTIAL	OPERATIONAL	ADVANCED	LEADING

A numeric score must never hide a failed mandatory gate, missing evidence, untested workload, or unacceptable residual risk.

MYTHOS-GUIDE-EDGE-0001 / OPERATIONAL TECHNOLOGY

Engineering Doctrine

Cross-domain guide for managed endpoints, cyber-physical systems, mobile and indoor RF, telephony, IoT, and disco

OPERATING POSITION

Start from mission, process, safety, communications, identity, and lifecycle consequences.

- Use local enforcement and safe degradation where cloud or management dependencies cannot be assumed.

MYTHOS-GUIDE-EDGE-0001 / OPERATIONAL TECHNOLOGY

Device Lifecycle

OPERATING POSITION

Bind procurement or manufacture, identity, commissioning, configuration, software, posture, data, support, transfer, reset, and retirement.

MYTHOS-GUIDE-EDGE-0001 / OPERATIONAL TECHNOLOGY

OT Change and Recovery

OPERATING POSITION

Use process ownership, hazard review, passive observation, versioned golden state, maintenance windows, rollback, and process validation.

MYTHOS-GUIDE-EDGE-0001 / OPERATIONAL TECHNOLOGY

Telecom and Radio

OPERATING POSITION

Correlate identity, signaling, user plane, radio, timing, QoS, emergency obligations, carriers, power, and service quality.

MYTHOS-GUIDE-EDGE-0001 / OPERATIONAL TECHNOLOGY

Field Evidence

OPERATING POSITION

Preserve as-built design, coverage, device, firmware, session, process, recovery, consent, safety, and authority records.

MYTHOS-GUIDE-EDGE-0001 / OPERATIONAL TECHNOLOGY

Publication Controls and Decision Record

Permanent identity, evidence, exceptions, and revision discipline

- Document identity remains stable across revision and packaging.
- Evidence grades and uncertainty accompany consequential claims.
- Exceptions are time-bounded and independently visible.
- Source currency is reviewed at assessment and scheduled publication review.
- Release artifacts are hashed and listed in the public manifest.

MYTHOS-GUIDE-EDGE-0001

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY

MYTHOS-GUIDE-EDGE-0001 / OPERATIONAL TECHNOLOGY

Publication Controls and Decision Record

Permanent identity, evidence, exceptions, and revision discipline

- Document identity remains stable across revision and packaging.
- Evidence grades and uncertainty accompany consequential claims.
- Exceptions are time-bounded and independently visible.
- Source currency is reviewed at assessment and scheduled publication review.
- Release artifacts are hashed and listed in the public manifest.

MYTHOS-GUIDE-EDGE-0001

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY

MYTHOS-GUIDE-EDGE-0001 / OPERATIONAL TECHNOLOGY

Publication Controls and Decision Record

Permanent identity, evidence, exceptions, and revision discipline

- Document identity remains stable across revision and packaging.
- Evidence grades and uncertainty accompany consequential claims.
- Exceptions are time-bounded and independently visible.
- Source currency is reviewed at assessment and scheduled publication review.
- Release artifacts are hashed and listed in the public manifest.

MYTHOS-GUIDE-EDGE-0001

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY

MYTHOS-GUIDE-EDGE-0001 / OPERATIONAL TECHNOLOGY

Publication Controls and Decision Record

Permanent identity, evidence, exceptions, and revision discipline

- Document identity remains stable across revision and packaging.
- Evidence grades and uncertainty accompany consequential claims.
- Exceptions are time-bounded and independently visible.
- Source currency is reviewed at assessment and scheduled publication review.
- Release artifacts are hashed and listed in the public manifest.

MYTHOS-GUIDE-EDGE-0001

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY

MYTHOS-GUIDE-EDGE-0001 / OPERATIONAL TECHNOLOGY

11. Source Authority and Reference Register

Primary sources informing interpretation; current obligations and provider behavior must be verified at assessment t

NIST CSF 2.0 — **Cybersecurity Framework 2.0**

https://doi.org/10.6028/NIST.CSWP.29

Enterprise governance and cybersecurity outcome framework.

NIST SP 800-53 Rev. 5 — **Security and Privacy Controls for Information Systems and Organizations**

https://doi.org/10.6028/NIST.SP.800-53r5

Broad control baseline for organizational systems and services.

NIST SP 800-82 Rev. 3 — **Guide to Operational Technology Security**

https://doi.org/10.6028/NIST.SP.800-82r3

OT architecture, threats, controls, safety context, and lifecycle guidance.

IEC 62443 Series — **Security for industrial automation and control systems**

https://www.iec.ch/industrial-cyber-security

Industrial security lifecycle, zones, conduits, and component requirements.

MITRE ATT&CK for ICS — **ATT&CK for Industrial Control Systems**

https://attack.mitre.org/matrices/ics/

Adversary behavior and technique knowledge base for ICS.

CISA ICS — **Industrial Control Systems Cybersecurity Guidance**

https://www.cisa.gov/topics/industrial-control-systems

Operational advisories, practices, and incident resources.

Source currency rule: authorities, specifications, legal obligations, provider services, firmware, browser behavior, carrier requirements, and operational evidence MUST be checked at assessment time. This publication records a 2026-07-24 source snapshot.

MYTHOS SYSTEMS

ENGINEERING GUIDE

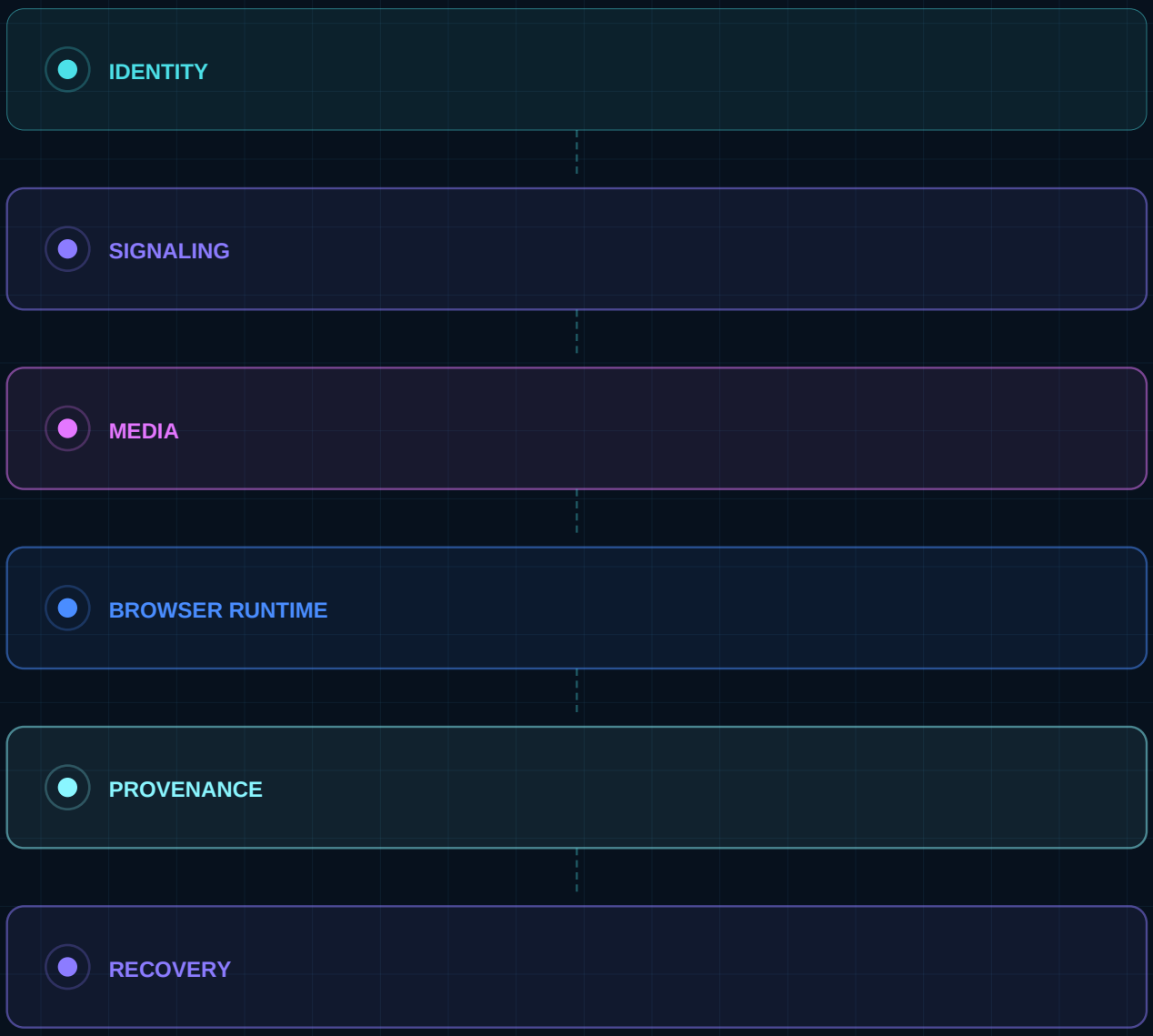
Realtime Media and Browser Compute Engineering Guide

Realtime signaling and media paths, browser/runtime isolation, WebGPU execution, provenance, privacy, quality, and recovery.

MYTHOS-GUIDE-MEDIAWEB-0001

PERMANENT PUBLICATION IDENTITY / CANDIDATE PUBLICATION / PUBLIC

Operating architecture



Evidence-bearing implementation path

01 DEFINE SCOPE & AUTHORITY

02 MODEL STATE & BOUNDARIES

03 IMPLEMENT POLICY & AUTOMATION

04 OBSERVE REAL EFFECTS

05 TEST FAILURE & RECOVERY

06 RECONCILE EVIDENCE

The guide remains informative; conformance is established only by the applicable permanent standards and evidence.

INFRASTRUCTURE AND CONTROL TOPOLOGY

Real-Time Media and Provenance

A trustworthy real-time media and browser compute engineering guide system is a governed chain of ownership, identity, desired state, enforcement, workload or device behavior, telemetry, recovery, and independently reviewable evidence.

OWNERSHIP

Purpose, service boundary, accountable owner, suppliers, users, and retained responsibility remain explicit.

ENFORCEMENT

Identity, policy, segmentation, configuration, and local safety mechanisms constrain every trust transition.

CONTINUITY

Capacity, dependency loss, safe degradation, backup, restoration, rollback, and exit are tested before reliance.

EVIDENCE

Inventory, desired state, runtime observation, tests, incidents, exceptions, and change records form the assurance chain.

GOVERNED INFRASTRUCTURE FLOW

Reconciliation, detection, response, restoration, and controlled evolution

INTERPRETATION AND ASSURANCE

How to apply this publication

Normative intent

Requirements define outcomes and constraints. Implementations may vary, but evidence must demonstrate equivalent control.

Evidence over assertion

Vendor documentation and design intent are not equivalent to reproduced behavior. Record evidence grade and uncertainty.

Risk-proportional depth

Higher consequence, autonomy, sensitivity, and irreversibility require stronger isolation, review, verification, and recovery.

Controlled exception

Exceptions require an owner, rationale, compensating control, residual-risk acceptance, expiration, and reevaluation trigger.

Normalized assessment scale

0	1	2	3	4	5
ABSENT	AD HOC	PARTIAL	OPERATIONAL	ADVANCED	LEADING

A numeric score must never hide a failed mandatory gate, missing evidence, untested workload, or unacceptable residual risk.

MYTHOS-GUIDE-MEDIAWEB-0001 / REAL-TIME MEDIA

Engineering Doctrine

Implementation guide for WebRTC, synthetic media, content provenance, WebGPU, WebAssembly, WebLLM, browser

OPERATING POSITION

Disclose who or what is participating, what is synthetic, what can access content, what is retained, and what leaves the device.

MYTHOS-GUIDE-MEDIAWEB-0001 / REAL-TIME MEDIA

Session Security

OPERATING POSITION

Protect signaling, session authorization, media keying, relay boundaries, recording, transcription, and user-visible permissions.

MYTHOS-GUIDE-MEDIAWEB-0001 / REAL-TIME MEDIA

Synthetic Media

OPERATING POSITION

Govern voice and likeness rights, persona identity, consent, content credentials, deepfake abuse, revocation, and dispute evidence.

MYTHOS-GUIDE-MEDIAWEB-0001 / REAL-TIME MEDIA

Browser Compute

OPERATING POSITION

Harden origin, CSP, dependencies, workers, storage, model integrity, WebAssembly, WebGPU, permissions, and resource budgets.

MYTHOS-GUIDE-MEDIAWEB-0001 / REAL-TIME MEDIA

Failure and Fallback

OPERATING POSITION

Define behavior when browser capability, GPU, relay, network, model, signer, identity, or storage is unavailable.

MYTHOS-GUIDE-MEDIAWEB-0001 / REAL-TIME MEDIA

Publication Controls and Decision Record

Permanent identity, evidence, exceptions, and revision discipline

- Document identity remains stable across revision and packaging.
- Evidence grades and uncertainty accompany consequential claims.
- Exceptions are time-bounded and independently visible.
- Source currency is reviewed at assessment and scheduled publication review.
- Release artifacts are hashed and listed in the public manifest.

MYTHOS-GUIDE-MEDIAWEB-0001

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY

MYTHOS-GUIDE-MEDIAWEB-0001 / REAL-TIME MEDIA

Publication Controls and Decision Record

Permanent identity, evidence, exceptions, and revision discipline

- Document identity remains stable across revision and packaging.
- Evidence grades and uncertainty accompany consequential claims.
- Exceptions are time-bounded and independently visible.
- Source currency is reviewed at assessment and scheduled publication review.
- Release artifacts are hashed and listed in the public manifest.

MYTHOS-GUIDE-MEDIAWEB-0001 / REAL-TIME MEDIA

Publication Controls and Decision Record

Permanent identity, evidence, exceptions, and revision discipline

- Document identity remains stable across revision and packaging.
- Evidence grades and uncertainty accompany consequential claims.
- Exceptions are time-bounded and independently visible.
- Source currency is reviewed at assessment and scheduled publication review.
- Release artifacts are hashed and listed in the public manifest.

MYTHOS-GUIDE-MEDIAWEB-0001 / REAL-TIME MEDIA

Publication Controls and Decision Record

Permanent identity, evidence, exceptions, and revision discipline

- Document identity remains stable across revision and packaging.
- Evidence grades and uncertainty accompany consequential claims.
- Exceptions are time-bounded and independently visible.
- Source currency is reviewed at assessment and scheduled publication review.
- Release artifacts are hashed and listed in the public manifest.

MYTHOS-GUIDE-MEDIAWEB-0001

MYTHOS SYSTEMS / ENGINEERING STANDARDS LIBRARY

MYTHOS-GUIDE-MEDIAWEB-0001 / REAL-TIME MEDIA

11. Source Authority and Reference Register

Primary sources informing interpretation; current obligations and provider behavior must be verified at assessment t

NIST CSF 2.0 — **Cybersecurity Framework 2.0**

https://doi.org/10.6028/NIST.CSWP.29

Enterprise governance and cybersecurity outcome framework.

NIST SP 800-53 Rev. 5 — **Security and Privacy Controls for Information Systems and Organizations**

https://doi.org/10.6028/NIST.SP.800-53r5

Broad control baseline for organizational systems and services.

RFC 8826 — **Security Considerations for WebRTC**

https://www.rfc-editor.org/rfc/rfc8826

WebRTC threat model, identity, signaling, media, and browser security considerations.

W3C WebRTC — **WebRTC 1.0: Real-Time Communication Between Browsers**

https://www.w3.org/TR/webrtc/

Browser media and peer-connection API specification.

C2PA — **Coalition for Content Provenance and Authenticity Specification**

https://c2pa.org/specifications/specifications/

Content credentials, manifests, signing, validation, and provenance.

WCAG 2.2 — **Web Content Accessibility Guidelines 2.2**

https://www.w3.org/TR/WCAG22/

Accessibility requirements relevant to media, captions, input, and interaction.

Source currency rule: authorities, specifications, legal obligations, provider services, firmware, browser behavior, carrier requirements, and operational evidence MUST be checked at assessment time. This publication records a 2026-07-24 source snapshot.

CANONICAL LIVING OVERVIEW GUIDE

Public, Private, Hybrid, and Sovereign Cloud Foundations

A cloud-foundations guide covering service and deployment models, landing zones, identity, networking, data, security, operations, resilience, hybrid integration, sovereignty, FinOps, portability, and cloud exit.

PERMANENT ID
MYTHOS-FG-CLD-0001

PUBLICATION CLASS
FIELD GUIDE

VERSION / STATUS
1.0.0-candidate / CANDIDATE

PERMANENT PUBLICATION IDENTITY

Public, Private, Hybrid, and Sovereign Cloud Foundations

Document ID
MYTHOS-FG-CLD-0001

Version
1.0.0-candidate

Status
Candidate Focused Field Guide

Review date
2026-10-26

Publication position
This is a living field guide. It is designed for frequent revision while retaining a stable permanent identity. Candidate status means the content is ready for structured use and review but is not represented as an external certification or authority publication.

LEVEL	CLASS	CADENCE	EVIDENCE
L1-L4	FIELD GUIDE	QUARTERLY	SOURCE-BOUND
Foundational through frontier	Practical and educational	Interim updates as needed	Limitations remain visible

Reader orientation

Dimension	Engineering position
Primary domain	MYTHOS-CLD - Cloud, Platform & Distributed Systems
Secondary domain	MYTHOS-SEC; MYTHOS-NET; MYTHOS-DAT
Audience	Cloud, infrastructure, security, network, platform, enterprise-architecture, and operations teams.
Prerequisites	Networking, identity, virtualization, distributed systems, security, storage, and operations.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

Distinguish public, private, hybrid, community, sovereign, edge, and disconnected cloud models.

Design landing zones with organizational hierarchy, identity, networking, policy, logging, keys, and ownership.

Integrate cloud and on-premises systems through explicit routing, identity, data, and operational boundaries.

Implement sovereignty through jurisdiction, control, egress, keys, personnel, supply chain, and survivability.

Measure reliability, security, cost, portability, lock-in, and exit readiness.

How to use this guide

Read the guide as a progressive system rather than a disconnected encyclopedia. Foundational material establishes the mental model; practitioner sections convert it into repeatable implementation and operations; advanced sections expose architecture, scale, security, and failure behavior; frontier sections describe technologies whose evidence or maturity is still evolving.

Suggested reading path
New practitioners should follow the chapters in order and complete the laboratories. Experienced engineers may begin with the architecture, failure, validation, or frontier sections, then use the related-document map to deepen a specific topic.

Learning and assurance model

Level	Reader outcome
L1 - Foundational	Terminology, first principles, component roles, packet or state flow, and simple working examples.
L2 - Practitioner	Implementation, configuration, automation, testing, troubleshooting, and operational evidence.
L3 - Advanced	Architecture, scale, concurrency, resilience, threat models, performance, and complex migrations.
L4 - Frontier	Emerging systems, research directions, technical uncertainty, readiness gates, and adoption signals.

Contents

CHAPTER 01

Cloud definitions, actors, and responsibility

Establish a precise cloud operating model and control boundary.

Chapter operating position
Establish a precise cloud operating model and control boundary.

1.1 Essential characteristics

Essential characteristics must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Apply on-demand service, network access, resource pooling, rapid elasticity, and measured service. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for essential characteristics.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating essential characteristics as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for essential characteristics.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

1.2 Service models

Service models must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Distinguish infrastructure, platform, software, functions, managed data, AI, and shared services. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for service models.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
landing_zone:
  organization: mythos
  account_class: production
  regions: [us-west-approved]
  identity:
    workforce_federation: required
    standing_admin: prohibited
  network:
    internet_egress: inspected-and-allowlisted
    private_service_access: required
  keys:
    customer_controlled: true
  sovereignty:
    data_location: enforced
    support_access: approved-and-recorded
  recovery:
    cross-region-test: quarterly
```

Failure patterns

Treating service models as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for service models.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

1.3 Deployment models

Deployment models must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Compare public, private, hybrid, community, sovereign, edge, and air-gapped environments. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for deployment models.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating deployment models as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for deployment models.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

1.4 Shared responsibility

Shared responsibility must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Assign provider, customer, platform, application, data, identity, and user duties by service. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for shared responsibility.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating shared responsibility as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for shared responsibility.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 01 laboratory and review

Practical laboratory

Map responsibility for one workload across IaaS, managed Kubernetes, serverless, and SaaS alternatives.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore composable sovereign clouds and user-owned compute fabrics.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 02

Organization, accounts, and landing zones

Create a governed foundation for teams and workloads.

Chapter operating position
Create a governed foundation for teams and workloads.

2.1 Hierarchy and tenancy

Hierarchy and tenancy must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Design organizations, folders, accounts, subscriptions, projects, resource groups, tenants, and environments. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for hierarchy and tenancy.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating hierarchy and tenancy as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for hierarchy and tenancy.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

2.2 Identity and administrative access

Identity and administrative access must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Federate workforce identity, use phishing-resistant authentication, JIT privilege, service identity, and break glass. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for identity and administrative access.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
landing_zone:
  organization: mythos
  account_class: production
  regions: [us-west-approved]
  identity:
    workforce_federation: required
    standing_admin: prohibited
  network:
    internet_egress: inspected-and-allowlisted
    private_service_access: required
  keys:
    customer_controlled: true
  sovereignty:
    data_location: enforced
    support_access: approved-and-recorded
  recovery:
    cross-region-test: quarterly
```

Failure patterns

Treating identity and administrative access as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for identity and administrative access.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

2.3 Guardrails and baselines

Guardrails and baselines must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Apply policy, logging, network, key, region, tag, budget, image, and deployment requirements. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for guardrails and baselines.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating guardrails and baselines as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for guardrails and baselines.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

2.4 Account vending and lifecycle

Account vending and lifecycle must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Automate request, approval, provisioning, ownership, updates, suspension, decommission, and evidence. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for account vending and lifecycle.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating account vending and lifecycle as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for account vending and lifecycle.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 02 laboratory and review

Practical laboratory
Design a landing-zone account vending workflow with policy, network, identity, logging, cost, and decommission gates.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore self-service landing zones generated by control planes and verified policy bundles.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 03

Cloud networking and connectivity

Build resilient paths among users, applications, services, regions, and sites.

Chapter operating position
Build resilient paths among users, applications, services, regions, and sites.

3.1 Virtual networks and routing

Virtual networks and routing must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Design address plans, subnets, route domains, gateways, peering, transit, segmentation, and DNS. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for virtual networks and routing.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating virtual networks and routing as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for virtual networks and routing.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

3.2 Hybrid and private connectivity

Hybrid and private connectivity must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use VPN, private circuits, SD-WAN, direct connectivity, encryption, routing policy, and failover. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for hybrid and private connectivity.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
landing_zone:
  organization: mythos
  account_class: production
  regions: [us-west-approved]
  identity:
    workforce_federation: required
    standing_admin: prohibited
  network:
    internet_egress: inspected-and-allowlisted
    private_service_access: required
  keys:
    customer_controlled: true
  sovereignty:
    data_location: enforced
    support_access: approved-and-recorded
  recovery:
    cross-region-test: quarterly
```

Failure patterns

- Treating hybrid and private connectivity as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for hybrid and private connectivity.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

3.3 Ingress, egress, and service access

Ingress, egress, and service access must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Control public endpoints, private links, proxies, NAT, firewalls, gateways, and data exfiltration. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for ingress, egress, and service access.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating ingress, egress, and service access as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for ingress, egress, and service access.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

3.4 Multi-region and edge networking

Multi-region and edge networking must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Plan latency, replication, global routing, failure domains, sovereignty, and observation. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for multi-region and edge networking.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating multi-region and edge networking as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for multi-region and edge networking.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 03 laboratory and review

Practical laboratory
Build a hybrid topology and test route loss, DNS failure, regional isolation, and unintended egress.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore policy-routed multi-cloud fabrics and photonic cloud interconnects.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 04

Cloud identity, security, keys, and data

Protect control planes, workloads, information, and trust anchors.

Chapter operating position
Protect control planes, workloads, information, and trust anchors.

4.1 Human and workload identity

Human and workload identity must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use federation, short-lived credentials, workload identities, roles, attributes, and continuous authorization. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for human and workload identity.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating human and workload identity as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for human and workload identity.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

4.2 Data classification and protection

Data classification and protection must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Apply encryption, tokenization, key ownership, access, replication, retention, backup, and deletion. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for data classification and protection.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
landing_zone:
  organization: mythos
  account_class: production
  regions: [us-west-approved]
  identity:
    workforce_federation: required
    standing_admin: prohibited
  network:
    internet_egress: inspected-and-allowlisted
    private_service_access: required
  keys:
    customer_controlled: true
  sovereignty:
    data_location: enforced
    support_access: approved-and-recorded
  recovery:
    cross-region-test: quarterly
```

Failure patterns

- Treating data classification and protection as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for data classification and protection.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

4.3 Security services and visibility

Security services and visibility must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Centralize configuration, vulnerability, threat, identity, network, key, audit, and incident evidence. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for security services and visibility.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating security services and visibility as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for security services and visibility.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

4.4 Secrets and cryptographic control

Secrets and cryptographic control must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use vaults, KMS, HSMs, customer-controlled keys, external keys, rotation, revocation, and recovery. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for secrets and cryptographic control.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating secrets and cryptographic control as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for secrets and cryptographic control.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 04 laboratory and review

Practical laboratory
Threat-model a cloud data service and prove who can access plaintext, keys, backups, logs, and support channels.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore confidential cloud, threshold key custody, and post-quantum cloud trust.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 05

Workload, platform, and data architecture

Select managed and self-managed services according to lifecycle and control.

Chapter operating position
Select managed and self-managed services according to lifecycle and control.

5.1 Compute choices

Compute choices must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Compare VMs, containers, Kubernetes, functions, edge, accelerators, and batch systems. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for compute choices.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating compute choices as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for compute choices.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

5.2 State and managed data

State and managed data must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Choose databases, object, block, file, streams, caches, warehouses, and recovery models. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for state and managed data.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
landing_zone:
  organization: mythos
  account_class: production
  regions: [us-west-approved]
  identity:
    workforce_federation: required
    standing_admin: prohibited
  network:
    internet_egress: inspected-and-allowlisted
    private_service_access: required
  keys:
    customer_controlled: true
  sovereignty:
    data_location: enforced
    support_access: approved-and-recorded
  recovery:
    cross-region-test: quarterly
```

Failure patterns

Treating state and managed data as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for state and managed data.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

5.3 Integration and messaging

Integration and messaging must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use APIs, queues, events, workflows, service mesh, data pipelines, and replication. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for integration and messaging.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating integration and messaging as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for integration and messaging.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

5.4 Portability and abstraction

Portability and abstraction must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Balance native services, open APIs, containers, infrastructure as code, and operational simplicity. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for portability and abstraction.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating portability and abstraction as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for portability and abstraction.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 05 laboratory and review

Practical laboratory

Create an architecture decision comparing cloud-native managed services with portable self-managed components.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore distributed sovereign data planes and workload mobility based on verifiable state.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 06

Operations, reliability, and disaster recovery

Operate cloud as a changing distributed system.

Chapter operating position
Operate cloud as a changing distributed system.

6.1 Observability and SLOs

Observability and SLOs must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Measure users, applications, platforms, networks, data, control planes, cost, and provider health. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for observability and slos.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating observability and slos as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for observability and slos.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

6.2 Change and configuration

Change and configuration must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use infrastructure as code, GitOps, policy, previews, canaries, drift reconciliation, and rollback. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for change and configuration.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
landing_zone:
  organization: mythos
  account_class: production
  regions: [us-west-approved]
  identity:
    workforce_federation: required
    standing_admin: prohibited
  network:
    internet_egress: inspected-and-allowlisted
    private_service_access: required
  keys:
    customer_controlled: true
  sovereignty:
    data_location: enforced
    support_access: approved-and-recorded
  recovery:
    cross-region-test: quarterly
```

Failure patterns

Treating change and configuration as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for change and configuration.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

6.3 Resilience and failure testing

Resilience and failure testing must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Test zone, region, identity, network, DNS, key, provider API, quota, and dependency failures. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for resilience and failure testing.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating resilience and failure testing as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for resilience and failure testing.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

6.4 Backup, restore, and disaster recovery

Backup, restore, and disaster recovery must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Define RPO, RTO, immutable copies, cross-region or cross-provider recovery, runbooks, and evidence. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for backup, restore, and disaster recovery.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating backup, restore, and disaster recovery as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for backup, restore, and disaster recovery.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 06 laboratory and review

Practical laboratory
Run a cloud resilience exercise involving identity outage, regional failure, key-service loss, and quota exhaustion.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore autonomous cloud recovery under external safety and budget controls.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 07

Sovereignty, jurisdiction, and disconnected operation

Engineer meaningful control rather than using sovereignty as a location label.

Chapter operating position
Engineer meaningful control rather than using sovereignty as a location label.

7.1 Jurisdiction and legal control

Jurisdiction and legal control must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Assess data, metadata, operators, support, subpoenas, contracts, ownership, and applicable law. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for jurisdiction and legal control.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating jurisdiction and legal control as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for jurisdiction and legal control.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

7.2 Technical sovereignty

Technical sovereignty must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Control identities, keys, images, source, telemetry, egress, management planes, updates, and suppliers. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for technical sovereignty.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
landing_zone:
  organization: mythos
  account_class: production
  regions: [us-west-approved]
  identity:
    workforce_federation: required
    standing_admin: prohibited
  network:
    internet_egress: inspected-and-allowlisted
    private_service_access: required
  keys:
    customer_controlled: true
  sovereignty:
    data_location: enforced
    support_access: approved-and-recorded
  recovery:
    cross-region-test: quarterly
```

Failure patterns

- Treating technical sovereignty as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for technical sovereignty.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

7.3 Operational sovereignty

Operational sovereignty must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Develop local skills, support, spare capacity, runbooks, incident authority, and continuity. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for operational sovereignty.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating operational sovereignty as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for operational sovereignty.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

7.4 Disconnected and degraded modes

Disconnected and degraded modes must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Operate during provider, network, federation, licensing, or update isolation and reconcile later. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for disconnected and degraded modes.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating disconnected and degraded modes as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for disconnected and degraded modes.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 07 laboratory and review

Practical laboratory
Score a proposed sovereign cloud against jurisdictional, technical, operational, and supply-chain criteria.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore nationally federated cloud regions and delay-tolerant sovereign services.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 08

Economics, governance, portability, and exit

Make cloud decisions over the complete lifecycle.

Chapter operating position
Make cloud decisions over the complete lifecycle.

8.1 Cost and unit economics

Cost and unit economics must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Measure compute, storage, data transfer, licenses, managed services, idle capacity, support, and labor. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for cost and unit economics.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating cost and unit economics as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for cost and unit economics.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

8.2 FinOps and budget controls

FinOps and budget controls must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Allocate ownership, forecasts, anomaly detection, commitments, quotas, showback, and product decisions. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for finops and budget controls.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
landing_zone:
  organization: mythos
  account_class: production
  regions: [us-west-approved]
  identity:
    workforce_federation: required
    standing_admin: prohibited
  network:
    internet_egress: inspected-and-allowlisted
    private_service_access: required
  keys:
    customer_controlled: true
  sovereignty:
    data_location: enforced
    support_access: approved-and-recorded
  recovery:
    cross-region-test: quarterly
```

Failure patterns

Treating finops and budget controls as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for finops and budget controls.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

8.3 Lock-in and concentration risk

Lock-in and concentration risk must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Assess APIs, data gravity, identities, skills, contracts, regions, suppliers, and correlated failure. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for lock-in and concentration risk.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating lock-in and concentration risk as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for lock-in and concentration risk.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

8.4 Exit and migration readiness

Exit and migration readiness must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Maintain inventories, portable data, replacement paths, tests, timelines, contracts, and practiced recovery. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the organization, identity, network, workload, data, policy, telemetry, resilience, sovereignty, and lifecycle chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for exit and migration readiness.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating exit and migration readiness as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for exit and migration readiness.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 08 laboratory and review

Practical laboratory

Create a five-year cloud decision including unit economics, concentration risk, sovereign requirements, and exit rehearsal.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore policy engines that optimize cloud placement across cost, sovereignty, carbon, resilience, and performance.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

Integrated learning roadmap

The guide is intended to be revisited. First establish fluency, then build a small implementation, operate it under failure, automate repeatable work, and finally evaluate emerging techniques against a stable baseline. Progress is demonstrated through evidence rather than reading completion alone.

Stage	Demonstrated capability
Foundation	Explain the system from first principles and trace its critical path without relying on product terminology.
Build	Implement the smallest representative system with explicit contracts, identities, telemetry, and tests.
Operate	Diagnose injected failures, recover safely, and preserve evidence of the causal chain.
Automate	Convert a proven manual workflow into bounded, idempotent, observable automation.
Architect	Design for scale, resilience, security, interoperability, and lifecycle change.
Explore	Evaluate frontier techniques using stated hypotheses, limitations, and adoption gates.

Source authority register

Source policy
Official standards, specifications, and project documentation remain with their issuing organizations. Mythos Systems provides original engineering interpretation, synthesis, implementation guidance, and relationship mapping. Source verification dates do not guarantee that an authority has not changed since review.

01. NIST - SP 800-145 - The NIST Definition of Cloud Computing
Role: Cloud service and deployment model definitions.
Verified: 2026-07-26
Official source: https://csrc.nist.gov/pubs/sp/800/145/final

02. NIST - SP 500-292 - NIST Cloud Computing Reference Architecture
Role: Cloud actors, activities, functions, and reference architecture.
Verified: 2026-07-26
Official source: https://www.nist.gov/publications/nist-cloud-computing-reference-architecture

03. Kubernetes - Kubernetes Documentation
Role: Official architecture, API, workload, scheduling, security, storage, networking, and operations documentation.
Verified: 2026-07-26
Official source: https://kubernetes.io/docs/home/

04. Crossplane - Crossplane Documentation
Role: Control-plane framework for custom APIs, composition, managed resources, and platform engineering.
Verified: 2026-07-26
Official source: https://docs.crossplane.io/latest/

05. Open Policy Agent - Open Policy Agent Documentation
Role: General-purpose policy engine and Rego policy language.
Verified: 2026-07-26
Official source: https://www.openpolicyagent.org/docs

06. OpenTelemetry - OpenTelemetry
Role: Vendor-neutral telemetry APIs, SDKs, collectors, traces, metrics, and logs.
Verified: 2026-07-26
Official source: https://opentelemetry.io/

07. SPIFFE - The SPIFFE Standard
Role: Workload identities, SVIDs, trust domains, federation, and Workload API.
Verified: 2026-07-26
Official source: https://spiffe.io/docs/latest/spiffe-specs/

Authority application and refresh triggers

Authority class	Required library action
Protocols and specifications	Recheck interoperability, security considerations, defaults, and migration guidance when a referenced specification changes.
Security and assurance frameworks	Update control mappings, evidence expectations, risk language, and assessment procedures after material framework revision.
Languages, runtimes, and projects	Re-run examples, tests, deprecation checks, and operational recommendations against supported releases.
Benchmarks and evaluations	Re-execute claims when workloads, methods, models, compilers, drivers, hardware, or scoring rules change.
Operational evidence	Incorporate validated incidents, failure tests, reader corrections, and measured implementation outcomes into the next revision.

Limitations, freshness, and revision control

Boundary	Statement
Publication limitation	This guide synthesizes broad engineering practice. It does not replace product documentation, legal advice, safety certification, or environment-specific design review.
Evidence limitation	Not every pattern is benchmarked in every environment. Examples illustrate engineering methods and must be validated against actual versions, workloads, hardware, policy, and failure conditions.
Freshness limitation	Vendor behavior, software libraries, AI models, security threats, and emerging standards can change quickly. Volatile claims require source revalidation before operational adoption.
Adoption limitation	Frontier content is not an implicit adoption recommendation. Adoption requires defined value, qualification gates, controlled proof, rollback, ownership, and residual-risk acceptance.
Status boundary	Candidate Focused Field Guide describes the publication, not certification of a product, implementation, engineer, or organization.

Revision record

Version	Revision statement
1.0.0-candidate	Initial candidate living-guide edition. Published 2026-07-26; scheduled review 2026-10-26.

Search and relationship metadata

Dimension	Engineering position
Primary domain	MYTHOS-CLD - Cloud, Platform & Distributed Systems
Secondary domain	MYTHOS-SEC; MYTHOS-NET; MYTHOS-DAT
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	cloud foundations, public cloud, private cloud, hybrid cloud, sovereign cloud, landing zone, cloud networking, cloud security, resilience, FinOps
Canonical route	/library/field-guides/mythos-fg-cld-0001/

Living publication

Corrections, expanded laboratories, improved diagrams, authority changes, and new engineering evidence should revise this permanent publication rather than create disconnected replacements.

CANONICAL LIVING OVERVIEW GUIDE

Kubernetes, Controllers, Operators, and Cluster Engineering

A complete Kubernetes engineering guide covering API machinery, desired state, controllers, CRDs, operators, scheduling, nodes, networking, storage, security, upgrades, multi-cluster operations, batch and AI workloads, and cluster assurance.

PERMANENT ID
MYTHOS-FG-CLD-0002

PUBLICATION CLASS
FIELD GUIDE

VERSION / STATUS
1.0.0-candidate / CANDIDATE

PERMANENT PUBLICATION IDENTITY

Kubernetes, Controllers, Operators, and Cluster Engineering

Document ID
MYTHOS-FG-CLD-0002

Version
1.0.0-candidate

Status
Candidate Focused Field Guide

Review date
2026-10-26

Publication position
This is a living field guide. It is designed for frequent revision while retaining a stable permanent identity. Candidate status means the content is ready for structured use and review but is not represented as an external certification or authority publication.

LEVEL	CLASS	CADENCE	EVIDENCE
L1-L4	FIELD GUIDE	QUARTERLY	SOURCE-BOUND
Foundational through frontier	Practical and educational	Interim updates as needed	Limitations remain visible

Reader orientation

Dimension	Engineering position
Primary domain	MYTHOS-CLD - Cloud, Platform & Distributed Systems
Secondary domain	MYTHOS-SWE; MYTHOS-SEC; MYTHOS-DAT
Audience	Kubernetes, platform, cloud, SRE, software, security, and infrastructure engineers.
Prerequisites	Containers, Linux, networking, distributed systems, APIs, storage, identity, and Go fundamentals.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

Explain Kubernetes from API request and etcd state through reconciliation, scheduling, nodes, and workload outcome.

Build controllers and operators with correct ownership, idempotency, concurrency, status, finalizers, and recovery.

Engineer clusters for networking, storage, security, observability, upgrades, and disaster recovery.

Schedule services, batch, GPU, and AI workloads with quotas, topology, fairness, and failure isolation.

Validate desired state against applied state, workload behavior, SLOs, and recoverability.

How to use this guide

Read the guide as a progressive system rather than a disconnected encyclopedia. Foundational material establishes the mental model; practitioner sections convert it into repeatable implementation and operations; advanced sections expose architecture, scale, security, and failure behavior; frontier sections describe technologies whose evidence or maturity is still evolving.

Suggested reading path
New practitioners should follow the chapters in order and complete the laboratories. Experienced engineers may begin with the architecture, failure, validation, or frontier sections, then use the related-document map to deepen a specific topic.

Learning and assurance model

Level	Reader outcome
L1 - Foundational	Terminology, first principles, component roles, packet or state flow, and simple working examples.
L2 - Practitioner	Implementation, configuration, automation, testing, troubleshooting, and operational evidence.
L3 - Advanced	Architecture, scale, concurrency, resilience, threat models, performance, and complex migrations.
L4 - Frontier	Emerging systems, research directions, technical uncertainty, readiness gates, and adoption signals.

Contents

CHAPTER 01

Kubernetes architecture and API machinery

Trace desired state through the control plane and resource store.

Chapter operating position
Trace desired state through the control plane and resource store.

1.1 API server and resource model

API server and resource model must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Understand kinds, resources, namespaces, metadata, spec, status, subresources, validation, and discovery. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for api server and resource model.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating api server and resource model as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for api server and resource model.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

1.2 etcd and authoritative state

etcd and authoritative state must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use consistency, revisions, transactions, compaction, snapshots, quorum, and restore. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for etcd and authoritative state.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
apiVersion: platform.mythos.systems/v1
kind: AgentMission
metadata:
  name: mission-042
spec:
  workers: 16
  modelClasses: [local-fast, frontier-reasoner, verifier]
  budget:
    maxTokens: 2000000
    maxGpuHours: 8
  completion:
    requireIndependentVerification: true
status:
  observedGeneration: 4
  conditions:
    - type: Ready
      status: "False"
      reason: VerificationPending
```

Failure patterns

- Treating etcd and authoritative state as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for etcd and authoritative state.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

1.3 Watch, cache, and reconciliation

Watch, cache, and reconciliation must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Handle resource versions, informers, eventual caches, missed events, relist, and idempotency. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for watch, cache, and reconciliation.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating watch, cache, and reconciliation as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for watch, cache, and reconciliation.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

1.4 Admission and authorization

Admission and authorization must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Apply authentication, authorization, validation, mutation, policy, and audit before persistence. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for admission and authorization.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating admission and authorization as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for admission and authorization.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 01 laboratory and review

Practical laboratory

Trace a Deployment create request through admission, storage, controller reconciliation, scheduling, kubelet, and status.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore strongly typed control-plane APIs generated from domain specifications.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 02

Workloads, scheduling, and resource management

Place and operate pods according to resources, topology, and policy.

Chapter operating position
Place and operate pods according to resources, topology, and policy.

2.1 Pod and workload controllers

Pod and workload controllers must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use Deployments, StatefulSets, DaemonSets, Jobs, CronJobs, disruption, rollout, and ownership. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for pod and workload controllers.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating pod and workload controllers as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for pod and workload controllers.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

2.2 Scheduler pipeline

Scheduler pipeline must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Understand queue, filtering, scoring, binding, plugins, preemption, and profiles. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for scheduler pipeline.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
apiVersion: platform.mythos.systems/v1
kind: AgentMission
metadata:
  name: mission-042
spec:
  workers: 16
  modelClasses: [local-fast, frontier-reasoner, verifier]
  budget:
    maxTokens: 2000000
    maxGpuHours: 8
  completion:
    requireIndependentVerification: true
status:
  observedGeneration: 4
  conditions:
    - type: Ready
      status: "False"
      reason: VerificationPending
```

Failure patterns

- Treating scheduler pipeline as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for scheduler pipeline.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

2.3 Resources and QoS

Resources and QoS must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Set requests, limits, CPU, memory, huge pages, ephemeral storage, GPUs, QoS, and eviction. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for resources and qos.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating resources and qos as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for resources and qos.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

2.4 Topology and placement

Topology and placement must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use affinity, anti-affinity, topology spread, taints, tolerations, zones, nodes, and device locality. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for topology and placement.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating topology and placement as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for topology and placement.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 02 laboratory and review

Practical laboratory
Schedule mixed web, stateful, batch, and GPU workloads and test pressure, preemption, and zone failure.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore topology-aware AI workload placement and dynamic resource allocation.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 03

Controllers and reconciliation engineering

Build reliable feedback loops around desired and observed state.

Chapter operating position
Build reliable feedback loops around desired and observed state.

3.1 Reconcile contract

Reconcile contract must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Read current state, compute intent, apply minimal change, update status, and return retry or watch dependencies. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for reconcile contract.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating reconcile contract as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for reconcile contract.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

3.2 Idempotency and concurrency

Idempotency and concurrency must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use owner references, resource versions, patch, compare-and-swap, deduplication, and deterministic naming. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for idempotency and concurrency.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
apiVersion: platform.mythos.systems/v1
kind: AgentMission
metadata:
  name: mission-042
spec:
  workers: 16
  modelClasses: [local-fast, frontier-reasoner, verifier]
  budget:
    maxTokens: 2000000
    maxGpuHours: 8
  completion:
    requireIndependentVerification: true
status:
  observedGeneration: 4
  conditions:
    - type: Ready
      status: "False"
      reason: VerificationPending
```

Failure patterns

- Treating idempotency and concurrency as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for idempotency and concurrency.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

3.3 Finalizers and deletion

Finalizers and deletion must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Perform external cleanup, handle retries, timeouts, stuck resources, force removal, and evidence. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for finalizers and deletion.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating finalizers and deletion as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for finalizers and deletion.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

3.4 Conditions, status, and events

Conditions, status, and events must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Expose observed generation, readiness, progress, degradation, reason, message, and actionable diagnostics. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for conditions, status, and events.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating conditions, status, and events as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for conditions, status, and events.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 03 laboratory and review

Practical laboratory
Implement a small controller simulator and test duplicate events, conflicts, partial external effects, deletion, and recovery.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore formally verified reconciliation and controller synthesis.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 04

CRDs, operators, and platform APIs

Turn operational knowledge into safe declarative APIs.

Chapter operating position
Turn operational knowledge into safe declarative APIs.

4.1 API and schema design

API and schema design must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Define versioned spec, status, defaults, validation, immutability, list semantics, and compatibility. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for api and schema design.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating api and schema design as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for api and schema design.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

4.2 Operator capability levels

Operator capability levels must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Encode installation, configuration, upgrades, backup, failure recovery, and application-specific automation. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for operator capability levels.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
apiVersion: platform.mythos.systems/v1
kind: AgentMission
metadata:
  name: mission-042
spec:
  workers: 16
  modelClasses: [local-fast, frontier-reasoner, verifier]
  budget:
    maxTokens: 2000000
    maxGpuHours: 8
  completion:
    requireIndependentVerification: true
status:
  observedGeneration: 4
  conditions:
    - type: Ready
      status: "False"
      reason: VerificationPending
```

Failure patterns

- Treating operator capability levels as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for operator capability levels.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

4.3 Version conversion and migration

Version conversion and migration must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use storage versions, conversion, deprecation, data migration, rollback, and client compatibility. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for version conversion and migration.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating version conversion and migration as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for version conversion and migration.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

4.4 Packaging and lifecycle

Packaging and lifecycle must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Distribute controllers, CRDs, RBAC, webhooks, images, charts, operators, and release evidence. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for packaging and lifecycle.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating packaging and lifecycle as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for packaging and lifecycle.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 04 laboratory and review

Practical laboratory
Design an application CRD and operator that performs safe upgrades and backup with version migration.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore Crossplane-style compositions that combine applications, infrastructure, policy, and operations.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 05

Networking, service, ingress, and policy

Engineer communication from pod interfaces through service and gateway paths.

Chapter operating position
Engineer communication from pod interfaces through service and gateway paths.

5.1 CNI and pod networking

CNI and pod networking must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Understand IP allocation, routes, overlays, underlays, MTU, policy, eBPF, and network failure. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for cni and pod networking.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating cni and pod networking as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for cni and pod networking.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

5.2 Services and discovery

Services and discovery must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use ClusterIP, headless, load balancing, endpoint slices, DNS, session behavior, and topology. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for services and discovery.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
apiVersion: platform.mythos.systems/v1
kind: AgentMission
metadata:
  name: mission-042
spec:
  workers: 16
  modelClasses: [local-fast, frontier-reasoner, verifier]
  budget:
    maxTokens: 2000000
    maxGpuHours: 8
  completion:
    requireIndependentVerification: true
status:
  observedGeneration: 4
  conditions:
    - type: Ready
      status: "False"
      reason: VerificationPending
```

Failure patterns

- Treating services and discovery as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for services and discovery.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

5.3 Gateway API and ingress

Gateway API and ingress must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Separate infrastructure, application routing, policies, certificates, and implementation capabilities. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for gateway api and ingress.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating gateway api and ingress as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for gateway api and ingress.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

5.4 Network policy and service identity

Network policy and service identity must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Control ingress, egress, DNS, host paths, service mesh, mTLS, and workload authorization. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for network policy and service identity.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating network policy and service identity as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for network policy and service identity.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 05 laboratory and review

Practical laboratory
Build a service path using Gateway API and default-deny network policy, then troubleshoot DNS, MTU, and return-path faults.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore unified north-south and east-west policy through Gateway API GAMMA and ambient data planes.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 06

Storage, stateful systems, and data protection

Provide durable state through dynamic infrastructure and failure.

Chapter operating position
Provide durable state through dynamic infrastructure and failure.

6.1 Volumes and CSI

Volumes and CSI must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use classes, claims, binding, topology, snapshots, expansion, attachment, access modes, and reclaim. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for volumes and csi.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating volumes and csi as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for volumes and csi.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

6.2 Stateful applications

Stateful applications must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Design identity, ordering, replication, quorum, anti-affinity, backups, upgrades, and failover. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for stateful applications.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
apiVersion: platform.mythos.systems/v1
kind: AgentMission
metadata:
  name: mission-042
spec:
  workers: 16
  modelClasses: [local-fast, frontier-reasoner, verifier]
  budget:
    maxTokens: 2000000
    maxGpuHours: 8
  completion:
    requireIndependentVerification: true
status:
  observedGeneration: 4
  conditions:
    - type: Ready
      status: "False"
      reason: VerificationPending
```

Failure patterns

- Treating stateful applications as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for stateful applications.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

6.3 Data protection

Data protection must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Coordinate application consistency, snapshots, backups, object copies, encryption, restore, and validation. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for data protection.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating data protection as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for data protection.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

6.4 Storage incidents and recovery

Storage incidents and recovery must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Handle lost nodes, attach conflicts, corrupt volumes, full disks, zone failure, and control-plane restore. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for storage incidents and recovery.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating storage incidents and recovery as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for storage incidents and recovery.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 06 laboratory and review

Practical laboratory
Deploy a stateful service, take an application-consistent backup, fail a node and zone, and restore to a new cluster.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore disaggregated storage and verifiable state migration.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 07

Cluster security, observability, and operations

Protect and operate the cluster as critical infrastructure.

Chapter operating position
Protect and operate the cluster as critical infrastructure.

7.1 Identity, RBAC, and workload security

Identity, RBAC, and workload security must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use least privilege, service accounts, pod security, secrets, admission, image verification, and node controls. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for identity, rbac, and workload security.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating identity, rbac, and workload security as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for identity, rbac, and workload security.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

7.2 Telemetry and diagnostics

Telemetry and diagnostics must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Observe API, etcd, scheduler, controllers, nodes, runtimes, network, storage, workloads, and user outcomes. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for telemetry and diagnostics.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
apiVersion: platform.mythos.systems/v1
kind: AgentMission
metadata:
  name: mission-042
spec:
  workers: 16
  modelClasses: [local-fast, frontier-reasoner, verifier]
  budget:
    maxTokens: 2000000
    maxGpuHours: 8
  completion:
    requireIndependentVerification: true
status:
  observedGeneration: 4
  conditions:
    - type: Ready
      status: "False"
      reason: VerificationPending
```

Failure patterns

- Treating telemetry and diagnostics as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for telemetry and diagnostics.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

7.3 Upgrades and version skew

Upgrades and version skew must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Plan control-plane, node, add-on, CRD, API, runtime, network, storage, and workload compatibility. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for upgrades and version skew.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating upgrades and version skew as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for upgrades and version skew.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

7.4 Backup and disaster recovery

Backup and disaster recovery must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Protect etcd, manifests, certificates, external resources, images, data, policy, and runbooks. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for backup and disaster recovery.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating backup and disaster recovery as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for backup and disaster recovery.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 07 laboratory and review

Practical laboratory
Execute a cluster upgrade simulation and recover from an incompatible CRD, node failure, and etcd restore.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore immutable clusters and continuously verified upgrade twins.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 08

Multi-cluster, fleet, batch, and AI engineering

Scale beyond one cluster and one workload class.

Chapter operating position
Scale beyond one cluster and one workload class.

8.1 Fleet architecture

Fleet architecture must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Manage clusters, regions, versions, policy, identity, networking, configuration, and lifecycle. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for fleet architecture.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating fleet architecture as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for fleet architecture.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

8.2 Multi-cluster service and data

Multi-cluster service and data must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Handle discovery, routing, failover, replication, consistency, sovereignty, and observability. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for multi-cluster service and data.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
apiVersion: platform.mythos.systems/v1
kind: AgentMission
metadata:
  name: mission-042
spec:
  workers: 16
  modelClasses: [local-fast, frontier-reasoner, verifier]
  budget:
    maxTokens: 2000000
    maxGpuHours: 8
  completion:
    requireIndependentVerification: true
status:
  observedGeneration: 4
  conditions:
    - type: Ready
      status: "False"
      reason: VerificationPending
```

Failure patterns

- Treating multi-cluster service and data as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for multi-cluster service and data.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

8.3 Batch and queueing

Batch and queueing must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use Jobs, priorities, Kueue admission, quotas, cohorts, preemption, and fair sharing. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for batch and queueing.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating batch and queueing as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for batch and queueing.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

8.4 AI and accelerator workloads

AI and accelerator workloads must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Schedule distributed training, inference, GPUs, topology, checkpoints, model data, and failure recovery. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the API request, admission, persistence, watch, reconciliation, scheduling, node execution, service, status, and recovery chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for ai and accelerator workloads.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating ai and accelerator workloads as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for ai and accelerator workloads.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 08 laboratory and review

Practical laboratory
Design a three-cluster fleet with service failover, policy, GPU cohorts, and sovereign workload placement.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore global workload schedulers combining Kubernetes, Kueue, agent fabrics, and heterogeneous accelerators.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

Integrated learning roadmap

The guide is intended to be revisited. First establish fluency, then build a small implementation, operate it under failure, automate repeatable work, and finally evaluate emerging techniques against a stable baseline. Progress is demonstrated through evidence rather than reading completion alone.

Stage	Demonstrated capability
Foundation	Explain the system from first principles and trace its critical path without relying on product terminology.
Build	Implement the smallest representative system with explicit contracts, identities, telemetry, and tests.
Operate	Diagnose injected failures, recover safely, and preserve evidence of the causal chain.
Automate	Convert a proven manual workflow into bounded, idempotent, observable automation.
Architect	Design for scale, resilience, security, interoperability, and lifecycle change.
Explore	Evaluate frontier techniques using stated hypotheses, limitations, and adoption gates.

Source authority register

Source policy
Official standards, specifications, and project documentation remain with their issuing organizations. Mythos Systems provides original engineering interpretation, synthesis, implementation guidance, and relationship mapping. Source verification dates do not guarantee that an authority has not changed since review.

01. Kubernetes - Kubernetes Releases
Role: Current supported-release and lifecycle information.
Verified: 2026-07-26
Official source: https://kubernetes.io/releases/

02. Kubernetes - Kubernetes Documentation
Role: Official architecture, API, workload, scheduling, security, storage, networking, and operations documentation.
Verified: 2026-07-26
Official source: https://kubernetes.io/docs/home/

03. Kubernetes - Kubernetes Controllers
Role: Official reconciliation and controller-loop model.
Verified: 2026-07-26
Official source: https://kubernetes.io/docs/concepts/architecture/controller/

04. Kubernetes - Kubernetes API Concepts
Role: API object, resource version, watch, patch, consistency, and concurrency semantics.
Verified: 2026-07-26
Official source: https://kubernetes.io/docs/reference/using-api/api-concepts/

05. Kubernetes SIGs - controller-runtime
Role: Libraries and patterns for building Kubernetes controllers.
Verified: 2026-07-26
Official source: https://pkg.go.dev/sigs.k8s.io/controller-runtime

06. Operator Framework - Operator SDK Documentation
Role: Operator construction, scaffolding, testing, and lifecycle guidance.
Verified: 2026-07-26
Official source: https://sdk.operatorframework.io/

07. etcd - etcd Documentation
Role: Distributed key-value state, consistency, watch, transactions, operations, and recovery.
Verified: 2026-07-26
Official source: https://etcd.io/docs/

08. Kubernetes - Kueue Documentation
Role: Kubernetes-native batch and AI workload queueing, admission, cohorts, and resource sharing.
Verified: 2026-07-26
Official source: https://kueue.sigs.k8s.io/

09. Kubernetes SIG Network - Gateway API
Role: Kubernetes-native role-oriented L4/L7 routing and service-mesh APIs.
Verified: 2026-07-26
Official source: https://gateway-api.sigs.k8s.io/docs/introduction/

Authority application and refresh triggers

Authority class	Required library action
Protocols and specifications	Recheck interoperability, security considerations, defaults, and migration guidance when a referenced specification changes.
Security and assurance frameworks	Update control mappings, evidence expectations, risk language, and assessment procedures after material framework revision.
Languages, runtimes, and projects	Re-run examples, tests, deprecation checks, and operational recommendations against supported releases.
Benchmarks and evaluations	Re-execute claims when workloads, methods, models, compilers, drivers, hardware, or scoring rules change.
Operational evidence	Incorporate validated incidents, failure tests, reader corrections, and measured implementation outcomes into the next revision.

Limitations, freshness, and revision control

Boundary	Statement
Publication limitation	This guide synthesizes broad engineering practice. It does not replace product documentation, legal advice, safety certification, or environment-specific design review.
Evidence limitation	Not every pattern is benchmarked in every environment. Examples illustrate engineering methods and must be validated against actual versions, workloads, hardware, policy, and failure conditions.
Freshness limitation	Vendor behavior, software libraries, AI models, security threats, and emerging standards can change quickly. Volatile claims require source revalidation before operational adoption.
Adoption limitation	Frontier content is not an implicit adoption recommendation. Adoption requires defined value, qualification gates, controlled proof, rollback, ownership, and residual-risk acceptance.
Status boundary	Candidate Focused Field Guide describes the publication, not certification of a product, implementation, engineer, or organization.

Revision record

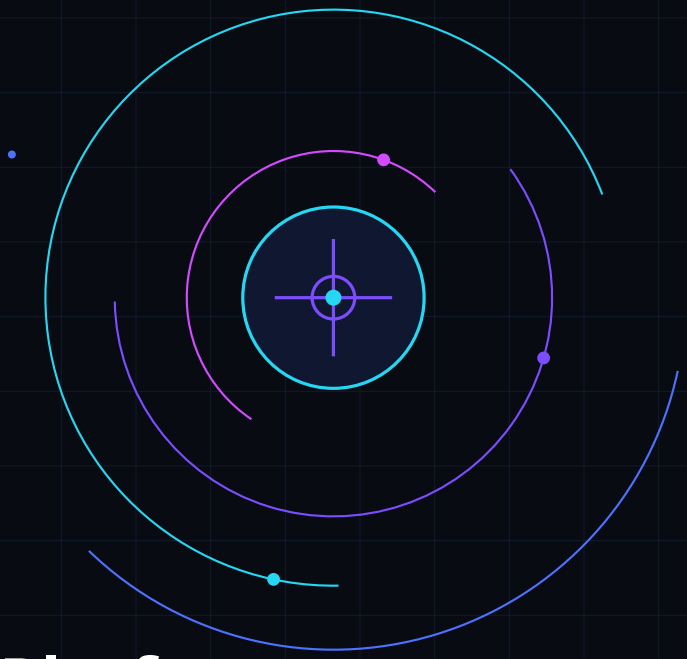
Version	Revision statement
1.0.0-candidate	Initial candidate living-guide edition. Published 2026-07-26; scheduled review 2026-10-26.

Search and relationship metadata

Dimension	Engineering position
Primary domain	MYTHOS-CLD - Cloud, Platform & Distributed Systems
Secondary domain	MYTHOS-SWE; MYTHOS-SEC; MYTHOS-DAT
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	Kubernetes, controllers, operators, CRD, scheduler, etcd, cluster engineering, Kueue, GPU workloads, multi-cluster
Canonical route	/library/field-guides/mythos-fg-cld-0002/

Living publication

Corrections, expanded laboratories, improved diagrams, authority changes, and new engineering evidence should revise this permanent publication rather than create disconnected replacements.



CANONICAL LIVING OVERVIEW GUIDE

Internal Developer Platforms, GitOps, and Policy as Code

A platform-engineering guide covering platform-as-product, developer portals, catalogs, golden paths, control planes, self-service APIs, Crossplane, GitOps, Argo CD, Flux, policy as code, OPA, Kyverno, provenance, observability, adoption, and governance.

PERMANENT ID
MYTHOS-FG-CLD-0003

PUBLICATION CLASS
FIELD GUIDE

VERSION / STATUS
1.0.0-candidate / CANDIDATE

PERMANENT PUBLICATION IDENTITY

Internal Developer Platforms, GitOps, and Policy as Code

Document ID
MYTHOS-FG-CLD-0003

Version
1.0.0-candidate

Status
Candidate Focused Field Guide

Review date
2026-10-26

Publication position
This is a living field guide. It is designed for frequent revision while retaining a stable permanent identity. Candidate status means the content is ready for structured use and review but is not represented as an external certification or authority publication.

LEVEL	CLASS	CADENCE	EVIDENCE
L1-L4	FIELD GUIDE	QUARTERLY	SOURCE-BOUND
Foundational through frontier	Practical and educational	Interim updates as needed	Limitations remain visible

Reader orientation

Dimension	Engineering position
Primary domain	MYTHOS-CLD - Cloud, Platform & Distributed Systems
Secondary domain	MYTHOS-SWE; MYTHOS-SEC; MYTHOS-DAT
Audience	Platform engineers, developer-experience teams, SREs, cloud teams, security teams, and engineering leaders.
Prerequisites	Software delivery, Kubernetes, cloud, APIs, identity, infrastructure as code, CI/CD, and security.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

Design internal platforms as products with defined users, outcomes, interfaces, ownership, and lifecycle.

Expose safe self-service through catalogs, templates, APIs, control planes, and golden paths.

Implement GitOps reconciliation with clear source authority, drift handling, promotions, and rollback.

Apply policy as code across source, CI, artifacts, infrastructure, admission, runtime, and exceptions.

Measure adoption, developer outcomes, reliability, security, cost, and platform effectiveness.

How to use this guide

Read the guide as a progressive system rather than a disconnected encyclopedia. Foundational material establishes the mental model; practitioner sections convert it into repeatable implementation and operations; advanced sections expose architecture, scale, security, and failure behavior; frontier sections describe technologies whose evidence or maturity is still evolving.

Suggested reading path
New practitioners should follow the chapters in order and complete the laboratories. Experienced engineers may begin with the architecture, failure, validation, or frontier sections, then use the related-document map to deepen a specific topic.

Learning and assurance model

Level	Reader outcome
L1 - Foundational	Terminology, first principles, component roles, packet or state flow, and simple working examples.
L2 - Practitioner	Implementation, configuration, automation, testing, troubleshooting, and operational evidence.
L3 - Advanced	Architecture, scale, concurrency, resilience, threat models, performance, and complex migrations.
L4 - Frontier	Emerging systems, research directions, technical uncertainty, readiness gates, and adoption signals.

Contents

CHAPTER 01

Platform product strategy and operating model

Define the platform's customers, promises, boundaries, and success measures.

Chapter operating position
Define the platform's customers, promises, boundaries, and success measures.

1.1 Users and journeys

Users and journeys must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Map developer, data, AI, operations, security, product, and support needs across delivery. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for users and journeys.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating users and journeys as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for users and journeys.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

1.2 Platform capabilities and contracts

Platform capabilities and contracts must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Define environments, runtime, data, identity, delivery, observability, security, and support services. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for platform capabilities and contracts.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
platform_request:
  api: ApplicationEnvironment/v2
  owner: team-payments
  environment: production
  composition:
    - kubernetes-namespace
    - workload-identity
    - database
    - gateway-route
    - telemetry
  policies:
    - signed-artifacts
    - restricted-pods
    - encrypted-data
  gitops:
    repository: environments-prod
    promotion: pull-request
  evidence:
    require: [policy-tests, drift-check, rollback-plan]
```

Failure patterns

- Treating platform capabilities and contracts as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for platform capabilities and contracts.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

1.3 Team topology and ownership

Team topology and ownership must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Assign platform product, engineering, SRE, security, enablement, service teams, and governance. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for team topology and ownership.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating team topology and ownership as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for team topology and ownership.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

1.4 Outcome and adoption metrics

Outcome and adoption metrics must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Measure lead time, cognitive load, reliability, security, support, self-service, and retention. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for outcome and adoption metrics.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating outcome and adoption metrics as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for outcome and adoption metrics.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 01 laboratory and review

Practical laboratory
Interview three developer archetypes and create a platform product charter with measurable outcomes.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore adaptive platforms that personalize interfaces without fragmenting control.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 02

Developer portal, catalog, and ownership graph

Make software, teams, dependencies, health, and actions discoverable.

Chapter operating position
Make software, teams, dependencies, health, and actions discoverable.

2.1 Catalog entities and metadata

Catalog entities and metadata must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Represent components, systems, APIs, resources, domains, users, groups, owners, lifecycle, and links. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for catalog entities and metadata.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating catalog entities and metadata as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for catalog entities and metadata.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

2.2 Source of truth and ingestion

Source of truth and ingestion must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use source-controlled metadata, providers, processors, validation, reconciliation, and conflict handling. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for source of truth and ingestion.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
platform_request:
  api: ApplicationEnvironment/v2
  owner: team-payments
  environment: production
  composition:
    - kubernetes-namespace
    - workload-identity
    - database
    - gateway-route
    - telemetry
  policies:
    - signed-artifacts
    - restricted-pods
    - encrypted-data
  gitops:
    repository: environments-prod
    promotion: pull-request
  evidence:
    require: [policy-tests, drift-check, rollback-plan]
```

Failure patterns

Treating source of truth and ingestion as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for source of truth and ingestion.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

2.3 Portal and documentation

Portal and documentation must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Provide discovery, TechDocs, scorecards, dashboards, runbooks, costs, incidents, and support. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection,

overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for portal and documentation.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating portal and documentation as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for portal and documentation.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

2.4 Ownership and dependency graph

Ownership and dependency graph must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Connect code, services, APIs, data, infrastructure, teams, SLOs, vulnerabilities, and changes. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for ownership and dependency graph.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating ownership and dependency graph as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for ownership and dependency graph.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 02 laboratory and review

Practical laboratory
Create a software catalog for one product and identify orphan services, undocumented APIs, and ownership conflicts.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore continuously generated architecture graphs from source, runtime, and cloud evidence.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 03

Golden paths, templates, and self-service

Turn organizational knowledge into safe repeatable workflows.

Chapter operating position
Turn organizational knowledge into safe repeatable workflows.

3.1 Golden path design

Golden path design must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Bundle architecture, repository, build, runtime, observability, security, docs, ownership, and lifecycle. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for golden path design.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating golden path design as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for golden path design.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

3.2 Software templates and scaffolding

Software templates and scaffolding must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Collect typed parameters, authorize actions, create repositories, generate code, and publish catalog entries. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for software templates and scaffolding.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
platform_request:
  api: ApplicationEnvironment/v2
  owner: team-payments
  environment: production
  composition:
    - kubernetes-namespace
    - workload-identity
    - database
    - gateway-route
    - telemetry
  policies:
    - signed-artifacts
    - restricted-pods
    - encrypted-data
  gitops:
    repository: environments-prod
    promotion: pull-request
  evidence:
    require: [policy-tests, drift-check, rollback-plan]
```

Failure patterns

- Treating software templates and scaffolding as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for software templates and scaffolding.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

3.3 Day-two actions

Day-two actions must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Support upgrades, scaling, database changes, certificate rotation, incident operations, and decommission. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for day-two actions.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating day-two actions as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for day-two actions.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

3.4 Exceptions and escape hatches

Exceptions and escape hatches must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Provide documented alternatives, review, compensating controls, expiry, and reintegration. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for exceptions and escape hatches.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating exceptions and escape hatches as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for exceptions and escape hatches.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 03 laboratory and review

Practical laboratory
Build a golden path for a service including repository, CI, deployment, policy, telemetry, catalog, and deletion.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore agent-generated golden paths whose outputs remain policy-verified and owner-approved.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 04

Control planes and platform APIs

Expose desired state instead of implementation-specific ticket queues.

Chapter operating position
Expose desired state instead of implementation-specific ticket queues.

4.1 Platform API design

Platform API design must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Define domain resources, schemas, defaults, composition, status, conditions, and lifecycle. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for platform api design.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating platform api design as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for platform api design.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

4.2 Crossplane composition

Crossplane composition must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Combine managed resources, Kubernetes resources, functions, packages, and operational workflows. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for crossplane composition.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
platform_request:
  api: ApplicationEnvironment/v2
  owner: team-payments
  environment: production
  composition:
    - kubernetes-namespace
    - workload-identity
    - database
    - gateway-route
    - telemetry
  policies:
    - signed-artifacts
    - restricted-pods
    - encrypted-data
  gitops:
    repository: environments-prod
    promotion: pull-request
  evidence:
    require: [policy-tests, drift-check, rollback-plan]
```

Failure patterns

- Treating crossplane composition as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for crossplane composition.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

4.3 Reconciliation and drift

Reconciliation and drift must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Continuously compare desired, composed, external, and observed state with safe correction. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for reconciliation and drift.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating reconciliation and drift as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for reconciliation and drift.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

4.4 Versioning and evolution

Versioning and evolution must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Migrate APIs, compositions, providers, resources, and consumers without uncontrolled breakage. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for versioning and evolution.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating versioning and evolution as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for versioning and evolution.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 04 laboratory and review

Practical laboratory

Design a platform API for an application environment and compose cloud, Kubernetes, data, identity, and policy resources.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore multi-control-plane fabrics spanning applications, agents, cloud, data, and sovereign infrastructure.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 05

GitOps architecture and promotion

Use declarative sources and reconciliation as the delivery control system.

Chapter operating position
Use declarative sources and reconciliation as the delivery control system.

5.1 Source authority and repository structure

Source authority and repository structure must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Define desired-state repositories, ownership, branches, environments, generators, secrets, and dependencies. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for source authority and repository structure.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating source authority and repository structure as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for source authority and repository structure.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

5.2 Reconciliation loops

Reconciliation loops must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Fetch, render, compare, apply, health-check, retry, suspend, prune, and report drift. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for reconciliation loops.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
platform_request:
  api: ApplicationEnvironment/v2
  owner: team-payments
  environment: production
  composition:
    - kubernetes-namespace
    - workload-identity
    - database
    - gateway-route
    - telemetry
  policies:
    - signed-artifacts
    - restricted-pods
    - encrypted-data
  gitops:
    repository: environments-prod
    promotion: pull-request
  evidence:
    require: [policy-tests, drift-check, rollback-plan]
```

Failure patterns

- Treating reconciliation loops as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for reconciliation loops.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

5.3 Promotion and release

Promotion and release must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Move immutable artifacts and configuration through environments with approvals, tests, and provenance. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for promotion and release.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating promotion and release as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for promotion and release.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

5.4 Rollback and disaster recovery

Rollback and disaster recovery must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Revert source, restore controllers, recover clusters, reconcile external state, and validate service. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for rollback and disaster recovery.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating rollback and disaster recovery as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for rollback and disaster recovery.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 05 laboratory and review

Practical laboratory
Implement or simulate a GitOps promotion from development to canary and production with drift and rollback.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore transactional multi-cluster GitOps and cryptographically witnessed reconciliation.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 06

Policy as code and enforcement architecture

Express organizational constraints as versioned, testable decisions.

Chapter operating position
Express organizational constraints as versioned, testable decisions.

6.1 Policy domains and decision points

Policy domains and decision points must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Apply policy in source, pull request, CI, artifact, deployment, admission, runtime, API, and data access. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for policy domains and decision points.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating policy domains and decision points as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for policy domains and decision points.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

6.2 OPA and Rego

OPA and Rego must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Separate policy decision from enforcement, use structured input, bundles, tests, partial evaluation, and decision logs. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for opa and rego.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
platform_request:
  api: ApplicationEnvironment/v2
  owner: team-payments
  environment: production
  composition:
    - kubernetes-namespace
    - workload-identity
    - database
    - gateway-route
    - telemetry
  policies:
    - signed-artifacts
    - restricted-pods
    - encrypted-data
  gitops:
    repository: environments-prod
    promotion: pull-request
  evidence:
    require: [policy-tests, drift-check, rollback-plan]
```

Failure patterns

- Treating opa and rego as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for opa and rego.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

6.3 Kubernetes-native and CEL policy

Kubernetes-native and CEL policy must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use Kyverno, admission policies, mutation, validation, generation, image verification, and exceptions. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for kubernetes-native and cel policy.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating kubernetes-native and cel policy as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for kubernetes-native and cel policy.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

6.4 Policy lifecycle and conflict

Policy lifecycle and conflict must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Manage ownership, precedence, versions, tests, rollout, audit, enforcement, exception, and retirement. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for policy lifecycle and conflict.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating policy lifecycle and conflict as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for policy lifecycle and conflict.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 06 laboratory and review

Practical laboratory
Create equivalent policy tests for an infrastructure plan, Kubernetes resource, and platform API request.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore policy synthesis from standards with human-reviewed executable controls.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 07

Platform security, supply chain, and evidence

Protect the platform because its automation has broad organizational authority.

Chapter operating position
Protect the platform because its automation has broad organizational authority.

7.1 Identity and privilege

Identity and privilege must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use workload identity, JIT administration, scoped controllers, separation, break glass, and audit. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for identity and privilege.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating identity and privilege as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for identity and privilege.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

7.2 Template, plugin, and provider supply chain

Template, plugin, and provider supply chain must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Verify publishers, source, dependencies, images, signatures, provenance, permissions, and updates. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for template, plugin, and provider supply chain.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
platform_request:
  api: ApplicationEnvironment/v2
  owner: team-payments
  environment: production
  composition:
    - kubernetes-namespace
    - workload-identity
    - database
    - gateway-route
    - telemetry
  policies:
    - signed-artifacts
    - restricted-pods
    - encrypted-data
  gitops:
    repository: environments-prod
    promotion: pull-request
  evidence:
    require: [policy-tests, drift-check, rollback-plan]
```

Failure patterns

- Treating template, plugin, and provider supply chain as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for template, plugin, and provider supply chain.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

7.3 Secrets and data boundaries

Secrets and data boundaries must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Broker credentials, isolate tenants, control logs, protect customer data, and rotate trust. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for secrets and data boundaries.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating secrets and data boundaries as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for secrets and data boundaries.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

7.4 Evidence and control validation

Evidence and control validation must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Record requests, policy decisions, changes, artifacts, reconciliations, tests, and user outcomes. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for evidence and control validation.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating evidence and control validation as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for evidence and control validation.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 07 laboratory and review

Practical laboratory
Threat-model the portal, scaffolder, GitOps controller, Crossplane providers, policy engine, and cloud credentials.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore confidential platform control planes and signed automation receipts.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 08

Operations, adoption, and continuous evolution

Operate the platform as a reliable product and avoid becoming a new bottleneck.

Chapter operating position
Operate the platform as a reliable product and avoid becoming a new bottleneck.

8.1 SLOs and support

SLOs and support must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Define platform API, provisioning, deployment, catalog, pipeline, policy, and incident objectives. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for slos and support.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating slos and support as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for slos and support.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

8.2 Feedback and discovery

Feedback and discovery must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use analytics, interviews, support cases, failed journeys, product research, and community. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for feedback and discovery.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
platform_request:
  api: ApplicationEnvironment/v2
  owner: team-payments
  environment: production
  composition:
    - kubernetes-namespace
    - workload-identity
    - database
    - gateway-route
    - telemetry
  policies:
    - signed-artifacts
    - restricted-pods
    - encrypted-data
  gitops:
    repository: environments-prod
    promotion: pull-request
  evidence:
    require: [policy-tests, drift-check, rollback-plan]
```

Failure patterns

- Treating feedback and discovery as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for feedback and discovery.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

8.3 Migration and enablement

Migration and enablement must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Onboard teams, import existing systems, document changes, train users, and retire old paths. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for migration and enablement.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating migration and enablement as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for migration and enablement.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

8.4 Platform roadmap and economics

Platform roadmap and economics must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Prioritize capabilities by user value, risk, leverage, cost, capacity, and organizational readiness. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the developer intent, catalog, template or API, policy, Git source, reconciliation, platform resource, evidence, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for platform roadmap and economics.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating platform roadmap and economics as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for platform roadmap and economics.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 08 laboratory and review

Practical laboratory
Create a twelve-month platform roadmap from developer evidence and define which capabilities should not be centralized.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore agentic platform operations with human product governance and automatically verified changes.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

Integrated learning roadmap

The guide is intended to be revisited. First establish fluency, then build a small implementation, operate it under failure, automate repeatable work, and finally evaluate emerging techniques against a stable baseline. Progress is demonstrated through evidence rather than reading completion alone.

Stage	Demonstrated capability
Foundation	Explain the system from first principles and trace its critical path without relying on product terminology.
Build	Implement the smallest representative system with explicit contracts, identities, telemetry, and tests.
Operate	Diagnose injected failures, recover safely, and preserve evidence of the causal chain.
Automate	Convert a proven manual workflow into bounded, idempotent, observable automation.
Architect	Design for scale, resilience, security, interoperability, and lifecycle change.
Explore	Evaluate frontier techniques using stated hypotheses, limitations, and adoption gates.

Source authority register

Source policy

Official standards, specifications, and project documentation remain with their issuing organizations. Mythos Systems provides original engineering interpretation, synthesis, implementation guidance, and relationship mapping. Source verification dates do not guarantee that an authority has not changed since review.

01. CNCF - Platform Engineering Technical Community Group
Role: Cloud-native platform-engineering practices and reference material.
Verified: 2026-07-26
Official source: https://tag-app-delivery.cncf.io/wgs/platforms/

02. Backstage - Backstage Software Catalog
Role: Software ownership and metadata catalog based on source-controlled entity definitions.
Verified: 2026-07-26
Official source: https://backstage.io/docs/features/software-catalog/

03. Backstage - Software Templates and Scaffolder
Role: Self-service software templates, actions, workflows, and authorization.
Verified: 2026-07-26
Official source: https://backstage.io/docs/features/software-templates/

04. Crossplane - Crossplane Documentation
Role: Control-plane framework for custom APIs, composition, managed resources, and platform engineering.
Verified: 2026-07-26
Official source: https://docs.crossplane.io/latest/

05. Crossplane - What's New in Crossplane v2
Role: Current v2 architecture, namespaced resources, composition, and operational workflows.
Verified: 2026-07-26
Official source: https://docs.crossplane.io/latest/whats-new/

06. Argo Project - Argo CD Documentation
Role: Declarative GitOps continuous delivery and reconciliation.
Verified: 2026-07-26
Official source: https://argo-cd.readthedocs.io/en/stable/

07. Flux - Flux Documentation
Role: GitOps toolkit, source reconciliation, delivery, notifications, and controllers.
Verified: 2026-07-26
Official source: https://fluxcd.io/flux/

08. Open Policy Agent - Open Policy Agent Documentation
Role: General-purpose policy engine and Rego policy language.
Verified: 2026-07-26
Official source: https://www.openpolicyagent.org/docs

09. Kyverno - Kyverno Documentation
Role: Kubernetes-native and CEL-oriented policy-as-code lifecycle.
Verified: 2026-07-26
Official source: https://kyverno.io/docs/

Authority application and refresh triggers

Authority class	Required library action
Protocols and specifications	Recheck interoperability, security considerations, defaults, and migration guidance when a referenced specification changes.
Security and assurance frameworks	Update control mappings, evidence expectations, risk language, and assessment procedures after material framework revision.
Languages, runtimes, and projects	Re-run examples, tests, deprecation checks, and operational recommendations against supported releases.
Benchmarks and evaluations	Re-execute claims when workloads, methods, models, compilers, drivers, hardware, or scoring rules change.
Operational evidence	Incorporate validated incidents, failure tests, reader corrections, and measured implementation outcomes into the next revision.

Limitations, freshness, and revision control

Boundary	Statement
Publication limitation	This guide synthesizes broad engineering practice. It does not replace product documentation, legal advice, safety certification, or environment-specific design review.
Evidence limitation	Not every pattern is benchmarked in every environment. Examples illustrate engineering methods and must be validated against actual versions, workloads, hardware, policy, and failure conditions.
Freshness limitation	Vendor behavior, software libraries, AI models, security threats, and emerging standards can change quickly. Volatile claims require source revalidation before operational adoption.
Adoption limitation	Frontier content is not an implicit adoption recommendation. Adoption requires defined value, qualification gates, controlled proof, rollback, ownership, and residual-risk acceptance.
Status boundary	Candidate Focused Field Guide describes the publication, not certification of a product, implementation, engineer, or organization.

Revision record

Version	Revision statement
1.0.0-candidate	Initial candidate living-guide edition. Published 2026-07-26; scheduled review 2026-10-26.

Search and relationship metadata

Dimension	Engineering position
Primary domain	MYTHOS-CLD - Cloud, Platform & Distributed Systems
Secondary domain	MYTHOS-SWE; MYTHOS-SEC; MYTHOS-DAT
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	platform engineering, internal developer platform, GitOps, policy as code, Backstage, Crossplane, Argo CD, Flux, OPA, Kyverno
Canonical route	/library/field-guides/mythos-fg-cld-0003/

Living publication

Corrections, expanded laboratories, improved diagrams, authority changes, and new engineering evidence should revise this permanent publication rather than create disconnected replacements.

CANONICAL LIVING OVERVIEW GUIDE

Serverless, Event-Driven, and Function Architecture

A serverless and event-driven engineering guide covering functions, scale-to-zero, event contracts, CloudEvents, brokers, queues, streams, workflows, idempotency, state, cold starts, observability, security, cost, and portability.

PERMANENT ID
MYTHOS-FG-CLD-0004

PUBLICATION CLASS
FIELD GUIDE

VERSION / STATUS
1.0.0-candidate / CANDIDATE

PERMANENT PUBLICATION IDENTITY

Serverless, Event-Driven, and Function Architecture

Document ID
MYTHOS-FG-CLD-0004

Version
1.0.0-candidate

Status
Candidate Focused Field Guide

Review date
2026-10-26

Publication position
This is a living field guide. It is designed for frequent revision while retaining a stable permanent identity. Candidate status means the content is ready for structured use and review but is not represented as an external certification or authority publication.

LEVEL	CLASS	CADENCE	EVIDENCE
L1-L4	FIELD GUIDE	QUARTERLY	SOURCE-BOUND
Foundational through frontier	Practical and educational	Interim updates as needed	Limitations remain visible

Reader orientation

Dimension	Engineering position
Primary domain	MYTHOS-CLD - Cloud, Platform & Distributed Systems
Secondary domain	MYTHOS-SWE; MYTHOS-DAT; MYTHOS-SEC
Audience	Cloud, platform, software, integration, data, SRE, and security engineers.
Prerequisites	Distributed systems, APIs, messaging, cloud, software reliability, and observability.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

Choose functions, containers, services, workflows, queues, streams, and brokers according to workload semantics.

Design event contracts with identity, schema, ordering, delivery, idempotency, and evolution.

Build reliable functions that tolerate duplicates, retries, partial failure, cold starts, and dependency outages.

Operate serverless systems with traces, metrics, replay, dead-letter handling, security, and cost controls.

Avoid provider lock-in through portable contracts, tested abstractions, and explicit exit strategies.

How to use this guide

Read the guide as a progressive system rather than a disconnected encyclopedia. Foundational material establishes the mental model; practitioner sections convert it into repeatable implementation and operations; advanced sections expose architecture, scale, security, and failure behavior; frontier sections describe technologies whose evidence or maturity is still evolving.

Suggested reading path
New practitioners should follow the chapters in order and complete the laboratories. Experienced engineers may begin with the architecture, failure, validation, or frontier sections, then use the related-document map to deepen a specific topic.

Learning and assurance model

Level	Reader outcome
L1 - Foundational	Terminology, first principles, component roles, packet or state flow, and simple working examples.
L2 - Practitioner	Implementation, configuration, automation, testing, troubleshooting, and operational evidence.
L3 - Advanced	Architecture, scale, concurrency, resilience, threat models, performance, and complex migrations.
L4 - Frontier	Emerging systems, research directions, technical uncertainty, readiness gates, and adoption signals.

Contents

CHAPTER 01

Serverless architecture and execution model

Understand what the platform manages and what the application still owns.

Chapter operating position
Understand what the platform manages and what the application still owns.

1.1 Functions and managed runtimes

Functions and managed runtimes must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use event or request handlers, runtime lifecycle, packaging, concurrency, limits, identity, and configuration. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for functions and managed runtimes.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating functions and managed runtimes as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for functions and managed runtimes.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

1.2 Scale-to-zero and elasticity

Scale-to-zero and elasticity must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Manage activation, cold start, warm instances, burst, maximum scale, fairness, and overload. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for scale-to-zero and elasticity.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
event:
  specversion: "1.0"
  id: order-042
  source: /orders
  type: com.mythos.order.accepted
  subject: order/042
  datacontenttype: application/json
handler:
  idempotency_key: event.id
  timeout_seconds: 20
  retry:
    max_attempts: 5
    backoff: exponential-jitter
  compensation:
    on_partial_failure: release-inventory
```

Failure patterns

Treating scale-to-zero and elasticity as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for scale-to-zero and elasticity.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

1.3 Service and function boundaries

Service and function boundaries must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Choose granularity according to state, latency, ownership, deployment, dependencies, and observability. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for service and function boundaries.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating service and function boundaries as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for service and function boundaries.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

1.4 Shared responsibility and limits

Shared responsibility and limits must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Account for provider runtime, networking, event delivery, logs, keys, quotas, and regional behavior. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for shared responsibility and limits.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating shared responsibility and limits as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for shared responsibility and limits.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 01 laboratory and review

Practical laboratory
Compare a synchronous API, background job, stream processor, and scheduled function across serverless and container services.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore WebAssembly functions and disaggregated serverless runtimes.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 02

Event contracts and CloudEvents

Represent events consistently across producers, routers, consumers, and platforms.

Chapter operating position
Represent events consistently across producers, routers, consumers, and platforms.

2.1 Event identity and metadata

Event identity and metadata must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use ID, source, type, subject, time, content type, schema, trace context, and extensions. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for event identity and metadata.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating event identity and metadata as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for event identity and metadata.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

2.2 Schema and semantic contract

Schema and semantic contract must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Define payload, invariants, optionality, units, identity, privacy, version, and compatibility. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for schema and semantic contract.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
event:
  specversion: "1.0"
  id: order-042
  source: /orders
  type: com.mythos.order.accepted
  subject: order/042
  datacontenttype: application/json
handler:
  idempotency_key: event.id
  timeout_seconds: 20
  retry:
    max_attempts: 5
    backoff: exponential-jitter
  compensation:
    on_partial_failure: release-inventory
```

Failure patterns

Treating schema and semantic contract as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for schema and semantic contract.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

2.3 Bindings and transport

Bindings and transport must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Map events to HTTP, messaging, Kafka, AMQP, WebSockets, and provider systems. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for bindings and transport.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating bindings and transport as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for bindings and transport.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

2.4 Filtering and routing

Filtering and routing must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use attributes, CloudEvents SQL, subscriptions, authorization, dead letters, and observability. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for filtering and routing.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating filtering and routing as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for filtering and routing.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 02 laboratory and review

Practical laboratory

Define a CloudEvents contract and route it across HTTP and a message broker while preserving identity and trace context.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore semantically typed event fabrics and cross-organization event federation.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 03

Queues, brokers, streams, and delivery semantics

Choose transport according to ordering, replay, scale, and failure behavior.

Chapter operating position
Choose transport according to ordering, replay, scale, and failure behavior.

3.1 Queue and broker semantics

Queue and broker semantics must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Handle visibility, acknowledgment, redelivery, dead letters, priority, delay, and consumer groups. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for queue and broker semantics.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating queue and broker semantics as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for queue and broker semantics.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

3.2 Streams and logs

Streams and logs must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use partitions, offsets, retention, replay, compaction, ordering, and stateful processing. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for streams and logs.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
event:
  specversion: "1.0"
  id: order-042
  source: /orders
  type: com.mythos.order.accepted
  subject: order/042
  datacontenttype: application/json
handler:
  idempotency_key: event.id
  timeout_seconds: 20
  retry:
    max_attempts: 5
    backoff: exponential-jitter
  compensation:
    on_partial_failure: release-inventory
```

Failure patterns

Treating streams and logs as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for streams and logs.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

3.3 At-most, at-least, and effectively-once

At-most, at-least, and effectively-once must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Separate transport promises from application idempotency and transactional effects. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for at-most, at-least, and effectively-once.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating at-most, at-least, and effectively-once as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for at-most, at-least, and effectively-once.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

3.4 Backpressure and overload

Backpressure and overload must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Bound producers, brokers, consumers, retries, lag, storage, and downstream capacity. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for backpressure and overload.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating backpressure and overload as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for backpressure and overload.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 03 laboratory and review

Practical laboratory
Simulate duplicate, reordered, delayed, and poisoned events and prove consumer correctness.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore globally replicated event logs with sovereignty-aware routing.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 04

Function design, state, and idempotency

Build handlers that remain correct across retries and concurrent execution.

Chapter operating position
Build handlers that remain correct across retries and concurrent execution.

4.1 Pure handler and side effects

Pure handler and side effects must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Separate validation, decision, state, external effects, response, and evidence. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for pure handler and side effects.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating pure handler and side effects as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for pure handler and side effects.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

4.2 Idempotency keys and deduplication

Idempotency keys and deduplication must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Choose business identity, storage, expiry, concurrency, result replay, and conflict behavior. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for idempotency keys and deduplication.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
event:
  specversion: "1.0"
  id: order-042
  source: /orders
  type: com.mythos.order.accepted
  subject: order/042
  datacontenttype: application/json
handler:
  idempotency_key: event.id
  timeout_seconds: 20
  retry:
    max_attempts: 5
    backoff: exponential-jitter
  compensation:
    on_partial_failure: release-inventory
```

Failure patterns

Treating idempotency keys and deduplication as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for idempotency keys and deduplication.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

4.3 State and coordination

State and coordination must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use databases, caches, object stores, workflows, durable objects, locks, and transactions. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for state and coordination.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating state and coordination as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for state and coordination.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

4.4 Timeout, retry, and compensation

Timeout, retry, and compensation must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Set deadlines, classify errors, use jitter, avoid retry storms, and reverse partial effects. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for timeout, retry, and compensation.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating timeout, retry, and compensation as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for timeout, retry, and compensation.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 04 laboratory and review

Practical laboratory
Implement a payment-like function with idempotency, duplicate delivery, timeout, and compensation tests.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore transactional functions over durable state abstractions.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 05

Workflow and saga orchestration

Coordinate long-running business processes beyond one function invocation.

Chapter operating position
Coordinate long-running business processes beyond one function invocation.

5.1 Orchestration and choreography

Orchestration and choreography must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Compare central workflow state with event-driven participant coordination. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for orchestration and choreography.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating orchestration and choreography as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for orchestration and choreography.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

5.2 Workflow state and determinism

Workflow state and determinism must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Persist decisions, timers, signals, retries, versions, compensation, and human tasks. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for workflow state and determinism.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
event:
  specversion: "1.0"
  id: order-042
  source: /orders
  type: com.mythos.order.accepted
  subject: order/042
  datacontenttype: application/json
handler:
  idempotency_key: event.id
  timeout_seconds: 20
  retry:
    max_attempts: 5
    backoff: exponential-jitter
  compensation:
    on_partial_failure: release-inventory
```

Failure patterns

Treating workflow state and determinism as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for workflow state and determinism.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

5.3 Saga patterns

Saga patterns must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Define local transactions, compensators, pivot steps, irreversibility, and recovery. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for saga patterns.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating saga patterns as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for saga patterns.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

5.4 Serverless Workflow portability

Serverless Workflow portability must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Represent states, events, functions, errors, timeouts, and data flow through portable definitions. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for serverless workflow portability.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating serverless workflow portability as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for serverless workflow portability.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 05 laboratory and review

Practical laboratory
Model an order workflow with payment, inventory, shipping, timeout, cancellation, and compensation.

Laboratory evidence package

Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.

Versioned configuration or code with explanatory comments and reproducible setup instructions.

Nominal-path validation plus at least two injected failure or boundary conditions.

Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.

A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore agent-assisted workflow generation constrained by formal state and effect contracts.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 06

Cold starts, performance, and capacity

Engineer latency and throughput under elastic runtime behavior.

Chapter operating position
Engineer latency and throughput under elastic runtime behavior.

6.1 Initialization and dependency load

Initialization and dependency load must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Measure runtime startup, imports, model load, network, credentials, clients, and cache. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for initialization and dependency load.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating initialization and dependency load as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for initialization and dependency load.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

6.2 Concurrency and instance lifecycle

Concurrency and instance lifecycle must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Control requests per instance, pools, connections, memory, CPU, and execution limits. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for concurrency and instance lifecycle.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
event:
  specversion: "1.0"
  id: order-042
  source: /orders
  type: com.mythos.order.accepted
  subject: order/042
  datacontenttype: application/json
handler:
  idempotency_key: event.id
  timeout_seconds: 20
  retry:
    max_attempts: 5
    backoff: exponential-jitter
  compensation:
    on_partial_failure: release-inventory
```

Failure patterns

Treating concurrency and instance lifecycle as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for concurrency and instance lifecycle.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

6.3 Provisioned and predictive capacity

Provisioned and predictive capacity must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use minimum instances, warm pools, schedules, traffic forecasts, and cost tradeoffs. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for provisioned and predictive capacity.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating provisioned and predictive capacity as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for provisioned and predictive capacity.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

6.4 Benchmarking and SLOs

Benchmarking and SLOs must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Measure user latency, activation, handler, dependency, queue, retries, success, and tails. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for benchmarking and slos.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating benchmarking and slos as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for benchmarking and slos.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 06 laboratory and review

Practical laboratory
Benchmark a function under cold, warm, burst, and dependency-failure conditions with a complete latency breakdown.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore snapshot-based instant start and hardware-accelerated serverless AI.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 07

Security, observability, and incident response

Preserve identity, evidence, and containment across ephemeral execution.

Chapter operating position
Preserve identity, evidence, and containment across ephemeral execution.

7.1 Function and service identity

Function and service identity must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use short-lived credentials, scopes, resource policy, network, secrets, and least privilege. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for function and service identity.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating function and service identity as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for function and service identity.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

7.2 Supply chain and deployment

Supply chain and deployment must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Verify source, dependencies, artifacts, signatures, provenance, configuration, and runtime. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for supply chain and deployment.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
event:
  specversion: "1.0"
  id: order-042
  source: /orders
  type: com.mythos.order.accepted
  subject: order/042
  datacontenttype: application/json
handler:
  idempotency_key: event.id
  timeout_seconds: 20
  retry:
    max_attempts: 5
    backoff: exponential-jitter
  compensation:
    on_partial_failure: release-inventory
```

Failure patterns

Treating supply chain and deployment as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for supply chain and deployment.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

7.3 Telemetry and correlation

Telemetry and correlation must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Trace event, queue, function, workflow, database, external API, retries, and business outcome. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for telemetry and correlation.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating telemetry and correlation as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for telemetry and correlation.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

7.4 Incident and recovery

Incident and recovery must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Disable triggers, isolate functions, revoke credentials, replay safely, restore state, and verify. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for incident and recovery.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating incident and recovery as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for incident and recovery.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 07 laboratory and review

Practical laboratory
Respond to a compromised function image and determine affected events, data, credentials, and downstream effects.

Laboratory evidence package

Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.

Versioned configuration or code with explanatory comments and reproducible setup instructions.

Nominal-path validation plus at least two injected failure or boundary conditions.

Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.

A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore confidential serverless execution and signed event-processing receipts.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 08

Portability, economics, and adoption

Choose serverless based on lifecycle outcomes rather than minimal code alone.

Chapter operating position
Choose serverless based on lifecycle outcomes rather than minimal code alone.

8.1 Cost model

Cost model must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Measure requests, duration, memory, provisioned capacity, brokers, storage, networking, logs, and support. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for cost model.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating cost model as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for cost model.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

8.2 Portability and abstraction

Portability and abstraction must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use CloudEvents, containers, Knative, open workflows, infrastructure as code, and testable adapters. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for portability and abstraction.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
event:
  specversion: "1.0"
  id: order-042
  source: /orders
  type: com.mythos.order.accepted
  subject: order/042
  datacontenttype: application/json
handler:
  idempotency_key: event.id
  timeout_seconds: 20
  retry:
    max_attempts: 5
    backoff: exponential-jitter
  compensation:
    on_partial_failure: release-inventory
```

Failure patterns

- Treating portability and abstraction as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for portability and abstraction.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

8.3 Lock-in and provider behavior

Lock-in and provider behavior must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Assess event sources, identity, state, networking, limits, observability, regions, and operational tooling. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for lock-in and provider behavior.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating lock-in and provider behavior as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for lock-in and provider behavior.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

8.4 Adoption decision

Adoption decision must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Compare serverless with services, Kubernetes, batch, edge, and managed integration according to workload. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the event or request, contract, broker, admission, function, state, external effect, retry or compensation, telemetry, and business outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for adoption decision.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating adoption decision as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for adoption decision.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 08 laboratory and review

Practical laboratory
Create an adoption decision for an event-driven product including cost, portability, SLOs, state, and exit.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore multi-cloud serverless fabrics and edge-to-cloud event continuity.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

Integrated learning roadmap

The guide is intended to be revisited. First establish fluency, then build a small implementation, operate it under failure, automate repeatable work, and finally evaluate emerging techniques against a stable baseline. Progress is demonstrated through evidence rather than reading completion alone.

Stage	Demonstrated capability
Foundation	Explain the system from first principles and trace its critical path without relying on product terminology.
Build	Implement the smallest representative system with explicit contracts, identities, telemetry, and tests.
Operate	Diagnose injected failures, recover safely, and preserve evidence of the causal chain.
Automate	Convert a proven manual workflow into bounded, idempotent, observable automation.
Architect	Design for scale, resilience, security, interoperability, and lifecycle change.
Explore	Evaluate frontier techniques using stated hypotheses, limitations, and adoption gates.

Source authority register

Source policy
Official standards, specifications, and project documentation remain with their issuing organizations. Mythos Systems provides original engineering interpretation, synthesis, implementation guidance, and relationship mapping. Source verification dates do not guarantee that an authority has not changed since review.

01. CloudEvents - CloudEvents Specification
Role: Common event metadata, bindings, SDKs, and interoperability.
Verified: 2026-07-26
Official source: https://cloudevents.io/

02. Knative - Knative Documentation
Role: Serving, Eventing, Functions, autoscaling, routing, and event delivery.
Verified: 2026-07-26
Official source: https://knative.dev/docs/

03. CNCF Serverless Workflow - Serverless Workflow Specification
Role: Portable workflow definition for event-driven and serverless orchestration.
Verified: 2026-07-26
Official source: https://serverlessworkflow.io/

04. OpenTelemetry - OpenTelemetry
Role: Vendor-neutral telemetry APIs, SDKs, collectors, traces, metrics, and logs.
Verified: 2026-07-26
Official source: https://opentelemetry.io/

05. NIST - SP 800-145 - The NIST Definition of Cloud Computing
Role: Cloud service and deployment model definitions.
Verified: 2026-07-26
Official source: https://csrc.nist.gov/pubs/sp/800/145/final

06. NIST - SP 500-292 - NIST Cloud Computing Reference Architecture
Role: Cloud actors, activities, functions, and reference architecture.
Verified: 2026-07-26
Official source: https://www.nist.gov/publications/nist-cloud-computing-reference-architecture

07. IETF / RFC Editor - RFC 9700 - Best Current Practice for OAuth 2.0 Security
Role: Current OAuth 2.0 security best current practice.
Verified: 2026-07-26
Official source: https://www.rfc-editor.org/info/rfc9700

Authority application and refresh triggers

Authority class	Required library action
Protocols and specifications	Recheck interoperability, security considerations, defaults, and migration guidance when a referenced specification changes.
Security and assurance frameworks	Update control mappings, evidence expectations, risk language, and assessment procedures after material framework revision.
Languages, runtimes, and projects	Re-run examples, tests, deprecation checks, and operational recommendations against supported releases.
Benchmarks and evaluations	Re-execute claims when workloads, methods, models, compilers, drivers, hardware, or scoring rules change.
Operational evidence	Incorporate validated incidents, failure tests, reader corrections, and measured implementation outcomes into the next revision.

Limitations, freshness, and revision control

Boundary	Statement
Publication limitation	This guide synthesizes broad engineering practice. It does not replace product documentation, legal advice, safety certification, or environment-specific design review.
Evidence limitation	Not every pattern is benchmarked in every environment. Examples illustrate engineering methods and must be validated against actual versions, workloads, hardware, policy, and failure conditions.
Freshness limitation	Vendor behavior, software libraries, AI models, security threats, and emerging standards can change quickly. Volatile claims require source revalidation before operational adoption.
Adoption limitation	Frontier content is not an implicit adoption recommendation. Adoption requires defined value, qualification gates, controlled proof, rollback, ownership, and residual-risk acceptance.
Status boundary	Candidate Focused Field Guide describes the publication, not certification of a product, implementation, engineer, or organization.

Revision record

Version	Revision statement
1.0.0-candidate	Initial candidate living-guide edition. Published 2026-07-26; scheduled review 2026-10-26.

Search and relationship metadata

Dimension	Engineering position
Primary domain	MYTHOS-CLD - Cloud, Platform & Distributed Systems
Secondary domain	MYTHOS-SWE; MYTHOS-DAT; MYTHOS-SEC
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	serverless, event driven, functions, CloudEvents, Knative, workflow, saga, idempotency, cold start, event architecture
Canonical route	/library/field-guides/mythos-fg-cld-0004/

Living publication

Corrections, expanded laboratories, improved diagrams, authority changes, and new engineering evidence should revise this permanent publication rather than create disconnected replacements.

CANONICAL LIVING OVERVIEW GUIDE

Service Mesh, API Gateway, and East-West Traffic Engineering

A traffic-platform guide covering gateways, proxies, service mesh, Gateway API, Envoy, Istio, workload identity, mTLS, routing, resilience, policy, telemetry, ambient and sidecar architectures, multi-cluster, egress, and emerging agent-aware gateways.

PERMANENT ID
MYTHOS-FG-CLD-0005

PUBLICATION CLASS
FIELD GUIDE

VERSION / STATUS
1.0.0-candidate / CANDIDATE

PERMANENT PUBLICATION IDENTITY

Service Mesh, API Gateway, and East-West Traffic Engineering

Document ID
MYTHOS-FG-CLD-0005

Version
1.0.0-candidate

Status
Candidate Focused Field Guide

Review date
2026-10-26

Publication position
This is a living field guide. It is designed for frequent revision while retaining a stable permanent identity. Candidate status means the content is ready for structured use and review but is not represented as an external certification or authority publication.

LEVEL	CLASS	CADENCE	EVIDENCE
L1-L4	FIELD GUIDE	QUARTERLY	SOURCE-BOUND
Foundational through frontier	Practical and educational	Interim updates as needed	Limitations remain visible

Reader orientation

Dimension	Engineering position
Primary domain	MYTHOS-CLD - Cloud, Platform & Distributed Systems
Secondary domain	MYTHOS-NET; MYTHOS-SEC; MYTHOS-AI
Audience	Platform, network, cloud, security, API, SRE, and AI-infrastructure engineers.
Prerequisites	TCP/IP, HTTP, TLS, Kubernetes, identity, distributed systems, APIs, and observability.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

Differentiate API management, ingress, gateway, load balancing, service mesh, and application policy.

Design north-south and east-west traffic using role-oriented APIs and explicit ownership.

Apply workload identity, mTLS, authorization, routing, resilience, rate limits, and egress controls.

Operate sidecar, ambient, gateway, and proxyless patterns with measurable failure and upgrade behavior.

Prepare traffic platforms for agent, model, MCP, A2A, streaming, and long-running AI workloads.

How to use this guide

Read the guide as a progressive system rather than a disconnected encyclopedia. Foundational material establishes the mental model; practitioner sections convert it into repeatable implementation and operations; advanced sections expose architecture, scale, security, and failure behavior; frontier sections describe technologies whose evidence or maturity is still evolving.

Suggested reading path
New practitioners should follow the chapters in order and complete the laboratories. Experienced engineers may begin with the architecture, failure, validation, or frontier sections, then use the related-document map to deepen a specific topic.

Learning and assurance model

Level	Reader outcome
L1 - Foundational	Terminology, first principles, component roles, packet or state flow, and simple working examples.
L2 - Practitioner	Implementation, configuration, automation, testing, troubleshooting, and operational evidence.
L3 - Advanced	Architecture, scale, concurrency, resilience, threat models, performance, and complex migrations.
L4 - Frontier	Emerging systems, research directions, technical uncertainty, readiness gates, and adoption signals.

Contents

CHAPTER 01

Traffic architecture and control boundaries

Define which layer owns routing, identity, policy, and reliability.

Chapter operating position
Define which layer owns routing, identity, policy, and reliability.

1.1 North-south and east-west traffic

North-south and east-west traffic must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Distinguish client ingress, partner APIs, service calls, cluster gateways, egress, and cross-region paths. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for north-south and east-west traffic.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating north-south and east-west traffic as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for north-south and east-west traffic.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

1.2 Gateway, proxy, and mesh roles

Gateway, proxy, and mesh roles must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Separate API management, load balancing, ingress, edge gateway, service mesh, and application logic. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for gateway, proxy, and mesh roles.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
traffic_policy:
  source_identity: spiffe://mythos/agent/planner
  destination: mcp://deployment-tools
  protocol: http2-streaming
  authorization:
    scopes: [deploy:canary]
  routing:
    model_backend: capability-aware
    retry_budget: 0.05
    timeout_seconds: 300
  safety:
    prompt_injection_filter: enabled
    production_confirmation: required
  telemetry:
    trace_mission_and_tool_effects: true
```

Failure patterns

Treating gateway, proxy, and mesh roles as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for gateway, proxy, and mesh roles.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

1.3 Control and data planes

Control and data planes must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Understand configuration, discovery, policy, certificate, telemetry, proxy state, and traffic processing. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for control and data planes.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating control and data planes as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for control and data planes.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

1.4 Ownership and multi-tenancy

Ownership and multi-tenancy must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Assign infrastructure, platform, application, security, network, tenant, and operator responsibilities. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for ownership and multi-tenancy.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating ownership and multi-tenancy as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for ownership and multi-tenancy.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 01 laboratory and review

Practical laboratory
Map one request from external client through gateway, services, database, and egress with every control owner.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore unified traffic control planes for humans, services, agents, models, and devices.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 02

Gateway API and declarative traffic policy

Use role-oriented resources for infrastructure and application routing.

Chapter operating position
Use role-oriented resources for infrastructure and application routing.

2.1 GatewayClass, Gateway, and listeners

GatewayClass, Gateway, and listeners must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Define implementation, infrastructure lifecycle, addresses, ports, protocols, certificates, and allowed routes. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for gatewayclass, gateway, and listeners.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating gatewayclass, gateway, and listeners as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for gatewayclass, gateway, and listeners.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

2.2 HTTPRoute, GRPCRoute, and other routes

HTTPRoute, GRPCRoute, and other routes must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Match hosts, paths, headers, methods, backends, weights, filters, and references. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for httproute, grpcroute, and other routes.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
traffic_policy:
  source_identity: spiffe://mythos/agent/planner
  destination: mcp://deployment-tools
  protocol: http2-streaming
  authorization:
    scopes: [deploy:canary]
  routing:
    model_backend: capability-aware
    retry_budget: 0.05
    timeout_seconds: 300
  safety:
    prompt_injection_filter: enabled
    production_confirmation: required
  telemetry:
    trace_mission_and_tool_effects: true
```

Failure patterns

Treating httproute, grpcroute, and other routes as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for httproute, grpcroute, and other routes.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

2.3 Policies and attachment

Policies and attachment must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Apply backend, security, timeout, retry, session, rate, and implementation-specific policy with clear scope. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for policies and attachment.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating policies and attachment as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for policies and attachment.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

2.4 GAMMA and service mesh

GAMMA and service mesh must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use service-attached routes and policies for east-west communication. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for gamma and service mesh.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating gamma and service mesh as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for gamma and service mesh.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 02 laboratory and review

Practical laboratory
Build a Gateway API design with separate infrastructure and application ownership, canary routing, and denied cross-namespace references.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore Gateway API as a common contract for ingress, mesh, egress, and agent gateways.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 03

Envoy, xDS, and data-plane engineering

Understand how dynamic proxies receive configuration and process traffic.

Chapter operating position
Understand how dynamic proxies receive configuration and process traffic.

3.1 Listeners, filters, clusters, and endpoints

Listeners, filters, clusters, and endpoints must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Trace connections through filter chains, routing, upstream pools, health, and load balancing. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for listeners, filters, clusters, and endpoints.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating listeners, filters, clusters, and endpoints as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for listeners, filters, clusters, and endpoints.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

3.2 xDS configuration

xDS configuration must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use discovery, versions, nonces, ACK/NACK, incremental updates, consistency, and failure behavior. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for xds configuration.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
traffic_policy:
  source_identity: spiffe://mythos/agent/planner
  destination: mcp://deployment-tools
  protocol: http2-streaming
  authorization:
    scopes: [deploy:canary]
  routing:
    model_backend: capability-aware
    retry_budget: 0.05
    timeout_seconds: 300
  safety:
    prompt_injection_filter: enabled
    production_confirmation: required
  telemetry:
    trace_mission_and_tool_effects: true
```

Failure patterns

Treating xds configuration as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for xds configuration.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

3.3 HTTP and transport behavior

HTTP and transport behavior must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Handle HTTP/1.1, HTTP/2, HTTP/3, gRPC, WebSockets, CONNECT, TLS, retries, and streaming. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for http and transport behavior.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating http and transport behavior as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for http and transport behavior.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

3.4 Performance and capacity

Performance and capacity must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Measure connections, requests, streams, buffers, memory, CPU, TLS, logs, and overload. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for performance and capacity.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating performance and capacity as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for performance and capacity.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 03 laboratory and review

Practical laboratory

Inspect or simulate an xDS update and identify how invalid configuration, stale endpoints, and proxy overload appear.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore programmable data planes and hardware-accelerated proxy processing.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 04

Service identity, mTLS, and authorization

Establish workload trust independent of network location.

Chapter operating position
Establish workload trust independent of network location.

4.1 SPIFFE identities and trust domains

SPIFFE identities and trust domains must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Issue short-lived SVIDs, use Workload API, rotate trust bundles, and federate domains. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for spiffe identities and trust domains.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating spiffe identities and trust domains as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for spiffe identities and trust domains.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

4.2 Mutual TLS and peer authentication

Mutual TLS and peer authentication must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Establish identity, encryption, certificate validation, mode, exceptions, and migration. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for mutual tls and peer authentication.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
traffic_policy:
  source_identity: spiffe://mythos/agent/planner
  destination: mcp://deployment-tools
  protocol: http2-streaming
  authorization:
    scopes: [deploy:canary]
  routing:
    model_backend: capability-aware
    retry_budget: 0.05
    timeout_seconds: 300
  safety:
    prompt_injection_filter: enabled
    production_confirmation: required
  telemetry:
    trace_mission_and_tool_effects: true
```

Failure patterns

Treating mutual tls and peer authentication as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for mutual tls and peer authentication.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

4.3 Service authorization

Service authorization must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Apply identity, namespace, service account, method, path, claims, network, and environmental policy. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for service authorization.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating service authorization as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for service authorization.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

4.4 Credential and certificate operations

Credential and certificate operations must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Monitor issuance, expiry, rotation, revocation, clock, CA health, and disaster recovery. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for credential and certificate operations.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating credential and certificate operations as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for credential and certificate operations.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 04 laboratory and review

Practical laboratory
Deploy or model service identity and mTLS, then test stale identity, wrong trust domain, expired SVID, and policy denial.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore post-quantum workload identity and attested service authorization.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 05

Routing, resilience, and traffic policy

Use traffic controls without amplifying failure.

Chapter operating position
Use traffic controls without amplifying failure.

5.1 Load balancing and locality

Load balancing and locality must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Choose algorithms, zones, priorities, health, panic thresholds, failover, and outlier ejection. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for load balancing and locality.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating load balancing and locality as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for load balancing and locality.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

5.2 Timeouts, retries, and hedging

Timeouts, retries, and hedging must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Set deadlines, retryable conditions, budgets, backoff, idempotency, and amplification limits. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for timeouts, retries, and hedging.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
traffic_policy:
  source_identity: spiffe://mythos/agent/planner
  destination: mcp://deployment-tools
  protocol: http2-streaming
  authorization:
    scopes: [deploy:canary]
  routing:
    model_backend: capability-aware
    retry_budget: 0.05
    timeout_seconds: 300
  safety:
    prompt_injection_filter: enabled
    production_confirmation: required
  telemetry:
    trace_mission_and_tool_effects: true
```

Failure patterns

Treating timeouts, retries, and hedging as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for timeouts, retries, and hedging.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

5.3 Circuit breaking and overload

Circuit breaking and overload must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Bound connections, requests, queues, pending work, rate, memory, and downstream saturation. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for circuit breaking and overload.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating circuit breaking and overload as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for circuit breaking and overload.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

5.4 Progressive delivery and experimentation

Progressive delivery and experimentation must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use weighted routes, headers, mirrors, canaries, A/B tests, metrics, and rollback. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for progressive delivery and experimentation.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating progressive delivery and experimentation as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for progressive delivery and experimentation.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 05 laboratory and review

Practical laboratory
Run a service failure test comparing safe retry budgets with an uncontrolled retry storm.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore adaptive traffic policy driven by causal SLO evidence.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 06

Mesh deployment patterns and lifecycle

Choose sidecar, ambient, gateway, or proxyless integration according to constraints.

Chapter operating position
Choose sidecar, ambient, gateway, or proxyless integration according to constraints.

6.1 Sidecar architecture

Sidecar architecture must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Understand injection, per-pod proxy, resource cost, interception, ordering, upgrades, and failure. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for sidecar architecture.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating sidecar architecture as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for sidecar architecture.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

6.2 Ambient and node-level data planes

Ambient and node-level data planes must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Separate secure transport from L7 policy and evaluate blast radius, visibility, and operations. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for ambient and node-level data planes.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
traffic_policy:
  source_identity: spiffe://mythos/agent/planner
  destination: mcp://deployment-tools
  protocol: http2-streaming
  authorization:
    scopes: [deploy:canary]
  routing:
    model_backend: capability-aware
    retry_budget: 0.05
    timeout_seconds: 300
  safety:
    prompt_injection_filter: enabled
    production_confirmation: required
  telemetry:
    trace_mission_and_tool_effects: true
```

Failure patterns

- Treating ambient and node-level data planes as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for ambient and node-level data planes.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

6.3 Gateway and proxyless patterns

Gateway and proxyless patterns must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use shared proxies, client libraries, service discovery, and application integration. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for gateway and proxyless patterns.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating gateway and proxyless patterns as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for gateway and proxyless patterns.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

6.4 Upgrade and compatibility

Upgrade and compatibility must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Manage control/data-plane skew, revisions, canaries, CRDs, APIs, certificates, rollback, and workload disruption. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for upgrade and compatibility.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating upgrade and compatibility as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for upgrade and compatibility.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 06 laboratory and review

Practical laboratory
Create an upgrade plan comparing sidecar and ambient meshes across two Kubernetes versions and mixed workloads.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore dynamically selected traffic enforcement per service and request.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 07

Observability, troubleshooting, and multi-cluster

Correlate application symptoms with proxy, policy, identity, network, and control state.

Chapter operating position
Correlate application symptoms with proxy, policy, identity, network, and control state.

7.1 Traffic telemetry

Traffic telemetry must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Collect traces, metrics, access logs, flow, TLS, routing, retry, policy, and upstream health. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for traffic telemetry.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating traffic telemetry as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for traffic telemetry.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

7.2 Troubleshooting workflow

Troubleshooting workflow must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Verify client, gateway, DNS, route, proxy config, identity, policy, endpoint, network, application, and return. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for troubleshooting workflow.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
traffic_policy:
  source_identity: spiffe://mythos/agent/planner
  destination: mcp://deployment-tools
  protocol: http2-streaming
  authorization:
    scopes: [deploy:canary]
  routing:
    model_backend: capability-aware
    retry_budget: 0.05
    timeout_seconds: 300
  safety:
    prompt_injection_filter: enabled
    production_confirmation: required
  telemetry:
    trace_mission_and_tool_effects: true
```

Failure patterns

- Treating troubleshooting workflow as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for troubleshooting workflow.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

7.3 Multi-cluster and multi-network

Multi-cluster and multi-network must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Design discovery, gateways, trust, routing, failover, locality, sovereignty, and split brain. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for multi-cluster and multi-network.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating multi-cluster and multi-network as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for multi-cluster and multi-network.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

7.4 Egress and external services

Egress and external services must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Control DNS, service entries, gateways, TLS origination, allowlists, data loss, and third parties. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for egress and external services.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating egress and external services as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for egress and external services.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 07 laboratory and review

Practical laboratory
Troubleshoot an intermittent cross-cluster service caused by stale discovery, certificate skew, and locality failover.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore global mesh fabrics with sovereign policy and verifiable route intent.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 08

AI-agent and model traffic frontier

Extend traffic engineering to MCP, A2A, inference, and long-running agent interactions.

Chapter operating position
Extend traffic engineering to MCP, A2A, inference, and long-running agent interactions.

8.1 Inference and streaming traffic

Inference and streaming traffic must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Handle large prompts, streaming responses, long requests, cancellations, retries, affinity, and model backends. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for inference and streaming traffic.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating inference and streaming traffic as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for inference and streaming traffic.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

8.2 MCP and capability gateways

MCP and capability gateways must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Discover servers, authorize tools, inspect schemas, route tenants, limit effects, and preserve evidence. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for mcp and capability gateways.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
traffic_policy:
  source_identity: spiffe://mythos/agent/planner
  destination: mcp://deployment-tools
  protocol: http2-streaming
  authorization:
    scopes: [deploy:canary]
  routing:
    model_backend: capability-aware
    retry_budget: 0.05
    timeout_seconds: 300
  safety:
    prompt_injection_filter: enabled
    production_confirmation: required
  telemetry:
    trace_mission_and_tool_effects: true
```

Failure patterns

Treating mcp and capability gateways as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for mcp and capability gateways.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

8.3 A2A agent communication

A2A agent communication must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Route agent cards, tasks, messages, artifacts, streaming updates, identity, and cross-organization policy. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for a2a agent communication.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating a2a agent communication as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for a2a agent communication.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

8.4 Agent-aware gateways

Agent-aware gateways must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Apply model routing, token budgets, prompt-injection controls, semantic policy, observability, and safe fallbacks. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the client, gateway, route, proxy, identity, policy, upstream, service, return path, telemetry, and agent or user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for agent-aware gateways.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating agent-aware gateways as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for agent-aware gateways.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 08 laboratory and review

Practical laboratory
Design an agent gateway that fronts model inference, MCP servers, and A2A peers with identity, policy, streaming, and telemetry.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore agentgateway-style data planes and semantic traffic management as an experimental frontier.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

Integrated learning roadmap

The guide is intended to be revisited. First establish fluency, then build a small implementation, operate it under failure, automate repeatable work, and finally evaluate emerging techniques against a stable baseline. Progress is demonstrated through evidence rather than reading completion alone.

Stage	Demonstrated capability
Foundation	Explain the system from first principles and trace its critical path without relying on product terminology.
Build	Implement the smallest representative system with explicit contracts, identities, telemetry, and tests.
Operate	Diagnose injected failures, recover safely, and preserve evidence of the causal chain.
Automate	Convert a proven manual workflow into bounded, idempotent, observable automation.
Architect	Design for scale, resilience, security, interoperability, and lifecycle change.
Explore	Evaluate frontier techniques using stated hypotheses, limitations, and adoption gates.

Source authority register

Source policy
Official standards, specifications, and project documentation remain with their issuing organizations. Mythos Systems provides original engineering interpretation, synthesis, implementation guidance, and relationship mapping. Source verification dates do not guarantee that an authority has not changed since review.

01. Kubernetes SIG Network - Gateway API
Role: Kubernetes-native role-oriented L4/L7 routing and service-mesh APIs.
Verified: 2026-07-26
Official source: https://gateway-api.sigs.k8s.io/docs/introduction/

02. Kubernetes SIG Network - Gateway API Specification
Role: Normative resource and field semantics.
Verified: 2026-07-26
Official source: https://gateway-api.sigs.k8s.io/reference/api-spec/

03. Istio - Istio Documentation
Role: Service-mesh traffic, security, telemetry, policy, and operations.
Verified: 2026-07-26
Official source: https://istio.io/latest/docs/

04. Istio - Istio 1.30 Release
Role: Current supported release including experimental agentgateway integration.
Verified: 2026-07-26
Official source: https://istio.io/latest/news/releases/1.30.x/announcing-1.30/

05. Envoy Proxy - Envoy Documentation
Role: L4/L7 data-plane proxy, xDS, routing, resilience, security, and telemetry.
Verified: 2026-07-26
Official source: https://www.envoyproxy.io/docs/envoy/latest/

06. Envoy Gateway - Envoy Gateway Documentation
Role: Gateway API implementation, traffic policy, security, and operations.
Verified: 2026-07-26
Official source: https://gateway.envoyproxy.io/latest/

07. SPIFFE - The SPIFFE Standard
Role: Workload identities, SVIDs, trust domains, federation, and Workload API.
Verified: 2026-07-26
Official source: https://spiffe.io/docs/latest/spiffe-specs/

08. Model Context Protocol - Model Context Protocol Specification 2025-11-25
Role: Stable protocol baseline for tool, resource, prompt, transport, authorization, lifecycle, and capability integration.
Verified: 2026-07-26
Official source: https://modelcontextprotocol.io/specification/2025-11-25

09. Agent2Agent Protocol - Agent2Agent Protocol Specification 1.0
Role: Open protocol for agent discovery, communication, task delegation, status, artifacts, and interoperability.
Verified: 2026-07-26
Official source: https://a2a-protocol.org/v1.0.0/

10. OpenTelemetry - OpenTelemetry
Role: Vendor-neutral telemetry APIs, SDKs, collectors, traces, metrics, and logs.
Verified: 2026-07-26
Official source: https://opentelemetry.io/

Authority application and refresh triggers

Authority class	Required library action
Protocols and specifications	Recheck interoperability, security considerations, defaults, and migration guidance when a referenced specification changes.
Security and assurance frameworks	Update control mappings, evidence expectations, risk language, and assessment procedures after material framework revision.
Languages, runtimes, and projects	Re-run examples, tests, deprecation checks, and operational recommendations against supported releases.
Benchmarks and evaluations	Re-execute claims when workloads, methods, models, compilers, drivers, hardware, or scoring rules change.
Operational evidence	Incorporate validated incidents, failure tests, reader corrections, and measured implementation outcomes into the next revision.

Limitations, freshness, and revision control

Boundary	Statement
Publication limitation	This guide synthesizes broad engineering practice. It does not replace product documentation, legal advice, safety certification, or environment-specific design review.
Evidence limitation	Not every pattern is benchmarked in every environment. Examples illustrate engineering methods and must be validated against actual versions, workloads, hardware, policy, and failure conditions.
Freshness limitation	Vendor behavior, software libraries, AI models, security threats, and emerging standards can change quickly. Volatile claims require source revalidation before operational adoption.
Adoption limitation	Frontier content is not an implicit adoption recommendation. Adoption requires defined value, qualification gates, controlled proof, rollback, ownership, and residual-risk acceptance.
Status boundary	Candidate Focused Field Guide describes the publication, not certification of a product, implementation, engineer, or organization.

Revision record

Version	Revision statement
1.0.0-candidate	Initial candidate living-guide edition. Published 2026-07-26; scheduled review 2026-10-26.

Search and relationship metadata

Dimension	Engineering position
Primary domain	MYTHOS-CLD - Cloud, Platform & Distributed Systems
Secondary domain	MYTHOS-NET; MYTHOS-SEC; MYTHOS-AI
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	service mesh, API gateway, Gateway API, Istio, Envoy, SPIFFE, mTLS, east-west traffic, agent gateway, multi-cluster
Canonical route	/library/field-guides/mythos-fg-cld-0005/

Living publication

Corrections, expanded laboratories, improved diagrams, authority changes, and new engineering evidence should revise this permanent publication rather than create disconnected replacements.

CANONICAL LIVING OVERVIEW GUIDE

Durable Workflows, Queues, Streams, and Sagas

A distributed-execution guide covering durable workflows, activity execution, queues, brokers, event streams, delivery guarantees, timers, signals, idempotency, transactions, sagas, compensation, replay, testing, observability, and failure recovery.

PERMANENT ID
MYTHOS-FG-CLD-0006

PUBLICATION CLASS
FIELD GUIDE

VERSION / STATUS
1.0.0-candidate / CANDIDATE

PERMANENT PUBLICATION IDENTITY

Durable Workflows, Queues, Streams, and Sagas

Document ID
MYTHOS-FG-CLD-0006

Version
1.0.0-candidate

Status
Candidate Focused Field Guide

Review date
2026-10-26

Publication position
This is a living field guide. It is designed for frequent revision while retaining a stable permanent identity. Candidate status means the content is ready for structured use and review but is not represented as an external certification or authority publication.

LEVEL	CLASS	CADENCE	EVIDENCE
L1-L4	FIELD GUIDE	QUARTERLY	SOURCE-BOUND
Foundational through frontier	Practical and educational	Interim updates as needed	Limitations remain visible

Reader orientation

Dimension	Engineering position
Primary domain	MYTHOS-CLD - Cloud, Platform & Distributed Systems
Secondary domain	MYTHOS-DAT; MYTHOS-SWE; MYTHOS-SEC
Audience	Cloud, platform, integration, software, data, SRE, and agent-runtime engineers.
Prerequisites	Distributed systems, APIs, messaging, databases, transactions, reliability, and observability.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

Choose workflow, queue, stream, scheduler, and event architectures according to state and failure semantics.

Build durable executions that recover from process, worker, dependency, and infrastructure failure.

Engineer idempotency, ordering, retries, timeouts, transactions, and compensation explicitly.

Operate workflows and event systems with replay, backpressure, evidence, and safe migration.

Validate business outcomes rather than equating delivery acknowledgement with completion.

How to use this guide

Read the guide as a progressive system rather than a disconnected encyclopedia. Foundational material establishes the mental model; practitioner sections convert it into repeatable implementation and operations; advanced sections expose architecture, scale, security, and failure behavior; frontier sections describe technologies whose evidence or maturity is still evolving.

Suggested reading path
New practitioners should follow the chapters in order and complete the laboratories. Experienced engineers may begin with the architecture, failure, validation, or frontier sections, then use the related-document map to deepen a specific topic.

Learning and assurance model

Level	Reader outcome
L1 - Foundational	Terminology, first principles, component roles, packet or state flow, and simple working examples.
L2 - Practitioner	Implementation, configuration, automation, testing, troubleshooting, and operational evidence.
L3 - Advanced	Architecture, scale, concurrency, resilience, threat models, performance, and complex migrations.
L4 - Frontier	Emerging systems, research directions, technical uncertainty, readiness gates, and adoption signals.

Contents

CHAPTER 01

Durable execution and workflow state

Define workflow code as recoverable stateful computation rather than a long-running process.

Chapter operating position
Define workflow code as recoverable stateful computation rather than a long-running process.

1.1 Workflow histories and replay

Workflow histories and replay must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Persist decisions, events, timers, signals, activity outcomes, and deterministic execution history. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for workflow histories and replay.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating workflow histories and replay as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for workflow histories and replay.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

1.2 Workflow and activity boundaries

Workflow and activity boundaries must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Keep orchestration deterministic while isolating nondeterministic I/O in retryable activities. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for workflow and activity boundaries.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
workflow:
  id: order-042
  state: payment_pending
  history_version: 18
  activities:
    reserve_inventory:
      idempotency_key: order-042:reserve
      retry: bounded-exponential
    charge_payment:
      idempotency_key: order-042:charge
      timeout_seconds: 30
  compensation:
    on_shipping_failure: [refund_payment, release_inventory]
  completion:
    require: [order_state_committed, external_effects_verified]
```

Failure patterns

- Treating workflow and activity boundaries as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for workflow and activity boundaries.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

1.3 Timers, signals, updates, and queries

Timers, signals, updates, and queries must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Handle time, external input, state mutation, and read-only inspection without losing causality. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for timers, signals, updates, and queries.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating timers, signals, updates, and queries as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for timers, signals, updates, and queries.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

1.4 Versioning and migration

Versioning and migration must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Evolve long-lived workflows with compatible code paths, markers, and controlled retirement. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for versioning and migration.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating versioning and migration as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for versioning and migration.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 01 laboratory and review

Practical laboratory

Implement a durable order workflow, terminate its worker during three different steps, and prove replay and recovery.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore durable agent missions whose plans, approvals, tools, and compensations survive model and harness replacement.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 02

Queues, brokers, and consumer engineering

Select work distribution according to ownership, visibility, acknowledgement, and retry semantics.

Chapter operating position
Select work distribution according to ownership, visibility, acknowledgement, and retry semantics.

2.1 Queue and consumer models

Queue and consumer models must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Use pull or push delivery, consumer groups, visibility, acknowledgements, leases, and horizontal scale. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for queue and consumer models.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating queue and consumer models as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for queue and consumer models.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

2.2 Retention and dead letters

Retention and dead letters must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Define limits, age, work-queue deletion, poison-message isolation, redrive, and evidence. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for retention and dead letters.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
workflow:
  id: order-042
  state: payment_pending
  history_version: 18
  activities:
    reserve_inventory:
      idempotency_key: order-042:reserve
      retry: bounded-exponential
    charge_payment:
      idempotency_key: order-042:charge
      timeout_seconds: 30
  compensation:
    on_shipping_failure: [refund_payment, release_inventory]
  completion:
    require: [order_state_committed, external_effects_verified]
```

Failure patterns

- Treating retention and dead letters as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for retention and dead letters.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

2.3 Ordering and partitioning

Ordering and partitioning must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Choose keys, partitions, affinity, sequencing, and parallelism according to domain invariants. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for ordering and partitioning.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating ordering and partitioning as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for ordering and partitioning.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

2.4 Backpressure and admission

Backpressure and admission must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Bound producers, queues, retries, consumers, dependencies, memory, and storage. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for backpressure and admission.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating backpressure and admission as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for backpressure and admission.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 02 laboratory and review

Practical laboratory
Simulate duplicate, delayed, reordered, expired, and poisoned work items across multiple consumers.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore globally scheduled work queues that combine agent, batch, GPU, and human work.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 03

Event streams and stateful processing

Treat ordered logs as durable facts that drive projections and continuous computation.

Chapter operating position
Treat ordered logs as durable facts that drive projections and continuous computation.

3.1 Topics, partitions, offsets, and retention

Topics, partitions, offsets, and retention must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Model event identity, ordering scope, replay horizon, compaction, and consumer progress. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for topics, partitions, offsets, and retention.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating topics, partitions, offsets, and retention as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for topics, partitions, offsets, and retention.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

3.2 Transactions and processing guarantees

Transactions and processing guarantees must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Use idempotent production, read-committed consumption, atomic offset updates, and exactly-once-v2 where supported. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for transactions and processing guarantees.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
workflow:
  id: order-042
  state: payment_pending
  history_version: 18
  activities:
    reserve_inventory:
      idempotency_key: order-042:reserve
      retry: bounded-exponential
    charge_payment:
      idempotency_key: order-042:charge
      timeout_seconds: 30
  compensation:
    on_shipping_failure: [refund_payment, release_inventory]
  completion:
    require: [order_state_committed, external_effects_verified]
```

Failure patterns

- Treating transactions and processing guarantees as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for transactions and processing guarantees.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

3.3 State stores and recovery

State stores and recovery must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Maintain local or remote state, changelogs, checkpoints, standby replicas, and restore. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for state stores and recovery.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating state stores and recovery as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for state stores and recovery.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

3.4 Stream-time and event-time

Stream-time and event-time must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Handle timestamps, watermarks, windows, late events, corrections, and deterministic aggregation. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for stream-time and event-time.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating stream-time and event-time as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for stream-time and event-time.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 03 laboratory and review

Practical laboratory
Build a stateful stream processor and inject consumer restart, late data, duplicate production, and partition movement.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore verifiable streaming where every derived fact carries source offsets and transformation provenance.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 04

Event contracts and interoperability

Make events understandable across producers, transports, consumers, and time.

Chapter operating position
Make events understandable across producers, transports, consumers, and time.

4.1 CloudEvents envelope

CloudEvents envelope must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Use stable IDs, source, type, subject, time, schema, trace context, and content type. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for cloudevents envelope.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating cloudevents envelope as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for cloudevents envelope.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

4.2 Domain event semantics

Domain event semantics must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Define business meaning, invariants, identity, privacy, units, causality, and compatibility. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for domain event semantics.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
workflow:
  id: order-042
  state: payment_pending
  history_version: 18
  activities:
    reserve_inventory:
      idempotency_key: order-042:reserve
      retry: bounded-exponential
    charge_payment:
      idempotency_key: order-042:charge
      timeout_seconds: 30
  compensation:
    on_shipping_failure: [refund_payment, release_inventory]
  completion:
    require: [order_state_committed, external_effects_verified]
```

Failure patterns

- Treating domain event semantics as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for domain event semantics.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

4.3 Schema governance

Schema governance must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Version schemas, defaults, optionality, evolution, validation, ownership, and deprecation. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for schema governance.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating schema governance as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for schema governance.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

4.4 Tracing and correlation

Tracing and correlation must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Propagate workflow, event, message, transaction, tenant, and business identifiers. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for tracing and correlation.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating tracing and correlation as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for tracing and correlation.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 04 laboratory and review

Practical laboratory
Create one portable event contract and transport it over HTTP, Kafka-style streams, and NATS-style messaging.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore semantic event registries that generate policies, tests, and observability conventions.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 05

Idempotency, retries, timeouts, and duplicate effects

Prevent delivery and execution uncertainty from creating incorrect external state.

Chapter operating position
Prevent delivery and execution uncertainty from creating incorrect external state.

5.1 Business idempotency keys

Business idempotency keys must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Choose stable operation identity, request scope, storage, expiry, conflict behavior, and result replay. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for business idempotency keys.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating business idempotency keys as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for business idempotency keys.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

5.2 Retry classification

Retry classification must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Separate transient, permanent, throttled, ambiguous, and unsafe-to-retry failures. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for retry classification.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
workflow:
  id: order-042
  state: payment_pending
  history_version: 18
  activities:
    reserve_inventory:
      idempotency_key: order-042:reserve
      retry: bounded-exponential
    charge_payment:
      idempotency_key: order-042:charge
      timeout_seconds: 30
  compensation:
    on_shipping_failure: [refund_payment, release_inventory]
  completion:
    require: [order_state_committed, external_effects_verified]
```

Failure patterns

- Treating retry classification as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for retry classification.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

5.3 Deadlines and cancellation

Deadlines and cancellation must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Propagate budgets, stop work, clean up resources, and prevent zombie effects. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for deadlines and cancellation.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating deadlines and cancellation as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for deadlines and cancellation.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

5.4 Inbox, outbox, and deduplication

Inbox, outbox, and deduplication must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Coordinate database state, event publication, consumer effects, and replay. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for inbox, outbox, and deduplication.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating inbox, outbox, and deduplication as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for inbox, outbox, and deduplication.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 05 laboratory and review

Practical laboratory
Implement an idempotent payment-like operation with ambiguous timeout, duplicate delivery, and safe result replay.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore cryptographically receipted effects and cross-system idempotency ledgers.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 06

Sagas, compensation, and long-running transactions

Coordinate business state when one atomic transaction cannot span participants.

Chapter operating position
Coordinate business state when one atomic transaction cannot span participants.

6.1 Orchestration and choreography

Orchestration and choreography must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Compare central workflow state with event-driven participant coordination and failure visibility. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for orchestration and choreography.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating orchestration and choreography as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for orchestration and choreography.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

6.2 Compensating actions

Compensating actions must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Define reversibility, semantic undo, pivot steps, irreversibility, and human intervention. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for compensating actions.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
workflow:
  id: order-042
  state: payment_pending
  history_version: 18
  activities:
    reserve_inventory:
      idempotency_key: order-042:reserve
      retry: bounded-exponential
    charge_payment:
      idempotency_key: order-042:charge
      timeout_seconds: 30
  compensation:
    on_shipping_failure: [refund_payment, release_inventory]
  completion:
    require: [order_state_committed, external_effects_verified]
```

Failure patterns

- Treating compensating actions as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for compensating actions.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

6.3 Partial failure and reconciliation

Partial failure and reconciliation must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Detect incomplete steps, stale participants, orphan effects, and divergent views. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for partial failure and reconciliation.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating partial failure and reconciliation as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for partial failure and reconciliation.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

6.4 Human tasks and approvals

Human tasks and approvals must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Pause durably for decisions, deadlines, escalations, evidence, and alternate paths. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for human tasks and approvals.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating human tasks and approvals as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for human tasks and approvals.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 06 laboratory and review

Practical laboratory
Model an order saga with inventory, payment, shipping, timeout, cancellation, and nonreversible actions.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore machine-generated sagas constrained by typed effects and verified compensation plans.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 07

Workflow testing, observability, and operations

Make durable systems reproducible and diagnosable across long time horizons.

Chapter operating position
Make durable systems reproducible and diagnosable across long time horizons.

7.1 Deterministic and time-skipping tests

Deterministic and time-skipping tests must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Test timers, retries, signals, activity failures, history replay, and versions quickly. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for deterministic and time-skipping tests.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating deterministic and time-skipping tests as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for deterministic and time-skipping tests.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

7.2 Operational telemetry

Operational telemetry must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Track workflow state, queue depth, lag, attempts, deadlines, history size, effects, and outcomes. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for operational telemetry.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
workflow:
  id: order-042
  state: payment_pending
  history_version: 18
  activities:
    reserve_inventory:
      idempotency_key: order-042:reserve
      retry: bounded-exponential
    charge_payment:
      idempotency_key: order-042:charge
      timeout_seconds: 30
  compensation:
    on_shipping_failure: [refund_payment, release_inventory]
  completion:
    require: [order_state_committed, external_effects_verified]
```

Failure patterns

- Treating operational telemetry as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for operational telemetry.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

7.3 Replay and repair

Replay and repair must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Reprocess safely, reset consumers, rebuild projections, patch state, and preserve audit evidence. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for replay and repair.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating replay and repair as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for replay and repair.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

7.4 Capacity and failure domains

Capacity and failure domains must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Plan partitions, workers, brokers, storage, regions, quotas, and dependency saturation. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for capacity and failure domains.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating capacity and failure domains as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for capacity and failure domains.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 07 laboratory and review

Practical laboratory
Run a chaos exercise involving broker loss, worker loss, dependency outage, replay, and projection rebuild.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore self-healing workflow fabrics with independent repair agents and immutable history.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 08

Architecture selection and governance

Choose the simplest durable mechanism that satisfies domain semantics and operations.

Chapter operating position
Choose the simplest durable mechanism that satisfies domain semantics and operations.

8.1 Workflow versus queue versus stream

Workflow versus queue versus stream must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Compare state, duration, ordering, replay, coordination, throughput, latency, and ownership. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for workflow versus queue versus stream.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating workflow versus queue versus stream as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for workflow versus queue versus stream.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

8.2 Managed and self-hosted platforms

Managed and self-hosted platforms must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Assess control, sovereignty, upgrade, cost, integration, support, and exit. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for managed and self-hosted platforms.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
workflow:
  id: order-042
  state: payment_pending
  history_version: 18
  activities:
    reserve_inventory:
      idempotency_key: order-042:reserve
      retry: bounded-exponential
    charge_payment:
      idempotency_key: order-042:charge
      timeout_seconds: 30
  compensation:
    on_shipping_failure: [refund_payment, release_inventory]
  completion:
    require: [order_state_committed, external_effects_verified]
```

Failure patterns

- Treating managed and self-hosted platforms as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for managed and self-hosted platforms.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

8.3 Evidence and acceptance

Evidence and acceptance must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Require business outcome, state reconciliation, failure tests, and recovery proof. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for evidence and acceptance.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating evidence and acceptance as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for evidence and acceptance.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

8.4 Lifecycle and retirement

Lifecycle and retirement must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Migrate histories, events, schemas, consumers, state stores, and retained evidence. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for lifecycle and retirement.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating lifecycle and retirement as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for lifecycle and retirement.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 08 laboratory and review

Practical laboratory
Create an architecture decision for a multi-day regulated process using at least three execution patterns.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Develop a unified durable-execution control plane spanning workflows, events, queues, agents, and people.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

Integrated learning roadmap

The guide is intended to be revisited. First establish fluency, then build a small implementation, operate it under failure, automate repeatable work, and finally evaluate emerging techniques against a stable baseline. Progress is demonstrated through evidence rather than reading completion alone.

Stage	Demonstrated capability
Foundation	Explain the system from first principles and trace its critical path without relying on product terminology.
Build	Implement the smallest representative system with explicit contracts, identities, telemetry, and tests.
Operate	Diagnose injected failures, recover safely, and preserve evidence of the causal chain.
Automate	Convert a proven manual workflow into bounded, idempotent, observable automation.
Architect	Design for scale, resilience, security, interoperability, and lifecycle change.
Explore	Evaluate frontier techniques using stated hypotheses, limitations, and adoption gates.

Source authority register

Source policy
Official standards, specifications, and project documentation remain with their issuing organizations. Mythos Systems provides original engineering interpretation, synthesis, implementation guidance, and relationship mapping. Source verification dates do not guarantee that an authority has not changed since review.

01. Temporal - Temporal Documentation - Workflow Execution
Role: Durable workflow state, replay, retries, timers, signals, and recovery.
Verified: 2026-07-26
Official source: https://docs.temporal.io/workflow-execution

02. Temporal - Activity Execution
Role: Activity retries, timeouts, heartbeats, cancellation, and failure handling.
Verified: 2026-07-26
Official source: https://docs.temporal.io/activity-execution

03. Apache Kafka - Apache Kafka Documentation
Role: Event streaming, durable topics, producers, consumers, transactions, and processing guarantees.
Verified: 2026-07-26
Official source: https://kafka.apache.org/documentation/

04. Apache Kafka - Kafka Streams
Role: Stateful stream processing, topology, state stores, recovery, and exactly-once-v2 semantics.
Verified: 2026-07-26
Official source: https://kafka.apache.org/documentation/streams/

05. NATS - JetStream Documentation
Role: Persistent streams, consumers, replay, retention, work queues, and replication.
Verified: 2026-07-26
Official source: https://docs.nats.io/nats-concepts/jetstream

06. CloudEvents - CloudEvents Specification
Role: Portable event envelope, attributes, bindings, and interoperability.
Verified: 2026-07-26
Official source: https://cloudevents.io/

07. CNCF Serverless Workflow - Serverless Workflow Specification
Role: Portable durable workflow and event-orchestration definitions.
Verified: 2026-07-26
Official source: https://serverlessworkflow.io/

Authority application and refresh triggers

Authority class	Required library action
Protocols and specifications	Recheck interoperability, security considerations, defaults, and migration guidance when a referenced specification changes.
Security and assurance frameworks	Update control mappings, evidence expectations, risk language, and assessment procedures after material framework revision.
Languages, runtimes, and projects	Re-run examples, tests, deprecation checks, and operational recommendations against supported releases.
Benchmarks and evaluations	Re-execute claims when workloads, methods, models, compilers, drivers, hardware, or scoring rules change.
Operational evidence	Incorporate validated incidents, failure tests, reader corrections, and measured implementation outcomes into the next revision.

Limitations, freshness, and revision control

Boundary	Statement
Publication limitation	This guide synthesizes broad engineering practice. It does not replace product documentation, legal advice, safety certification, or environment-specific design review.
Evidence limitation	Not every pattern is benchmarked in every environment. Examples illustrate engineering methods and must be validated against actual versions, workloads, hardware, policy, and failure conditions.
Freshness limitation	Vendor behavior, software libraries, AI models, security threats, and emerging standards can change quickly. Volatile claims require source revalidation before operational adoption.
Adoption limitation	Frontier content is not an implicit adoption recommendation. Adoption requires defined value, qualification gates, controlled proof, rollback, ownership, and residual-risk acceptance.
Status boundary	Candidate Focused Field Guide describes the publication, not certification of a product, implementation, engineer, or organization.

Revision record

Version	Revision statement
1.0.0-candidate	Initial candidate living-guide edition. Published 2026-07-26; scheduled review 2026-10-26.

Search and relationship metadata

Dimension	Engineering position
Primary domain	MYTHOS-CLD - Cloud, Platform & Distributed Systems
Secondary domain	MYTHOS-DAT; MYTHOS-SWE; MYTHOS-SEC
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	durable workflows, queues, streams, sagas, Temporal, Kafka, NATS JetStream, idempotency, event contracts, compensation
Canonical route	/library/field-guides/mythos-fg-cld-0006/

Living publication

Corrections, expanded laboratories, improved diagrams, authority changes, and new engineering evidence should revise this permanent publication rather than create disconnected replacements.

CANONICAL LIVING OVERVIEW GUIDE

Edge Computing, Remote Sites, and Distributed Control Planes

An edge-systems guide covering remote-site architecture, fog and MEC models, autonomous local operation, distributed control planes, intermittent links, fleet lifecycle, data locality, security, observability, failover, synchronization, and recovery.

PERMANENT ID
MYTHOS-FG-CLD-0007

PUBLICATION CLASS
FIELD GUIDE

VERSION / STATUS
1.0.0-candidate / CANDIDATE

PERMANENT PUBLICATION IDENTITY

Edge Computing, Remote Sites, and Distributed Control Planes

Document ID
MYTHOS-FG-CLD-0007

Version
1.0.0-candidate

Status
Candidate Focused Field Guide

Review date
2026-10-26

Publication position
This is a living field guide. It is designed for frequent revision while retaining a stable permanent identity. Candidate status means the content is ready for structured use and review but is not represented as an external certification or authority publication.

LEVEL	CLASS	CADENCE	EVIDENCE
L1-L4	FIELD GUIDE	QUARTERLY	SOURCE-BOUND
Foundational through frontier	Practical and educational	Interim updates as needed	Limitations remain visible

Reader orientation

Dimension	Engineering position
Primary domain	MYTHOS-CLD - Cloud, Platform & Distributed Systems
Secondary domain	MYTHOS-NET; MYTHOS-SEC; MYTHOS-DAT
Audience	Edge, cloud, network, platform, OT, IoT, security, and remote-operations engineers.
Prerequisites	Cloud, networking, Linux, containers, Kubernetes, distributed systems, identity, and site operations.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

Design edge systems around latency, autonomy, data locality, survivability, and constrained operations.

Separate global intent from local authority and safe disconnected behavior.

Manage large fleets of heterogeneous remote nodes without assuming continuous connectivity.

Protect edge identities, software, models, data, devices, and physical environments.

Reconcile remote state and evidence after outages without overwriting legitimate local changes.

How to use this guide

Read the guide as a progressive system rather than a disconnected encyclopedia. Foundational material establishes the mental model; practitioner sections convert it into repeatable implementation and operations; advanced sections expose architecture, scale, security, and failure behavior; frontier sections describe technologies whose evidence or maturity is still evolving.

Suggested reading path
New practitioners should follow the chapters in order and complete the laboratories. Experienced engineers may begin with the architecture, failure, validation, or frontier sections, then use the related-document map to deepen a specific topic.

Learning and assurance model

Level	Reader outcome
L1 - Foundational	Terminology, first principles, component roles, packet or state flow, and simple working examples.
L2 - Practitioner	Implementation, configuration, automation, testing, troubleshooting, and operational evidence.
L3 - Advanced	Architecture, scale, concurrency, resilience, threat models, performance, and complex migrations.
L4 - Frontier	Emerging systems, research directions, technical uncertainty, readiness gates, and adoption signals.

Contents

CHAPTER 01

Edge, fog, MEC, and remote-site models

Define where computation, data, and control should execute.

Chapter operating position
Define where computation, data, and control should execute.

1.1 Edge and fog hierarchy

Edge and fog hierarchy must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Place device, gateway, site, regional, and central functions according to latency, bandwidth, state, and trust. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for edge and fog hierarchy.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating edge and fog hierarchy as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for edge and fog hierarchy.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

1.2 MEC hosts and management

MEC hosts and management must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Map applications, platform services, virtualization, orchestration, and network information at the access edge. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for mec hosts and management.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
edge_site:
  site_id: remote-017
  desired_generation: 42
  local_generation: 43
  connectivity: disconnected
  local_override:
    reason: safety-mode
    expires_at: 2026-07-27T12:00:00Z
  sync_policy:
    commands: priority
    telemetry: summarize
    artifacts: resumable-delta
  reconcile_on_connect: preserve-local-emergency-decision
```

Failure patterns

- Treating mec hosts and management as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for mec hosts and management.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

1.3 Remote-site archetypes

Remote-site archetypes must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Distinguish branch, industrial, retail, clinical, vehicle, maritime, military, and field deployments. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for remote-site archetypes.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating remote-site archetypes as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for remote-site archetypes.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

1.4 Application placement

Application placement must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Choose local, regional, cloud, or split execution based on deadline, data gravity, safety, and cost. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for application placement.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating application placement as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for application placement.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 01 laboratory and review

Practical laboratory
Create a placement model for a latency-sensitive, privacy-sensitive, intermittently connected application.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore heterogeneous compute placement across CPU, GPU, NPU, radio, and photonic edge fabrics.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 02

Distributed control-plane architecture

Separate global desired state from local admission, execution, and emergency authority.

Chapter operating position
Separate global desired state from local admission, execution, and emergency authority.

2.1 Global policy and intent

Global policy and intent must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Publish signed configurations, workload definitions, models, routes, identities, and lifecycle policy. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for global policy and intent.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating global policy and intent as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for global policy and intent.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

2.2 Site-local controllers

Site-local controllers must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Reconcile workloads, devices, storage, network, and safety state without central availability. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for site-local controllers.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
edge_site:
  site_id: remote-017
  desired_generation: 42
  local_generation: 43
  connectivity: disconnected
  local_override:
    reason: safety-mode
    expires_at: 2026-07-27T12:00:00Z
  sync_policy:
    commands: priority
    telemetry: summarize
    artifacts: resumable-delta
  reconcile_on_connect: preserve-local-emergency-decision
```

Failure patterns

- Treating site-local controllers as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for site-local controllers.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

2.3 Hierarchical and federated management

Hierarchical and federated management must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Aggregate sites through regions, domains, tenants, and sovereign boundaries. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for hierarchical and federated management.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating hierarchical and federated management as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for hierarchical and federated management.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

2.4 Conflict and precedence

Conflict and precedence must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Resolve central updates, local overrides, emergency modes, stale generations, and operator decisions. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for conflict and precedence.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating conflict and precedence as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for conflict and precedence.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 02 laboratory and review

Practical laboratory
Simulate a central policy change during a 12-hour disconnection and reconcile it with an emergency local override.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore verifiable hierarchical control planes with signed intent and local proof of enforcement.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 03

Connectivity, delay, and disruption tolerance

Operate under intermittent, expensive, asymmetric, and high-latency links.

Chapter operating position
Operate under intermittent, expensive, asymmetric, and high-latency links.

3.1 Link characterization

Link characterization must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Measure bandwidth, latency, loss, jitter, asymmetry, outages, data caps, and path changes. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for link characterization.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating link characterization as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for link characterization.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

3.2 Store-and-forward transport

Store-and-forward transport must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Queue commands, events, updates, telemetry, and artifacts with priorities and expiry. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for store-and-forward transport.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
edge_site:
  site_id: remote-017
  desired_generation: 42
  local_generation: 43
  connectivity: disconnected
  local_override:
    reason: safety-mode
    expires_at: 2026-07-27T12:00:00Z
  sync_policy:
    commands: priority
    telemetry: summarize
    artifacts: resumable-delta
  reconcile_on_connect: preserve-local-emergency-decision
```

Failure patterns

- Treating store-and-forward transport as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for store-and-forward transport.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

3.3 Synchronization windows

Synchronization windows must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Schedule large transfers, use delta updates, compression, deduplication, and resumable chunks. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for synchronization windows.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating synchronization windows as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for synchronization windows.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

3.4 Multi-path and failover

Multi-path and failover must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Use wired, wireless, cellular, satellite, mesh, and removable media with policy. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for multi-path and failover.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating multi-path and failover as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for multi-path and failover.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 03 laboratory and review

Practical laboratory

Test control and data synchronization across packet loss, long outage, expensive backup link, and limited transfer windows.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore delay-tolerant networking and autonomous data mules for disconnected infrastructure.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 04

Edge workload and fleet lifecycle

Deploy, update, observe, and retire software and models across heterogeneous nodes.

Chapter operating position
Deploy, update, observe, and retire software and models across heterogeneous nodes.

4.1 Node identity and inventory

Node identity and inventory must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Track hardware, firmware, OS, runtime, accelerators, location class, owner, and lifecycle. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for node identity and inventory.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating node identity and inventory as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for node identity and inventory.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

4.2 Policy-based placement

Policy-based placement must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Match workloads to capabilities, data, network, trust, power, and environmental limits. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for policy-based placement.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
edge_site:
  site_id: remote-017
  desired_generation: 42
  local_generation: 43
  connectivity: disconnected
  local_override:
    reason: safety-mode
    expires_at: 2026-07-27T12:00:00Z
  sync_policy:
    commands: priority
    telemetry: summarize
    artifacts: resumable-delta
  reconcile_on_connect: preserve-local-emergency-decision
```

Failure patterns

- Treating policy-based placement as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for policy-based placement.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

4.3 Signed update and rollback

Signed update and rollback must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Distribute images, models, configuration, certificates, and firmware with canaries and anti-rollback. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for signed update and rollback.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating signed update and rollback as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for signed update and rollback.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

4.4 Autonomous remediation

Autonomous remediation must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Restart, isolate, roll back, reschedule, or enter safe mode according to local evidence. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for autonomous remediation.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating autonomous remediation as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for autonomous remediation.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 04 laboratory and review

Practical laboratory
Roll out a signed application and model update to fifty simulated nodes with two failures and one revoked version.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore self-maintaining edge fleets with attested nodes and peer-assisted content distribution.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 05

Local data, state, and synchronization

Keep essential data useful locally while maintaining global consistency expectations.

Chapter operating position
Keep essential data useful locally while maintaining global consistency expectations.

5.1 Local authoritative state

Local authoritative state must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Define which records the site owns, which are cached, and which require central approval. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for local authoritative state.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating local authoritative state as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for local authoritative state.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

5.2 Replication and conflict

Replication and conflict must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Use logs, CRDTs, version vectors, domain merges, and human review according to semantics. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for replication and conflict.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
edge_site:
  site_id: remote-017
  desired_generation: 42
  local_generation: 43
  connectivity: disconnected
  local_override:
    reason: safety-mode
    expires_at: 2026-07-27T12:00:00Z
  sync_policy:
    commands: priority
    telemetry: summarize
    artifacts: resumable-delta
  reconcile_on_connect: preserve-local-emergency-decision
```

Failure patterns

- Treating replication and conflict as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for replication and conflict.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

5.3 Filtering and aggregation

Filtering and aggregation must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Process data locally, transmit derived results, preserve raw evidence, and control loss. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for filtering and aggregation.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating filtering and aggregation as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for filtering and aggregation.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

5.4 Retention and storage pressure

Retention and storage pressure must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Prioritize data, rotate buffers, protect critical evidence, and handle full disks. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for retention and storage pressure.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating retention and storage pressure as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for retention and storage pressure.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 05 laboratory and review

Practical laboratory
Design a site database that remains writable offline and reconciles inventory and telemetry after reconnection.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore edge knowledge fabrics that exchange verified summaries instead of unrestricted raw data.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 06

Security, physical exposure, and zero trust

Protect systems deployed outside controlled data centers.

Chapter operating position
Protect systems deployed outside controlled data centers.

6.1 Device and workload identity

Device and workload identity must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Use hardware roots, secure boot, attestation, certificates, workload identity, and rotation. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for device and workload identity.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating device and workload identity as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for device and workload identity.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

6.2 Physical and supply-chain threats

Physical and supply-chain threats must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Address theft, tampering, replacement, debug access, malicious peripherals, and hostile maintenance. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for physical and supply-chain threats.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
edge_site:
  site_id: remote-017
  desired_generation: 42
  local_generation: 43
  connectivity: disconnected
  local_override:
    reason: safety-mode
    expires_at: 2026-07-27T12:00:00Z
  sync_policy:
    commands: priority
    telemetry: summarize
    artifacts: resumable-delta
  reconcile_on_connect: preserve-local-emergency-decision
```

Failure patterns

- Treating physical and supply-chain threats as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for physical and supply-chain threats.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

6.3 Network and capability policy

Network and capability policy must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Treat every link as untrusted, limit egress, segment devices, and broker credentials. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for network and capability policy.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating network and capability policy as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for network and capability policy.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

6.4 Incident containment

Incident containment must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Quarantine nodes, revoke identities, preserve evidence, operate safely, and rebuild trust. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for incident containment.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating incident containment as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for incident containment.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 06 laboratory and review

Practical laboratory

Threat-model a remote node that is stolen, cloned, and later reconnects with valid but revoked credentials.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore confidential edge computing with attestation-gated models and data.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 07

Edge observability and remote operations

Diagnose sites when bandwidth and local expertise are limited.

Chapter operating position
Diagnose sites when bandwidth and local expertise are limited.

7.1 Local telemetry pipeline

Local telemetry pipeline must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Collect health, workload, network, power, storage, sensor, security, and user outcome data. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for local telemetry pipeline.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating local telemetry pipeline as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for local telemetry pipeline.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

7.2 Priority and summarization

Priority and summarization must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Transmit critical alerts first, aggregate metrics, sample traces, and retain forensic detail locally. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for priority and summarization.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
edge_site:
  site_id: remote-017
  desired_generation: 42
  local_generation: 43
  connectivity: disconnected
  local_override:
    reason: safety-mode
    expires_at: 2026-07-27T12:00:00Z
  sync_policy:
    commands: priority
    telemetry: summarize
    artifacts: resumable-delta
  reconcile_on_connect: preserve-local-emergency-decision
```

Failure patterns

- Treating priority and summarization as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for priority and summarization.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

7.3 Remote diagnostics

Remote diagnostics must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Provide safe shells, logs, captures, bundles, commands, approvals, and session recording. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for remote diagnostics.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating remote diagnostics as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for remote diagnostics.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

7.4 Local operator experience

Local operator experience must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Offer clear status, offline runbooks, manual controls, and recovery guidance. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for local operator experience.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating local operator experience as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for local operator experience.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 07 laboratory and review

Practical laboratory
Diagnose a remote service failure using only a constrained telemetry bundle and one ten-minute maintenance window.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore agent-assisted remote operations with bounded local authority and human escalation.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 08

Resilience, continuity, and fleet governance

Prepare remote sites for prolonged isolation and coordinated recovery.

Chapter operating position
Prepare remote sites for prolonged isolation and coordinated recovery.

8.1 Power and environmental resilience

Power and environmental resilience must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Plan UPS, generators, thermal limits, ruggedization, maintenance, and safe shutdown. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for power and environmental resilience.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating power and environmental resilience as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for power and environmental resilience.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

8.2 Local service continuity

Local service continuity must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Define essential functions, degraded modes, cached identity, local DNS, and dependency substitution. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for local service continuity.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
edge_site:
  site_id: remote-017
  desired_generation: 42
  local_generation: 43
  connectivity: disconnected
  local_override:
    reason: safety-mode
    expires_at: 2026-07-27T12:00:00Z
  sync_policy:
    commands: priority
    telemetry: summarize
    artifacts: resumable-delta
  reconcile_on_connect: preserve-local-emergency-decision
```

Failure patterns

- Treating local service continuity as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for local service continuity.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

8.3 Fleet recovery

Fleet recovery must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Rebuild nodes, restore site data, reissue identities, validate workloads, and reconcile central records. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for fleet recovery.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating fleet recovery as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for fleet recovery.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

8.4 Governance and adoption

Governance and adoption must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Assign site, platform, security, network, application, vendor, and business responsibility. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the global intent, site policy, remote node, local execution, data state, link disruption, synchronization, evidence, and service continuity chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for governance and adoption.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating governance and adoption as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for governance and adoption.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 08 laboratory and review

Practical laboratory
Run a remote-site disaster exercise involving power loss, WAN outage, storage failure, and replacement hardware.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Develop sovereign edge zones capable of months-long autonomous operation and later verifiable reconciliation.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

Integrated learning roadmap

The guide is intended to be revisited. First establish fluency, then build a small implementation, operate it under failure, automate repeatable work, and finally evaluate emerging techniques against a stable baseline. Progress is demonstrated through evidence rather than reading completion alone.

Stage	Demonstrated capability
Foundation	Explain the system from first principles and trace its critical path without relying on product terminology.
Build	Implement the smallest representative system with explicit contracts, identities, telemetry, and tests.
Operate	Diagnose injected failures, recover safely, and preserve evidence of the causal chain.
Automate	Convert a proven manual workflow into bounded, idempotent, observable automation.
Architect	Design for scale, resilience, security, interoperability, and lifecycle change.
Explore	Evaluate frontier techniques using stated hypotheses, limitations, and adoption gates.

Source authority register

Source policy
Official standards, specifications, and project documentation remain with their issuing organizations. Mythos Systems provides original engineering interpretation, synthesis, implementation guidance, and relationship mapping. Source verification dates do not guarantee that an authority has not changed since review.

01. NIST - SP 500-325 - Fog Computing Conceptual Model
Role: Fog and edge computing actors, layers, nodes, services, and management concepts.
Verified: 2026-07-26
Official source: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.500-325.pdf

02. ETSI - GS MEC 002 V3.2.1 - MEC Requirements
Role: Requirements for multi-access edge-computing frameworks and deployments.
Verified: 2026-07-26
Official source: https://www.etsi.org/deliver/etsi_gs/MEC/001_099/002/03.02.01_60/gs_MEC002v030201p.pdf

03. ETSI - GS MEC 003 V3.1.1 - MEC Framework and Reference Architecture
Role: MEC hosts, platform, management, orchestration, services, and reference points.
Verified: 2026-07-26
Official source: https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/03.01.01_60/gs_mec003v030101p.pdf

04. ETSI - GR MEC 041 V3.1.1 - Study on MEC Security
Role: Edge trust, isolation, identity, platform, network, and operational security considerations.
Verified: 2026-07-26
Official source: https://www.etsi.org/deliver/etsi_gr/MEC/001_099/041/03.01.01_60/gr_MEC041v030101p.pdf

05. Open Horizon - Open Horizon Documentation
Role: Autonomous lifecycle management for distributed edge nodes and machine-learning assets.
Verified: 2026-07-26
Official source: https://open-horizon.github.io/

06. NIST - SP 800-207 - Zero Trust Architecture
Role: Resource-centric access, continuous authorization, policy enforcement, and distributed trust.
Verified: 2026-07-26
Official source: https://csrc.nist.gov/pubs/sp/800/207/final

07. OpenTelemetry - OpenTelemetry
Role: Vendor-neutral traces, metrics, logs, baggage, APIs, SDKs, and collectors.
Verified: 2026-07-26
Official source: https://opentelemetry.io/

Authority application and refresh triggers

Authority class	Required library action
Protocols and specifications	Recheck interoperability, security considerations, defaults, and migration guidance when a referenced specification changes.
Security and assurance frameworks	Update control mappings, evidence expectations, risk language, and assessment procedures after material framework revision.
Languages, runtimes, and projects	Re-run examples, tests, deprecation checks, and operational recommendations against supported releases.
Benchmarks and evaluations	Re-execute claims when workloads, methods, models, compilers, drivers, hardware, or scoring rules change.
Operational evidence	Incorporate validated incidents, failure tests, reader corrections, and measured implementation outcomes into the next revision.

Limitations, freshness, and revision control

Boundary	Statement
Publication limitation	This guide synthesizes broad engineering practice. It does not replace product documentation, legal advice, safety certification, or environment-specific design review.
Evidence limitation	Not every pattern is benchmarked in every environment. Examples illustrate engineering methods and must be validated against actual versions, workloads, hardware, policy, and failure conditions.
Freshness limitation	Vendor behavior, software libraries, AI models, security threats, and emerging standards can change quickly. Volatile claims require source revalidation before operational adoption.
Adoption limitation	Frontier content is not an implicit adoption recommendation. Adoption requires defined value, qualification gates, controlled proof, rollback, ownership, and residual-risk acceptance.
Status boundary	Candidate Focused Field Guide describes the publication, not certification of a product, implementation, engineer, or organization.

Revision record

Version	Revision statement
1.0.0-candidate	Initial candidate living-guide edition. Published 2026-07-26; scheduled review 2026-10-26.

Search and relationship metadata

Dimension	Engineering position
Primary domain	MYTHOS-CLD - Cloud, Platform & Distributed Systems
Secondary domain	MYTHOS-NET; MYTHOS-SEC; MYTHOS-DAT
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	edge computing, remote sites, distributed control plane, MEC, fog computing, offline operation, fleet management, edge security, intermittent connectivity, site autonomy
Canonical route	/library/field-guides/mythos-fg-cld-0007/

Living publication

Corrections, expanded laboratories, improved diagrams, authority changes, and new engineering evidence should revise this permanent publication rather than create disconnected replacements.

CANONICAL LIVING OVERVIEW GUIDE

DePIN and Verifiable Distributed Infrastructure

A critical engineering guide to decentralized physical infrastructure networks, including participants, resources, incentives, identity, attestations, verifiable service, coordination, ledgers, privacy, economics, governance, attacks, and adoption limits.

PERMANENT ID
MYTHOS-FG-CLD-0008

PUBLICATION CLASS
FIELD GUIDE

VERSION / STATUS
1.0.0-candidate / CANDIDATE

PERMANENT PUBLICATION IDENTITY

DePIN and Verifiable Distributed Infrastructure

Document ID
MYTHOS-FG-CLD-0008

Version
1.0.0-candidate

Status
Candidate Focused Field Guide

Review date
2026-10-26

Publication position
This is a living field guide. It is designed for frequent revision while retaining a stable permanent identity. Candidate status means the content is ready for structured use and review but is not represented as an external certification or authority publication.

LEVEL	CLASS	CADENCE	EVIDENCE
L1-L4	FIELD GUIDE	QUARTERLY	SOURCE-BOUND
Foundational through frontier	Practical and educational	Interim updates as needed	Limitations remain visible

Reader orientation

Dimension	Engineering position
Primary domain	MYTHOS-CLD - Cloud, Platform & Distributed Systems
Secondary domain	MYTHOS-EMT; MYTHOS-SEC; MYTHOS-DAT
Audience	Distributed-systems, infrastructure, security, cryptography, economics, edge, and emerging-technology teams.
Prerequisites	Distributed systems, networking, cryptography, identity, economics, edge computing, and governance.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

Separate useful decentralized infrastructure from token, marketing, and governance claims.

Model resources, operators, clients, verifiers, coordinators, ledgers, and incentive flows explicitly.

Design verifiable service claims using identity, attestations, measurements, receipts, and audit.

Evaluate Sybil, collusion, spoofing, manipulation, privacy, and concentration risks.

Make adoption decisions based on service quality, economics, governance, reversibility, and evidence.

How to use this guide

Read the guide as a progressive system rather than a disconnected encyclopedia. Foundational material establishes the mental model; practitioner sections convert it into repeatable implementation and operations; advanced sections expose architecture, scale, security, and failure behavior; frontier sections describe technologies whose evidence or maturity is still evolving.

Suggested reading path
New practitioners should follow the chapters in order and complete the laboratories. Experienced engineers may begin with the architecture, failure, validation, or frontier sections, then use the related-document map to deepen a specific topic.

Learning and assurance model

Level	Reader outcome
L1 - Foundational	Terminology, first principles, component roles, packet or state flow, and simple working examples.
L2 - Practitioner	Implementation, configuration, automation, testing, troubleshooting, and operational evidence.
L3 - Advanced	Architecture, scale, concurrency, resilience, threat models, performance, and complex migrations.
L4 - Frontier	Emerging systems, research directions, technical uncertainty, readiness gates, and adoption signals.

Contents

CHAPTER 01

DePIN architecture and participant model

Define the infrastructure service before defining its token or network story.

Chapter operating position
Define the infrastructure service before defining its token or network story.

1.1 Physical resource classes

Physical resource classes must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Model compute, storage, wireless, sensing, energy, mobility, mapping, bandwidth, and edge capacity. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for physical resource classes.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating physical resource classes as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for physical resource classes.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

1.2 Actors and roles

Actors and roles must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Identify resource providers, clients, coordinators, verifiers, delegators, developers, governors, and auditors. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for actors and roles.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
service_receipt:
  resource_id: did:example:edge-node-9
  service_contract: compute-v2
  workload_digest: sha256:...
  measurement:
    started_at: 2026-07-26T18:00:00Z
    completed_at: 2026-07-26T18:04:12Z
    result_digest: sha256:...
  verification:
    independent_witnesses: 2
    transparency_receipt: required
  settlement:
    payable_only_if: quality_and_receipt_verified
```

Failure patterns

- Treating actors and roles as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for actors and roles.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

1.3 Control and data paths

Control and data paths must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Trace discovery, admission, scheduling, service delivery, measurement, settlement, and dispute. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for control and data paths.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating control and data paths as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for control and data paths.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

1.4 Trust and failure boundaries

Trust and failure boundaries must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Separate claims that require cryptography, measurement, reputation, governance, or legal enforcement. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for trust and failure boundaries.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating trust and failure boundaries as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for trust and failure boundaries.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 01 laboratory and review

Practical laboratory

Create an architecture for a decentralized edge-compute network and identify every unverifiable assumption.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore composable DePIN services that expose standardized capabilities to agent and cloud control planes.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 02

Identity, credentials, and participant admission

Bind resources and operators to stable, privacy-aware identities and qualifications.

Chapter operating position
Bind resources and operators to stable, privacy-aware identities and qualifications.

2.1 Device and operator identity

Device and operator identity must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Use keys, hardware roots, DIDs, certificates, accounts, organizations, and lifecycle. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for device and operator identity.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating device and operator identity as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for device and operator identity.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

2.2 Verifiable credentials

Verifiable credentials must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Represent capability, ownership, location class, compliance, maintenance, and authorization claims. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for verifiable credentials.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
service_receipt:
  resource_id: did:example:edge-node-9
  service_contract: compute-v2
  workload_digest: sha256:...
  measurement:
    started_at: 2026-07-26T18:00:00Z
    completed_at: 2026-07-26T18:04:12Z
    result_digest: sha256:...
  verification:
    independent_witnesses: 2
    transparency_receipt: required
  settlement:
    payable_only_if: quality_and_receipt_verified
```

Failure patterns

- Treating verifiable credentials as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for verifiable credentials.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

2.3 Admission and revocation

Admission and revocation must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Evaluate proof, stake, reputation, audits, challenges, appeals, and removal. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for admission and revocation.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating admission and revocation as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for admission and revocation.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

2.4 Privacy and unlinkability

Privacy and unlinkability must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Minimize participant correlation, location exposure, customer leakage, and unnecessary public metadata. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for privacy and unlinkability.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating privacy and unlinkability as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for privacy and unlinkability.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 02 laboratory and review

Practical laboratory
Design admission for a resource provider and test forged credentials, cloned devices, revoked operators, and privacy leakage.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore selective-disclosure infrastructure credentials and post-quantum decentralized identity.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 03

Resource discovery, scheduling, and service contracts

Match clients to distributed resources through typed requirements and measurable obligations.

Chapter operating position

Match clients to distributed resources through typed requirements and measurable obligations.

3.1 Capability advertisement

Capability advertisement must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Publish resource type, capacity, location class, software, security, price, availability, and restrictions. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for capability advertisement.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating capability advertisement as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for capability advertisement.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

3.2 Matching and scheduling

Matching and scheduling must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Select providers by latency, capability, trust, sovereignty, cost, reputation, and diversity. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for matching and scheduling.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
service_receipt:
  resource_id: did:example:edge-node-9
  service_contract: compute-v2
  workload_digest: sha256:...
  measurement:
    started_at: 2026-07-26T18:00:00Z
    completed_at: 2026-07-26T18:04:12Z
    result_digest: sha256:...
  verification:
    independent_witnesses: 2
    transparency_receipt: required
  settlement:
    payable_only_if: quality_and_receipt_verified
```

Failure patterns

- Treating matching and scheduling as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for matching and scheduling.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

3.3 Service-level contracts

Service-level contracts must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Define work, duration, data, quality, availability, measurement, payment, dispute, and termination. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for service-level contracts.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating service-level contracts as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for service-level contracts.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

3.4 Multi-provider composition

Multi-provider composition must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Split, replicate, hedge, quorum, route, or fail over work across independent providers. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for multi-provider composition.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating multi-provider composition as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for multi-provider composition.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 03 laboratory and review

Practical laboratory
Schedule a workload across ten providers while enforcing location, latency, trust, and concentration limits.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore market-based schedulers constrained by verifiable performance and public-interest policy.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 04

Measurement, attestation, and proof of service

Convert physical or digital work into defensible service evidence.

Chapter operating position
Convert physical or digital work into defensible service evidence.

4.1 Measurement architecture

Measurement architecture must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Collect resource identity, time, location class, workload, capacity, output, availability, and quality. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for measurement architecture.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating measurement architecture as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for measurement architecture.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

4.2 Attestation and signed statements

Attestation and signed statements must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Bind measurements to hardware, software, operator, verifier, and policy. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for attestation and signed statements.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
service_receipt:
  resource_id: did:example:edge-node-9
  service_contract: compute-v2
  workload_digest: sha256:...
  measurement:
    started_at: 2026-07-26T18:00:00Z
    completed_at: 2026-07-26T18:04:12Z
    result_digest: sha256:...
  verification:
    independent_witnesses: 2
    transparency_receipt: required
  settlement:
    payable_only_if: quality_and_receipt_verified
```

Failure patterns

- Treating attestation and signed statements as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for attestation and signed statements.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

4.3 Challenges and independent verification

Challenges and independent verification must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Use spot checks, redundancy, canaries, witnesses, proof sampling, and external observations. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for challenges and independent verification.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating challenges and independent verification as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for challenges and independent verification.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

4.4 Transparency and receipts

Transparency and receipts must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Register statements, obtain receipts, verify inclusion, audit consistency, and handle privacy. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for transparency and receipts.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating transparency and receipts as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for transparency and receipts.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 04 laboratory and review

Practical laboratory
Build a proof-of-service record and identify which fields are observed, self-asserted, derived, or independently verified.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore SCITT-based infrastructure receipts and zero-knowledge service proofs; clearly retain draft and research status.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 05

Incentives, settlement, and unit economics

Align rewards with useful service rather than easily gamed proxy activity.

Chapter operating position
Align rewards with useful service rather than easily gamed proxy activity.

5.1 Reward function

Reward function must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Tie payment to delivered quality, availability, scarcity, verified work, cost, and client value. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for reward function.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating reward function as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for reward function.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

5.2 Stake, slashing, and collateral

Stake, slashing, and collateral must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Assess deterrence, false punishment, governance capture, volatility, and legal enforceability. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for stake, slashing, and collateral.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
service_receipt:
  resource_id: did:example:edge-node-9
  service_contract: compute-v2
  workload_digest: sha256:...
  measurement:
    started_at: 2026-07-26T18:00:00Z
    completed_at: 2026-07-26T18:04:12Z
    result_digest: sha256:...
  verification:
    independent_witnesses: 2
    transparency_receipt: required
  settlement:
    payable_only_if: quality_and_receipt_verified
```

Failure patterns

- Treating stake, slashing, and collateral as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for stake, slashing, and collateral.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

5.3 Pricing and settlement

Pricing and settlement must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Handle bids, fixed rates, auctions, credits, tokens, fiat, taxes, disputes, and reversals. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for pricing and settlement.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating pricing and settlement as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for pricing and settlement.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

5.4 Sustainability and utilization

Sustainability and utilization must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Measure hardware cost, energy, depreciation, idle capacity, support, demand, and subsidies. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for sustainability and utilization.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating sustainability and utilization as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for sustainability and utilization.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 05 laboratory and review

Practical laboratory
Model provider and client economics under honest service, low demand, fraud, volatility, and equipment failure.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore stable service credits and automated settlement backed by verifiable receipts rather than speculative demand.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 06

Security threats and adversarial networks

Assume participants may be malicious, colluding, unavailable, or strategically rational.

Chapter operating position
Assume participants may be malicious, colluding, unavailable, or strategically rational.

6.1 Sybil and identity farming

Sybil and identity farming must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Prevent one actor from appearing as many independent resources or locations. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for sybil and identity farming.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating sybil and identity farming as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for sybil and identity farming.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

6.2 Spoofed work and measurement

Spoofed work and measurement must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Detect fabricated capacity, replayed results, fake location, proxy execution, and manipulated telemetry. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for spoofed work and measurement.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
service_receipt:
  resource_id: did:example:edge-node-9
  service_contract: compute-v2
  workload_digest: sha256:...
  measurement:
    started_at: 2026-07-26T18:00:00Z
    completed_at: 2026-07-26T18:04:12Z
    result_digest: sha256:...
  verification:
    independent_witnesses: 2
    transparency_receipt: required
  settlement:
    payable_only_if: quality_and_receipt_verified
```

Failure patterns

- Treating spoofed work and measurement as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for spoofed work and measurement.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

6.3 Collusion and verifier capture

Collusion and verifier capture must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Limit provider-verifier collusion, bribery, censorship, frontrunning, and governance abuse. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for collusion and verifier capture.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating collusion and verifier capture as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for collusion and verifier capture.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

6.4 Client and workload threats

Client and workload threats must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Protect providers from malicious code, data exfiltration, prohibited use, resource exhaustion, and legal risk. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for client and workload threats.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating client and workload threats as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for client and workload threats.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 06 laboratory and review

Practical laboratory

Red-team a network with Sybil providers, colluding verifiers, malicious clients, and a compromised coordinator.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore Byzantine-resilient infrastructure verification and confidential workload execution.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 07

Governance, upgrades, and dispute resolution

Define how a distributed network changes and who can make binding decisions.

Chapter operating position
Define how a distributed network changes and who can make binding decisions.

7.1 Protocol and parameter governance

Protocol and parameter governance must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Control software, contracts, reward functions, admission, fees, security, and emergency changes. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for protocol and parameter governance.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating protocol and parameter governance as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for protocol and parameter governance.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

7.2 Software and hardware lifecycle

Software and hardware lifecycle must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Require signed releases, provenance, compatibility, rollout, rollback, maintenance, and retirement. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for software and hardware lifecycle.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
service_receipt:
  resource_id: did:example:edge-node-9
  service_contract: compute-v2
  workload_digest: sha256:...
  measurement:
    started_at: 2026-07-26T18:00:00Z
    completed_at: 2026-07-26T18:04:12Z
    result_digest: sha256:...
  verification:
    independent_witnesses: 2
    transparency_receipt: required
  settlement:
    payable_only_if: quality_and_receipt_verified
```

Failure patterns

- Treating software and hardware lifecycle as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for software and hardware lifecycle.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

7.3 Disputes and appeals

Disputes and appeals must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Preserve evidence, identify jurisdiction, provide arbitration, reverse settlement, and repair reputation. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for disputes and appeals.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating disputes and appeals as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for disputes and appeals.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

7.4 Concentration and public interest

Concentration and public interest must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Measure operators, stake, geography, suppliers, clients, governance, and infrastructure dependency. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for concentration and public interest.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating concentration and public interest as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for concentration and public interest.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 07 laboratory and review

Practical laboratory
Design a governance process for a critical protocol vulnerability and a disputed slashing event.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore bicameral governance combining technical safety authority with participant and public-interest representation.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 08

Adoption, integration, and readiness

Determine where decentralized infrastructure improves outcomes and where it adds avoidable complexity.

Chapter operating position
Determine where decentralized infrastructure improves outcomes and where it adds avoidable complexity.

8.1 Service comparison

Service comparison must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Compare centralized, federated, marketplace, cooperative, and DePIN models by quality and control. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for service comparison.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating service comparison as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for service comparison.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

8.2 Enterprise integration

Enterprise integration must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Connect identity, procurement, scheduling, data, observability, billing, incident, and compliance systems. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for enterprise integration.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
service_receipt:
  resource_id: did:example:edge-node-9
  service_contract: compute-v2
  workload_digest: sha256:...
  measurement:
    started_at: 2026-07-26T18:00:00Z
    completed_at: 2026-07-26T18:04:12Z
    result_digest: sha256:...
  verification:
    independent_witnesses: 2
    transparency_receipt: required
  settlement:
    payable_only_if: quality_and_receipt_verified
```

Failure patterns

- Treating enterprise integration as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for enterprise integration.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

8.3 Readiness and exit

Readiness and exit must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Require mature clients, support, evidence, interoperability, economics, portability, and provider alternatives. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for readiness and exit.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating readiness and exit as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for readiness and exit.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

8.4 Limitations and claims

Limitations and claims must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Separate production evidence, pilots, research, token incentives, projections, and unsupported narratives. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the participant identity, resource advertisement, scheduling, service delivery, measurement, attestation, receipt, settlement, dispute, and governance chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for limitations and claims.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating limitations and claims as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for limitations and claims.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 08 laboratory and review

Practical laboratory
Produce an Adopt, Pilot, Observe, Hold, or Reject decision for a real infrastructure workload.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Develop verifiable public infrastructure commons with interoperable identity, service receipts, and sovereign policy.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

Integrated learning roadmap

The guide is intended to be revisited. First establish fluency, then build a small implementation, operate it under failure, automate repeatable work, and finally evaluate emerging techniques against a stable baseline. Progress is demonstrated through evidence rather than reading completion alone.

Stage	Demonstrated capability
Foundation	Explain the system from first principles and trace its critical path without relying on product terminology.
Build	Implement the smallest representative system with explicit contracts, identities, telemetry, and tests.
Operate	Diagnose injected failures, recover safely, and preserve evidence of the causal chain.
Automate	Convert a proven manual workflow into bounded, idempotent, observable automation.
Architect	Design for scale, resilience, security, interoperability, and lifecycle change.
Explore	Evaluate frontier techniques using stated hypotheses, limitations, and adoption gates.

Source authority register

Source policy
Official standards, specifications, and project documentation remain with their issuing organizations. Mythos Systems provides original engineering interpretation, synthesis, implementation guidance, and relationship mapping. Source verification dates do not guarantee that an authority has not changed since review.

01. W3C - Decentralized Identifiers v1.1
Role: Decentralized identifier data model, operations, and verifiable data registries.
Verified: 2026-07-26
Official source: https://www.w3.org/TR/did-1.1/

02. W3C - Verifiable Credentials Data Model v2.0
Role: Issuer-holder-verifier model and tamper-resistant credential data model.
Verified: 2026-07-26
Official source: https://www.w3.org/TR/vc-data-model-2.0/

03. W3C - Verifiable Credential Data Integrity 1.0
Role: Cryptographic integrity and authenticity mechanisms for verifiable digital documents.
Verified: 2026-07-26
Official source: https://www.w3.org/TR/vc-data-integrity/

04. IETF - SCITT Architecture - Internet-Draft
Role: Signed statements, transparency services, receipts, auditors, and verifiable data structures; draft status.
Verified: 2026-07-26
Official source: https://datatracker.ietf.org/doc/draft-ietf-scitt-architecture/

05. in-toto - in-toto
Role: Open metadata framework for proving supply-chain steps, actors, and ordering.
Verified: 2026-07-26
Official source: https://in-toto.io/

06. SLSA - SLSA Specification v1.2
Role: Current approved source and build provenance, verification, and assurance requirements.
Verified: 2026-07-26
Official source: https://slsa.dev/spec/v1.2/

07. NIST - SP 500-325 - Fog Computing Conceptual Model
Role: Fog and edge computing actors, layers, nodes, services, and management concepts.
Verified: 2026-07-26
Official source: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.500-325.pdf

Authority application and refresh triggers

Authority class	Required library action
Protocols and specifications	Recheck interoperability, security considerations, defaults, and migration guidance when a referenced specification changes.
Security and assurance frameworks	Update control mappings, evidence expectations, risk language, and assessment procedures after material framework revision.
Languages, runtimes, and projects	Re-run examples, tests, deprecation checks, and operational recommendations against supported releases.
Benchmarks and evaluations	Re-execute claims when workloads, methods, models, compilers, drivers, hardware, or scoring rules change.
Operational evidence	Incorporate validated incidents, failure tests, reader corrections, and measured implementation outcomes into the next revision.

Limitations, freshness, and revision control

Boundary	Statement
Publication limitation	This guide synthesizes broad engineering practice. It does not replace product documentation, legal advice, safety certification, or environment-specific design review.
Evidence limitation	Not every pattern is benchmarked in every environment. Examples illustrate engineering methods and must be validated against actual versions, workloads, hardware, policy, and failure conditions.
Freshness limitation	Vendor behavior, software libraries, AI models, security threats, and emerging standards can change quickly. Volatile claims require source revalidation before operational adoption.
Adoption limitation	Frontier content is not an implicit adoption recommendation. Adoption requires defined value, qualification gates, controlled proof, rollback, ownership, and residual-risk acceptance.
Status boundary	Candidate Focused Field Guide describes the publication, not certification of a product, implementation, engineer, or organization.

Revision record

Version	Revision statement
1.0.0-candidate	Initial candidate living-guide edition. Published 2026-07-26; scheduled review 2026-10-26.

Search and relationship metadata

Dimension	Engineering position
Primary domain	MYTHOS-CLD - Cloud, Platform & Distributed Systems
Secondary domain	MYTHOS-EMT; MYTHOS-SEC; MYTHOS-DAT
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	DePIN, decentralized infrastructure, verifiable service, DID, verifiable credentials, SCITT, proof of service, Sybil resistance, incentives, distributed governance
Canonical route	/library/field-guides/mythos-fg-cld-0008/

Living publication

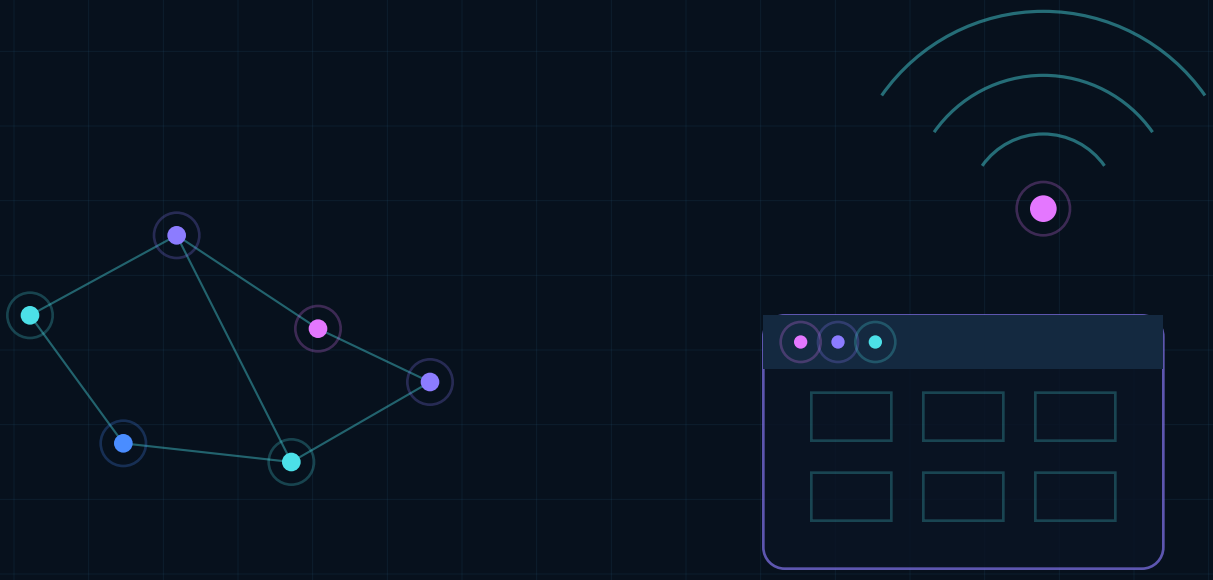
Corrections, expanded laboratories, improved diagrams, authority changes, and new engineering evidence should revise this permanent publication rather than create disconnected replacements.

MYTHOS SYSTEMS

RESEARCH & ADOPTION ANNEX

Cloud, Edge, Telecom, Hardware, Media, and Web Research Annex

Evidence-bounded research posture for browser-local compute, accelerated edge inference, photonics, confidential compute, decentralized infrastructure, RAN automation, and network slicing.



MYTHOS-RES-ANNEX-CLOUDEDGE-0001

PERMANENT PUBLICATION IDENTITY / CANDIDATE PUBLICATION / PUBLIC

Research adoption map

RES-006	**Browser-Local WebLLM: Architecture, Constraints, and WebGPU** PILOT	A-
RES-018	**NPU Integration and Local Inference Acceleration** ADOPT HARDWARE-ABSTRACTION STRATEGY	A-
RES-020	**Silicon Photonics and Chip-to-Chip Optical Interconnects** MONITOR AND PARTNER PILOT	A-
RES-023	**Trusted Execution Environments: TDX, SEV-SNP, and Remote** CONDITIONAL ADOPTION	A-
RES-025	**Decentralized Physical Infrastructure Networks (DePIN) Compute** MONITOR - DO NOT USE FOR SENSITIVE WORKLOADS	B-
RES-027	**AI-Driven Radio Access Networks (RAN) and vRAN Orchestration** MONITOR AND TELECOM PILOT	B+
RES-028	**5G/6G Network Slicing and QoS Enforcement** ADOPT 5G CONTROLS; MONITOR 6G	A-

Claim -> authority -> experiment -> evidence -> decisio

No research result becomes a production guarantee. Every adoption decision remains bounded by the tested implementation, source freshness, environment, evidence period, and stated limitations.

REVALIDATION

90-180d

Review faster when specifications, hardware support, threat models, browser/runtime behavior, or telecom releases change.

FAIL-CLOSED RULE

NO INFERENCE

Unknown maturity, stale evidence, or unsupported deployment context blocks stronger readiness claims.

Where the research changes engineering decisions

WEB

WebGPU/WebLLM constraints determine browser-local model size, storage, isolation, and fallback.

HW

NPU, photonics, and TEE evidence shape accelerator placement, trust, memory, and interconnect design.

CLD

Confidential compute and DePIN findings constrain cloud sovereignty and sensitive-workload placement.

TEL

RAN automation and slicing research govern what can be automated, isolated, measured, and safely piloted.

EDGE

Local inference and remote-site constraints define degraded modes, sync, update, and resource budgets.

Boundaries on interpretation

- Research reports are not certifications, procurement approvals, legal opinions, or vendor guarantees.
- Hardware and browser behavior must be reproduced on the deployed platform and version.
- Telecom standards and 6G work remain release- and implementation-dependent; speculative capability is labeled as such.
- Performance results require workload, hardware, model, runtime, topology, and configuration disclosure.
- Security claims require threat-model alignment and direct evidence of enforcement, not feature presence alone.

RES-006

Browser-Local WebLLM (Architecture, Constraints, and WebGPU bindings)

Enterprise Research Report, Source-Freshness Upgrade, and Adoption Decision

RESEARCH DOMAIN	ADOPTION POSTURE
Browser-Local AI	PILOT
EVIDENCE GRADE	**VALIDATED THROUGH**
A-	2026-07-22

Candidate Research Report

Mythos Systems | Research Intelligence Program | Public Technical Publication

This report preserves the complete source note while adding current authority validation, explicit limitations, an adoption decision, and a reproducible validation plan.

Document Control

Field	Value
Document ID	RES-006
Version	2.0.0
Status	Candidate Research Report
Classification	Public
Publisher	Mythos Systems
Program	Research Intelligence Program
Validated Through	2026-07-22
Scheduled Review	90 days
Adoption Posture	PILOT
Evidence Grade	A-
Source Lineage	RES-006_Browser_Local_WebLLM_Architecture_Constraints_WebGPU_Bindings.md
Series Integrity	RES-019 remains reserved; no source note was present in the corpus.

Contents

1. Research at a Glance

ADOPTION	EVIDENCE	REVIEW	SOURCE SET
PILOT	A-	90 days	3 current authorities

CURRENT ADOPTION DECISION

Pilot browser-local inference for privacy-sensitive, latency-tolerant, bounded workloads. Gate deployment on WebGPU support, memory pressure, model size, caching, browser isolation, and measured device compatibility.

The original source note contains a strong architectural thesis, but its legacy “definitive,” “mandated,” or absolute language is not itself evidence. This edition preserves the complete source content while replacing unsupported certainty with a controlled research posture, validation conditions, and explicit limits.

2. Executive Research Position

Research domain: Browser-Local AI. Current posture: PILOT. The subject is assessed using primary standards, official project documentation, vendor technical material where unavoidable, and primary research. Adoption is conditional on reproducing the relevant behavior in the intended environment.

The upgraded position distinguishes architectural usefulness from implementation maturity. A concept may be valuable enough to influence interfaces, data models, or long-term strategy while remaining unsuitable as a mandatory production dependency. Conversely, mature mechanisms with strong authority may be adopted immediately while still requiring environment-specific testing.

3. Current Authority and Freshness

Source	Authority and Status	Freshness Control
W3C WebGPU Specification https://www.w3.org/TR/webgpu/	Primary web standard Living W3C specification Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.
WebLLM Documentation https://webllm.mlc.ai/docs/index.html	Primary project documentation Living Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.
WebLLM: A High-Performance In-Browser LLM Inference Engine https://arxiv.org/abs/2412.15803	Primary research preprint Preprint Published: 2024	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.

Freshness is controlled through both a scheduled review interval and event-driven triggers. A report becomes stale before its scheduled date when a governing standard changes, a cited project introduces a breaking release, a material vulnerability changes the risk posture, a research result is retracted, or internal validation contradicts the published position.

4. Explicit Research Limitations

- This report is a technical research publication, not a certification, legal opinion, procurement approval, or guarantee of production suitability.

- Source authority does not eliminate implementation risk. Product behavior, interoperability, performance, security, and operational fit require reproducible validation in the adopting environment.
- Fast-moving specifications, product documentation, open-source repositories, and research preprints may change before the next scheduled review.
- Quantitative claims from vendors or research papers are not treated as Mythos results unless independently reproduced under documented conditions.
- Adoption decisions are conditional on applicable policy, risk, licensing, export-control, privacy, safety, regulatory, and support requirements.
- Model and agent behavior is probabilistic, provider-dependent, prompt-sensitive, and affected by context, data, evaluation design, and runtime controls.

5. Adoption Decision and Guardrails

Pilot browser-local inference for privacy-sensitive, latency-tolerant, bounded workloads. Gate deployment on WebGPU support, memory pressure, model size, caching, browser isolation, and measured device compatibility.

Decision dimension	Required direction
Production authority	No deployment or control authority is granted by this report. Aegis policy, formal change control, and system ownership remain authoritative.
Implementation	Use versioned adapters, explicit interfaces, bounded scope, observability, rollback, and evidence collection.
Validation	Reproduce mandatory claims using representative data, target hardware or software, failure cases, and documented acceptance criteria.
Procurement	Treat vendor claims as evidence inputs, not verified outcomes. Require support, lifecycle, licensing, security, and exit-plan analysis.
Retirement	Define conditions for rollback, replacement, data export, key rotation, state migration, and evidence preservation before adoption.

6. Validation and Reproducibility Plan

Stage	Required evidence	Exit condition
Source validation	Exact versions, publication status, revisions, and source hashes or retrieval records.	No high-impact claim depends on an untraceable or superseded source.
Prototype	Minimal implementation with representative data and instrumented execution.	Core mechanism is demonstrated without relying on presentation-only behavior.
Adversarial and failure testing	Malformed inputs, unavailable dependencies, stale state, rollback, isolation, resource exhaustion, and misuse cases.	Failure is bounded, observable, and recoverable.
Comparative baseline	Current production method or conventional alternative tested under the same workload.	Benefit is measurable against a credible baseline.
Operational trial	Ownership, alerts, support, patching, capacity, evidence, and exit procedures exercised.	The operating model is accepted by accountable owners.
Adoption review	Risk, legal, security, privacy, safety, cost, and architecture decision record.	Named authority approves the bounded use case.

7. Review and Expiration Triggers

- Scheduled review no later than 90 days after 2026-07-22.
- A cited standard, specification, API, product, model, runtime, or reference implementation introduces a material revision.
- A security advisory, safety event, interoperability defect, or legal change alters the risk or applicability.
- Internal benchmark, proof-of-concept, or production evidence contradicts a material claim.
- The adoption posture changes, the technology becomes a critical dependency, or a retirement decision is proposed.

8. Complete Source Research Note

SOURCE-LINEAGE NOTICE

The following material preserves the complete original source note, excluding only the conversational wrapper and outer Markdown fence. Legacy status labels and absolute language are retained for traceability and do not override the candidate status, limitations, or adoption decision in this edition.

Document ID: RES-006 Version: 1.0.0 (Definitive Registry Edition) Status: Published Research Note Classification: Public Organization: Mythos Systems (MYTHOS AI) Owner: Aaron Stovall Effective Date: 2026-07-16 Applies To: Front-end architects, AI engineers building local-first web clients (CALLISTO), and platform engineers evaluating browser-based inference capabilities Companion Standards: MYTHOS-WEB-0001 (WebGPU, WebLLM & Browser Isolation), MYTHOS-EMT-0001 (WASM & Edge Sandbox), MYTHOS-DAT-0001 (Vector DB & Local-First Sync), MYTHOS-AI-0014 (Inference Serving)

The architecture of web-based AI has historically been a lie. Applications claim to be "local-first" while silently routing every keystroke to a third-party cloud LLM API for inference. This introduces crippling latency (200ms+ per token) and violates basic data sovereignty.

The introduction of WebGPU and the WebLLM runtime (e.g., `mlc-ai/web-llm`) shatters this limitation. It is now architecturally possible to download a quantized LLM directly to the browser, cache it in the Origin Private File System (OPFS), and execute tensor operations natively on the user's GPU via compute shaders.

This research note defines the mechanics of the Browser-Local WebLLM architecture. It dissects the WebGPU bindings (how LLM math maps to GPU workgroups), the browser storage lifecycle (OPFS), and the hard physical constraints (VRAM ceilings, tab eviction) that govern what is and is not possible in a purely client-side sovereign AI application.

To execute an LLM in the browser without a backend server, the architecture must solve three problems: computation, memory, and storage.

- Compute (WebGPU): The browser does not have access to CUDA. It utilizes the WebGPU API, which provides a low-overhead, cross-platform abstraction layer to the underlying graphics driver (Vulkan, Metal, Direct3D 12).
- Memory (VRAM): The model weights and the KV-cache must reside in the GPU's VRAM. WebGPU allocates `GPUBuffer` objects for this purpose.
- Storage (OPFS): A 4GB model cannot be downloaded on every page load. The browser utilizes the Origin Private File System (OPFS) to permanently cache the model weights on the user's disk.

The WebLLM runtime (like MLC-LLM or `transformers.js`) acts as the orchestrator. It fetches the model from OPFS, loads it into WebGPU buffers, compiles the LLM architecture into WebGPU Shading Language (WGSL) compute pipelines, and executes the prefill and decode loops entirely within the browser tab's sandbox.

Traditional native inference engines (vLLM, llama.cpp) use C++ and CUDA. WebLLM must map these same matrix multiplications to WebGPU compute shaders.

2.1 The WGSL Compute Pipeline

In WebGPU, the LLM's attention heads and Multi-Layer Perceptron (MLP) blocks are compiled into WGSL compute shaders.

- The Mechanic: The WebLLM runtime creates a `GPUComputePipeline` for the matrix multiplication. It binds the weight buffers and the activation buffers to `@group(0)` binding indices.

- Execution: The GPU executes the shader in parallel across thousands of workgroups. Each workgroup computes a tile of the output matrix.
- Constraint: WebGPU lacks the fine-grained hardware intrinsics (like Tensor Cores via PTX) that native CUDA utilizes. WebGPU relies on standard 32-bit floating point or emerging 16-bit (f16) support. This means browser inference is currently slower per-FLOP than native CUDA, though still highly usable for sub-7B models.

2.2 The KV-Cache in Browser VRAM

During the decode phase, the model must store the Key and Value tensors for previous tokens.

- The Mechanic: WebLLM allocates a `GPUBuffer` mapped as `STORAGE` to hold the KV-cache. As the context window grows, this buffer expands.
- Constraint: Unlike native vLLM, which utilizes PagedAttention to manage VRAM fragmentation, browser WebLLM runtimes often pre-allocate a contiguous buffer for the max context length. If the context exceeds the buffer, the application crashes (Out of Memory).

Executing AI in a browser tab is an act of brute force against a highly constrained environment. Engineers must understand these hard limits.

3.1 The VRAM Ceiling

A browser tab does not get access to the entire GPU. The OS and the browser partition GPU memory. If a user has an 8GB GPU, the browser might only allow WebGPU to allocate 2-4GB of VRAM before throwing a `GPUOutOfMemoryError`.

- Impact: You cannot run a 70B model in the browser. You cannot even run a 13B FP16 model. WebLLM is strictly constrained to heavily quantized models (INT4 AWQ or GGUF), typically in the 1B to 7B parameter range (e.g., Llama-3-8B-Instruct-Q4, Phi-3-Mini).

3.2 The Cold-Start Penalty

Loading a 4GB model from OPFS into VRAM is an I/O-intensive process.

- Impact: The first prompt a user sends will take 5 to 15 seconds to process (depending on disk speed and GPU bandwidth) as the weights are transferred. If the user navigates away or the tab is evicted, the process must restart.
- Mitigation: The model loading MUST occur inside a Web Worker. If it runs on the main thread, the browser UI will freeze, and the OS will display a "Page Unresponsive" warning.

3.3 Tab Eviction and Lifecycle

Browsers aggressively reclaim memory. If a user switches to another memory-intensive application, the OS may suspend the browser tab. When the tab is suspended, the WebGPU context is lost, and the VRAM is zeroed.

- Impact: The user's in-progress conversation (the KV-cache) is destroyed. When they return, the application must either restart the session or reload the KV-cache from memory, causing latency spikes.

To make WebLLM production-ready (as mandated by MYTHOS-WEB-0001), the architecture must actively manage the browser's constraints.

4.1 Dynamic Model Routing

Do not force the browser to run a model it cannot handle. The CALLISTO architecture requires hardware discovery.

- The app queries `navigator.gpu` and `GPUAdapter` to estimate available VRAM.
- If VRAM is high (e.g., > 6GB available), load `Llama-3-8B-Q4`.

- If VRAM is low (e.g., < 2GB available), load `Phi-3-Mini-Q4` or route the request to a local edge node (RA-006) via WebRTC.

4.2 OPFS Caching and Service Workers

A 4GB download is unacceptable on every page load.

- The app must use a Service Worker to intercept the model download request.
- The Service Worker fetches the model from the CDN the first time, writes it to OPFS, and serves all subsequent requests from the local disk. This guarantees zero-latency model loading on return visits.

4.3 Context Compaction (KV-Cache Eviction)

Because VRAM is scarce, the WebLLM runtime cannot hold an infinite context window.

- The architecture MUST implement context compaction. When the KV-cache approaches 80% of the allocated VRAM, the runtime must summarize the earliest parts of the conversation, drop those KV blocks, and inject the summary as a system prompt. This prevents OOM crashes during long chat sessions.

Browser-Local WebLLM is not a replacement for datacenter-scale inference. It is a tool for sovereign, zero-latency AI. By mapping LLM tensor operations to WebGPU compute pipelines and caching weights in OPFS, we transform the browser from a dumb terminal into a self-sufficient AI compute node. While hard VRAM constraints limit the model size, the CALLISTO architecture's dynamic routing and context compaction ensure that the user retains absolute cognitive privacy and offline resilience without sacrificing usability.

```
A cloud API is a leash.
A browser tab is a sandbox, but it is a sovereign sandbox.

WebGPU binds the math to the silicon.
OPFS binds the weights to the disk.

The VRAM is small, but the privacy is absolute.

> The web may be distributed.
> Local-first execution must be absolute.
```

End of Research Note RES-006

9. Source Register

Source	Authority and Status	Freshness Control
W3C WebGPU Specification https://www.w3.org/TR/webgpu/	Primary web standard Living W3C specification Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.
WebLLM Documentation https://webllm.mlc.ai/docs/index.html	Primary project documentation Living Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.
WebLLM: A High-Performance In-Browser LLM Inference Engine https://arxiv.org/abs/2412.15803	Primary research preprint Preprint Published: 2024	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.

10. Lineage, Review, and Publication Record

Record	Value
Original source file	RES-006_Browser_Local_WebLLM_Architecture_Constraints_WebGPU_Bindings.md
Original source words	1272
Upgrade type	Enterprise research report; source and freshness validation; limitations; adoption decision; reproducibility plan
Generated on	2026-07-22
Current status	Candidate Research Report
Publication authority	Formal Mythos approval not yet recorded
Next review	90 days or event-driven trigger, whichever occurs first

RES-018

NPU Integration & Local Inference Acceleration

Enterprise Research Report, Source-Freshness Upgrade, and Adoption Decision

RESEARCH DOMAIN	ADOPTION POSTURE
Local AI Acceleration	ADOPT HARDWARE-ABSTRACTION STRATEGY
EVIDENCE GRADE	**VALIDATED THROUGH**
A-	2026-07-22

Candidate Research Report

Mythos Systems | Research Intelligence Program | Public Technical Publication

This report preserves the complete source note while adding current authority validation, explicit limitations, an adoption decision, and a reproducible validation plan.

Document Control

Field	Value
Document ID	RES-018
Version	2.0.0
Status	Candidate Research Report
Classification	Public
Publisher	Mythos Systems
Program	Research Intelligence Program
Validated Through	2026-07-22
Scheduled Review	90 days
Adoption Posture	ADOPT HARDWARE-ABSTRACTION STRATEGY
Evidence Grade	A-
Source Lineage	RES-018_NPU_Integration_Local_Inference_Acceleration.md
Series Integrity	RES-019 remains reserved; no source note was present in the corpus.

Contents

1. Research at a Glance

ADOPTION	EVIDENCE	REVIEW	SOURCE SET
ADOPT HARDWARE-ABSTRACTION STRATEGY	A-	90 days	3 current authorities

CURRENT ADOPTION DECISION

Adopt a runtime abstraction across CPU, GPU, and NPU execution providers. Benchmark per device, model, quantization, operator coverage, memory movement, and power envelope.

The original source note contains a strong architectural thesis, but its legacy “definitive,” “mandated,” or absolute language is not itself evidence. This edition preserves the complete source content while replacing unsupported certainty with a controlled research posture, validation conditions, and explicit limits.

2. Executive Research Position

Research domain: Local AI Acceleration. Current posture: ADOPT HARDWARE-ABSTRACTION STRATEGY. The subject is assessed using primary standards, official project documentation, vendor technical material where unavoidable, and primary research. Adoption is conditional on reproducing the relevant behavior in the intended environment.

The upgraded position distinguishes architectural usefulness from implementation maturity. A concept may be valuable enough to influence interfaces, data models, or long-term strategy while remaining unsuitable as a mandatory production dependency. Conversely, mature mechanisms with strong authority may be adopted immediately while still requiring environment-specific testing.

3. Current Authority and Freshness

Source	Authority and Status	Freshness Control
Windows ML Overview https://learn.microsoft.com/en-us/windows/ai/new-windows-ml/overview	Primary platform documentation Living - fast moving Published: 2026	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.
OpenVINO NPU Device Documentation https://docs.openvino.ai/2026/openvino-workflow/running-inference/inference-devices-and-modes/npu-device.html	Primary runtime documentation Current version Published: 2026	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.
Apple Core ML Documentation https://developer.apple.com/documentation/coreml	Primary platform documentation Living Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.

Freshness is controlled through both a scheduled review interval and event-driven triggers. A report becomes stale before its scheduled date when a governing standard changes, a cited project introduces a breaking release, a material vulnerability changes the risk posture, a research result is retracted, or internal validation contradicts the published position.

4. Explicit Research Limitations

- This report is a technical research publication, not a certification, legal opinion, procurement approval, or guarantee of production suitability.

- Source authority does not eliminate implementation risk. Product behavior, interoperability, performance, security, and operational fit require reproducible validation in the adopting environment.
- Fast-moving specifications, product documentation, open-source repositories, and research preprints may change before the next scheduled review.
- Quantitative claims from vendors or research papers are not treated as Mythos results unless independently reproduced under documented conditions.
- Adoption decisions are conditional on applicable policy, risk, licensing, export-control, privacy, safety, regulatory, and support requirements.
- Model and agent behavior is probabilistic, provider-dependent, prompt-sensitive, and affected by context, data, evaluation design, and runtime controls.

5. Adoption Decision and Guardrails

Adopt a runtime abstraction across CPU, GPU, and NPU execution providers. Benchmark per device, model, quantization, operator coverage, memory movement, and power envelope.

Decision dimension	Required direction
Production authority	No deployment or control authority is granted by this report. Aegis policy, formal change control, and system ownership remain authoritative.
Implementation	Use versioned adapters, explicit interfaces, bounded scope, observability, rollback, and evidence collection.
Validation	Reproduce mandatory claims using representative data, target hardware or software, failure cases, and documented acceptance criteria.
Procurement	Treat vendor claims as evidence inputs, not verified outcomes. Require support, lifecycle, licensing, security, and exit-plan analysis.
Retirement	Define conditions for rollback, replacement, data export, key rotation, state migration, and evidence preservation before adoption.

6. Validation and Reproducibility Plan

Stage	Required evidence	Exit condition
Source validation	Exact versions, publication status, revisions, and source hashes or retrieval records.	No high-impact claim depends on an untraceable or superseded source.
Prototype	Minimal implementation with representative data and instrumented execution.	Core mechanism is demonstrated without relying on presentation-only behavior.
Adversarial and failure testing	Malformed inputs, unavailable dependencies, stale state, rollback, isolation, resource exhaustion, and misuse cases.	Failure is bounded, observable, and recoverable.
Comparative baseline	Current production method or conventional alternative tested under the same workload.	Benefit is measurable against a credible baseline.
Operational trial	Ownership, alerts, support, patching, capacity, evidence, and exit procedures exercised.	The operating model is accepted by accountable owners.
Adoption review	Risk, legal, security, privacy, safety, cost, and architecture decision record.	Named authority approves the bounded use case.

7. Review and Expiration Triggers

- Scheduled review no later than 90 days after 2026-07-22.
- A cited standard, specification, API, product, model, runtime, or reference implementation introduces a material revision.
- A security advisory, safety event, interoperability defect, or legal change alters the risk or applicability.
- Internal benchmark, proof-of-concept, or production evidence contradicts a material claim.
- The adoption posture changes, the technology becomes a critical dependency, or a retirement decision is proposed.

8. Complete Source Research Note

SOURCE-LINEAGE NOTICE

The following material preserves the complete original source note, excluding only the conversational wrapper and outer Markdown fence. Legacy status labels and absolute language are retained for traceability and do not override the candidate status, limitations, or adoption decision in this edition.

Document ID: RES-018 Version: 1.0.0 (Definitive Registry Edition) Status: Published Research Note Classification: Public Organization: Mythos Systems (MYTHOS AI) Owner: Aaron Stovall Effective Date: 2026-07-16 Applies To: Edge AI engineers, mobile platform developers, and system architects designing local-first AI inference (CALLISTO), heterogeneous compute pipelines, and sovereign edge runtimes Companion Standards: MYTHOS-EMT-0003 (Neuromorphic Computing & NPU Standard), MYTHOS-EMT-0002 (Sovereign AI), MYTHOS-WEB-0001 (WebGPU & WebLLM), RA-006 (Local-First AI Inference Cluster)

The architecture of local AI compute has failed. Organizations attempt to run continuous edge inference—wake-word detection, real-time transcription, and local LLM reasoning—using high-wattage GPUs or general-purpose CPUs. The GPU drains device batteries in minutes and requires active cooling; the CPU bottlenecks the system, causing UI jank and thermal throttling. Meanwhile, the dedicated Neural Processing Unit (NPU) sits idle on the silicon because standard AI frameworks (PyTorch/TensorFlow) default to CUDA or CPU backends.

The Mythos Architecture mandates the absolute utilization of heterogeneous silicon. NPU integration shatters the monolithic compute paradigm. By dynamically discovering the underlying hardware (Apple Neural Engine, Intel NPU, Qualcomm Hexagon) and partitioning the compute graph, we offload dense matrix multiplications to specialized, sub-milliwatt silicon.

This research note defines the architectural dynamics of NPU integration. It dissects the vendor-specific SDKs (CoreML, OpenVINO, QNN), the non-differentiable boundary of operator support, and the memory bandwidth isolation required to execute concurrent AI workloads (e.g., running an LLM on the GPU while the NPU handles continuous audio transcription) without OOM crashes. Understanding these dynamics is a prerequisite for building the sovereign, zero-latency CALLISTO web client and PANTHEON edge runtimes.

To utilize NPUs, we must understand that they are not miniature GPUs. They are fundamentally different architectures designed for extreme power efficiency.

1.1 Apple Neural Engine (ANE)

The ANE is a fixed-function, highly optimized matrix coprocessor integrated into Apple Silicon (M-series, A-series).

- The Architecture: It features a massive 2D systolic array capable of trillions of operations per second (TOPS). It is strictly optimized for INT8/FP16 execution.
- The Constraint: The ANE is rigid. It cannot execute arbitrary code. If a neural network layer (e.g., a custom CUDA kernel) is not natively supported by Apple's `MPSCNN` or `CoreML` compiler, it falls back to the GPU or CPU, introducing massive latency spikes.
- The Advantage: Unmatched power efficiency. The ANE can execute complex vision models at under 1 watt, allowing always-on inference without battery degradation.

1.2 Intel NPU (Meteor Lake & Beyond)

Intel's NPU is a dedicated AI inference engine embedded alongside the CPU and Arc GPU in modern Core Ultra chipsets.

- The Architecture: A highly programmable VLIW (Very Long Instruction Word) architecture designed for sustained AI workloads. It supports INT8, INT4, and FP16.

- The Constraint: Intel NPUs are newer to the market. The software stack (OpenVINO) is mature, but driver support for dynamic shapes (crucial for LLM KV-caches) is historically fragile.
- The Advantage: It offloads the CPU for background AI tasks (e.g., Windows Studio Effects, background blur, local RAG indexing) without waking the high-power discrete GPU.

1.3 Qualcomm Hexagon NPU

The Hexagon NPU (part of the Snapdragon platform) is a vector and tensor processor built for mobile and edge compute.

- The Architecture: A highly scalable architecture utilizing HVX (Hexagon Vector Extensions) and HTA (Hexagon Tensor Accelerator). It is deeply integrated with Qualcomm's AI Engine (QNN).
- The Constraint: The QNN SDK is notoriously complex. Compiling a standard ONNX model to Hexagon requires strict quantization (INT8) and adherence to specific operator constraints to avoid CPU fallback.
- The Advantage: Industry-leading performance-per-watt for mobile devices, enabling multi-day continuous inference on battery power.

Routing AI models across CPUs, GPUs, and NPUs introduces severe physical and software constraints that monolithic GPU deployments do not face.

2.1 The Memory Transfer Tax

In a monolithic GPU system, the model weights and activations live in GPU VRAM. In a heterogeneous system, the NPU, GPU, and CPU often have separate memory pools or caches.

- The Flaw: If Layer A runs on the GPU and Layer B runs on the NPU, the activation tensor must be copied from GPU VRAM to system RAM, and then to NPU SRAM. This PCIe/memory bus transfer takes longer than the math itself.
- The Mitigation: The compiler MUST perform operator fusion and graph partitioning. It should only assign a contiguous block of layers to the NPU if the block is large enough to amortize the memory transfer cost.

2.2 The Operator Support Boundary

NPUs are not Turing-complete. They only support specific mathematical operations (Conv2D, MatMul, ReLU).

- The Flaw: Modern LLMs utilize complex operations (Rotary Positional Embeddings, FlashAttention, dynamic KV-cache shifting). NPUs often do not support these natively. If the graph is not carefully partitioned, the system will constantly bounce between the NPU and CPU, destroying performance.
- The Mitigation: The architecture MUST utilize hardware-aware compilers (Apache TVM, ONNX Runtime) that analyze the compute graph and mathematically prove which sub-graphs can be safely offloaded to the NPU without breaking the execution context.

2.3 The Quantization Mismatch

NPUs are highly optimized for INT8 or INT4. LLMs often require FP16 for coherent reasoning.

- The Constraint: Forcing an FP16 LLM onto an INT8 NPU requires aggressive quantization (AWQ/GPTQ), which can degrade reasoning quality. Conversely, running an INT8 vision model on an FP16 GPU wastes VRAM and power.

The true power of NPU integration is realized when the runtime dynamically routes workloads based on real-time hardware availability and power budgets.

3.1 Heterogeneous Execution Routing

The local-first runtime (e.g., CALLISTO or PANTHEON edge node) MUST NOT hardcode the execution target.

- The Mechanic: At boot, the runtime queries the OS hardware abstraction layer (e.g., `IORegistry` on macOS, `WMI` on Windows, `/sys/class/devfreq` on Linux) to discover NPU capabilities and current thermal headroom.
- The Routing Decision: If the user is on battery power, route the vision/audio models to the NPU (low power). If the user is plugged in and requesting complex LLM reasoning, route the LLM to the discrete GPU (high performance) and offload the background audio transcription to the NPU.

3.2 Unified Memory Architecture (UMA)

Modern edge chips (Apple Silicon, Intel Core Ultra) utilize a Unified Memory Architecture, where the CPU, GPU, and NPU share the same physical RAM pool.

- The Advantage: Zero-copy execution. A tensor computed by the GPU can be immediately accessed by the NPU via a pointer, without physically copying the data across a bus.
- The Requirement: The runtime MUST utilize low-level APIs (Metal, Level-Zero, OpenVINO) that support shared memory handles (e.g., `IOSurface` on Apple, USM in OpenVINO) to enable true zero-copy heterogeneous execution.

The fundamental shift of NPU integration is the ability to run continuous, concurrent AI without degrading the user experience.

- Always-On Perception: The NPU handles continuous audio wake-word detection (Iris STT) and camera tracking (Medusa) at < 1 watt.
- On-Demand Cognition: When the user speaks, the GPU is instantly spun up to execute the heavy LLM reasoning loop.
- The Result: The device maintains a 15-hour battery life while running a fully autonomous, local-first AI agent. Without the NPU, the GPU would drain the battery in 2 hours.

To deploy NPU-accelerated AI in production, the Mythos Architecture enforces strict hardware abstraction and dynamic compilation.

5.1 Declarative Hardware Abstraction

The PANTHEON Nona (Planner) module MUST NOT be compiled for a specific chip. The agent logic is defined in a high-level, hardware-agnostic format (ONNX or PyTorch).

- The Mitigation: At deployment time, the local gateway utilizes an Execution Provider (ONNX Runtime / CoreML / OpenVINO) to dynamically partition the graph based on the discovered hardware. If the target is an iPad, the graph is compiled for the ANE. If the target is a Linux workstation, it is compiled for an Intel NPU or AMD NPU.

5.2 NPU Thermal Quotas

Because NPUs are passively cooled in thin-and-light laptops, sustained 100% utilization will cause thermal throttling.

- The Mitigation: The runtime MUST enforce a thermal quota. If the NPU temperature exceeds 80°C, the runtime dynamically downgrades the model (e.g., switching from a 7B model to a 3B model) or reduces the batch size to maintain the thermal envelope.

5.3 The Memory Isolation Boundary

When running an LLM on the GPU and an audio model on the NPU concurrently, they are competing for the same Unified Memory bandwidth.

- The Mitigation: The OS MUST enforce Quality of Service (QoS) memory prioritization. The interactive LLM (foreground) is given high-priority memory access, while the background NPU tasks are throttled to prevent memory bus saturation and OOM crashes.

Defaulting to the GPU for all AI tasks is an artifact of the cloud era. For sovereign, local-first AI to succeed on edge devices, the architecture must embrace heterogeneous compute. By dynamically discovering the NPU, partitioning the compute graph, and enforcing zero-copy memory boundaries, we transform edge devices from battery-draining space heaters into ultra-efficient, continuously reasoning autonomous agents.

```
A GPU is a power plant; an NPU is a watch battery.
A CPU is a generalist; an NPU is a specialist.
A monolithic graph is a bottleneck; a partitioned graph is a flow.

The model must adapt to the silicon; the silicon must not adapt to the model.

> Compute may be heterogeneous.
> Sovereign efficiency must be absolute.
```

End of Research Note RES-018

9. Source Register

Source	Authority and Status	Freshness Control
Windows ML Overview https://learn.microsoft.com/en-us/windows/ai/new-windows-ml/overview	Primary platform documentation Living - fast moving Published: 2026	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.
OpenVINO NPU Device Documentation https://docs.openvino.ai/2026/openvino-workflow/running-inference/inference-devices-and-modes/npu-device.html	Primary runtime documentation Current version Published: 2026	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.
Apple Core ML Documentation https://developer.apple.com/documentation/coreml	Primary platform documentation Living Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.

10. Lineage, Review, and Publication Record

Record	Value
Original source file	RES-018_NPU_Integration_Local_Inference_Acceleration.md
Original source words	1617
Upgrade type	Enterprise research report; source and freshness validation; limitations; adoption decision; reproducibility plan
Generated on	2026-07-22
Current status	Candidate Research Report
Publication authority	Formal Mythos approval not yet recorded
Next review	90 days or event-driven trigger, whichever occurs first

RES-020

Silicon Photonics & Chip-to-Chip Optical Interconnects

Enterprise Research Report, Source-Freshness Upgrade, and Adoption Decision

RESEARCH DOMAIN	ADOPTION POSTURE
Photonics and Interconnect	MONITOR AND PARTNER PILOT
EVIDENCE GRADE	**VALIDATED THROUGH**
A-	2026-07-22

Candidate Research Report

Mythos Systems | Research Intelligence Program | Public Technical Publication

This report preserves the complete source note while adding current authority validation, explicit limitations, an adoption decision, and a reproducible validation plan.

Document Control

Field	Value
Document ID	RES-020
Version	2.0.0
Status	Candidate Research Report
Classification	Public
Publisher	Mythos Systems
Program	Research Intelligence Program
Validated Through	2026-07-22
Scheduled Review	180 days
Adoption Posture	MONITOR AND PARTNER PILOT
Evidence Grade	A-
Source Lineage	RES-020_Silicon_Photonics_Chip_to_Chip_Optical_Interconnects.md
Series Integrity	RES-019 remains reserved; no source note was present in the corpus.

Contents

1. Research at a Glance

ADOPTION	EVIDENCE	REVIEW	SOURCE SET
MONITOR AND PARTNER PILOT	A-	180 days	1 current authorities

CURRENT ADOPTION DECISION

Track silicon photonics and co-packaged optics as infrastructure roadmap inputs. Avoid production dependency until interoperability, thermal, serviceability, packaging, and supply-chain requirements are validated.

The original source note contains a strong architectural thesis, but its legacy “definitive,” “mandated,” or absolute language is not itself evidence. This edition preserves the complete source content while replacing unsupported certainty with a controlled research posture, validation conditions, and explicit limits.

2. Executive Research Position

Research domain: Photonics and Interconnect. Current posture: MONITOR AND PARTNER PILOT. The subject is assessed using primary standards, official project documentation, vendor technical material where unavoidable, and primary research. Adoption is conditional on reproducing the relevant behavior in the intended environment.

The upgraded position distinguishes architectural usefulness from implementation maturity. A concept may be valuable enough to influence interfaces, data models, or long-term strategy while remaining unsuitable as a mandatory production dependency. Conversely, mature mechanisms with strong authority may be adopted immediately while still requiring environment-specific testing.

3. Current Authority and Freshness

Source	Authority and Status	Freshness Control
OIF Co-Packaging Implementation Agreements https://www.oiforum.com/technical-work/implementation-agreements-ias/	Primary interoperability agreements Living portfolio Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.

Freshness is controlled through both a scheduled review interval and event-driven triggers. A report becomes stale before its scheduled date when a governing standard changes, a cited project introduces a breaking release, a material vulnerability changes the risk posture, a research result is retracted, or internal validation contradicts the published position.

4. Explicit Research Limitations

- This report is a technical research publication, not a certification, legal opinion, procurement approval, or guarantee of production suitability.
- Source authority does not eliminate implementation risk. Product behavior, interoperability, performance, security, and operational fit require reproducible validation in the adopting environment.
- Fast-moving specifications, product documentation, open-source repositories, and research preprints may change before the next scheduled review.
- Quantitative claims from vendors or research papers are not treated as Mythos results unless independently reproduced under documented conditions.

- Adoption decisions are conditional on applicable policy, risk, licensing, export-control, privacy, safety, regulatory, and support requirements.
- The technology is emerging and may have limited independent production evidence, immature tooling, scarce hardware access, or rapidly changing performance claims.

5. Adoption Decision and Guardrails

Track silicon photonics and co-packaged optics as infrastructure roadmap inputs. Avoid production dependency until interoperability, thermal, serviceability, packaging, and supply-chain requirements are validated.

Decision dimension	Required direction
Production authority	No deployment or control authority is granted by this report. Aegis policy, formal change control, and system ownership remain authoritative.
Implementation	Use versioned adapters, explicit interfaces, bounded scope, observability, rollback, and evidence collection.
Validation	Reproduce mandatory claims using representative data, target hardware or software, failure cases, and documented acceptance criteria.
Procurement	Treat vendor claims as evidence inputs, not verified outcomes. Require support, lifecycle, licensing, security, and exit-plan analysis.
Retirement	Define conditions for rollback, replacement, data export, key rotation, state migration, and evidence preservation before adoption.

6. Validation and Reproducibility Plan

Stage	Required evidence	Exit condition
Source validation	Exact versions, publication status, revisions, and source hashes or retrieval records.	No high-impact claim depends on an untraceable or superseded source.
Prototype	Minimal implementation with representative data and instrumented execution.	Core mechanism is demonstrated without relying on presentation-only behavior.
Adversarial and failure testing	Malformed inputs, unavailable dependencies, stale state, rollback, isolation, resource exhaustion, and misuse cases.	Failure is bounded, observable, and recoverable.
Comparative baseline	Current production method or conventional alternative tested under the same workload.	Benefit is measurable against a credible baseline.
Operational trial	Ownership, alerts, support, patching, capacity, evidence, and exit procedures exercised.	The operating model is accepted by accountable owners.
Adoption review	Risk, legal, security, privacy, safety, cost, and architecture decision record.	Named authority approves the bounded use case.

7. Review and Expiration Triggers

- Scheduled review no later than 180 days after 2026-07-22.
- A cited standard, specification, API, product, model, runtime, or reference implementation introduces a material revision.
- A security advisory, safety event, interoperability defect, or legal change alters the risk or applicability.
- Internal benchmark, proof-of-concept, or production evidence contradicts a material claim.
- The adoption posture changes, the technology becomes a critical dependency, or a retirement decision is proposed.

8. Complete Source Research Note

SOURCE-LINEAGE NOTICE

The following material preserves the complete original source note, excluding only the conversational wrapper and outer Markdown fence. Legacy status labels and absolute language are retained for traceability and do not override the candidate status, limitations, or adoption decision in this edition.

Document ID: RES-020 Version: 1.0.0 (Definitive Registry Edition) Status: Published Research Note Classification: Public Organization: Mythos Systems (MYTHOS AI) Owner: Aaron Stovall Effective Date: 2026-07-16 Applies To: Hardware architects, HPC engineers, and silicon designers evaluating optical interconnects, Co-Packaged Optics (CPO), and Network-on-Chip (NoC) paradigms for exascale AI clusters Companion Standards: MYTHOS-EMT-0004 (Silicon Photonics & Next-Gen Interconnect Standard), MYTHOS-HW-0001 (AI GPU Clusters), MYTHOS-HW-0002 (Trusted Compute & DPU), MYTHOS-CLD-0001 (Multi-Cloud Control Plane)

The architecture of high-performance computing has hit a physical wall. Organizations attempt to scale AI clusters by connecting thousands of GPUs using high-speed electrical SerDes (Serialization/Deserialization) over copper traces and pluggable optical transceivers. As data rates push beyond 112 Gbps and toward 224 Gbps, copper suffers from severe signal attenuation, frequency-dependent loss, and crosstalk. The power required to drive these electrical signals—and the physical distance from the ASIC to the front-panel pluggable module—creates a "power wall," where 20% to 30% of a switch's power is consumed merely by moving data, not computing it.

Silicon Photonics shatters this electrical bottleneck. By integrating optical components—lasers, modulators, and waveguides—directly onto the silicon substrate, we replace electrons with photons for data transmission.

This research note defines the architectural dynamics of chip-to-chip optical interconnects. It dissects the mechanics of Micro-Ring Modulators (MRMs), the thermal resonance challenges, the sub-picojoule-per-bit efficiency milestone, and the ultimate goal: Zero-Conversion Optical Routing, where data remains in the optical domain from origin core to destination core. Understanding these dynamics is a prerequisite for building the exascale, zero-latency AI fabrics mandated by the Mythos Hardware Standards.

To understand optical interconnects, we must move from electrical RC (Resistor-Capacitor) delays to photonic waveguide dynamics.

1.1 Silicon Waveguides

A silicon waveguide is a sub-micron channel etched into a Silicon-on-Insulator (SOI) wafer. Because silicon has a high refractive index compared to the surrounding silicon dioxide cladding, it acts as a pipe for light (typically in the 1310nm to 1550nm infrared spectrum).

- The Advantage: Unlike electrical wires, waveguides do not suffer from capacitive load or electromagnetic interference (EMI). Photons do not interact with each other, allowing massive bandwidth density via Wavelength Division Multiplexing (WDM).

1.2 Micro-Ring Modulators (MRMs)

To encode digital data onto a continuous wave laser, we use MRMs. An MRM is a microscopic circular loop of silicon waveguide placed adjacent to the main waveguide.

- The Mechanic: By applying a voltage to the MRM, the refractive index of the silicon changes (via the plasma dispersion effect). This shifts the resonant frequency of the ring, allowing it to either capture light from the waveguide (blocking it, a "0") or let it pass (a "1").

- The Advantage: MRMs are incredibly small (microns in diameter) and consume femtojoules of energy per bit, enabling massive multiplexing density on a single chip.

1.3 Co-Packaged Optics (CPO)

CPO moves the optical transceiver (the laser driver and photodetector) from the front panel of the switch directly onto the same substrate as the compute ASIC (GPU or Switch).

- The Paradigm Shift: The electrical trace length drops from 30+ centimeters (to the front panel) to less than 5 millimeters. The signal integrity is perfect, the power consumption plummets, and the front panel is abolished.

While photons move at the speed of light, integrating photonic generation and modulation onto thermal, logic-focused silicon introduces severe physical constraints.

2.1 The Thermal Resonance Crisis

Micro-Ring Modulators are highly sensitive to temperature. A change of just 1°C can detune the ring by 1nm, ruining its resonance and dropping the optical signal.

- The Flaw: AI ASICs generate massive, fluctuating heat. If an MRM drifts off resonance due to a sudden thermal spike from a heavy matrix multiplication, the optical link dies.
- The Mitigation: CPO architectures MUST integrate localized micro-heaters and thermoelectric coolers onto the MRMs. A closed-loop feedback system constantly applies heat to lock the ring at its resonant wavelength, consuming additional static power.

2.2 The Laser Source Problem

Silicon is an indirect bandgap material—it cannot emit light efficiently. Therefore, the laser source MUST be external.

- The Constraint: High-power, multi-wavelength lasers (necessary for WDM) are fragile and degrade over time. They must be housed off-chip, and their light piped into the silicon waveguides via grating couplers or edge coupling.
- The Impact: If the external laser fails, the entire optical fabric goes dark. CPO architectures require redundant, failover laser arrays.

2.3 The O-E-O Conversion Tax

In current CPO architectures, data leaves the GPU as electricity, is converted to light (E-O), traverses the fiber, is converted back to electricity (O-E), and enters the destination GPU.

- The Flaw: The O-E-O conversion consumes power and adds latency. While better than all-copper links, it is not the optimal end state.

The ultimate evolution of silicon photonics is the elimination of the electrical domain entirely for data transit.

3.1 Optical Network-on-Chip (NoC)

Instead of routing electrical signals through a CPU/ASIC to reach the optical transceiver, the optical waveguides are routed directly to the compute cores.

- The Mechanic: A core generates an electrical signal, which is immediately modulated onto an optical wavelength by a local MRM. The photon travels across the wafer or rack, hits a photodetector at the destination core, and is converted back to an electrical signal directly at the register level.
- The Impact: The electrical interconnect fabric (routers, crossbars) is abolished. Latency drops to the physical speed of light across the die.

3.2 All-Optical Switching (Wavelength Routing)

The holy grail is zero-conversion routing. Data arrives as light, is routed as light, and departs as light, with zero O-E-O conversion in the middle.

- The Mechanic: Utilizing Semiconductor Optical Amplifiers (SOAs) or Mach-Zehnder Interferometers (MZIs), an optical switch can route a specific wavelength to a specific output port based on the wavelength itself, without ever reading the digital 1s and 0s.
- The Impact: A network switch that consumes zero power per bit routed. A 100-Terabit switch becomes a passive prism, directing light rather than processing packets.

The fundamental shift of silicon photonics is the decoupling of bandwidth from power and distance.

- Electrical SerDes: Moving a terabit of data across a rack electrically costs ~15 to 20 picojoules per bit. The power draw is so high it limits the number of links an ASIC can support.
- Silicon Photonics: Moving a terabit of data optically costs < 1 picojoule per bit. The power draw is negligible, allowing an ASIC to support thousands of high-bandwidth optical links.

The Topology Shift: Because electrical signals degrade over distance, GPU clusters are forced into rigid, physical spine-leaf topologies with limited cable lengths. Because optical signals do not degrade over rack-scale distances, silicon photonics enables "flat" topologies. Any GPU can talk to any other GPU with near-zero latency, creating a true, non-blocking, all-to-all supercomputer fabric.

To deploy silicon photonics in production, the Mythos Hardware Standard (MYTHOS-EMT-0004) enforces strict operational and architectural boundaries.

5.1 Mandatory CPO for Scale-Out AI

The Mythos standard prohibits the use of front-panel pluggable optics (QSFP) for scale-out AI fabrics.

- The Mitigation: All AI accelerators (GPUs, NPUs, WSEs) and top-of-rack switches MUST utilize Co-Packaged Optics (CPO). The electrical trace length from ASIC to optical engine MUST NOT exceed 10mm. This guarantees that the power budget is spent on compute, not SerDes.

5.2 AI-Driven Thermal Stabilization

The thermal resonance crisis of MRMs cannot be solved by static PID controllers; AI workloads generate heat too dynamically.

- The Mitigation: The CPO control plane MUST utilize a lightweight, localized AI model (running on an NPU) to predict thermal spikes based on the compute workload. The model pre-heats or pre-cools the MRMs milliseconds before the ASIC temperature shifts, ensuring zero optical link drops.

5.3 The Optical Sovereignty Boundary

Optical links can be physically tapped (e.g., bending a fiber to leak light) without breaking the connection.

- The Mitigation: For sovereign and classified environments, all optical waveguides (both on-die and in-fiber) MUST be physically shielded and monitored. Furthermore, because optical signals can be intercepted, end-to-end encryption (MACsec or PQC MACsec) MUST be applied at the electrical boundary before modulation, ensuring that even if light is tapped, the data remains cryptographically secure.

The strategy of scaling AI bandwidth by pushing more electricity through copper wires has reached the limits of physics. Silicon Photonics proves that the path to exascale AI requires a shift from electrons to photons. By integrating waveguides and modulators directly onto compute silicon, we abolish the SerDes power wall, enable zero-conversion routing, and unlock the flat, all-to-all topologies required for trillion-parameter model training.

```
A copper trace is a resistor; a waveguide is a free path.
An electrical SerDes is a power tax; a micro-ring is a whisper.
An O-E-O conversion is a bottleneck; an all-optical switch is a prism.
A front-panel pluggable is a relic; co-packaged optics is the absolute.

The interconnect is the enemy; the photon is the solution.

> Bandwidth may be massive.
> Power-per-bit must be absolute.
```

End of Research Note RES-020

9. Source Register

Source	Authority and Status	Freshness Control
OIF Co-Packaging Implementation Agreements https://www.oiforum.com/technical-work/implementation-agreements-ias/	Primary interoperability agreements Living portfolio Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.

10. Lineage, Review, and Publication Record

Record	Value
Original source file	RES-020_Silicon_Photonics_Chip_to_Chip_Optical_Interconnects.md
Original source words	1568
Upgrade type	Enterprise research report; source and freshness validation; limitations; adoption decision; reproducibility plan
Generated on	2026-07-22
Current status	Candidate Research Report
Publication authority	Formal Mythos approval not yet recorded
Next review	180 days or event-driven trigger, whichever occurs first

RES-023

Trusted Execution Environments (TEEs: TDX/SEV-SNP) Remote Attestation

Enterprise Research Report, Source-Freshness Upgrade, and Adoption Decision

RESEARCH DOMAIN	ADOPTION POSTURE
Confidential Computing	CONDITIONAL ADOPTION
EVIDENCE GRADE	**VALIDATED THROUGH**
A-	2026-07-22

Candidate Research Report

Mythos Systems | Research Intelligence Program | Public Technical Publication

This report preserves the complete source note while adding current authority validation, explicit limitations, an adoption decision, and a reproducible validation plan.

Document Control

Field	Value
Document ID	RES-023
Version	2.0.0
Status	Candidate Research Report
Classification	Public
Publisher	Mythos Systems
Program	Research Intelligence Program
Validated Through	2026-07-22
Scheduled Review	90 days
Adoption Posture	CONDITIONAL ADOPTION
Evidence Grade	A-
Source Lineage	RES-023_TEEs_TDX_SEV_SNP_Remote_Attestation.md
Series Integrity	RES-019 remains reserved; no source note was present in the corpus.

Contents

1. Research at a Glance

ADOPTION	EVIDENCE	REVIEW	SOURCE SET
CONDITIONAL ADOPTION	A-	90 days	3 current authorities

CURRENT ADOPTION DECISION

Adopt confidential-computing profiles for sensitive workloads where attestation, firmware trust, TCB scope, side-channel exposure, key release, and recovery are independently validated.

The original source note contains a strong architectural thesis, but its legacy “definitive,” “mandated,” or absolute language is not itself evidence. This edition preserves the complete source content while replacing unsupported certainty with a controlled research posture, validation conditions, and explicit limits.

2. Executive Research Position

Research domain: Confidential Computing. Current posture: CONDITIONAL ADOPTION. The subject is assessed using primary standards, official project documentation, vendor technical material where unavoidable, and primary research. Adoption is conditional on reproducing the relevant behavior in the intended environment.

The upgraded position distinguishes architectural usefulness from implementation maturity. A concept may be valuable enough to influence interfaces, data models, or long-term strategy while remaining unsuitable as a mandatory production dependency. Conversely, mature mechanisms with strong authority may be adopted immediately while still requiring environment-specific testing.

3. Current Authority and Freshness

Source	Authority and Status	Freshness Control
Intel Trust Domain Extensions Documentation https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/overview.html	Primary vendor specification and guidance Living Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.
AMD SEV-SNP Documentation https://www.amd.com/en/developer/sev.html	Primary vendor documentation Living Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.
Confidential Computing Consortium https://confidentialcomputing.io/	Primary industry consortium Living Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.

Freshness is controlled through both a scheduled review interval and event-driven triggers. A report becomes stale before its scheduled date when a governing standard changes, a cited project introduces a breaking release, a material vulnerability changes the risk posture, a research result is retracted, or internal validation contradicts the published position.

4. Explicit Research Limitations

- This report is a technical research publication, not a certification, legal opinion, procurement approval, or guarantee of production suitability.

- Source authority does not eliminate implementation risk. Product behavior, interoperability, performance, security, and operational fit require reproducible validation in the adopting environment.
- Fast-moving specifications, product documentation, open-source repositories, and research preprints may change before the next scheduled review.
- Quantitative claims from vendors or research papers are not treated as Mythos results unless independently reproduced under documented conditions.
- Adoption decisions are conditional on applicable policy, risk, licensing, export-control, privacy, safety, regulatory, and support requirements.

5. Adoption Decision and Guardrails

Adopt confidential-computing profiles for sensitive workloads where attestation, firmware trust, TCB scope, side-channel exposure, key release, and recovery are independently validated.

Decision dimension	Required direction
Production authority	No deployment or control authority is granted by this report. Aegis policy, formal change control, and system ownership remain authoritative.
Implementation	Use versioned adapters, explicit interfaces, bounded scope, observability, rollback, and evidence collection.
Validation	Reproduce mandatory claims using representative data, target hardware or software, failure cases, and documented acceptance criteria.
Procurement	Treat vendor claims as evidence inputs, not verified outcomes. Require support, lifecycle, licensing, security, and exit-plan analysis.
Retirement	Define conditions for rollback, replacement, data export, key rotation, state migration, and evidence preservation before adoption.

6. Validation and Reproducibility Plan

Stage	Required evidence	Exit condition
Source validation	Exact versions, publication status, revisions, and source hashes or retrieval records.	No high-impact claim depends on an untraceable or superseded source.
Prototype	Minimal implementation with representative data and instrumented execution.	Core mechanism is demonstrated without relying on presentation-only behavior.
Adversarial and failure testing	Malformed inputs, unavailable dependencies, stale state, rollback, isolation, resource exhaustion, and misuse cases.	Failure is bounded, observable, and recoverable.
Comparative baseline	Current production method or conventional alternative tested under the same workload.	Benefit is measurable against a credible baseline.
Operational trial	Ownership, alerts, support, patching, capacity, evidence, and exit procedures exercised.	The operating model is accepted by accountable owners.
Adoption review	Risk, legal, security, privacy, safety, cost, and architecture decision record.	Named authority approves the bounded use case.

7. Review and Expiration Triggers

- Scheduled review no later than 90 days after 2026-07-22.

- A cited standard, specification, API, product, model, runtime, or reference implementation introduces a material revision.
- A security advisory, safety event, interoperability defect, or legal change alters the risk or applicability.
- Internal benchmark, proof-of-concept, or production evidence contradicts a material claim.
- The adoption posture changes, the technology becomes a critical dependency, or a retirement decision is proposed.

8. Complete Source Research Note

SOURCE-LINEAGE NOTICE

The following material preserves the complete original source note, excluding only the conversational wrapper and outer Markdown fence. Legacy status labels and absolute language are retained for traceability and do not override the candidate status, limitations, or adoption decision in this edition.

Document ID: RES-023 Version: 1.0.0 (Definitive Registry Edition) Status: Published Research Note Classification: Public Organization: Mythos Systems (MYTHOS AI) Owner: Aaron Stovall Effective Date: 2026-07-16 Applies To: Security architects, cloud platform engineers, and AI engineers designing sovereign cloud deployments, zero-trust AI inference, and confidential computing pipelines Companion Standards: MYTHOS-SEC-0009 (Confidential Computing & TEE Standard), MYTHOS-HW-0002 (Trusted Compute & DPU), MYTHOS-DAT-0003 (Data Sovereignty & Egress), MYTHOS-EMT-0002 (Sovereign AI)

The architecture of cloud data sovereignty has failed. Organizations attempt to secure proprietary AI model weights and sensitive user data by encrypting it at rest (disk) and in transit (TLS). However, to process the data, it MUST be decrypted into the server's system RAM. At that moment, the data is fully exposed to the cloud provider, the hypervisor, and any malicious insider or privileged malware on the host. "Encrypt at rest" is a legal fiction when the cloud provider holds the keys and the RAM.

Trusted Execution Environments (TEEs), specifically Intel Trust Domain Extensions (TDX) and AMD Secure Encrypted Virtualization - Secure Nested Paging (SEV-SNP), shatter this vulnerability. They introduce Confidential VMs (CVMs), where the memory and CPU registers of a virtual machine are encrypted by dedicated hardware on the physical CPU package. The hypervisor is demoted to an untrusted resource allocator; it cannot read the VM's memory.

This research note defines the architectural dynamics of TEEs. It dissects the mechanics of hardware memory encryption, the cryptographic measurement of boot states, and the Remote Attestation pipeline. Understanding these dynamics is a prerequisite for building the zero-trust, sovereign AI inference clusters mandated by the Mythos Standards, where an organization can mathematically prove that a cloud provider cannot read their AI weights.

To understand TEEs, we must move from software-based encryption to hardware-rooted memory isolation.

1.1 Hardware Memory Encryption (MKTME / AES-XTS)

Modern x86 processors feature a Memory Encryption Engine (MEE) integrated into the memory controller.

- The Mechanic: When a TEE (TDX Guest or SEV-SNP VM) is launched, the CPU generates a unique, ephemeral Data Encryption Key (DEK) for that specific VM. This key never leaves the CPU silicon.
- The Execution: When the VM writes data to RAM, the CPU encrypts it on the fly. When the VM reads data, the CPU decrypts it on the fly. If the hypervisor, the host OS, or another VM attempts to read that RAM, they see only ciphertext.

1.2 Intel TDX (Trust Domain Extensions)

TDX introduces a new mode: the Trust Domain (TD).

- The Architecture: A TD runs in a hardware-isolated enclave that spans multiple vCPUs. The hypervisor (KVM/ESXi) manages memory allocation (assigning pages), but it cannot access the content of those pages. A secure arbiter (the TDX Module) sits between the hypervisor and the TD, enforcing memory access rights.

1.3 AMD SEV-SNP (Secure Nested Paging)

SEV-SNP protects VMs by encrypting the page tables and the memory itself.

- The Architecture: It prevents the hypervisor from performing "malicious hypervisor attacks" (e.g., remapping a VM's memory page to read its contents or causing a denial-of-service by altering page tables). It uses a Reverse Map Table (RMP) to ensure that only the designated VM can access its own pages.

While TEEs protect data in-use, they introduce severe physical and operational constraints that standard VMs do not face.

2.1 The Attestation Cold-Start Penalty

Before a TEE can process sensitive data, the client must verify the TEE's hardware state (Remote Attestation).

- The Flaw: This cryptographic challenge-response protocol adds 5 to 15 seconds of latency to the VM boot process.
- The Impact: TEEs cannot be used for rapid, sub-second serverless cold starts without maintaining warm pools of pre-attested CVMs.

2.2 The Memory Overhead

Memory encryption introduces a physical performance tax.

- The Constraint: Every memory read/write requires an AES-XTS cryptographic operation. Additionally, TEEs require a "memory integrity tree" (to prevent replay attacks) that consumes additional RAM.
- The Impact: TEE workloads typically incur a 5% to 15% performance overhead compared to plaintext VMs. For ultra-low-latency LLM inference, this overhead must be factored into the SLO.

2.3 The Side-Channel Blind Spot

TEEs protect against logical memory reads, but they do not inherently protect against micro-architectural side-channel attacks.

- The Flaw: Attackers can measure cache timing, power fluctuations, or electromagnetic emissions to infer what the TEE is computing.
- The Mitigation: TEEs MUST be paired with OS-level mitigations (e.g., page table isolation, cache flushing) and hardware-constant-time cryptographic libraries.

The true power of a TEE is not memory encryption; it is the ability to mathematically prove to a remote client that the memory is encrypted and running the correct software.

3.1 The Cryptographic Measurement (The Quote)

When the TEE boots, the CPU hashes every component: the firmware, the bootloader, the kernel, and the initial ramdisk.

- The Mechanic: These hashes are recorded in Platform Configuration Registers (PCRs) or TDX Measurement Registers (TMRs). The CPU signs this measurement log with a hardware-attested key (derived from a burned-in AMD/Intel root of trust), producing a "Quote."

3.2 Remote Attestation (The Verification)

The client (e.g., the PANTHEON control plane) requests the Quote.

- The Verification: The client checks the signature against a public key infrastructure (e.g., AMD Key Distribution Service or Intel SGX PCCS). If the signature is valid, the client knows the Quote originated from genuine hardware.
- The Policy Match: The client then compares the measurement hashes in the Quote against a known-good database (e.g., a signed manifest of the approved OS image). If the hashes match, the client mathematically proves that the VM is running the exact, unmodified approved software.

3.3 Sealing AI Weights (The Payload Release)

Once attestation is successful, the client can securely provision secrets.

- The Mechanic: The client retrieves the TEE's public key (derived from the hardware measurement). The client encrypts the AI model weights (or database credentials) using this key and sends the ciphertext to the TEE.
- The Absolute Guarantee: Only that specific, verified TEE possesses the private key to decrypt the weights. If the cloud provider tries to copy the weights to another machine, or alters the OS to read them, the hardware measurement changes, the private key is lost, and the weights remain encrypted ciphertext.

The fundamental shift of TEEs is the abolition of implicit trust in the cloud provider.

- Proprietary AI Protection: An organization can deploy a 70B parameter proprietary LLM to AWS or Azure. The model weights are sealed to the TEE. The cloud provider cannot extract the weights, even with physical access to the server.
- Privacy-Preserving Inference: Users can submit encrypted prompts to the TEE. The TEE decrypts the prompt internally, runs the inference, and returns the encrypted result. The cloud provider cannot read the user's prompt or the model's output.
- Verifiable Compute: Regulators can audit the attestation logs to mathematically prove that a financial AI model was running on approved, unmodified hardware.

To deploy TEEs in production, the Mythos Architecture enforces strict attestation gates and operational boundaries.

5.1 The Zero-Trust Provisioning Gate

The Mythos control plane MUST NOT push AI weights or data to a CVM until a valid attestation Quote is verified.

- The Mitigation: The deployment pipeline (Clio/Morta) intercepts the deployment. The CVM boots with a generic OS. The CVM sends its Quote to the control plane. The control plane verifies the Quote against a SBOM/Reference Value. Only upon a 100% match are the weights streamed via a sealed TLS tunnel to the TEE.

5.2 Attested Observability (The Verifiable Telemetry)

If the TEE is a black box, how does the organization know what the agent inside it is doing?

- The Mitigation: The observability pipeline (Argus/OpenTelemetry) MUST sign the cognitive traces inside the TEE using the TEE's hardware key. The downstream SIEM verifies the signature. If the hypervisor attempts to alter the logs in transit, the signature breaks, alerting the SOC to a host compromise.

5.3 The TEE-FHE Hybrid (Maximum Security)

For Level 5 sovereign data (e.g., classified CUI), TEEs alone are insufficient because they rely on the integrity of the CPU firmware.

- The Mitigation: The architecture MUST combine TEEs with Fully Homomorphic Encryption (FHE). The user sends an FHE-encrypted prompt. The TEE receives the ciphertext. The TEE performs the LLM inference homomorphically (utilizing the TEE's hardware to accelerate FHE). The result is returned as FHE ciphertext. Even if the TEE's memory encryption is somehow bypassed, the data is still mathematically encrypted by FHE.

Trust in cloud providers is a liability, not a security strategy. Legal agreements and compliance audits cannot prevent a hypervisor zero-day or a malicious insider from reading plaintext RAM. Trusted Execution Environments, via hardware memory encryption and remote attestation, transform the cloud into a blind, untrusted utility. By sealing proprietary AI weights to verified hardware states, we achieve the ultimate mandate of data sovereignty: the compute happens in the cloud, but the ownership remains absolute.

```
Encryption at rest is a legal fiction; encryption in-use is a mathematical truth.
A hypervisor with read access is an insider threat; a demoted hypervisor is a utility.
```

```
A promise of security is a liability; a hardware attestation is a proof.
A sealed weight is a cipher; an attested TEE is the key.

The cloud provider sees only ciphertext; the math sees all.

> Clouds may be shared.
> Memory sovereignty must be absolute.
```

End of Research Note RES-023

9. Source Register

Source	Authority and Status	Freshness Control
Intel Trust Domain Extensions Documentation https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/overview.html	Primary vendor specification and guidance Living Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.
AMD SEV-SNP Documentation https://www.amd.com/en/developer/sev.html	Primary vendor documentation Living Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.
Confidential Computing Consortium https://confidentialcomputing.io/	Primary industry consortium Living Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.

10. Lineage, Review, and Publication Record

Record	Value
Original source file	RES-023_TEEs_TDX_SEV_SNP_Remote_Attestation.md
Original source words	1638
Upgrade type	Enterprise research report; source and freshness validation; limitations; adoption decision; reproducibility plan
Generated on	2026-07-22
Current status	Candidate Research Report
Publication authority	Formal Mythos approval not yet recorded
Next review	90 days or event-driven trigger, whichever occurs first

RES-025

Decentralized Physical Infrastructure Networks (DePIN) Compute Models

Enterprise Research Report, Source-Freshness Upgrade, and Adoption Decision

RESEARCH DOMAIN	ADOPTION POSTURE
Decentralized Infrastructure	MONITOR - DO NOT USE FOR SENSITIVE WORKLOADS
EVIDENCE GRADE	**VALIDATED THROUGH**
B-	2026-07-22

Candidate Research Report

Mythos Systems | Research Intelligence Program | Public Technical Publication

This report preserves the complete source note while adding current authority validation, explicit limitations, an adoption decision, and a reproducible validation plan.

Document Control

Field	Value
Document ID	RES-025
Version	2.0.0
Status	Candidate Research Report
Classification	Public
Publisher	Mythos Systems
Program	Research Intelligence Program
Validated Through	2026-07-22
Scheduled Review	90 days
Adoption Posture	MONITOR - DO NOT USE FOR SENSITIVE WORKLOADS
Evidence Grade	B-
Source Lineage	RES-025_DePIN_Compute_Models.md
Series Integrity	RES-019 remains reserved; no source note was present in the corpus.

Contents

1. Research at a Glance

ADOPTION	EVIDENCE	REVIEW	SOURCE SET
MONITOR - DO NOT USE FOR SENSITIVE WORKLOADS	B-	90 days	3 current authorities

CURRENT ADOPTION DECISION

Monitor DePIN economics and scheduling. Do not place secrets, regulated data, proprietary models, or high-integrity workloads on untrusted nodes without independently verified confidentiality and execution.

The original source note contains a strong architectural thesis, but its legacy “definitive,” “mandated,” or absolute language is not itself evidence. This edition preserves the complete source content while replacing unsupported certainty with a controlled research posture, validation conditions, and explicit limits.

2. Executive Research Position

Research domain: Decentralized Infrastructure. Current posture: MONITOR - DO NOT USE FOR SENSITIVE WORKLOADS. The subject is assessed using primary standards, official project documentation, vendor technical material where unavoidable, and primary research. Adoption is conditional on reproducing the relevant behavior in the intended environment.

The upgraded position distinguishes architectural usefulness from implementation maturity. A concept may be valuable enough to influence interfaces, data models, or long-term strategy while remaining unsuitable as a mandatory production dependency. Conversely, mature mechanisms with strong authority may be adopted immediately while still requiring environment-specific testing.

3. Current Authority and Freshness

Source	Authority and Status	Freshness Control
Akash Network Documentation https://akash.network/docs/	Primary network documentation Living Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.
Helium Documentation https://docs.helium.com/	Primary network documentation Living Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.
Confidential Computing Consortium https://confidentialcomputing.io/	Primary industry consortium Living Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.

Freshness is controlled through both a scheduled review interval and event-driven triggers. A report becomes stale before its scheduled date when a governing standard changes, a cited project introduces a breaking release, a material vulnerability changes the risk posture, a research result is retracted, or internal validation contradicts the published position.

4. Explicit Research Limitations

- This report is a technical research publication, not a certification, legal opinion, procurement approval, or guarantee of production suitability.
- Source authority does not eliminate implementation risk. Product behavior, interoperability, performance, security, and operational fit require reproducible validation in the adopting environment.
- Fast-moving specifications, product documentation, open-source repositories, and research preprints may change before the next scheduled review.
- Quantitative claims from vendors or research papers are not treated as Mythos results unless independently reproduced under documented conditions.
- Adoption decisions are conditional on applicable policy, risk, licensing, export-control, privacy, safety, regulatory, and support requirements.
- The technology is emerging and may have limited independent production evidence, immature tooling, scarce hardware access, or rapidly changing performance claims.

5. Adoption Decision and Guardrails

Monitor DePIN economics and scheduling. Do not place secrets, regulated data, proprietary models, or high-integrity workloads on untrusted nodes without independently verified confidentiality and execution.

Decision dimension	Required direction
Production authority	No deployment or control authority is granted by this report. Aegis policy, formal change control, and system ownership remain authoritative.
Implementation	Use versioned adapters, explicit interfaces, bounded scope, observability, rollback, and evidence collection.
Validation	Reproduce mandatory claims using representative data, target hardware or software, failure cases, and documented acceptance criteria.
Procurement	Treat vendor claims as evidence inputs, not verified outcomes. Require support, lifecycle, licensing, security, and exit-plan analysis.
Retirement	Define conditions for rollback, replacement, data export, key rotation, state migration, and evidence preservation before adoption.

6. Validation and Reproducibility Plan

Stage	Required evidence	Exit condition
Source validation	Exact versions, publication status, revisions, and source hashes or retrieval records.	No high-impact claim depends on an untraceable or superseded source.
Prototype	Minimal implementation with representative data and instrumented execution.	Core mechanism is demonstrated without relying on presentation-only behavior.
Adversarial and failure testing	Malformed inputs, unavailable dependencies, stale state, rollback, isolation, resource exhaustion, and misuse cases.	Failure is bounded, observable, and recoverable.
Comparative baseline	Current production method or conventional alternative tested under the same workload.	Benefit is measurable against a credible baseline.
Operational trial	Ownership, alerts, support, patching, capacity, evidence, and exit procedures exercised.	The operating model is accepted by accountable owners.
Adoption review	Risk, legal, security, privacy, safety, cost, and architecture decision record.	Named authority approves the bounded use case.

7. Review and Expiration Triggers

- Scheduled review no later than 90 days after 2026-07-22.
- A cited standard, specification, API, product, model, runtime, or reference implementation introduces a material revision.
- A security advisory, safety event, interoperability defect, or legal change alters the risk or applicability.
- Internal benchmark, proof-of-concept, or production evidence contradicts a material claim.
- The adoption posture changes, the technology becomes a critical dependency, or a retirement decision is proposed.

8. Complete Source Research Note

SOURCE-LINEAGE NOTICE

The following material preserves the complete original source note, excluding only the conversational wrapper and outer Markdown fence. Legacy status labels and absolute language are retained for traceability and do not override the candidate status, limitations, or adoption decision in this edition.

Document ID: RES-025 Version: 1.0.0 (Definitive Registry Edition) Status: Published Research Note Classification: Public Organization: Mythos Systems (MYTHOS AI) Owner: Aaron Stovall Effective Date: 2026-07-16 Applies To: Sovereign infrastructure architects, Web3/decentralized compute engineers, and security professionals designing verifiable distributed AI pipelines, zero-trust compute markets, and air-gapped burst-compute environments Companion Standards: MYTHOS-CLD-0009 (DePIN Standard), MYTHOS-EMT-0002 (Sovereign AI), MYTHOS-SEC-0009 (Confidential Computing), MYTHOS-SEC-0010 (FHE & ZKP), MYTHOS-HW-0002 (Trusted Compute)

The architecture of cloud computing has been crippled by monopolistic centralization. Organizations are forced to surrender their proprietary AI weights and data to a handful of centralized cloud providers (AWS, Azure, GCP) to access massive GPU compute. This model destroys data sovereignty, creates single points of failure, and locks organizations into vendor-controlled pricing.

Decentralized Physical Infrastructure Networks (DePIN) offer an alternative: a global, blockchain-validated market of distributed compute nodes. However, naively sending proprietary workloads to anonymous, internet-connected GPUs is a catastrophic security failure. An anonymous node operator can easily exfiltrate model weights or return hallucinated, unexecuted inference results.

This research note defines the architectural dynamics of Sovereign DePIN Compute. It dissects the mechanics of smart-contract Service Level Agreements (SLAs), Proof of Compute (PoC) via Zero-Knowledge Proofs (zk-ML) and Trusted Execution Environments (TEEs), and the cryptographic attestation required to enable highly secure, air-gapped organizations to safely bid on distributed compute. Understanding these dynamics is a prerequisite for building the verifiable, zero-trust decentralized compute fabrics mandated by the Mythos Cloud Standards.

To utilize decentralized compute securely, we must map how a compute workload is advertised, bid upon, executed, and verified on a blockchain.

1.1 The Compute Bidding Protocol (Smart Contracts)

DePIN compute is governed by smart contracts acting as an automated escrow and matching engine.

- The Mechanic: A sovereign organization submits an encrypted compute request (e.g., "Inference for Model X on Encrypted Input Y for Z tokens"). Decentralized node operators stake collateral and bid on the contract. The smart contract automatically matches the bid to the highest-staked, lowest-latency node.
- The SLA Enforcement: If the node fails to return the result within the block time, or if the result fails cryptographic verification, the smart contract automatically slashes the node's staked collateral.

1.2 Proof of Compute (PoC)

A node cannot simply return a JSON payload and claim it ran a 70-billion parameter model. The network must mathematically verify the execution.

- The Mechanic: The node must return the result alongside a cryptographic proof. This is achieved via zk-ML (Zero-Knowledge Machine Learning) or TEE Remote Attestation. The blockchain's consensus layer verifies the proof before releasing the escrowed payment to the node.

1.3 Sybil Resistance & Hardware Attestation

A single attacker could spin up 1,000 virtual nodes to monopolize the network.

- The Mechanic: DePIN networks MUST require Proof of Physical Work (PoPW) or hardware-bound attestation. Nodes must provide a TPM 2.0 or TEE signature proving they are running on physical, unique silicon, preventing virtual Sybil cloning.

While DePIN breaks the cloud monopoly, it introduces severe physical and security constraints that centralized clouds abstract away.

2.1 The Untrusted Node Problem (Model Exfiltration)

An anonymous node operator receives a proprietary AI model to execute. The operator can easily copy the `.safetensors` weights from system RAM and sell them to a competitor.

- The Flaw: Standard API-based compute guarantees data exfiltration if the endpoint is compromised.
- The Mitigation: DePIN compute MUST utilize TEEs (Intel TDX / AMD SEV-SNP). The model is encrypted in transit and only decrypted inside the hardware-isolated enclave. The node operator's host OS and hypervisor cannot read the weights.

2.2 Network Topology and Jitter

Centralized datacenters possess uniform, 400Gbps non-blocking spine-leaf fabrics. DePIN nodes are scattered across the globe on consumer broadband and varied ISPs.

- The Constraint: Distributed training (which requires microsecond All-Reduce synchronization) is physically impossible across a DePIN mesh due to jitter and latency.
- The Mitigation: DePIN compute MUST be restricted to embarrassingly parallel workloads (e.g., batch inference, hyperparameter search, data parallelization) where nodes do not need to synchronize state.

2.3 Economic Volatility (The SLA Stability Problem)

If a DePIN network's native token crashes in value, node operators will instantly turn off their machines to avoid electricity costs, halting global compute mid-workflow.

- The Constraint: Volatile tokenomics cannot guarantee enterprise SLAs.
- The Mitigation: The smart contract SLA MUST be denominated in a stablecoin or dual-token architecture. Furthermore, the compute orchestrator MUST implement checkpointing; if a node drops offline, the durable runtime (Temporal/Clio) seamlessly re-bids the workload to another node without losing state.

The true power of a Mythos-compliant DePIN architecture is realized when highly secure, air-gapped environments (e.g., DoD, classified enterprise) can safely leverage external, decentralized compute without breaching their physical perimeter.

3.1 The Data Diode Bid

An air-gapped organization cannot connect to the public internet to query a DePIN network.

- The Mechanic: The organization utilizes a one-way data diode. The compute request (an FHE-encrypted prompt and a ZK-verification key) is pushed through the diode to a gateway node. The gateway node interacts with the DePIN smart contract, bids on the compute, and waits for the result.
- The Impact: The air-gapped organization leverages global compute power, but the physical network boundary is never breached. The data flowing back through the diode is mathematically verified (ZKP) and encrypted (FHE).

3.2 Fully Homomorphic Encryption (FHE) on DePIN

Even with TEEs, some organizations classify the hardware itself as untrusted.

- The Mechanic: The organization encrypts the prompt and the model using FHE. The DePIN node executes the inference homomorphically. The node never possesses the decryption key.
- The Impact: The compute is mathematically blind. The node returns an encrypted result, which the air-gapped organization decrypts locally. The DePIN node could be owned by a hostile nation-state, and the data would remain perfectly secure.

The fundamental shift of DePIN compute is the transition from "trust the cloud provider" to "trust the math."

- Burst Compute: An organization training a 1-trillion parameter model can burst 50% of the hyperparameter search to a DePIN network, cutting cloud costs by 80% while maintaining cryptographic proof of execution.
- Censorship Resistance: A sovereign AI researcher in a heavily regulated jurisdiction can publish their model to a DePIN network. Because the nodes are decentralized and the smart contracts are autonomous, no single government can force a takedown of the compute.
- Verifiable AI: When an agent retrieves an inference result from a DePIN node, it attaches the ZK-proof or TEE-attestation to its Evidence Bundle. The downstream consumer can mathematically verify that the AI output was generated by the correct, unmodified model on the correct input.

To deploy DePIN compute in production, the Mythos Architecture enforces strict orchestration and verification boundaries.

5.1 The DePIN Routing Gateway

The PANTHEON Nona (Planner) module MUST NOT directly interact with a blockchain RPC.

- The Mitigation: The architecture MUST utilize a DePIN Routing Gateway. Nona proposes a standard tool call. The Gateway intercepts it, formats it into a DePIN smart-contract bid, manages the escrow, verifies the ZK-proof/attestation returned by the node, and passes the result back to Nona. The cognitive loop is completely abstracted from the blockchain mechanics.

5.2 Mandatory Verification Gates

No DePIN compute result enters the active memory graph (Mnemosyne) without passing a verification gate.

- The Mitigation: If the DePIN node returns a result lacking a valid TEE attestation quote or a valid zk-STARK proof, the Routing Gateway instantly drops the result, slashes the node via the smart contract, and re-bids the workload. Zero unverified data touches the core agent state.

5.3 Stable Economic Quorums

To prevent token volatility from crashing the compute pipeline, the Mythos architecture mandates economic abstraction.

- The Mitigation: The organization pre-funds a smart-contract wallet with stablecoin. The DePIN Routing Gateway draws from this pool. The node operators are paid in stablecoin, insulating them from token volatility and ensuring enterprise-grade uptime guarantees.

Decentralized Physical Infrastructure Networks break the centralized cloud monopoly, offering limitless, censorship-resistant compute. However, leveraging anonymous hardware for sovereign AI requires absolute cryptographic enforcement. By combining smart-contract SLAs, TEE hardware isolation, and Zero-Knowledge verifiability, we transform a chaotic mesh of internet-connected GPUs into a mathematically verifiable, zero-trust compute utility. In a Mythos-compliant architecture, the compute provider does not need to be trusted; the math guarantees the result.

```
A centralized cloud is a monopoly; a DePIN mesh is a market.
```

```
An anonymous node is a thief; a TEE enclave is a vault.
A JSON response is a claim; a ZK proof is a settlement.
A volatile token is a risk; a stablecoin SLA is a guarantee.

The compute is decentralized; the verification is absolute.

> Nodes may be anonymous.
> Cryptographic provenance must be absolute.
```

End of Research Note RES-025

9. Source Register

Source	Authority and Status	Freshness Control
Akash Network Documentation https://akash.network/docs/	Primary network documentation Living Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.
Helium Documentation https://docs.helium.com/	Primary network documentation Living Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.
Confidential Computing Consortium https://confidentialcomputing.io/	Primary industry consortium Living Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.

10. Lineage, Review, and Publication Record

Record	Value
Original source file	RES-025_DePIN_Compute_Models.md
Original source words	1528
Upgrade type	Enterprise research report; source and freshness validation; limitations; adoption decision; reproducibility plan
Generated on	2026-07-22
Current status	Candidate Research Report
Publication authority	Formal Mythos approval not yet recorded
Next review	90 days or event-driven trigger, whichever occurs first

RES-027

AI-Driven Radio Access Networks (RAN) & vRAN Orchestration

Enterprise Research Report, Source-Freshness Upgrade, and Adoption Decision

RESEARCH DOMAIN	ADOPTION POSTURE
Radio Access Networks	MONITOR AND TELECOM PILOT
EVIDENCE GRADE	**VALIDATED THROUGH**
B+	2026-07-22

Candidate Research Report

Mythos Systems | Research Intelligence Program | Public Technical Publication

This report preserves the complete source note while adding current authority validation, explicit limitations, an adoption decision, and a reproducible validation plan.

Document Control

Field	Value
Document ID	RES-027
Version	2.0.0
Status	Candidate Research Report
Classification	Public
Publisher	Mythos Systems
Program	Research Intelligence Program
Validated Through	2026-07-22
Scheduled Review	120 days
Adoption Posture	MONITOR AND TELECOM PILOT
Evidence Grade	B+
Source Lineage	RES-027_AI_Driven_RAN_vRAN_Orchestration.md
Series Integrity	RES-019 remains reserved; no source note was present in the corpus.

Contents

1. Research at a Glance

ADOPTION	EVIDENCE	REVIEW	SOURCE SET
MONITOR AND TELECOM PILOT	B+	120 days	2 current authorities

CURRENT ADOPTION DECISION

Pilot AI-assisted RAN optimization only in operator-grade test environments with deterministic policy bounds, rollback, timing validation, and O-RAN interface conformance.

The original source note contains a strong architectural thesis, but its legacy “definitive,” “mandated,” or absolute language is not itself evidence. This edition preserves the complete source content while replacing unsupported certainty with a controlled research posture, validation conditions, and explicit limits.

2. Executive Research Position

Research domain: Radio Access Networks. Current posture: MONITOR AND TELECOM PILOT. The subject is assessed using primary standards, official project documentation, vendor technical material where unavoidable, and primary research. Adoption is conditional on reproducing the relevant behavior in the intended environment.

The upgraded position distinguishes architectural usefulness from implementation maturity. A concept may be valuable enough to influence interfaces, data models, or long-term strategy while remaining unsuitable as a mandatory production dependency. Conversely, mature mechanisms with strong authority may be adopted immediately while still requiring environment-specific testing.

3. Current Authority and Freshness

Source	Authority and Status	Freshness Control
O-RAN Alliance Specifications https://www.o-ran.org/specifications	Primary industry specifications Living - access and version dependent Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.
O-RAN Software Community Documentation https://docs.o-ran-sc.org/	Primary open implementation documentation Living Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.

Freshness is controlled through both a scheduled review interval and event-driven triggers. A report becomes stale before its scheduled date when a governing standard changes, a cited project introduces a breaking release, a material vulnerability changes the risk posture, a research result is retracted, or internal validation contradicts the published position.

4. Explicit Research Limitations

- This report is a technical research publication, not a certification, legal opinion, procurement approval, or guarantee of production suitability.
- Source authority does not eliminate implementation risk. Product behavior, interoperability, performance, security, and operational fit require reproducible validation in the adopting environment.

- Fast-moving specifications, product documentation, open-source repositories, and research preprints may change before the next scheduled review.
- Quantitative claims from vendors or research papers are not treated as Mythos results unless independently reproduced under documented conditions.
- Adoption decisions are conditional on applicable policy, risk, licensing, export-control, privacy, safety, regulatory, and support requirements.
- Protocol support, silicon behavior, vendor implementation, licensing, topology, and operational maturity can materially differ from the specification or lab model.

5. Adoption Decision and Guardrails

Pilot AI-assisted RAN optimization only in operator-grade test environments with deterministic policy bounds, rollback, timing validation, and O-RAN interface conformance.

Decision dimension	Required direction
Production authority	No deployment or control authority is granted by this report. Aegis policy, formal change control, and system ownership remain authoritative.
Implementation	Use versioned adapters, explicit interfaces, bounded scope, observability, rollback, and evidence collection.
Validation	Reproduce mandatory claims using representative data, target hardware or software, failure cases, and documented acceptance criteria.
Procurement	Treat vendor claims as evidence inputs, not verified outcomes. Require support, lifecycle, licensing, security, and exit-plan analysis.
Retirement	Define conditions for rollback, replacement, data export, key rotation, state migration, and evidence preservation before adoption.

6. Validation and Reproducibility Plan

Stage	Required evidence	Exit condition
Source validation	Exact versions, publication status, revisions, and source hashes or retrieval records.	No high-impact claim depends on an untraceable or superseded source.
Prototype	Minimal implementation with representative data and instrumented execution.	Core mechanism is demonstrated without relying on presentation-only behavior.
Adversarial and failure testing	Malformed inputs, unavailable dependencies, stale state, rollback, isolation, resource exhaustion, and misuse cases.	Failure is bounded, observable, and recoverable.
Comparative baseline	Current production method or conventional alternative tested under the same workload.	Benefit is measurable against a credible baseline.
Operational trial	Ownership, alerts, support, patching, capacity, evidence, and exit procedures exercised.	The operating model is accepted by accountable owners.
Adoption review	Risk, legal, security, privacy, safety, cost, and architecture decision record.	Named authority approves the bounded use case.

7. Review and Expiration Triggers

- Scheduled review no later than 120 days after 2026-07-22.

- A cited standard, specification, API, product, model, runtime, or reference implementation introduces a material revision.
- A security advisory, safety event, interoperability defect, or legal change alters the risk or applicability.
- Internal benchmark, proof-of-concept, or production evidence contradicts a material claim.
- The adoption posture changes, the technology becomes a critical dependency, or a retirement decision is proposed.

8. Complete Source Research Note

SOURCE-LINEAGE NOTICE

The following material preserves the complete original source note, excluding only the conversational wrapper and outer Markdown fence. Legacy status labels and absolute language are retained for traceability and do not override the candidate status, limitations, or adoption decision in this edition.

Document ID: RES-027 Version: 1.0.0 (Definitive Registry Edition) Status: Published Research Note Classification: Public Organization: Mythos Systems (MYTHOS AI) Owner: Aaron Stovall Effective Date: 2026-07-16 Applies To: Telecom architects, 5G/6G service providers, private 5G network engineers, and AI researchers designing O-RAN architectures, vRAN orchestration, and AI-driven RF optimization Companion Standards: MYTHOS-TEL-0001 (5G/6G Core, Open RAN & Slicing), MYTHOS-EMT-0003 (Neuromorphic & NPU), MYTHOS-AI-0014 (Inference Serving), MYTHOS-PLT-0006 (Digital Marketplace & Extension Architecture)

The architecture of mobile telecommunications has failed. For decades, service providers have been held hostage by monolithic, proprietary Radio Access Network (RAN) vendors (e.g., Ericsson, Nokia, Huawei). The baseband processing—the brain of the cell tower—is a closed black box. RF optimization relies on static, rules-based algorithms that cannot adapt to dynamic environments, resulting in chronic interference, wasted spectrum, and massive operational expenditure.

The O-RAN (Open Radio Access Network) architecture shatters this proprietary monopoly. It disaggregates the RAN into software-defined Virtualized RAN (vRAN) units—Radio Units (RU), Distributed Units (DU), and Centralized Units (CU)—connected by open fronthaul interfaces.

This research note defines the architectural dynamics of AI-Driven vRAN. It dissects the mechanics of the RAN Intelligent Controller (RIC), the deployment of xApps and rApps for real-time AI/ML optimization, and the transition from static codebooks to dynamic, AI-driven beamforming and interference mitigation. Understanding these dynamics is a prerequisite for building the sovereign, intelligent telecom fabrics mandated by the Mythos Telecom Standards.

To understand AI-driven RAN, we must move from purpose-built silicon to software-defined components connected by open interfaces.

1.1 The Disaggregated RAN (RU, DU, CU)

Traditional base stations combine radio, MAC, and network processing into one proprietary box. O-RAN splits this:

- Radio Unit (RU): The physical antenna and RF transceiver. Converts digital signals to radio waves.
- Distributed Unit (DU): Handles the real-time, latency-critical Layer 1 (Physical) and Layer 2 (MAC/RLC) processing. Usually deployed at the cell tower edge.
- Centralized Unit (CU): Handles the non-real-time Layer 3 (RRC) processing and connectivity to the 5G Core. Usually deployed in a regional datacenter.

1.2 The Open Fronthaul (7.2x)

The interface between the RU and the DU. Historically, this was a proprietary fiber connection. O-RAN defines an open standard (eCPRI over UDP/IP), allowing an RU from Vendor A to communicate with a DU from Vendor B.

- The Constraint: This interface requires sub-100 microsecond latency and nanosecond timing synchronization. Any jitter causes radio desynchronization and dropped calls.

1.3 The RAN Intelligent Controller (RIC)

The brain of the AI-driven RAN. The RIC is a cloud-native platform deployed in the CU or 5G Core. It hosts third-party applications (xApps/rApps) that ingest telemetry from the RAN and use AI/ML to optimize the network in real-time.

- Near-RT RIC: Sub-10ms to 1s latency. Handles real-time radio resource management, interference mitigation, and beamforming.
- Non-RT RIC: > 1s latency. Handles service and policy management, guiding the Near-RT RIC with high-level objectives.

While vRAN breaks the vendor monopoly, it introduces severe physical and computational constraints that proprietary hardware abstracted away.

2.1 The L1 Compute Tax (vRAN Processing)

The Layer 1 (Physical Layer) of 5G—handling OFDM modulation, channel coding (LDPC/Polar), and Fast Fourier Transforms (FFT)—is incredibly compute-intensive.

- The Flaw: Proprietary baseband units use custom ASICs to do this efficiently. Running L1 on general-purpose x86 CPUs in a vRAN DU exhausts the CPU cores and struggles to hit 100MHz bandwidths.
- The Mitigation: vRAN DUs MUST utilize hardware accelerators. Modern architectures offload L1 processing to FPGAs, eFPGAs, or dedicated vRAN ASICs (e.g., Intel vRAN Boost, Qualcomm RAN accelerators).

2.2 The Fronthaul Bandwidth and Latency Wall

Splitting the RU and DU requires massive bandwidth. A standard 4T4R 100MHz 5G cell requires ~10 to 25 Gbps of fronthaul bandwidth per cell site.

- The Constraint: Fiber is mandatory. Microwave or copper cannot support O-RAN fronthaul. Furthermore, the DU cannot be geographically far from the RU; the round-trip latency must remain under 100 microseconds to prevent timing violations.

2.3 The xApp Supply Chain Threat

The Near-RT RIC is an open platform. Third-party vendors write xApps (e.g., an AI interference mitigator) and deploy them to the RIC.

- The Flaw: A malicious xApp could intentionally drop calls for specific users, act as a wiretap, or crash the cell tower. Treating xApps as "trusted" code is a critical failure.

The true power of O-RAN is realized when the RIC utilizes AI/ML to replace static, rules-based RF engineering.

3.1 Dynamic Interference Mitigation

Traditional networks statically allocate frequency bands to cells to avoid overlap. This wastes spectrum when cells are idle.

- The Mechanic: An AI xApp in the Near-RT RIC ingests real-time User Equipment (UE) signal-to-interference-plus-noise ratio (SINR) reports. The AI predicts interference patterns and dynamically reallocates resource blocks (PRBs) and transmission power in < 10ms, maximizing throughput without wasting spectrum.

3.2 Massive MIMO (mMIMO) Beamforming Optimization

5G utilizes Massive MIMO—arrays of 32, 64, or 128 antennas. Traditional systems use fixed "codebooks" of beam patterns.

- The Mechanic: Deep Reinforcement Learning (RL) models in the RIC analyze user mobility and multipath reflections. The AI dynamically computes custom, narrow beam weights for individual users, focusing RF energy precisely where it is needed, extending cell range and reducing battery drain on UEs.

3.3 Predictive Handover (Zero-Drop Mobility)

When a user moves between cell towers, the network must hand over the connection. Traditional handovers are reactive; they wait for the signal to drop before switching, causing dropped calls.

- The Mechanic: The Non-RT RIC analyzes historical traffic patterns and UE trajectory. It predicts that a user will enter a tunnel in 30 seconds and proactively steers the connection to a macro cell, ensuring a zero-drop handover before the signal is lost.

The fundamental shift of AI-driven vRAN is the transition from static hardware to cognitive, self-optimizing software.

- Energy Efficiency: The AI RIC learns traffic patterns. At 3 AM, when a cell site has zero traffic, the AI puts the DU and RU into deep sleep, waking them up milliseconds before the first morning packet arrives. This cuts cell site power consumption by 40%.
- Private 5G Sovereignty: Enterprises can deploy Private 5G networks using off-the-shelf white-box RUs and open-source DUs. The AI RIC optimizes the network specifically for AGVs (Automated Guided Vehicles) and robots, completely decoupled from public carrier infrastructure.
- Spectrum Sharing (CBRS): AI models dynamically negotiate shared spectrum (e.g., CBRS in the US), allowing military and commercial users to share the same frequency bands without interference, mathematically guaranteed by the RIC.

To deploy AI-driven vRAN in production, the Mythos Architecture enforces strict hardware isolation, supply chain governance, and NPU offloading.

5.1 xApp Sandboxing (Zero-Trust RIC)

The RIC MUST NOT execute xApps as native processes.

- The Mitigation: The Mythos standard mandates that all xApps and rApps MUST be compiled to WebAssembly (WASM) (`MYTHOS-EMT-0001`). The RIC executes xApps in a WASM sandbox with deny-by-default capabilities. An xApp can read telemetry, but if it attempts to execute a malicious network command (e.g., `DROP ALL CALLS`), the RIC blocks the action.

5.2 NPU/FPGA Acceleration for RIC Inference

Running AI inference models (for beamforming or interference) on the x86 CPU of the RIC adds latency.

- The Mitigation: The RIC MUST utilize heterogeneous compute. The AI models (TensorFlow/PyTorch) MUST be compiled to execute on local NPUs or FPGAs (`MYTHOS-EMT-0003`), ensuring the < 10ms inference loop is mathematically bounded.

5.3 Continuous A/B Testing via Digital Twins

Deploying a new xApp to a live 5G network is risky; a bug can take down a city's connectivity.

- The Mitigation: The xApp deployment pipeline MUST utilize a Containerlab/Batfish digital twin (`MYTHOS-NET-0003`). The CI/CD pipeline simulates RF traffic, injects interference, and verifies the xApp optimizes the network without crashing before it is deployed to production RICs.

The era of proprietary, static, rules-based telecom hardware is over. O-RAN and vRAN disaggregate the cell tower into software, and AI transforms that software into a cognitive, self-healing fabric. By deploying AI-driven xApps in zero-trust WASM sandboxes and accelerating L1 processing on dedicated silicon, we break the vendor monopoly and unlock the ultimate goal of telecom: a network that adapts to reality in real-time, ensuring absolute connectivity for sovereign and enterprise infrastructures.

```
A proprietary baseband is a monopoly; a vRAN is a commodity.
A static codebook is a guess; an AI beamformer is a laser.
A reactive handover drops calls; a predictive handover is seamless.
```

```
The RAN is no longer a radio; it is a cognitive application.

> Spectrum may be physical.
> Cognitive optimization must be absolute.
```

End of Research Note RES-027

9. Source Register

Source	Authority and Status	Freshness Control
O-RAN Alliance Specifications https://www.o-ran.org/specifications	Primary industry specifications Living - access and version dependent Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.
O-RAN Software Community Documentation https://docs.o-ran-sc.org/	Primary open implementation documentation Living Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.

10. Lineage, Review, and Publication Record

Record	Value
Original source file	RES-027_AI_Driven_RAN_vRAN_Orchestration.md
Original source words	1502
Upgrade type	Enterprise research report; source and freshness validation; limitations; adoption decision; reproducibility plan
Generated on	2026-07-22
Current status	Candidate Research Report
Publication authority	Formal Mythos approval not yet recorded
Next review	120 days or event-driven trigger, whichever occurs first

RES-028

5G/6G Network Slicing & QoS Enforcement

Enterprise Research Report, Source-Freshness Upgrade, and Adoption Decision

RESEARCH DOMAIN	ADOPTION POSTURE
5G and 6G Networks	ADOPT 5G CONTROLS; MONITOR 6G
EVIDENCE GRADE	**VALIDATED THROUGH**
A-	2026-07-22

Candidate Research Report

Mythos Systems | Research Intelligence Program | Public Technical Publication

This report preserves the complete source note while adding current authority validation, explicit limitations, an adoption decision, and a reproducible validation plan.

Document Control

Field	Value
Document ID	RES-028
Version	2.0.0
Status	Candidate Research Report
Classification	Public
Publisher	Mythos Systems
Program	Research Intelligence Program
Validated Through	2026-07-22
Scheduled Review	90 days
Adoption Posture	ADOPT 5G CONTROLS; MONITOR 6G
Evidence Grade	A-
Source Lineage	RES-028_5G_6G_Network_Slicing_QoS_Enforcement.md
Series Integrity	RES-019 remains reserved; no source note was present in the corpus.

Contents

1. Research at a Glance

ADOPTION	EVIDENCE	REVIEW	SOURCE SET
ADOPT 5G CONTROLS; MONITOR 6G	A-	90 days	2 current authorities

CURRENT ADOPTION DECISION

Adopt explicit end-to-end 5G slice assurance and QoS verification where slicing is used. Treat 6G architecture and Release 21 material as evolving research and standards work.

The original source note contains a strong architectural thesis, but its legacy “definitive,” “mandated,” or absolute language is not itself evidence. This edition preserves the complete source content while replacing unsupported certainty with a controlled research posture, validation conditions, and explicit limits.

2. Executive Research Position

Research domain: 5G and 6G Networks. Current posture: ADOPT 5G CONTROLS; MONITOR 6G. The subject is assessed using primary standards, official project documentation, vendor technical material where unavoidable, and primary research. Adoption is conditional on reproducing the relevant behavior in the intended environment.

The upgraded position distinguishes architectural usefulness from implementation maturity. A concept may be valuable enough to influence interfaces, data models, or long-term strategy while remaining unsuitable as a mandatory production dependency. Conversely, mature mechanisms with strong authority may be adopted immediately while still requiring environment-specific testing.

3. Current Authority and Freshness

Source	Authority and Status	Freshness Control
3GPP TS 23.501 - System Architecture for the 5G System https://www.3gpp.org/dynareport/23501.htm	Primary telecom specification Versioned and actively maintained Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.
3GPP Release 21 Timeline https://www.3gpp.org/news-events/3gpp-news/rel21-timeline	Primary standards-program update Current planning baseline Published: 2026	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.

Freshness is controlled through both a scheduled review interval and event-driven triggers. A report becomes stale before its scheduled date when a governing standard changes, a cited project introduces a breaking release, a material vulnerability changes the risk posture, a research result is retracted, or internal validation contradicts the published position.

4. Explicit Research Limitations

- This report is a technical research publication, not a certification, legal opinion, procurement approval, or guarantee of production suitability.
- Source authority does not eliminate implementation risk. Product behavior, interoperability, performance, security, and operational fit require reproducible validation in the adopting environment.

- Fast-moving specifications, product documentation, open-source repositories, and research preprints may change before the next scheduled review.
- Quantitative claims from vendors or research papers are not treated as Mythos results unless independently reproduced under documented conditions.
- Adoption decisions are conditional on applicable policy, risk, licensing, export-control, privacy, safety, regulatory, and support requirements.
- Protocol support, silicon behavior, vendor implementation, licensing, topology, and operational maturity can materially differ from the specification or lab model.

5. Adoption Decision and Guardrails

Adopt explicit end-to-end 5G slice assurance and QoS verification where slicing is used. Treat 6G architecture and Release 21 material as evolving research and standards work.

Decision dimension	Required direction
Production authority	No deployment or control authority is granted by this report. Aegis policy, formal change control, and system ownership remain authoritative.
Implementation	Use versioned adapters, explicit interfaces, bounded scope, observability, rollback, and evidence collection.
Validation	Reproduce mandatory claims using representative data, target hardware or software, failure cases, and documented acceptance criteria.
Procurement	Treat vendor claims as evidence inputs, not verified outcomes. Require support, lifecycle, licensing, security, and exit-plan analysis.
Retirement	Define conditions for rollback, replacement, data export, key rotation, state migration, and evidence preservation before adoption.

6. Validation and Reproducibility Plan

Stage	Required evidence	Exit condition
Source validation	Exact versions, publication status, revisions, and source hashes or retrieval records.	No high-impact claim depends on an untraceable or superseded source.
Prototype	Minimal implementation with representative data and instrumented execution.	Core mechanism is demonstrated without relying on presentation-only behavior.
Adversarial and failure testing	Malformed inputs, unavailable dependencies, stale state, rollback, isolation, resource exhaustion, and misuse cases.	Failure is bounded, observable, and recoverable.
Comparative baseline	Current production method or conventional alternative tested under the same workload.	Benefit is measurable against a credible baseline.
Operational trial	Ownership, alerts, support, patching, capacity, evidence, and exit procedures exercised.	The operating model is accepted by accountable owners.
Adoption review	Risk, legal, security, privacy, safety, cost, and architecture decision record.	Named authority approves the bounded use case.

7. Review and Expiration Triggers

- Scheduled review no later than 90 days after 2026-07-22.

- A cited standard, specification, API, product, model, runtime, or reference implementation introduces a material revision.
- A security advisory, safety event, interoperability defect, or legal change alters the risk or applicability.
- Internal benchmark, proof-of-concept, or production evidence contradicts a material claim.
- The adoption posture changes, the technology becomes a critical dependency, or a retirement decision is proposed.

8. Complete Source Research Note

SOURCE-LINEAGE NOTICE

The following material preserves the complete original source note, excluding only the conversational wrapper and outer Markdown fence. Legacy status labels and absolute language are retained for traceability and do not override the candidate status, limitations, or adoption decision in this edition.

Document ID: RES-028 Version: 1.0.0 (Definitive Registry Edition) Status: Published Research Note Classification: Public Organization: Mythos Systems (MYTHOS AI) Owner: Aaron Stovall Effective Date: 2026-07-16 Applies To: Telecom architects, private 5G network engineers, industrial IoT specialists, and AI engineers designing ultra-reliable low-latency communication (URLLC) pipelines for physical AI and sovereign edge compute Companion Standards: MYTHOS-TEL-0001 (5G/6G Core, Open RAN & Slicing), MYTHOS-ROB-0001 (Physical AI & Autonomous Fleets), MYTHOS-IOT-0001 (IoT & Edge Sensors), MYTHOS-SEC-0007 (Microsegmentation), MYTHOS-NET-0004 (Zero-Trust Networking)

The architecture of wireless broadband has failed. Organizations attempt to support divergent workloads—autonomous robotics requiring sub-10ms latency, and massive video streams requiring gigabit bandwidth—over a single, "best-effort" RF network. They attempt to manage traffic using DiffServ/QoS tags, which is a prioritization suggestion, not a physical guarantee. When a massive video download saturates the cell tower, the autonomous robot's control link drops, resulting in a physical crash.

5G Standalone (SA) Network Slicing shatters this best-effort paradigm. A network slice is not a VLAN or a QoS tag; it is a logically isolated, end-to-end network built over a shared physical infrastructure. It spans from the device (UE), through the Radio Access Network (RAN), across the transport fabric, and through dedicated 5G Core (5GC) functions.

This research note defines the architectural dynamics of 5G/6G Network Slicing. It dissects the mechanics of NSSAI (Network Slice Selection Assistance Information), the hard reservation of Physical Resource Blocks (PRBs) at the RAN MAC scheduler, and the strict isolation of User Plane Functions (UPF) to guarantee mathematically bounded SLAs. Understanding these dynamics is a prerequisite for building the sovereign, physical AI, and industrial IoT fabrics mandated by the Mythos Telecom Standards.

To understand slicing, we must move from prioritization tags to dedicated, virtualized network functions bound to specific radio resources.

1.1 The S-NSSAI (The Slice ID)

A 5G device does not just connect to a network; it requests a specific slice.

- The Mechanic: The device sends a registration request containing its requested NSSAI (Network Slice Selection Assistance Information). The 5G Core's Access and Mobility Management Function (AMF) evaluates this request and routes the device to the specific Slice/Session Management Function (SMF) and User Plane Function (UPF) dedicated to that slice.

1.2 End-to-End Isolation (RAN + Transport + Core)

A true network slice is not just a core construct; it spans the entire physical network.

- RAN Slicing: The gNodeB (base station) reserves specific Physical Resource Blocks (PRBs)—the actual time/frequency units of the RF spectrum—exclusively for the slice.
- Transport Slicing: The fronthaul and backhaul network (typically SRv6 or MPLS) provisions dedicated, traffic-engineered paths with guaranteed bandwidth and latency for the slice.
- Core Slicing: The 5GC spins up dedicated UPF (User Plane Function) instances, either in the cloud or at the edge, specifically to process the traffic for that slice, completely isolating the data plane from other slices.

1.3 The 3GPP Slice Types (eMBB, URLLC, mMTC)

5G defines standardized slice types to match the physics of the RF to the application:

- eMBB (Enhanced Mobile Broadband): Optimized for high throughput (e.g., 4K video, massive data dumps). Prioritizes bandwidth over latency.
- URLLC (Ultra-Reliable Low-Latency Communication): Optimized for sub-10ms latency and 99.999% reliability (e.g., physical robotics, autonomous vehicles, remote surgery). Prioritizes latency over bandwidth.
- mMTC (Massive Machine Type Communications): Optimized for millions of low-power, low-data sensors (e.g., smart meters).

While network slicing promises isolation, the underlying RF spectrum is a shared, physical medium. This introduces severe physical and mathematical constraints.

2.1 The "Soft" Slicing Fallacy (Noisy Neighbor)

Many early 5G deployments implement "soft slicing." They share the RAN PRBs among all slices and use QoS prioritization to give URLLC traffic preference.

- The Flaw: If an eMBB slice is saturating the RAN with a massive download, the URLLC traffic gets priority, but it still must wait in the MAC scheduler queue behind the eMBB packets already being transmitted. The latency spike violates the URLLC SLA.
- The Mitigation: True slicing MUST utilize "hard" RAN slicing. The MAC scheduler MUST physically partition the PRBs. For example, 80% of PRBs go to eMBB, 20% are strictly reserved for URLLC. The eMBB slice cannot use the URLLC PRBs, even if they are idle. This guarantees URLLC latency bounds.

2.2 The Inter-Slice Pivot Threat

If a slice is compromised (e.g., a massive IoT sensor botnet is breached), the attacker might attempt to pivot from the mMTC slice into the URLLC slice controlling the factory robotics.

- The Flaw: If all slices share a common UPF or a common GTP-U (GPRS Tunneling Protocol) gateway, a compromise of the shared infrastructure breaks the isolation.
- The Mitigation: Each slice MUST possess dedicated, microsegmentated UPF instances. Inter-slice communication MUST be blocked by default and routed through a highly scrutinized, policy-enforced Application Function (AF) gateway.

2.3 The Spectrum Exhaustion Ceiling

RF spectrum is finite. A cell tower has a hard physical limit (e.g., 100 MHz of bandwidth).

- The Constraint: If you allocate 20 MHz to URLLC, that spectrum is permanently unavailable to eMBB. Over-provisioning hard slices starves the network of general-purpose capacity.

The true power of 5G/6G slicing is realized when hard RAN partitioning is combined with edge compute, creating mathematically deterministic data planes.

3.1 URLLC and Time-Sensitive Networking (TSN)

For industrial robotics (`MYTHOS-ROB-0001`), a 5G slice can be configured as a TSN bridge.

- The Mechanic: The URLLC slice utilizes specific 5G numerologies (e.g., mini-slots) that reduce transmission time intervals (TTI) to less than 1ms. Combined with edge compute (MEC) deployed at the base station, the round-trip latency from sensor to AI agent to actuator is physically bounded to under 10ms.

- The Impact: The factory floor replaces miles of rigid physical Ethernet cabling with wireless 5G URLLC, allowing robots to move freely without losing deterministic control.

3.2 Network Slicing as Zero-Trust Microsegmentation

In a Mythos-compliant architecture, a 5G slice is treated as the ultimate physical zero-trust boundary.

- The Mechanic: A smart camera (eMBB slice) and an industrial valve (URLLC slice) might be in the same physical room, connected to the same gNodeB. However, because their NSSAI differs, they are routed to completely different UPFs, different subnets, and different firewalls. The camera is physically incapable of addressing the valve.

The fundamental shift of network slicing is the viability of Private 5G as a sovereign enterprise fabric.

- The Industrial Metaverse: A manufacturing plant deploys a Private 5G network. They provision a URLLC slice for AGVs (Automated Guided Vehicles), an eMBB slice for AR (Augmented Reality) headsets worn by maintenance workers, and an mMTC slice for environmental sensors. All coexist on one RF spectrum without interfering.
- First Responder Priority: In a public 5G network, a slice is reserved for emergency services. During a disaster when civilian eMBB traffic saturates the tower, the first responder URLLC slice operates flawlessly on its reserved PRBs.
- Bounded SLA AI Inference: An autonomous AI agent requires cloud inference but cannot tolerate internet latency. A dedicated 5G slice connects the agent directly to a localized MEC (Multi-access Edge Computing) datacenter, guaranteeing a 5ms end-to-end RTT.

To deploy 5G slicing in production, the Mythos Architecture enforces strict isolation, GTP-U inspection, and edge compute boundaries.

5.1 Dedicated UPF Microsegmentation

The 5G Core MUST NOT use a monolithic UPF for all slices.

- The Mitigation: The Mythos standard mandates dedicated, containerized UPFs (cUPF) deployed at the edge for each critical slice. The cUPF is treated as a zero-trust workload, integrated with a Service Mesh (Istio/Linkerd) to enforce mTLS and L7 authorization on all downstream API calls.

5.2 GTP-U Firewalling

The GTP-U (GPRS Tunneling Protocol - User Plane) tunnels traffic between the RAN and the Core.

- The Mitigation: The architecture MUST implement a 5G-aware firewall (e.g., Palo Alto Networks or Cisco 5G Secure Gateway) that inspects the GTP-U headers. It must drop any GTP-U packet where the NSSAI (slice ID) does not match the expected source/destination tunnel endpoints, preventing cross-slice pivoting.

5.3 Hard PRB Reservation Verification

The RAN MAC scheduler configuration MUST be verified.

- The Mitigation: The continuous reconciliation plane (NetBox/Ansible) MUST audit the gNodeB configuration. If an engineer accidentally configures soft slicing (shared PRBs) for a URLLC slice, the pipeline must detect the deviation and auto-revert the configuration to hard slicing before a robotic failure occurs.

The strategy of treating wireless networks as best-effort pipes is obsolete for the era of physical AI and industrial automation. 5G/6G Network Slicing, powered by Standalone cores and hard RAN partitioning, transforms the shared RF spectrum into a deterministic, mathematically bounded fabric. By dedicating radio resources, transport paths, and core functions to specific workloads, we guarantee that a massive video stream will never cause a physical robot to crash. In a Mythos-compliant architecture, the slice is not a QoS tag; it is an absolute physical boundary.

```
A QoS tag is a suggestion; a hard PRB reservation is a physical law.
A shared UPF is a pivot vector; a dedicated UPF is a boundary.
```

```
A best-effort network drops packets; a sliced network guarantees delivery.

The spectrum is shared; the slice is absolute.

> RF may be physical.
> Slice isolation must be absolute.
```

End of Research Note RES-028

9. Source Register

Source	Authority and Status	Freshness Control
3GPP TS 23.501 - System Architecture for the 5G System https://www.3gpp.org/dynareport/23501.htm	Primary telecom specification Versioned and actively maintained Published: Living	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.
3GPP Release 21 Timeline https://www.3gpp.org/news-events/3gpp-news/rel21-timeline	Primary standards-program update Current planning baseline Published: 2026	Validated: 2026-07-22 Review on material revision, withdrawal, supersession, or scheduled report review.

10. Lineage, Review, and Publication Record

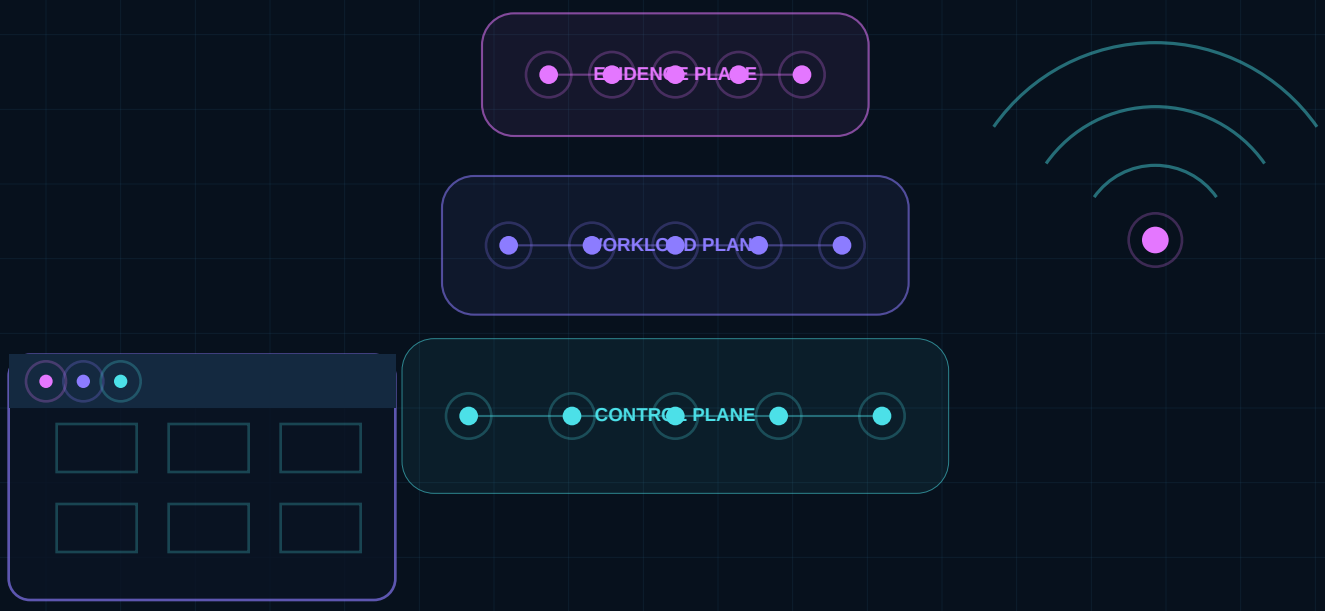
Record	Value
Original source file	RES-028_5G_6G_Network_Slicing_QoS_Enforcement.md
Original source words	1605
Upgrade type	Enterprise research report; source and freshness validation; limitations; adoption decision; reproducibility plan
Generated on	2026-07-22
Current status	Candidate Research Report
Publication authority	Formal Mythos approval not yet recorded
Next review	90 days or event-driven trigger, whichever occurs first

MYTHOS SYSTEMS

REFERENCE ARCHITECTURE COMPENDIUM

Cloud, Edge, and Trusted Compute Reference Architecture Compendium

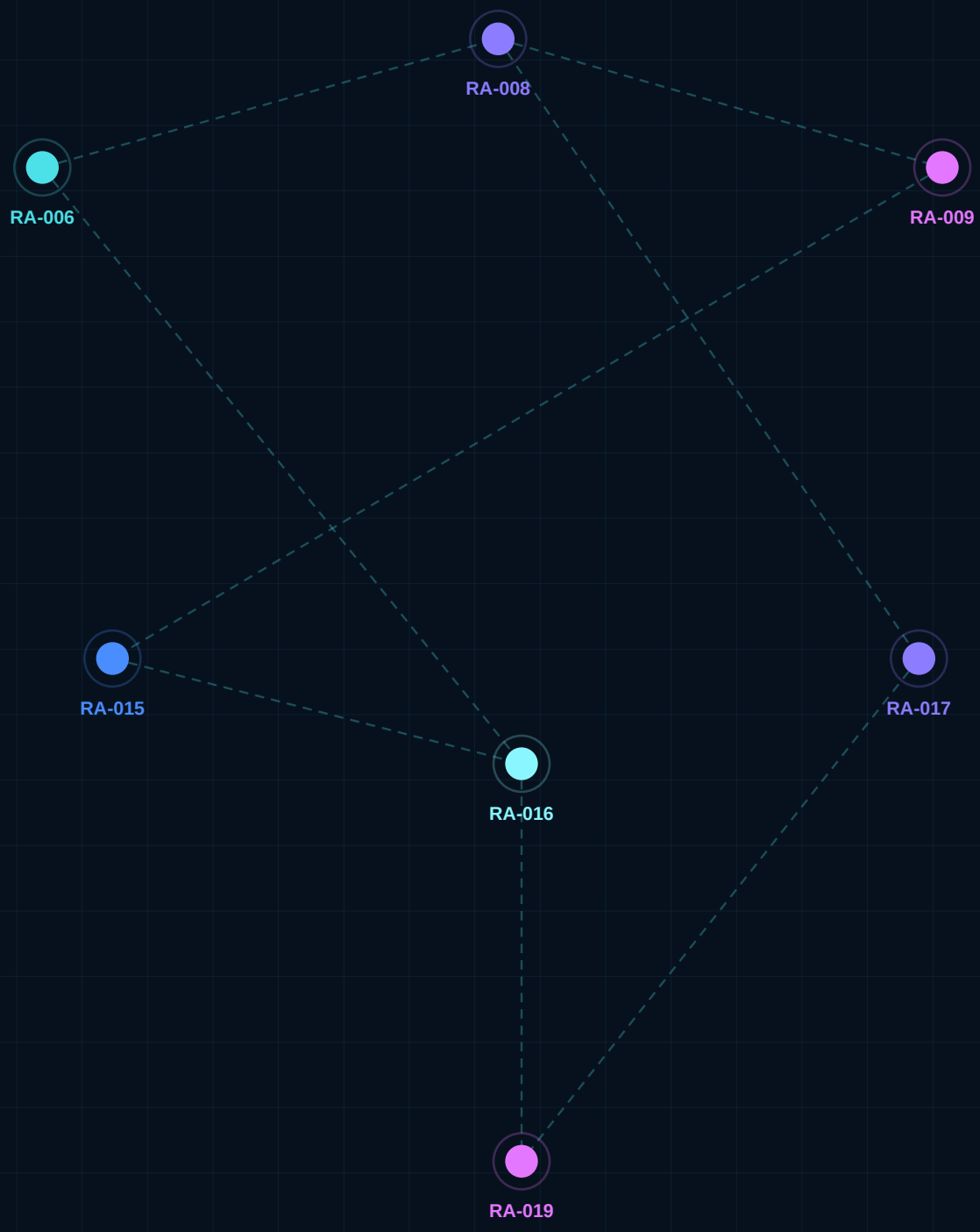
Reference architectures for cloud landing zones, local inference, zero-trust access, OT/IoT convergence, realtime media, confidential compute, and sovereign browser execution.



MYTHOS-RA-COMP-CLOUDEDGE-0001

PERMANENT PUBLICATION IDENTITY / CANDIDATE PUBLICATION / PUBLIC

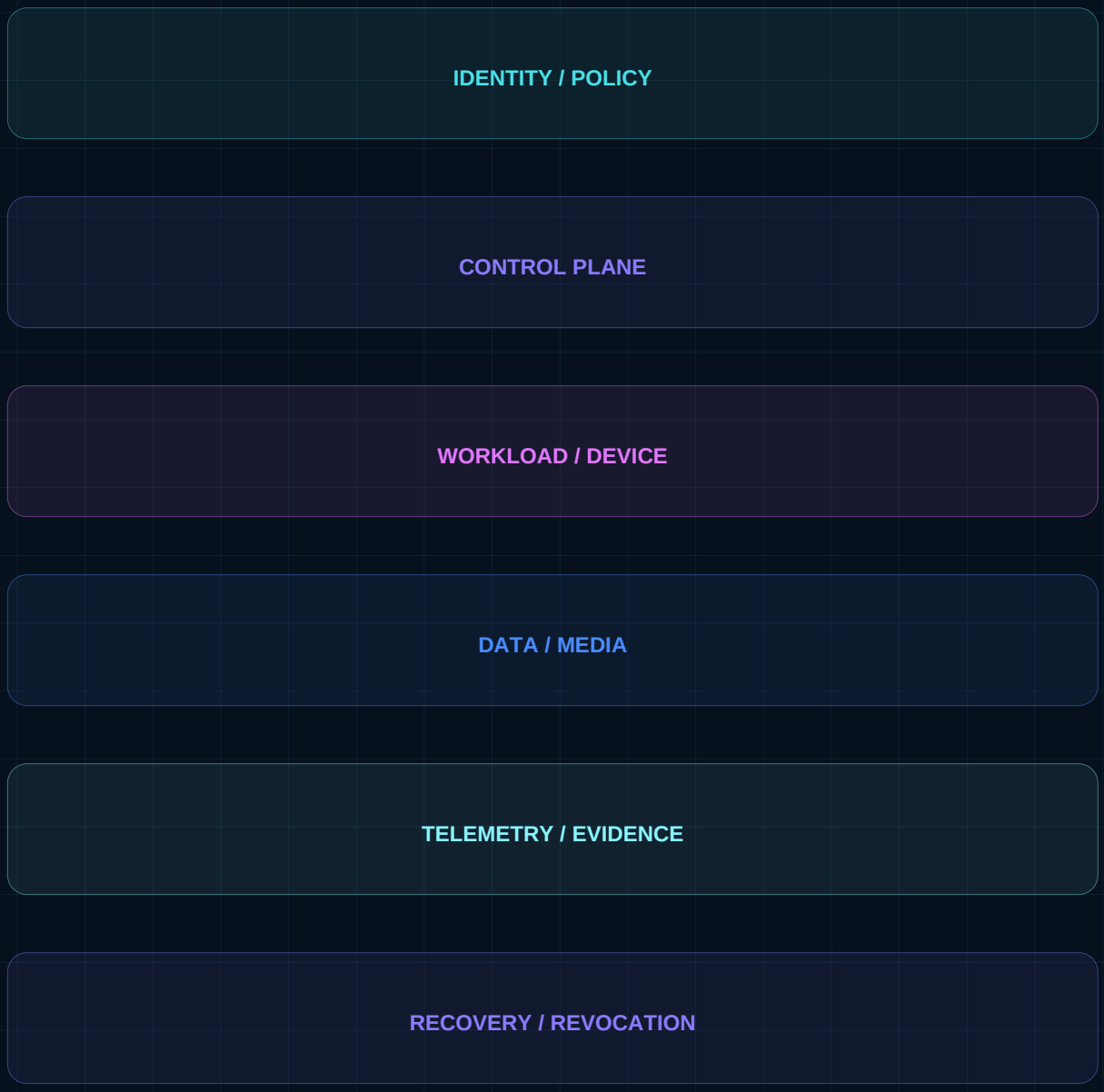
Architecture dependency fabric



Permanent architecture register

RA-006	Local-First AI Inference Cluster and Edge Node
RA-008	Multi-Cloud Landing Zone and Internal Developer Platform
RA-009	Zero-Trust Identity, Policy, Secrets, and Access Fabric
RA-015	Edge, OT, IoT, and Telecommunications Convergence
RA-016	Realtime Voice, Media, and Multimodal Interaction Platform
RA-017	Quantum-Safe, Confidential, and Verifiable Compute
RA-019	Sovereign Browser, Remote Access, and Web Automation

Trust, control, data, and evidence remain separate



Architecture acceptance gates

GATE 01

Boundary model is explicit

GATE 02

Identity and authority are externalized

GATE 03

Failure domains are observable

GATE 04

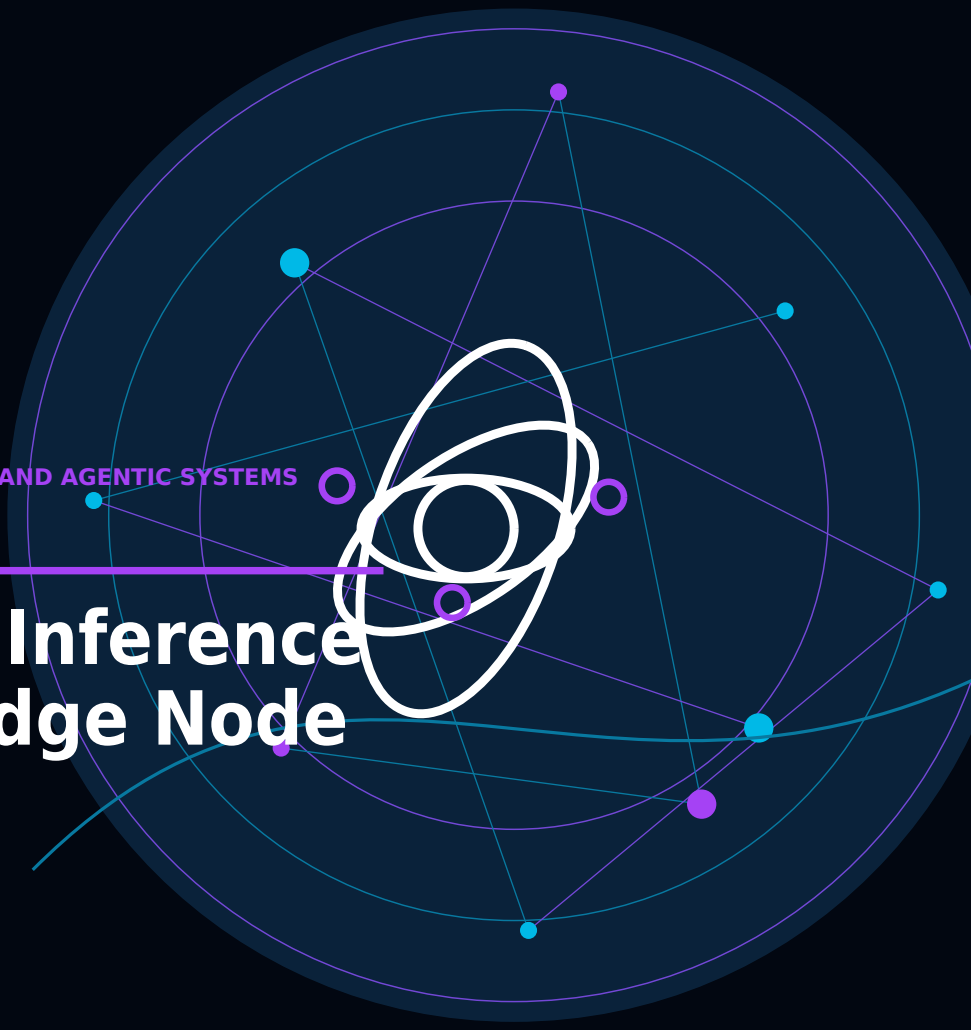
Recovery is demonstrated

GATE 05

Evidence links desired to observed state

GATE 06

Lifecycle and decommission paths exist



ENGINEERING STANDARDS LIBRARY / AI AND AGENTIC SYSTEMS

Local-First AI Inference Cluster and Edge Node

A local-first architecture for model acquisition, inference, routing, isolation, acceleration, retrieval, observability, privacy, and resilient edge operation.

PERMANENT DOCUMENT ID

RA-006

PUBLICATION

Candidate Reference Architecture
Version 1.0.0-candidate

ASSURANCE PROFILE

Candidate Architecture
High Assurance

PERMANENT PUBLICATION IDENTITY

Local-First AI Inference Cluster and Edge Node

A local-first architecture for model acquisition, inference, routing, isolation, acceleration, retrieval, observability, privacy, and resilient edge operation.

DOCUMENT ID
RA-006

VERSION
1.0.0-candidate

STATUS
Candidate Reference Architecture

ARCHITECTURE SET
Canonical Set

DOMAIN
Artificial Intelligence

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

6	6	18	4
ARCHITECTURE LAYERS	CORE COMPONENTS	ACCEPTANCE CRITERIA	DEPENDENCIES

Architecture position. Local-first means local capability is a first-class operating mode; it does not imply that every model, dataset, or workload must run locally or that cloud services are prohibited.

REFERENCE ARCHITECTURE TOPOLOGY

ARTIFICIAL INTELLIGENCE

Local-first means local capability is a first-class operating mode; it does not imply that every model, dataset, or workload must run locally or that cloud services are prohibited.

GOVERNED ARCHITECTURE FLOW

INTERPRETATION AND ASSURANCE

HOW TO APPLY THIS ARCHITECTURE

INFORMATIVE ARCHITECTURE

The architecture describes a deployable pattern and assurance model. Normative obligations remain in mapped standards, policy, law, contracts, and approved design decisions.

EVIDENCE OVER DIAGRAM

A diagram is not implementation evidence. Conformance requires inventories, configurations, tests, telemetry, incidents, recovery results, and accountable decisions.

PROFILE WITHOUT DILUTION

Profiles may change scale and technology, but cannot remove mandatory trust, safety, privacy, recovery, or evidence outcomes.

PRODUCT-NEUTRAL CORE

Named technologies are illustrative unless an organization explicitly adopts them. Interfaces, outcomes, and evidence remain primary.

BOUNDED CLAIMS

Performance, security, autonomy, availability, privacy, and compliance claims apply only to tested configurations and conditions.

CONTROLLED EVOLUTION

Architecture change preserves version, migration, rollback, source currency, exceptions, and residual-risk records.

Adoption requires environment-specific design, threat and hazard analysis, capacity engineering, implementation, verification, approval, operation, and periodic reassessment.

INTERACTIVE NAVIGATION

REFERENCE ARCHITECTURE CONTENTS

PART I - EXECUTIVE DIRECTION

OUTCOMES, RISKS, AND DECISIONS

ARCHITECTURE MANDATE

A local-first architecture for model acquisition, inference, routing, isolation, acceleration, retrieval, observability, privacy, and resilient edge operation.

EXECUTIVE OUTCOMES

- Run approved models near the user or data when privacy, latency, resilience, or cost justify it.
- Route workloads across CPU, GPU, NPU, accelerator, workstation, cluster, and approved cloud resources.
- Protect model, data, prompt, context, cache, and tool boundaries.
- Operate through resource pressure, model failure, offline periods, and node loss.

DECISIONS REQUIRED

- Accountable service and risk ownership.
- Accepted architecture profile and consequence class.
- Authoritative identity, data, policy, configuration, and evidence systems.
- Mandatory approval, recovery, privacy, safety, and supplier-exit gates.
- Funding for operations, observability, validation, and lifecycle—not only implementation.

PART II - SCOPE, PRINCIPLES, AND SYSTEM CONTEXT

ARTIFICIAL INTELLIGENCE

IN SCOPE

A local-first architecture for model acquisition, inference, routing, isolation, acceleration, retrieval, observability, privacy, and resilient edge operation.

OUT OF SCOPE

Vendor selection, exact sizing, contractual obligations, detailed low-level design, product certification, regulatory approval, and proof that any named platform is implemented.

ARCHITECTURE INVARIANTS

Explicit ownership; least authority; separable desired, runtime, observed, and evidence state; bounded automation; tested recovery; product-neutral interfaces; source and claim currency.

ASSUMPTIONS

Organizations tailor the pattern through documented risk, environment, criticality, data, safety, regulatory, workforce, and supplier analysis.

SYSTEM CONTEXT AND RESPONSIBILITY MODEL

ACTORS, ENVIRONMENTS, AND EXTERNAL DEPENDENCIES

ACTORS AND RESPONSIBILITY

- Service consumers and affected users
- Product, platform, network, security, data, reliability, and support teams
- Human administrators, operators, reviewers, incident commanders, and risk owners
- Machine identities, agents, workloads, devices, controllers, and external services
- Suppliers, providers, carriers, integrators, assessors, and regulators where applicable

RESPONSIBILITY BOUNDARY

- The adopting organization retains accountability for purpose, risk, configuration, data, identity, operation, recovery, and evidence even where a provider performs underlying functions.
- Inherited controls are recorded with provider, scope, evidence, limitations, shared responsibility, incident path, and exit conditions.
- No component is considered trusted solely because it is internal, managed, signed, cloud-hosted, model-generated, or supplied by a recognized vendor.

ARCHITECTURE POSITION

Local-first means local capability is a first-class operating mode; it does not imply that every model, dataset, or workload must run locally or that cloud services are prohibited.

PART III - LOGICAL ARCHITECTURE

LAYERS AND RESPONSIBILITIES

01

Model Registry and Supply Chain

Model Registry and Supply Chain owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

02

Scheduling and Model Routing

Scheduling and Model Routing owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

03

Inference Runtime and Isolation

Inference Runtime and Isolation owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

04

Data, Retrieval, and Context

Data, Retrieval, and Context owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

05

Acceleration, Capacity, and Thermal Control

Acceleration, Capacity, and Thermal Control owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

06

Observation, Safety, and Recovery

Observation, Safety, and Recovery owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

CORE COMPONENTS - A

COMPONENTS 01-03

COMPONENT 01

Signed model, tokenizer, adapter, runtime, license, and evaluation registry

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 02

Capability profiles, workload queues, budgets, placement, admission, and fallback policy

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 03

Sandboxed runtimes, local APIs, batching, streaming, quantization, and resource limits

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

CORE COMPONENTS - B

COMPONENTS 04-06

COMPONENT 04

Local retrieval, vector and structured stores, ephemeral context, cache, and privacy policy

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 05

GPU/NPU/CPU topology, memory planning, power, cooling, driver, and health management

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 06

Quality evaluation, guardrails, telemetry, incident handling, node rebuild, and evidence

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

PART IV - DATA, CONTROL, AUTHORITY, AND EVIDENCE FLOW

FLOW CONTRACTS

1. Model artifacts are acquired, scanned, evaluated, signed, and approved.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

2. Requests are classified by data sensitivity, capability, latency, cost, and consequence.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

3. Router selects an eligible profile and records the exact model/runtime configuration.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

4. Inference runs within resource, context, tool, network, and retention boundaries.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

5. Outputs are evaluated, surfaced with provenance and uncertainty, and retained according to policy.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

6. Node or model failure triggers bounded retry, smaller model, alternate node, approved cloud, or safe refusal.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

TRUST BOUNDARIES AND ENFORCEMENT

EVERY CROSSING REQUIRES AN EXPLICIT CONTRACT

TRUST BOUNDARY 01

Untrusted model artifact versus trusted registry

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 02

User data versus shared model service

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 03

Inference runtime versus host operating system

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 04

Model process versus tools and network

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 05

Local cluster versus approved external provider

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 06

Tenant or project context versus shared cache and memory

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

PART V - DEPLOYMENT PROFILES

PROFILE CHANGES SCALE, NOT ASSURANCE OUTCOMES

PROFILE 01 / SINGLE ACCELERATED WORKSTATION

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PROFILE 02 / MULTI-GPU LOCAL INFERENCE SERVER

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PROFILE 03 / SMALL EDGE CLUSTER

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PROFILE 04 / FEDERATED LOCAL-FIRST FLEET WITH APPROVED CLOUD OVERFLOW

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PART VI - OPERATING LIFECYCLE

FROM DISCOVERY THROUGH RETIREMENT

01 / Inventory workloads, data, and hardware

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

02 / Establish signed model registry

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

03 / Deploy isolated runtime and API

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

04 / Add routing and resource governance

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

05 / Integrate retrieval and tools cautiously

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

06 / Exercise failure and offline modes

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

07 / Optimize capacity with measured evidence

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

SECURITY, PRIVACY, SAFETY, AND RIGHTS

CROSS-CUTTING ASSURANCE

SECURITY

Threat model identities, trust boundaries, supply chain, control planes, data, protocols, privilege, abuse, detection, response, recovery, and residual risk.

PRIVACY

Purpose limitation, minimization, transparency, consent where required, access, correction, retention, deletion, sovereignty, metadata, and third-party processing.

SAFETY

Hazards, unsafe states, human intervention, physical consequence, dependency failure, automation limits, emergency stop, recovery sequence, and validation.

RIGHTS AND USABILITY

Accessibility, contestability, explanation, operator workload, affected-user communication, grievance, misuse prevention, and equitable performance.

SUPPLY CHAIN

Provenance, version, SBOM or component inventory, signature, vulnerability, support, update, provider, subcontractor, and exit.

EVIDENCE

Design decisions, tests, telemetry, incidents, approvals, exceptions, recovery exercises, source currency, and review findings remain linked to the architecture version.

FAILURE MODES AND RESILIENCE

DESIGN FOR SAFE DEGRADATION AND TRUSTWORTHY RECOVERY

FAILURE SCENARIO 01

A model file, adapter, tokenizer, or runtime contains malicious or incompatible content.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 02

Memory pressure or thermal throttling destabilizes the host.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 03

Shared cache, retrieval, or batching leaks data across projects or users.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 04

Fallback silently moves sensitive data to an external provider.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 05

Quantization or runtime conversion changes safety or task quality.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 06

Driver, accelerator, model, or orchestration updates cannot be reproduced or rolled back.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

PART VII - INTEGRATION AND DEPENDENCY MODEL

CONNECTED REFERENCE ARCHITECTURES AND STANDARDS

REFERENCE ARCHITECTURE DEPENDENCIES

- RA-008 - Multi-Cloud Landing Zone and Internal Developer Platform
- RA-010 - Local-First Knowledge, Memory, and Evidence Platform
- RA-016 - Realtime Voice, Media, and Multimodal Interaction Platform
- RA-017 - Quantum-Safe, Confidential, and Verifiable Compute

MAPPED MYTHOS STANDARDS

- MYTHOS-AI-0003
- MYTHOS-AI-0005
- MYTHOS-AI-0006
- MYTHOS-DAT-0001
- MYTHOS-HW-0001
- MYTHOS-WEB-0001

DEPENDENCY DOCTRINE

A dependency is not a blanket trust grant. Each connection requires an interface, identity, policy, data, availability, version, failure, recovery, and evidence contract. Correlated failure and circular authority must be analyzed across the full architecture graph.

PART VIII - IMPLEMENTATION AND ADOPTION ROADMAP

PHASED, EVIDENCE-DRIVEN TRANSITION

0-30 DAYS	Inventory purpose, owners, users, systems, data, dependencies, risks, current diagrams, incidents, recovery, and evidence gaps.
31-90 DAYS	Establish authoritative identity, configuration, policy, data, source, and architecture records; define target profile and acceptance tests.
3-6 MONTHS	Implement bounded foundations, isolation, telemetry, change control, backup, recovery, and representative pilot workflows.
6-12 MONTHS	Expand through standard patterns, automation, service ownership, SLOs, independent assessment, failure injection, and supplier-exit exercises.
ONGOING	Reconcile drift, review sources, reassess risk, measure outcomes, exercise recovery, renew exceptions, migrate dependencies, and retire obsolete paths.

ARCHITECTURE ACCEPTANCE CRITERIA - A

CRITERIA 01-09

AC-01	The architecture has a named service owner, technical owner, security owner, data owner, and recovery authority.
AC-02	The system boundary, external dependencies, retained responsibilities, and trust transitions are documented and reviewed.
AC-03	Identity, authorization, secrets, policy, and privileged operations are explicit and independently auditable.
AC-04	Desired state, runtime state, observed state, and evidence state are distinguishable and reconciled.
AC-05	Consequential changes require preconditions, approval policy, bounded execution, verification, and rollback.
AC-06	Telemetry supports service health, security detection, forensic reconstruction, and conformance evidence.
AC-07	Failure modes include safe degradation, dependency loss, corrupted state, operator error, and malicious action.
AC-08	Backup, restoration, reconstitution, and exit procedures are exercised against defined recovery objectives.
AC-09	Data classification, retention, privacy, sovereignty, and disposition controls follow the actual information flow.

ARCHITECTURE ACCEPTANCE CRITERIA - B

CRITERIA 10-18

AC-10	Supplier, model, protocol, platform, and software dependencies have provenance, version, support, and replacement records.
AC-11	Representative performance and capacity tests include saturation, contention, latency, and degraded conditions.
AC-12	Exceptions are time-bounded, visible, approved by accountable risk owners, and linked to compensating controls.
AC-13	Every served profile identifies model, tokenizer, adapter, runtime, quantization, hardware, driver, safety configuration, license, and evaluation state.
AC-14	Routing policy proves that sensitive requests cannot cross an unapproved provider, tenant, cache, or network boundary.
AC-15	Admission control prevents memory, power, thermal, storage, and queue saturation from destabilizing critical workloads.
AC-16	The platform exposes model cold-start, load, queue, token, latency, quality, failure, resource, and fallback telemetry.
AC-17	Offline operation, node loss, accelerator loss, and approved cloud overflow are exercised with representative tasks.
AC-18	Model retirement removes serving authority, cache, indexes, credentials, and retained data without deleting required evidence.

METRICS, EVIDENCE, AND DECISION GATES

MEASURE SERVICE OUTCOMES AND ASSURANCE

COVERAGE

Percent of components, interfaces, identities, data flows, dependencies, risks, and controls represented in current architecture evidence.

CHANGE QUALITY

Plan success, rollback, verification pass, escaped defect, drift, and emergency-change rates.

RELIABILITY

Availability, latency, saturation, dependency health, recovery time, recovery point, and safe-degradation success.

SECURITY

Privilege exposure, policy denial, identity compromise, vulnerable dependency, detection coverage, response, and evidence completeness.

HUMAN OUTCOME

Task success, operator workload, accessibility, override, contestability, training, and user-impact measures.

LIFECYCLE

Version currency, unsupported component, exception age, migration readiness, restore exercise, and retirement completeness.

Release or expansion gate: mandatory acceptance criteria pass; failed safety, privacy, identity, recovery, evidence, or legal gates block promotion regardless of aggregate score.

SOURCE AUTHORITY REGISTER

CURRENT PRIMARY AND OFFICIAL REFERENCES

SOURCE 01 **NIST AI Risk Management Framework 1.0 (revision in progress as of publication date)**

https://www.nist.gov/itl/ai-risk-management-framework

SOURCE 02 **NIST AI 600-1, Generative AI Profile**

https://doi.org/10.6028/NIST.AI.600-1

SOURCE 03 **NIST SP 800-53 Rev. 5**

https://doi.org/10.6028/NIST.SP.800-53r5

SOURCE 04 **SLSA Specification 1.2**

https://slsa.dev/spec/v1.2/

SOURCE 05 **Kubernetes Documentation**

https://kubernetes.io/docs/

SOURCE 06 **OpenTelemetry Specification**

https://opentelemetry.io/docs/specs/

Source currency note. Official frameworks, specifications, and implementation guidance evolve. Assessors must verify the current version, status, errata, applicability, and effective date at adoption and scheduled review. NIST AI RMF 1.0 was under revision at the publication date.

PUBLICATION CONTROL AND REVISION

PERMANENT IDENTITY / CONTROLLED EVOLUTION

PERMANENT DOCUMENT IDENTITY	RA-006
VERSION	1.0.0-candidate
PUBLICATION DATE	2026-07-24
SCHEDULED REVIEW	2027-01-24
STATUS	Candidate Reference Architecture
ARCHITECTURE SET	Canonical Set
CLASSIFICATION	Public
PUBLISHER	Mythos Systems

REVISION RECORD

1.0.0-candidate (2026-07-24) - Permanent-publication candidate edition regenerated under the Mythos Systems architecture publication system. Production sequence metadata is excluded from public identity.

ENGINEERING STANDARDS LIBRARY / PLATFORM AND SOFTWARE ENGINEERING

Multi-Cloud Landing Zone and Internal Developer Platform

A governed multi-cloud foundation and product-oriented developer platform that standardizes identity, policy, networking, data, delivery, observability, cost, and service ownership.

PERMANENT DOCUMENT ID

RA-008

PUBLICATION

Candidate Reference Architecture
Version 1.0.0-candidate

ASSURANCE PROFILE

Candidate Architecture
High Assurance

PERMANENT PUBLICATION IDENTITY

Multi-Cloud Landing Zone and Internal Developer Platform

A governed multi-cloud foundation and product-oriented developer platform that standardizes identity, policy, networking, data, delivery, observability, cost, and service ownership.

DOCUMENT ID
RA-008

VERSION
1.0.0-candidate

STATUS
Candidate Reference Architecture

ARCHITECTURE SET
Future Domain Set

DOMAIN
Platform Engineering

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

6	7	18	4
ARCHITECTURE LAYERS	CORE COMPONENTS	ACCEPTANCE CRITERIA	DEPENDENCIES

Architecture position. This future-domain candidate defines a stable architectural destination and assurance model. Product choices, scale, regulatory obligations, and implementation details remain environment-specific.

REFERENCE ARCHITECTURE TOPOLOGY

PLATFORM ENGINEERING

This future-domain candidate defines a stable architectural destination and assurance model. Product choices, scale, regulatory obligations, and implementation details remain environment-specific.

GOVERNED ARCHITECTURE FLOW

INTERPRETATION AND ASSURANCE

HOW TO APPLY THIS ARCHITECTURE

INFORMATIVE ARCHITECTURE

The architecture describes a deployable pattern and assurance model. Normative obligations remain in mapped standards, policy, law, contracts, and approved design decisions.

EVIDENCE OVER DIAGRAM

A diagram is not implementation evidence. Conformance requires inventories, configurations, tests, telemetry, incidents, recovery results, and accountable decisions.

PROFILE WITHOUT DILUTION

Profiles may change scale and technology, but cannot remove mandatory trust, safety, privacy, recovery, or evidence outcomes.

PRODUCT-NEUTRAL CORE

Named technologies are illustrative unless an organization explicitly adopts them. Interfaces, outcomes, and evidence remain primary.

BOUNDED CLAIMS

Performance, security, autonomy, availability, privacy, and compliance claims apply only to tested configurations and conditions.

CONTROLLED EVOLUTION

Architecture change preserves version, migration, rollback, source currency, exceptions, and residual-risk records.

Adoption requires environment-specific design, threat and hazard analysis, capacity engineering, implementation, verification, approval, operation, and periodic reassessment.

INTERACTIVE NAVIGATION

REFERENCE ARCHITECTURE CONTENTS

PART I - EXECUTIVE DIRECTION

OUTCOMES, RISKS, AND DECISIONS

ARCHITECTURE MANDATE

A governed multi-cloud foundation and product-oriented developer platform that standardizes identity, policy, networking, data, delivery, observability, cost, and service ownership.

EXECUTIVE OUTCOMES

- Establish a deployable, product-neutral target architecture for multi-cloud landing zone and internal developer platform.
- Make ownership, trust boundaries, data and control flows, failure handling, and evidence explicit.
- Support multiple implementation profiles without weakening mandatory assurance outcomes.
- Provide a governed adoption path from discovery through production operation and retirement.

DECISIONS REQUIRED

- Accountable service and risk ownership.
- Accepted architecture profile and consequence class.
- Authoritative identity, data, policy, configuration, and evidence systems.
- Mandatory approval, recovery, privacy, safety, and supplier-exit gates.
- Funding for operations, observability, validation, and lifecycle—not only implementation.

PART II - SCOPE, PRINCIPLES, AND SYSTEM CONTEXT

PLATFORM ENGINEERING

IN SCOPE

A governed multi-cloud foundation and product-oriented developer platform that standardizes identity, policy, networking, data, delivery, observability, cost, and service ownership.

OUT OF SCOPE

Vendor selection, exact sizing, contractual obligations, detailed low-level design, product certification, regulatory approval, and proof that any named platform is implemented.

ARCHITECTURE INVARIANTS

Explicit ownership; least authority; separable desired, runtime, observed, and evidence state; bounded automation; tested recovery; product-neutral interfaces; source and claim currency.

ASSUMPTIONS

Organizations tailor the pattern through documented risk, environment, criticality, data, safety, regulatory, workforce, and supplier analysis.

SYSTEM CONTEXT AND RESPONSIBILITY MODEL

ACTORS, ENVIRONMENTS, AND EXTERNAL DEPENDENCIES

ACTORS AND RESPONSIBILITY

- Service consumers and affected users
- Product, platform, network, security, data, reliability, and support teams
- Human administrators, operators, reviewers, incident commanders, and risk owners
- Machine identities, agents, workloads, devices, controllers, and external services
- Suppliers, providers, carriers, integrators, assessors, and regulators where applicable

RESPONSIBILITY BOUNDARY

- The adopting organization retains accountability for purpose, risk, configuration, data, identity, operation, recovery, and evidence even where a provider performs underlying functions.
- Inherited controls are recorded with provider, scope, evidence, limitations, shared responsibility, incident path, and exit conditions.
- No component is considered trusted solely because it is internal, managed, signed, cloud-hosted, model-generated, or supplied by a recognized vendor.

ARCHITECTURE POSITION

This future-domain candidate defines a stable architectural destination and assurance model. Product choices, scale, regulatory obligations, and implementation details remain environment-specific.

PART III - LOGICAL ARCHITECTURE

LAYERS AND RESPONSIBILITIES

01

Organization and tenancy

Organization and tenancy owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

02

Landing zones and shared services

Landing zones and shared services owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

03

Developer portal and service catalog

Developer portal and service catalog owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

04

Golden paths and templates

Golden paths and templates owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

05

Workload identity and policy

Workload identity and policy owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

06

Delivery, observability, FinOps, and exit

Delivery, observability, FinOps, and exit owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

CORE COMPONENTS - A

COMPONENTS 01-04

COMPONENT 01

Account and environment hierarchy

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 02

Federated identity and privileged administration

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 03

Network and egress control

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 04

Policy-as-code guardrails

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

CORE COMPONENTS - B

COMPONENTS 05-07

COMPONENT 05

Developer portal, catalog, scorecards, and ownership

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 06

Templates, pipelines, artifact registry, and deployment control

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 07

Telemetry, SLO, cost, quota, recovery, and provider exit

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

PART IV - DATA, CONTROL, AUTHORITY, AND EVIDENCE FLOW

FLOW CONTRACTS

1. Intent, service demand, policy, and environment constraints enter the architecture through explicit contracts.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

2. Authoritative identity, data, configuration, and dependency state are validated before execution or access.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

3. Control planes perform bounded orchestration while local enforcement preserves least privilege and safe behavior.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

4. Telemetry, tests, user outcomes, and incidents update evidence and trigger correction or rollback.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

5. Lifecycle governance manages versioning, capacity, exceptions, migration, and retirement.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

TRUST BOUNDARIES AND ENFORCEMENT

EVERY CROSSING REQUIRES AN EXPLICIT CONTRACT

TRUST BOUNDARY 01

Human or machine actor versus authorization service

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 02

Control plane versus workload, device, or enforcement point

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 03

Tenant or project versus shared platform

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 04

Local environment versus external provider or network

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 05

Mutable runtime state versus authoritative evidence

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 06

Normal operation versus emergency or break-glass authority

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

PART V - DEPLOYMENT PROFILES

PROFILE CHANGES SCALE, NOT ASSURANCE OUTCOMES

PROFILE 01 / PILOT OR LABORATORY PROFILE

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PROFILE 02 / DEPARTMENT OR BOUNDED PRODUCTION PROFILE

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PROFILE 03 / ENTERPRISE FEDERATED PROFILE

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PROFILE 04 / HIGH-ASSURANCE SOVEREIGN OR REGULATED PROFILE

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PART VI - OPERATING LIFECYCLE

FROM DISCOVERY THROUGH RETIREMENT

01 / Discover and classify demand

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

02 / Define service boundary and owners

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

03 / Model architecture and acceptance criteria

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

04 / Build isolated proof and challenge assumptions

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

05 / Pilot with stop conditions and rollback

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

06 / Scale through standard profiles and automation

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

07 / Operate, reassess, migrate, and retire

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

SECURITY, PRIVACY, SAFETY, AND RIGHTS

CROSS-CUTTING ASSURANCE

SECURITY

Threat model identities, trust boundaries, supply chain, control planes, data, protocols, privilege, abuse, detection, response, recovery, and residual risk.

PRIVACY

Purpose limitation, minimization, transparency, consent where required, access, correction, retention, deletion, sovereignty, metadata, and third-party processing.

SAFETY

Hazards, unsafe states, human intervention, physical consequence, dependency failure, automation limits, emergency stop, recovery sequence, and validation.

RIGHTS AND USABILITY

Accessibility, contestability, explanation, operator workload, affected-user communication, grievance, misuse prevention, and equitable performance.

SUPPLY CHAIN

Provenance, version, SBOM or component inventory, signature, vulnerability, support, update, provider, subcontractor, and exit.

EVIDENCE

Design decisions, tests, telemetry, incidents, approvals, exceptions, recovery exercises, source currency, and review findings remain linked to the architecture version.

FAILURE MODES AND RESILIENCE

DESIGN FOR SAFE DEGRADATION AND TRUSTWORTHY RECOVERY

FAILURE SCENARIO 01

Control-plane compromise or excessive privilege.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 02

Identity, policy, data, or configuration state becomes stale or inconsistent.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 03

A shared dependency creates correlated failure across tenants or domains.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 04

Automation succeeds technically but violates intended service, safety, or privacy outcomes.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 05

Recovery restores components without restoring trustworthy service state.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 06

Exit, migration, revocation, or deletion cannot be completed within the required time.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

PART VII - INTEGRATION AND DEPENDENCY MODEL

CONNECTED REFERENCE ARCHITECTURES AND STANDARDS

REFERENCE ARCHITECTURE DEPENDENCIES

- RA-009 - Zero-Trust Identity, Policy, Secrets, and Access Fabric
- RA-012 - Enterprise Observability, SRE, and Incident Command Platform
- RA-013 - Secure Software Supply Chain and Release Factory
- RA-014 - Enterprise Data, API, Event, and Integration Fabric

MAPPED MYTHOS STANDARDS

- MYTHOS-CLD-0001
- MYTHOS-CLD-0007
- MYTHOS-PLT-0002
- MYTHOS-SRE-0001

DEPENDENCY DOCTRINE

A dependency is not a blanket trust grant. Each connection requires an interface, identity, policy, data, availability, version, failure, recovery, and evidence contract. Correlated failure and circular authority must be analyzed across the full architecture graph.

PART VIII - IMPLEMENTATION AND ADOPTION ROADMAP

PHASED, EVIDENCE-DRIVEN TRANSITION

0-30 DAYS	Inventory purpose, owners, users, systems, data, dependencies, risks, current diagrams, incidents, recovery, and evidence gaps.
31-90 DAYS	Establish authoritative identity, configuration, policy, data, source, and architecture records; define target profile and acceptance tests.
3-6 MONTHS	Implement bounded foundations, isolation, telemetry, change control, backup, recovery, and representative pilot workflows.
6-12 MONTHS	Expand through standard patterns, automation, service ownership, SLOs, independent assessment, failure injection, and supplier-exit exercises.
ONGOING	Reconcile drift, review sources, reassess risk, measure outcomes, exercise recovery, renew exceptions, migrate dependencies, and retire obsolete paths.

ARCHITECTURE ACCEPTANCE CRITERIA - A

CRITERIA 01-09

AC-01	The architecture has a named service owner, technical owner, security owner, data owner, and recovery authority.
AC-02	The system boundary, external dependencies, retained responsibilities, and trust transitions are documented and reviewed.
AC-03	Identity, authorization, secrets, policy, and privileged operations are explicit and independently auditable.
AC-04	Desired state, runtime state, observed state, and evidence state are distinguishable and reconciled.
AC-05	Consequential changes require preconditions, approval policy, bounded execution, verification, and rollback.
AC-06	Telemetry supports service health, security detection, forensic reconstruction, and conformance evidence.
AC-07	Failure modes include safe degradation, dependency loss, corrupted state, operator error, and malicious action.
AC-08	Backup, restoration, reconstitution, and exit procedures are exercised against defined recovery objectives.
AC-09	Data classification, retention, privacy, sovereignty, and disposition controls follow the actual information flow.

ARCHITECTURE ACCEPTANCE CRITERIA - B

CRITERIA 10-18

AC-10	Supplier, model, protocol, platform, and software dependencies have provenance, version, support, and replacement records.
AC-11	Representative performance and capacity tests include saturation, contention, latency, and degraded conditions.
AC-12	Exceptions are time-bounded, visible, approved by accountable risk owners, and linked to compensating controls.
AC-13	Architecture diagrams and inventories remain synchronized with deployed components and interfaces.
AC-14	Mandatory trust, safety, privacy, recovery, and evidence gates cannot be bypassed by a lower assurance profile.
AC-15	External dependencies have contractual, technical, operational, data, and exit boundaries.
AC-16	Representative acceptance tests exercise normal, degraded, adversarial, recovery, and retirement scenarios.
AC-17	The architecture can explain why an action, decision, deployment, or access was permitted and what evidence supports it.
AC-18	A formal decision record names selected and rejected alternatives, residual risk, review date, and change triggers.

METRICS, EVIDENCE, AND DECISION GATES

MEASURE SERVICE OUTCOMES AND ASSURANCE

COVERAGE

Percent of components, interfaces, identities, data flows, dependencies, risks, and controls represented in current architecture evidence.

CHANGE QUALITY

Plan success, rollback, verification pass, escaped defect, drift, and emergency-change rates.

RELIABILITY

Availability, latency, saturation, dependency health, recovery time, recovery point, and safe-degradation success.

SECURITY

Privilege exposure, policy denial, identity compromise, vulnerable dependency, detection coverage, response, and evidence completeness.

HUMAN OUTCOME

Task success, operator workload, accessibility, override, contestability, training, and user-impact measures.

LIFECYCLE

Version currency, unsupported component, exception age, migration readiness, restore exercise, and retirement completeness.

Release or expansion gate: mandatory acceptance criteria pass; failed safety, privacy, identity, recovery, evidence, or legal gates block promotion regardless of aggregate score.

SOURCE AUTHORITY REGISTER

CURRENT PRIMARY AND OFFICIAL REFERENCES

SOURCE 01 **NIST SP 800-207, Zero Trust Architecture**

https://csrc.nist.gov/pubs/sp/800/207/final

SOURCE 02 **NIST SP 800-218, Secure Software Development Framework**

https://csrc.nist.gov/pubs/sp/800/218/final

SOURCE 03 **SLSA Specification 1.2**

https://slsa.dev/spec/v1.2/

SOURCE 04 **Kubernetes Documentation**

https://kubernetes.io/docs/

SOURCE 05 **OpenTelemetry Specification**

https://opentelemetry.io/docs/specs/

Source currency note. Official frameworks, specifications, and implementation guidance evolve. Assessors must verify the current version, status, errata, applicability, and effective date at adoption and scheduled review. NIST AI RMF 1.0 was under revision at the publication date.

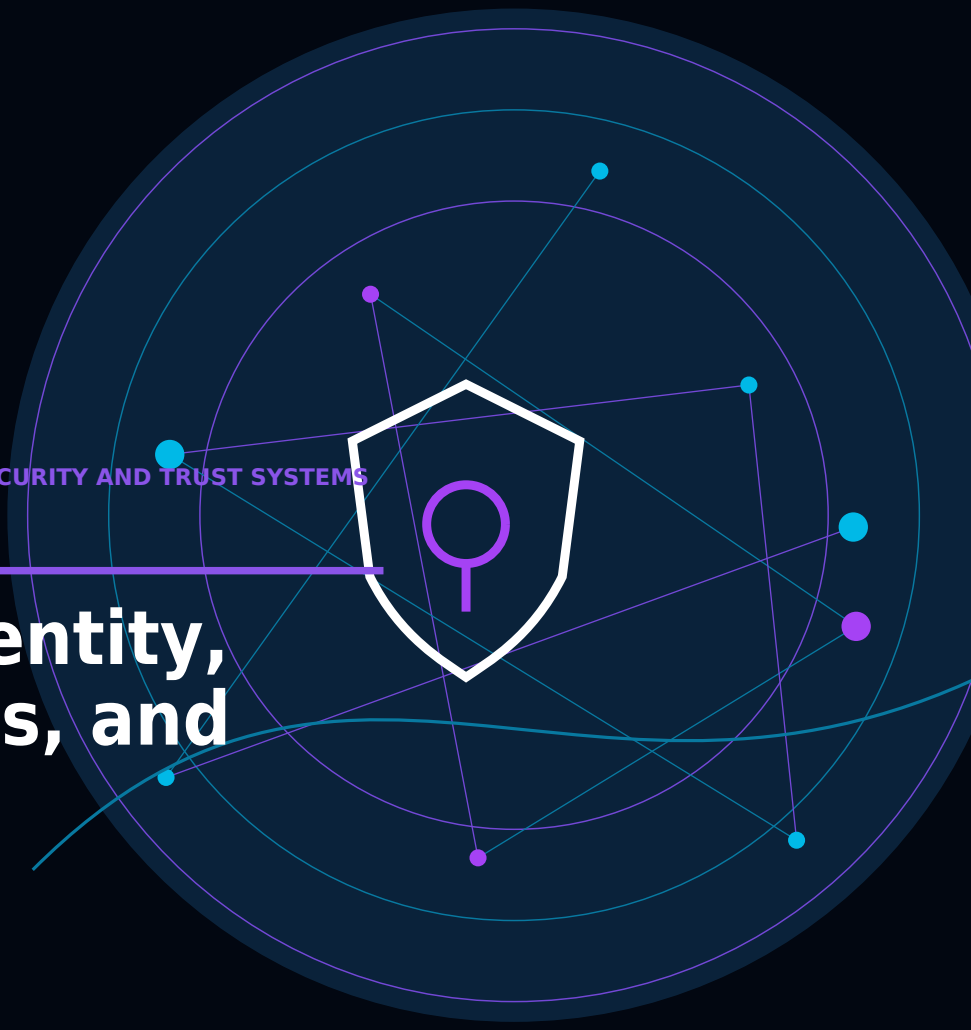
PUBLICATION CONTROL AND REVISION

PERMANENT IDENTITY / CONTROLLED EVOLUTION

PERMANENT DOCUMENT IDENTITY	RA-008
VERSION	1.0.0-candidate
PUBLICATION DATE	2026-07-24
SCHEDULED REVIEW	2027-01-24
STATUS	Candidate Reference Architecture
ARCHITECTURE SET	Future Domain Set
CLASSIFICATION	Public
PUBLISHER	Mythos Systems

REVISION RECORD

1.0.0-candidate (2026-07-24) - Permanent-publication candidate edition regenerated under the Mythos Systems architecture publication system. Production sequence metadata is excluded from public identity.



ENGINEERING STANDARDS LIBRARY / SECURITY AND TRUST SYSTEMS

Zero-Trust Identity, Policy, Secrets, and Access Fabric

A unified identity and authorization architecture for humans, workloads, agents, devices, services, privileged sessions, policy, credentials, secrets, and continuous trust decisions.

PERMANENT DOCUMENT ID

RA-009

PUBLICATION

Candidate Reference Architecture
Version 1.0.0-candidate

ASSURANCE PROFILE

Candidate Architecture
High Assurance

PERMANENT PUBLICATION IDENTITY

Zero-Trust Identity, Policy, Secrets, and Access Fabric

A unified identity and authorization architecture for humans, workloads, agents, devices, services, privileged sessions, policy, credentials, secrets, and continuous trust decisions.

DOCUMENT ID
RA-009

VERSION
1.0.0-candidate

STATUS
Candidate Reference Architecture

ARCHITECTURE SET
Future Domain Set

DOMAIN
Cybersecurity and Network Security

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

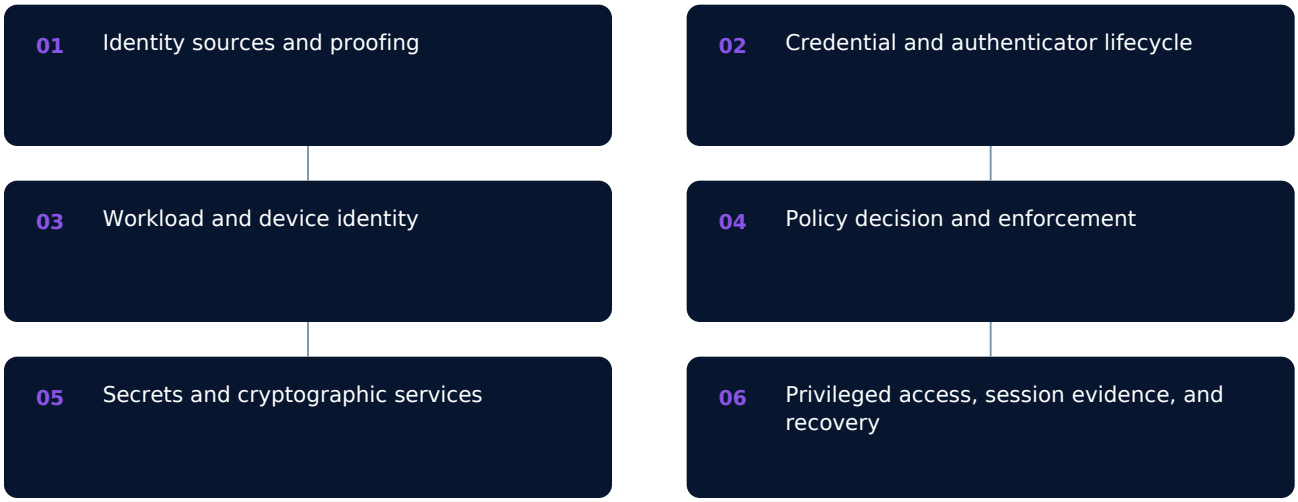
6	6	18	1
ARCHITECTURE LAYERS	CORE COMPONENTS	ACCEPTANCE CRITERIA	DEPENDENCIES

Architecture position. This future-domain candidate defines a stable architectural destination and assurance model. Product choices, scale, regulatory obligations, and implementation details remain environment-specific.

REFERENCE ARCHITECTURE TOPOLOGY

CYBERSECURITY AND NETWORK SECURITY

This future-domain candidate defines a stable architectural destination and assurance model. Product choices, scale, regulatory obligations, and implementation details remain environment-specific.



GOVERNED ARCHITECTURE FLOW

INTERPRETATION AND ASSURANCE

HOW TO APPLY THIS ARCHITECTURE

INFORMATIVE ARCHITECTURE

The architecture describes a deployable pattern and assurance model. Normative obligations remain in mapped standards, policy, law, contracts, and approved design decisions.

EVIDENCE OVER DIAGRAM

A diagram is not implementation evidence. Conformance requires inventories, configurations, tests, telemetry, incidents, recovery results, and accountable decisions.

PROFILE WITHOUT DILUTION

Profiles may change scale and technology, but cannot remove mandatory trust, safety, privacy, recovery, or evidence outcomes.

PRODUCT-NEUTRAL CORE

Named technologies are illustrative unless an organization explicitly adopts them. Interfaces, outcomes, and evidence remain primary.

BOUNDED CLAIMS

Performance, security, autonomy, availability, privacy, and compliance claims apply only to tested configurations and conditions.

CONTROLLED EVOLUTION

Architecture change preserves version, migration, rollback, source currency, exceptions, and residual-risk records.

Adoption requires environment-specific design, threat and hazard analysis, capacity engineering, implementation, verification, approval, operation, and periodic reassessment.

INTERACTIVE NAVIGATION

REFERENCE ARCHITECTURE CONTENTS

PART I - EXECUTIVE DIRECTION

OUTCOMES, RISKS, AND DECISIONS

ARCHITECTURE MANDATE

A unified identity and authorization architecture for humans, workloads, agents, devices, services, privileged sessions, policy, credentials, secrets, and continuous trust decisions.

EXECUTIVE OUTCOMES

- Establish a deployable, product-neutral target architecture for zero-trust identity, policy, secrets, and access fabric.
- Make ownership, trust boundaries, data and control flows, failure handling, and evidence explicit.
- Support multiple implementation profiles without weakening mandatory assurance outcomes.
- Provide a governed adoption path from discovery through production operation and retirement.

DECISIONS REQUIRED

- Accountable service and risk ownership.
- Accepted architecture profile and consequence class.
- Authoritative identity, data, policy, configuration, and evidence systems.
- Mandatory approval, recovery, privacy, safety, and supplier-exit gates.
- Funding for operations, observability, validation, and lifecycle—not only implementation.

PART II - SCOPE, PRINCIPLES, AND SYSTEM CONTEXT

CYBERSECURITY AND NETWORK SECURITY

IN SCOPE

A unified identity and authorization architecture for humans, workloads, agents, devices, services, privileged sessions, policy, credentials, secrets, and continuous trust decisions.

OUT OF SCOPE

Vendor selection, exact sizing, contractual obligations, detailed low-level design, product certification, regulatory approval, and proof that any named platform is implemented.

ARCHITECTURE INVARIANTS

Explicit ownership; least authority; separable desired, runtime, observed, and evidence state; bounded automation; tested recovery; product-neutral interfaces; source and claim currency.

ASSUMPTIONS

Organizations tailor the pattern through documented risk, environment, criticality, data, safety, regulatory, workforce, and supplier analysis.

SYSTEM CONTEXT AND RESPONSIBILITY MODEL

ACTORS, ENVIRONMENTS, AND EXTERNAL DEPENDENCIES

ACTORS AND RESPONSIBILITY

- Service consumers and affected users
- Product, platform, network, security, data, reliability, and support teams
- Human administrators, operators, reviewers, incident commanders, and risk owners
- Machine identities, agents, workloads, devices, controllers, and external services
- Suppliers, providers, carriers, integrators, assessors, and regulators where applicable

RESPONSIBILITY BOUNDARY

- The adopting organization retains accountability for purpose, risk, configuration, data, identity, operation, recovery, and evidence even where a provider performs underlying functions.
- Inherited controls are recorded with provider, scope, evidence, limitations, shared responsibility, incident path, and exit conditions.
- No component is considered trusted solely because it is internal, managed, signed, cloud-hosted, model-generated, or supplied by a recognized vendor.

ARCHITECTURE POSITION

This future-domain candidate defines a stable architectural destination and assurance model. Product choices, scale, regulatory obligations, and implementation details remain environment-specific.

PART III - LOGICAL ARCHITECTURE

LAYERS AND RESPONSIBILITIES

01

Identity sources and proofing

Identity sources and proofing owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

02

Credential and authenticator lifecycle

Credential and authenticator lifecycle owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

03

Workload and device identity

Workload and device identity owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

04

Policy decision and enforcement

Policy decision and enforcement owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

05

Secrets and cryptographic services

Secrets and cryptographic services owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

06

Privileged access, session evidence, and recovery

Privileged access, session evidence, and recovery owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

CORE COMPONENTS - A

COMPONENTS 01-03

COMPONENT 01

Human identity, federation, passkeys, and lifecycle

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 02

Workload identity, certificates, SPIFFE-like service identity, and attestation

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 03

Policy decision points and distributed enforcement

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

CORE COMPONENTS - B

COMPONENTS 04-06

COMPONENT 04

Secrets vault, KMS/HSM, rotation, and short-lived credentials

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 05

Privileged access management, approval, recording, and just-in-time elevation

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 06

Risk signals, posture, anomaly, revocation, break-glass, and evidence

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

PART IV - DATA, CONTROL, AUTHORITY, AND EVIDENCE FLOW

FLOW CONTRACTS

1. Intent, service demand, policy, and environment constraints enter the architecture through explicit contracts.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

2. Authoritative identity, data, configuration, and dependency state are validated before execution or access.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

3. Control planes perform bounded orchestration while local enforcement preserves least privilege and safe behavior.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

4. Telemetry, tests, user outcomes, and incidents update evidence and trigger correction or rollback.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

5. Lifecycle governance manages versioning, capacity, exceptions, migration, and retirement.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

TRUST BOUNDARIES AND ENFORCEMENT

EVERY CROSSING REQUIRES AN EXPLICIT CONTRACT

TRUST BOUNDARY 01

Human or machine actor versus authorization service

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 02

Control plane versus workload, device, or enforcement point

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 03

Tenant or project versus shared platform

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 04

Local environment versus external provider or network

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 05

Mutable runtime state versus authoritative evidence

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 06

Normal operation versus emergency or break-glass authority

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

PART V - DEPLOYMENT PROFILES

PROFILE CHANGES SCALE, NOT ASSURANCE OUTCOMES

PROFILE 01 / PILOT OR LABORATORY PROFILE

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PROFILE 02 / DEPARTMENT OR BOUNDED PRODUCTION PROFILE

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PROFILE 03 / ENTERPRISE FEDERATED PROFILE

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PROFILE 04 / HIGH-ASSURANCE SOVEREIGN OR REGULATED PROFILE

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PART VI - OPERATING LIFECYCLE

FROM DISCOVERY THROUGH RETIREMENT

01 / Discover and classify demand

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

02 / Define service boundary and owners

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

03 / Model architecture and acceptance criteria

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

04 / Build isolated proof and challenge assumptions

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

05 / Pilot with stop conditions and rollback

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

06 / Scale through standard profiles and automation

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

07 / Operate, reassess, migrate, and retire

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

SECURITY, PRIVACY, SAFETY, AND RIGHTS

CROSS-CUTTING ASSURANCE

SECURITY

Threat model identities, trust boundaries, supply chain, control planes, data, protocols, privilege, abuse, detection, response, recovery, and residual risk.

PRIVACY

Purpose limitation, minimization, transparency, consent where required, access, correction, retention, deletion, sovereignty, metadata, and third-party processing.

SAFETY

Hazards, unsafe states, human intervention, physical consequence, dependency failure, automation limits, emergency stop, recovery sequence, and validation.

RIGHTS AND USABILITY

Accessibility, contestability, explanation, operator workload, affected-user communication, grievance, misuse prevention, and equitable performance.

SUPPLY CHAIN

Provenance, version, SBOM or component inventory, signature, vulnerability, support, update, provider, subcontractor, and exit.

EVIDENCE

Design decisions, tests, telemetry, incidents, approvals, exceptions, recovery exercises, source currency, and review findings remain linked to the architecture version.

FAILURE MODES AND RESILIENCE

DESIGN FOR SAFE DEGRADATION AND TRUSTWORTHY RECOVERY

FAILURE SCENARIO 01

Control-plane compromise or excessive privilege.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 02

Identity, policy, data, or configuration state becomes stale or inconsistent.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 03

A shared dependency creates correlated failure across tenants or domains.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 04

Automation succeeds technically but violates intended service, safety, or privacy outcomes.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 05

Recovery restores components without restoring trustworthy service state.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 06

Exit, migration, revocation, or deletion cannot be completed within the required time.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

PART VII - INTEGRATION AND DEPENDENCY MODEL

CONNECTED REFERENCE ARCHITECTURES AND STANDARDS

REFERENCE ARCHITECTURE DEPENDENCIES

- RA-019 - Sovereign Browser, Remote Access, and Web Automation

MAPPED MYTHOS STANDARDS

- MYTHOS-ID-0001
- MYTHOS-ID-0002
- MYTHOS-ID-0003
- MYTHOS-SEC-0003

DEPENDENCY DOCTRINE

A dependency is not a blanket trust grant. Each connection requires an interface, identity, policy, data, availability, version, failure, recovery, and evidence contract. Correlated failure and circular authority must be analyzed across the full architecture graph.

PART VIII - IMPLEMENTATION AND ADOPTION ROADMAP

PHASED, EVIDENCE-DRIVEN TRANSITION

0-30 DAYS	Inventory purpose, owners, users, systems, data, dependencies, risks, current diagrams, incidents, recovery, and evidence gaps.
31-90 DAYS	Establish authoritative identity, configuration, policy, data, source, and architecture records; define target profile and acceptance tests.
3-6 MONTHS	Implement bounded foundations, isolation, telemetry, change control, backup, recovery, and representative pilot workflows.
6-12 MONTHS	Expand through standard patterns, automation, service ownership, SLOs, independent assessment, failure injection, and supplier-exit exercises.
ONGOING	Reconcile drift, review sources, reassess risk, measure outcomes, exercise recovery, renew exceptions, migrate dependencies, and retire obsolete paths.

ARCHITECTURE ACCEPTANCE CRITERIA - A

CRITERIA 01-09

AC-01	The architecture has a named service owner, technical owner, security owner, data owner, and recovery authority.
AC-02	The system boundary, external dependencies, retained responsibilities, and trust transitions are documented and reviewed.
AC-03	Identity, authorization, secrets, policy, and privileged operations are explicit and independently auditable.
AC-04	Desired state, runtime state, observed state, and evidence state are distinguishable and reconciled.
AC-05	Consequential changes require preconditions, approval policy, bounded execution, verification, and rollback.
AC-06	Telemetry supports service health, security detection, forensic reconstruction, and conformance evidence.
AC-07	Failure modes include safe degradation, dependency loss, corrupted state, operator error, and malicious action.
AC-08	Backup, restoration, reconstitution, and exit procedures are exercised against defined recovery objectives.
AC-09	Data classification, retention, privacy, sovereignty, and disposition controls follow the actual information flow.

ARCHITECTURE ACCEPTANCE CRITERIA - B

CRITERIA 10-18

AC-10	Supplier, model, protocol, platform, and software dependencies have provenance, version, support, and replacement records.
AC-11	Representative performance and capacity tests include saturation, contention, latency, and degraded conditions.
AC-12	Exceptions are time-bounded, visible, approved by accountable risk owners, and linked to compensating controls.
AC-13	Architecture diagrams and inventories remain synchronized with deployed components and interfaces.
AC-14	Mandatory trust, safety, privacy, recovery, and evidence gates cannot be bypassed by a lower assurance profile.
AC-15	External dependencies have contractual, technical, operational, data, and exit boundaries.
AC-16	Representative acceptance tests exercise normal, degraded, adversarial, recovery, and retirement scenarios.
AC-17	The architecture can explain why an action, decision, deployment, or access was permitted and what evidence supports it.
AC-18	A formal decision record names selected and rejected alternatives, residual risk, review date, and change triggers.

METRICS, EVIDENCE, AND DECISION GATES

MEASURE SERVICE OUTCOMES AND ASSURANCE

COVERAGE

Percent of components, interfaces, identities, data flows, dependencies, risks, and controls represented in current architecture evidence.

CHANGE QUALITY

Plan success, rollback, verification pass, escaped defect, drift, and emergency-change rates.

RELIABILITY

Availability, latency, saturation, dependency health, recovery time, recovery point, and safe-degradation success.

SECURITY

Privilege exposure, policy denial, identity compromise, vulnerable dependency, detection coverage, response, and evidence completeness.

HUMAN OUTCOME

Task success, operator workload, accessibility, override, contestability, training, and user-impact measures.

LIFECYCLE

Version currency, unsupported component, exception age, migration readiness, restore exercise, and retirement completeness.

Release or expansion gate: mandatory acceptance criteria pass; failed safety, privacy, identity, recovery, evidence, or legal gates block promotion regardless of aggregate score.

SOURCE AUTHORITY REGISTER

CURRENT PRIMARY AND OFFICIAL REFERENCES

SOURCE 01 **NIST SP 800-207, Zero Trust Architecture**

https://csrc.nist.gov/pubs/sp/800/207/final

SOURCE 02 **NIST SP 1800-35, Implementing a Zero Trust Architecture**

https://www.nccoe.nist.gov/projects/implementing-zero-trust-architecture

SOURCE 03 **NIST SP 800-53 Rev. 5**

https://doi.org/10.6028/NIST.SP.800-53r5

SOURCE 04 **W3C Web Authentication Level 3**

https://www.w3.org/TR/webauthn-3/

Source currency note. Official frameworks, specifications, and implementation guidance evolve. Assessors must verify the current version, status, errata, applicability, and effective date at adoption and scheduled review. NIST AI RMF 1.0 was under revision at the publication date.

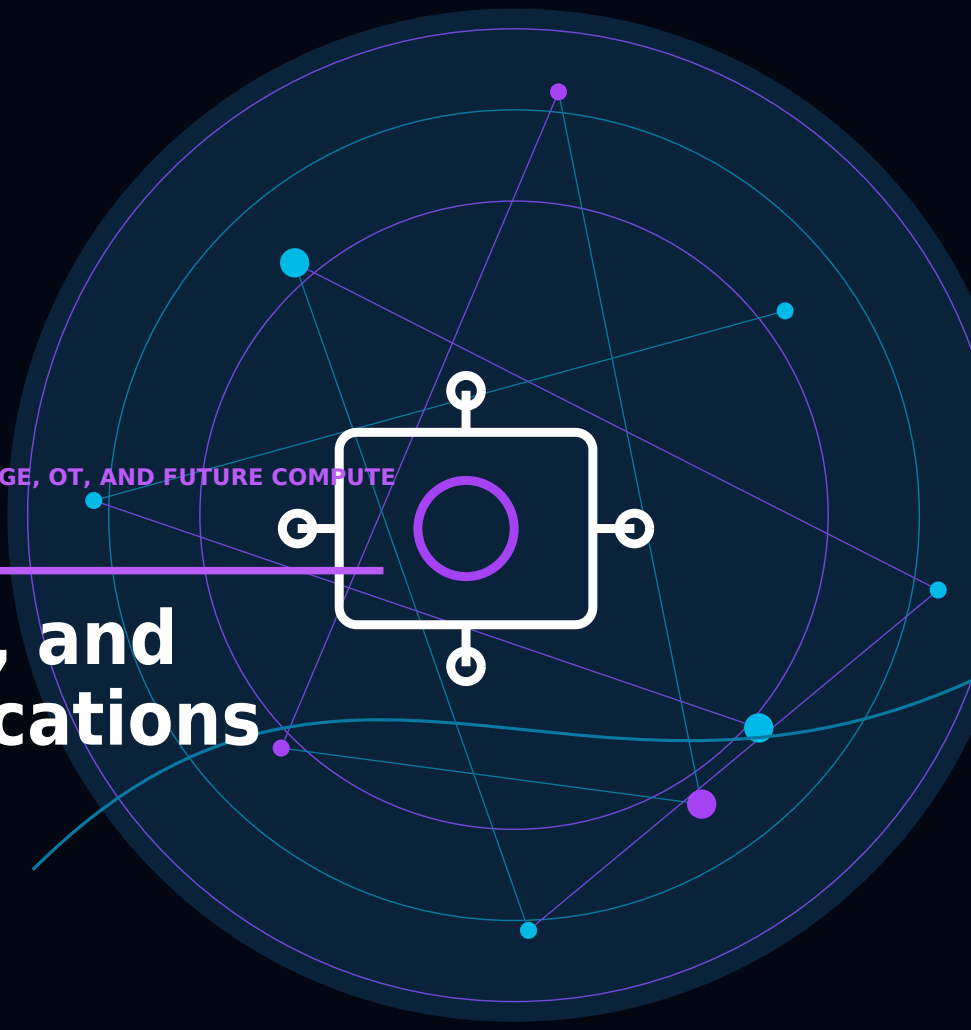
PUBLICATION CONTROL AND REVISION

PERMANENT IDENTITY / CONTROLLED EVOLUTION

PERMANENT DOCUMENT IDENTITY	RA-009
VERSION	1.0.0-candidate
PUBLICATION DATE	2026-07-24
SCHEDULED REVIEW	2027-01-24
STATUS	Candidate Reference Architecture
ARCHITECTURE SET	Future Domain Set
CLASSIFICATION	Public
PUBLISHER	Mythos Systems

REVISION RECORD

1.0.0-candidate (2026-07-24) - Permanent-publication candidate edition regenerated under the Mythos Systems architecture publication system. Production sequence metadata is excluded from public identity.



ENGINEERING STANDARDS LIBRARY / EDGE, OT, AND FUTURE COMPUTE

Edge, OT, IoT, and Telecommunications Convergence

A converged cyber-physical architecture for field devices, industrial control, edge compute, private wireless, public telecom, IoT, safety, disconnected operation, and fleet lifecycle.

PERMANENT DOCUMENT ID

RA-015

PUBLICATION

Candidate Reference Architecture
Version 1.0.0-candidate

ASSURANCE PROFILE

Candidate Architecture
High Assurance

PERMANENT PUBLICATION IDENTITY

Edge, OT, IoT, and Telecommunications Convergence

A converged cyber-physical architecture for field devices, industrial control, edge compute, private wireless, public telecom, IoT, safety, disconnected operation, and fleet lifecycle.

DOCUMENT ID
RA-015

VERSION
1.0.0-candidate

STATUS
Candidate Reference Architecture

ARCHITECTURE SET
Future Domain Set

DOMAIN
Edge, OT, and Future Compute

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

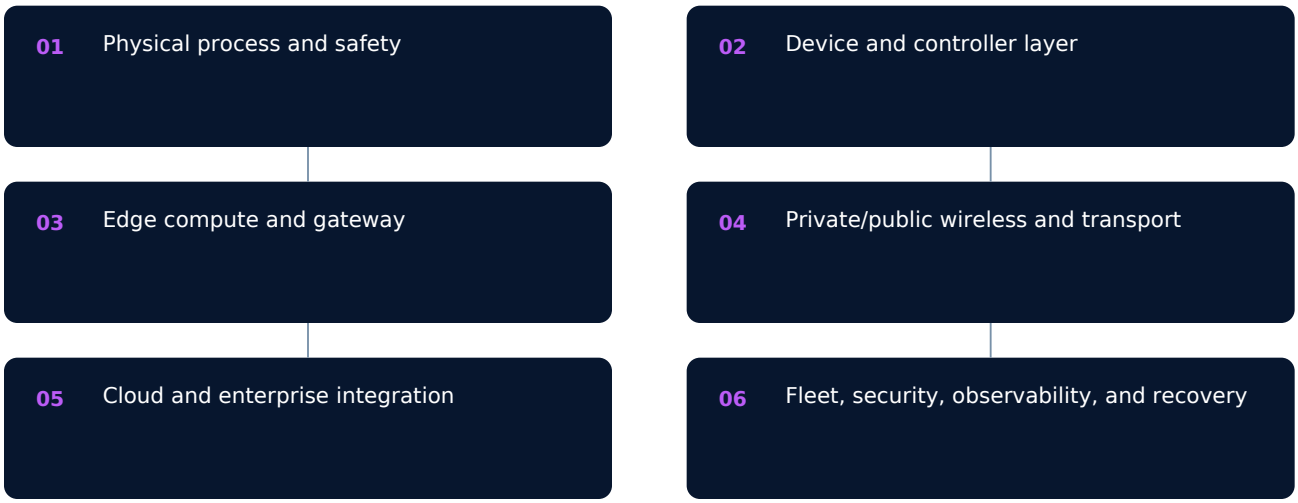
6	6	18	2
ARCHITECTURE LAYERS	CORE COMPONENTS	ACCEPTANCE CRITERIA	DEPENDENCIES

Architecture position. This future-domain candidate defines a stable architectural destination and assurance model. Product choices, scale, regulatory obligations, and implementation details remain environment-specific.

REFERENCE ARCHITECTURE TOPOLOGY

EDGE, OT, AND FUTURE COMPUTE

This future-domain candidate defines a stable architectural destination and assurance model. Product choices, scale, regulatory obligations, and implementation details remain environment-specific.



GOVERNED ARCHITECTURE FLOW

INTERPRETATION AND ASSURANCE

HOW TO APPLY THIS ARCHITECTURE

INFORMATIVE ARCHITECTURE

The architecture describes a deployable pattern and assurance model. Normative obligations remain in mapped standards, policy, law, contracts, and approved design decisions.

EVIDENCE OVER DIAGRAM

A diagram is not implementation evidence. Conformance requires inventories, configurations, tests, telemetry, incidents, recovery results, and accountable decisions.

PROFILE WITHOUT DILUTION

Profiles may change scale and technology, but cannot remove mandatory trust, safety, privacy, recovery, or evidence outcomes.

PRODUCT-NEUTRAL CORE

Named technologies are illustrative unless an organization explicitly adopts them. Interfaces, outcomes, and evidence remain primary.

BOUNDED CLAIMS

Performance, security, autonomy, availability, privacy, and compliance claims apply only to tested configurations and conditions.

CONTROLLED EVOLUTION

Architecture change preserves version, migration, rollback, source currency, exceptions, and residual-risk records.

Adoption requires environment-specific design, threat and hazard analysis, capacity engineering, implementation, verification, approval, operation, and periodic reassessment.

INTERACTIVE NAVIGATION

REFERENCE ARCHITECTURE CONTENTS

PART I - EXECUTIVE DIRECTION

OUTCOMES, RISKS, AND DECISIONS

ARCHITECTURE MANDATE

A converged cyber-physical architecture for field devices, industrial control, edge compute, private wireless, public telecom, IoT, safety, disconnected operation, and fleet lifecycle.

EXECUTIVE OUTCOMES

- Establish a deployable, product-neutral target architecture for edge, ot, iot, and telecommunications convergence.
- Make ownership, trust boundaries, data and control flows, failure handling, and evidence explicit.
- Support multiple implementation profiles without weakening mandatory assurance outcomes.
- Provide a governed adoption path from discovery through production operation and retirement.

DECISIONS REQUIRED

- Accountable service and risk ownership.
- Accepted architecture profile and consequence class.
- Authoritative identity, data, policy, configuration, and evidence systems.
- Mandatory approval, recovery, privacy, safety, and supplier-exit gates.
- Funding for operations, observability, validation, and lifecycle—not only implementation.

PART II - SCOPE, PRINCIPLES, AND SYSTEM CONTEXT

EDGE, OT, AND FUTURE COMPUTE

IN SCOPE

A converged cyber-physical architecture for field devices, industrial control, edge compute, private wireless, public telecom, IoT, safety, disconnected operation, and fleet lifecycle.

OUT OF SCOPE

Vendor selection, exact sizing, contractual obligations, detailed low-level design, product certification, regulatory approval, and proof that any named platform is implemented.

ARCHITECTURE INVARIANTS

Explicit ownership; least authority; separable desired, runtime, observed, and evidence state; bounded automation; tested recovery; product-neutral interfaces; source and claim currency.

ASSUMPTIONS

Organizations tailor the pattern through documented risk, environment, criticality, data, safety, regulatory, workforce, and supplier analysis.

SYSTEM CONTEXT AND RESPONSIBILITY MODEL

ACTORS, ENVIRONMENTS, AND EXTERNAL DEPENDENCIES

ACTORS AND RESPONSIBILITY

- Service consumers and affected users
- Product, platform, network, security, data, reliability, and support teams
- Human administrators, operators, reviewers, incident commanders, and risk owners
- Machine identities, agents, workloads, devices, controllers, and external services
- Suppliers, providers, carriers, integrators, assessors, and regulators where applicable

RESPONSIBILITY BOUNDARY

- The adopting organization retains accountability for purpose, risk, configuration, data, identity, operation, recovery, and evidence even where a provider performs underlying functions.
- Inherited controls are recorded with provider, scope, evidence, limitations, shared responsibility, incident path, and exit conditions.
- No component is considered trusted solely because it is internal, managed, signed, cloud-hosted, model-generated, or supplied by a recognized vendor.

ARCHITECTURE POSITION

This future-domain candidate defines a stable architectural destination and assurance model. Product choices, scale, regulatory obligations, and implementation details remain environment-specific.

PART III - LOGICAL ARCHITECTURE

LAYERS AND RESPONSIBILITIES

01

Physical process and safety

Physical process and safety owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

02

Device and controller layer

Device and controller layer owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

03

Edge compute and gateway

Edge compute and gateway owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

04

Private/public wireless and transport

Private/public wireless and transport owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

05

Cloud and enterprise integration

Cloud and enterprise integration owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

06

Fleet, security, observability, and recovery

Fleet, security, observability, and recovery owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

CORE COMPONENTS - A

COMPONENTS 01-03

COMPONENT 01

Sensors, actuators, PLCs, safety systems, robots, and physical process

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 02

Device identity, firmware, secure boot, commissioning, and local control

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 03

Edge clusters, gateways, brokers, inference, storage, and local autonomy

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

CORE COMPONENTS - B

COMPONENTS 04-06

COMPONENT 04

Ethernet, Wi-Fi, private 5G, carrier, satellite, timing, and QoS

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 05

APIs, events, data platforms, digital twin, remote operations, and enterprise services

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 06

Fleet management, segmentation, monitoring, maintenance, incident and disaster recovery

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

PART IV - DATA, CONTROL, AUTHORITY, AND EVIDENCE FLOW

FLOW CONTRACTS

1. Intent, service demand, policy, and environment constraints enter the architecture through explicit contracts.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

2. Authoritative identity, data, configuration, and dependency state are validated before execution or access.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

3. Control planes perform bounded orchestration while local enforcement preserves least privilege and safe behavior.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

4. Telemetry, tests, user outcomes, and incidents update evidence and trigger correction or rollback.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

5. Lifecycle governance manages versioning, capacity, exceptions, migration, and retirement.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

TRUST BOUNDARIES AND ENFORCEMENT

EVERY CROSSING REQUIRES AN EXPLICIT CONTRACT

TRUST BOUNDARY 01

Human or machine actor versus authorization service

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 02

Control plane versus workload, device, or enforcement point

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 03

Tenant or project versus shared platform

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 04

Local environment versus external provider or network

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 05

Mutable runtime state versus authoritative evidence

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 06

Normal operation versus emergency or break-glass authority

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

PART V - DEPLOYMENT PROFILES

PROFILE CHANGES SCALE, NOT ASSURANCE OUTCOMES

PROFILE 01 / PILOT OR LABORATORY PROFILE

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PROFILE 02 / DEPARTMENT OR BOUNDED PRODUCTION PROFILE

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PROFILE 03 / ENTERPRISE FEDERATED PROFILE

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PROFILE 04 / HIGH-ASSURANCE SOVEREIGN OR REGULATED PROFILE

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PART VI - OPERATING LIFECYCLE

FROM DISCOVERY THROUGH RETIREMENT

01 / Discover and classify demand

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

02 / Define service boundary and owners

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

03 / Model architecture and acceptance criteria

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

04 / Build isolated proof and challenge assumptions

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

05 / Pilot with stop conditions and rollback

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

06 / Scale through standard profiles and automation

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

07 / Operate, reassess, migrate, and retire

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

SECURITY, PRIVACY, SAFETY, AND RIGHTS

CROSS-CUTTING ASSURANCE

SECURITY

Threat model identities, trust boundaries, supply chain, control planes, data, protocols, privilege, abuse, detection, response, recovery, and residual risk.

PRIVACY

Purpose limitation, minimization, transparency, consent where required, access, correction, retention, deletion, sovereignty, metadata, and third-party processing.

SAFETY

Hazards, unsafe states, human intervention, physical consequence, dependency failure, automation limits, emergency stop, recovery sequence, and validation.

RIGHTS AND USABILITY

Accessibility, contestability, explanation, operator workload, affected-user communication, grievance, misuse prevention, and equitable performance.

SUPPLY CHAIN

Provenance, version, SBOM or component inventory, signature, vulnerability, support, update, provider, subcontractor, and exit.

EVIDENCE

Design decisions, tests, telemetry, incidents, approvals, exceptions, recovery exercises, source currency, and review findings remain linked to the architecture version.

FAILURE MODES AND RESILIENCE

DESIGN FOR SAFE DEGRADATION AND TRUSTWORTHY RECOVERY

FAILURE SCENARIO 01

Control-plane compromise or excessive privilege.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 02

Identity, policy, data, or configuration state becomes stale or inconsistent.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 03

A shared dependency creates correlated failure across tenants or domains.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 04

Automation succeeds technically but violates intended service, safety, or privacy outcomes.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 05

Recovery restores components without restoring trustworthy service state.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 06

Exit, migration, revocation, or deletion cannot be completed within the required time.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

PART VII - INTEGRATION AND DEPENDENCY MODEL

CONNECTED REFERENCE ARCHITECTURES AND STANDARDS

REFERENCE ARCHITECTURE DEPENDENCIES

- RA-017 - Quantum-Safe, Confidential, and Verifiable Compute
- RA-018 - Physical AI and Autonomous Fleet Operations

MAPPED MYTHOS STANDARDS

- MYTHOS-OT-0001
- MYTHOS-IOT-0001
- MYTHOS-TEL-0001
- MYTHOS-CLD-0010

DEPENDENCY DOCTRINE

A dependency is not a blanket trust grant. Each connection requires an interface, identity, policy, data, availability, version, failure, recovery, and evidence contract. Correlated failure and circular authority must be analyzed across the full architecture graph.

PART VIII - IMPLEMENTATION AND ADOPTION ROADMAP

PHASED, EVIDENCE-DRIVEN TRANSITION

0-30 DAYS	Inventory purpose, owners, users, systems, data, dependencies, risks, current diagrams, incidents, recovery, and evidence gaps.
31-90 DAYS	Establish authoritative identity, configuration, policy, data, source, and architecture records; define target profile and acceptance tests.
3-6 MONTHS	Implement bounded foundations, isolation, telemetry, change control, backup, recovery, and representative pilot workflows.
6-12 MONTHS	Expand through standard patterns, automation, service ownership, SLOs, independent assessment, failure injection, and supplier-exit exercises.
ONGOING	Reconcile drift, review sources, reassess risk, measure outcomes, exercise recovery, renew exceptions, migrate dependencies, and retire obsolete paths.

ARCHITECTURE ACCEPTANCE CRITERIA - A

CRITERIA 01-09

AC-01	The architecture has a named service owner, technical owner, security owner, data owner, and recovery authority.
AC-02	The system boundary, external dependencies, retained responsibilities, and trust transitions are documented and reviewed.
AC-03	Identity, authorization, secrets, policy, and privileged operations are explicit and independently auditable.
AC-04	Desired state, runtime state, observed state, and evidence state are distinguishable and reconciled.
AC-05	Consequential changes require preconditions, approval policy, bounded execution, verification, and rollback.
AC-06	Telemetry supports service health, security detection, forensic reconstruction, and conformance evidence.
AC-07	Failure modes include safe degradation, dependency loss, corrupted state, operator error, and malicious action.
AC-08	Backup, restoration, reconstitution, and exit procedures are exercised against defined recovery objectives.
AC-09	Data classification, retention, privacy, sovereignty, and disposition controls follow the actual information flow.

ARCHITECTURE ACCEPTANCE CRITERIA - B

CRITERIA 10-18

AC-10	Supplier, model, protocol, platform, and software dependencies have provenance, version, support, and replacement records.
AC-11	Representative performance and capacity tests include saturation, contention, latency, and degraded conditions.
AC-12	Exceptions are time-bounded, visible, approved by accountable risk owners, and linked to compensating controls.
AC-13	Architecture diagrams and inventories remain synchronized with deployed components and interfaces.
AC-14	Mandatory trust, safety, privacy, recovery, and evidence gates cannot be bypassed by a lower assurance profile.
AC-15	External dependencies have contractual, technical, operational, data, and exit boundaries.
AC-16	Representative acceptance tests exercise normal, degraded, adversarial, recovery, and retirement scenarios.
AC-17	The architecture can explain why an action, decision, deployment, or access was permitted and what evidence supports it.
AC-18	A formal decision record names selected and rejected alternatives, residual risk, review date, and change triggers.

METRICS, EVIDENCE, AND DECISION GATES

MEASURE SERVICE OUTCOMES AND ASSURANCE

COVERAGE

Percent of components, interfaces, identities, data flows, dependencies, risks, and controls represented in current architecture evidence.

CHANGE QUALITY

Plan success, rollback, verification pass, escaped defect, drift, and emergency-change rates.

RELIABILITY

Availability, latency, saturation, dependency health, recovery time, recovery point, and safe-degradation success.

SECURITY

Privilege exposure, policy denial, identity compromise, vulnerable dependency, detection coverage, response, and evidence completeness.

HUMAN OUTCOME

Task success, operator workload, accessibility, override, contestability, training, and user-impact measures.

LIFECYCLE

Version currency, unsupported component, exception age, migration readiness, restore exercise, and retirement completeness.

Release or expansion gate: mandatory acceptance criteria pass; failed safety, privacy, identity, recovery, evidence, or legal gates block promotion regardless of aggregate score.

SOURCE AUTHORITY REGISTER

CURRENT PRIMARY AND OFFICIAL REFERENCES

SOURCE 01 **NIST SP 800-207, Zero Trust Architecture**

https://csrc.nist.gov/pubs/sp/800/207/final

SOURCE 02 **NIST SP 800-53 Rev. 5**

https://doi.org/10.6028/NIST.SP.800-53r5

SOURCE 03 **OpenTelemetry Specification**

https://opentelemetry.io/docs/specs/

SOURCE 04 **CISA Secure by Design**

https://www.cisa.gov/resources-tools/resources/secure-by-design

Source currency note. Official frameworks, specifications, and implementation guidance evolve. Assessors must verify the current version, status, errata, applicability, and effective date at adoption and scheduled review. NIST AI RMF 1.0 was under revision at the publication date.

PUBLICATION CONTROL AND REVISION

PERMANENT IDENTITY / CONTROLLED EVOLUTION

PERMANENT DOCUMENT IDENTITY	RA-015
VERSION	1.0.0-candidate
PUBLICATION DATE	2026-07-24
SCHEDULED REVIEW	2027-01-24
STATUS	Candidate Reference Architecture
ARCHITECTURE SET	Future Domain Set
CLASSIFICATION	Public
PUBLISHER	Mythos Systems

REVISION RECORD

1.0.0-candidate (2026-07-24) - Permanent-publication candidate edition regenerated under the Mythos Systems architecture publication system. Production sequence metadata is excluded from public identity.

ENGINEERING STANDARDS LIBRARY / HUMAN SYSTEMS AND LEARNING

Realtime Voice, Media, and Multimodal Interaction Platform

A realtime interaction architecture for speech, audio, video, avatars or presence, transcription, translation, multimodal models, accessibility, consent, provenance, and low-latency orchestration.

PERMANENT DOCUMENT ID

RA-016

PUBLICATION

Candidate Reference Architecture
Version 1.0.0-candidate

ASSURANCE PROFILE

Candidate Architecture
High Assurance

PERMANENT PUBLICATION IDENTITY

Realtime Voice, Media, and Multimodal Interaction Platform

A realtime interaction architecture for speech, audio, video, avatars or presence, transcription, translation, multimodal models, accessibility, consent, provenance, and low-latency orchestration.

DOCUMENT ID
RA-016

VERSION
1.0.0-candidate

STATUS
Candidate Reference Architecture

ARCHITECTURE SET
Future Domain Set

DOMAIN
Human Systems and Learning

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

6	6	18	2
ARCHITECTURE LAYERS	CORE COMPONENTS	ACCEPTANCE CRITERIA	DEPENDENCIES

Architecture position. This future-domain candidate defines a stable architectural destination and assurance model. Product choices, scale, regulatory obligations, and implementation details remain environment-specific.

REFERENCE ARCHITECTURE TOPOLOGY

HUMAN SYSTEMS AND LEARNING

This future-domain candidate defines a stable architectural destination and assurance model. Product choices, scale, regulatory obligations, and implementation details remain environment-specific.

GOVERNED ARCHITECTURE FLOW

INTERPRETATION AND ASSURANCE

HOW TO APPLY THIS ARCHITECTURE

INFORMATIVE ARCHITECTURE

The architecture describes a deployable pattern and assurance model. Normative obligations remain in mapped standards, policy, law, contracts, and approved design decisions.

EVIDENCE OVER DIAGRAM

A diagram is not implementation evidence. Conformance requires inventories, configurations, tests, telemetry, incidents, recovery results, and accountable decisions.

PROFILE WITHOUT DILUTION

Profiles may change scale and technology, but cannot remove mandatory trust, safety, privacy, recovery, or evidence outcomes.

PRODUCT-NEUTRAL CORE

Named technologies are illustrative unless an organization explicitly adopts them. Interfaces, outcomes, and evidence remain primary.

BOUNDED CLAIMS

Performance, security, autonomy, availability, privacy, and compliance claims apply only to tested configurations and conditions.

CONTROLLED EVOLUTION

Architecture change preserves version, migration, rollback, source currency, exceptions, and residual-risk records.

Adoption requires environment-specific design, threat and hazard analysis, capacity engineering, implementation, verification, approval, operation, and periodic reassessment.

INTERACTIVE NAVIGATION

REFERENCE ARCHITECTURE CONTENTS

PART I - EXECUTIVE DIRECTION

OUTCOMES, RISKS, AND DECISIONS

ARCHITECTURE MANDATE

A realtime interaction architecture for speech, audio, video, avatars or presence, transcription, translation, multimodal models, accessibility, consent, provenance, and low-latency orchestration.

EXECUTIVE OUTCOMES

- Establish a deployable, product-neutral target architecture for realtime voice, media, and multimodal interaction platform.
- Make ownership, trust boundaries, data and control flows, failure handling, and evidence explicit.
- Support multiple implementation profiles without weakening mandatory assurance outcomes.
- Provide a governed adoption path from discovery through production operation and retirement.

DECISIONS REQUIRED

- Accountable service and risk ownership.
- Accepted architecture profile and consequence class.
- Authoritative identity, data, policy, configuration, and evidence systems.
- Mandatory approval, recovery, privacy, safety, and supplier-exit gates.
- Funding for operations, observability, validation, and lifecycle—not only implementation.

PART II - SCOPE, PRINCIPLES, AND SYSTEM CONTEXT

HUMAN SYSTEMS AND LEARNING

IN SCOPE

A realtime interaction architecture for speech, audio, video, avatars or presence, transcription, translation, multimodal models, accessibility, consent, provenance, and low-latency orchestration.

OUT OF SCOPE

Vendor selection, exact sizing, contractual obligations, detailed low-level design, product certification, regulatory approval, and proof that any named platform is implemented.

ARCHITECTURE INVARIANTS

Explicit ownership; least authority; separable desired, runtime, observed, and evidence state; bounded automation; tested recovery; product-neutral interfaces; source and claim currency.

ASSUMPTIONS

Organizations tailor the pattern through documented risk, environment, criticality, data, safety, regulatory, workforce, and supplier analysis.

SYSTEM CONTEXT AND RESPONSIBILITY MODEL

ACTORS, ENVIRONMENTS, AND EXTERNAL DEPENDENCIES

ACTORS AND RESPONSIBILITY

- Service consumers and affected users
- Product, platform, network, security, data, reliability, and support teams
- Human administrators, operators, reviewers, incident commanders, and risk owners
- Machine identities, agents, workloads, devices, controllers, and external services
- Suppliers, providers, carriers, integrators, assessors, and regulators where applicable

RESPONSIBILITY BOUNDARY

- The adopting organization retains accountability for purpose, risk, configuration, data, identity, operation, recovery, and evidence even where a provider performs underlying functions.
- Inherited controls are recorded with provider, scope, evidence, limitations, shared responsibility, incident path, and exit conditions.
- No component is considered trusted solely because it is internal, managed, signed, cloud-hosted, model-generated, or supplied by a recognized vendor.

ARCHITECTURE POSITION

This future-domain candidate defines a stable architectural destination and assurance model. Product choices, scale, regulatory obligations, and implementation details remain environment-specific.

PART III - LOGICAL ARCHITECTURE

LAYERS AND RESPONSIBILITIES

01

Capture and device interaction

Capture and device interaction owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

02

Realtime media transport

Realtime media transport owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

03

Speech and multimodal intelligence

Speech and multimodal intelligence owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

04

Conversation and presence runtime

Conversation and presence runtime owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

05

Consent, privacy, safety, and provenance

Consent, privacy, safety, and provenance owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

06

Quality, accessibility, storage, and recovery

Quality, accessibility, storage, and recovery owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

CORE COMPONENTS - A

COMPONENTS 01-03

COMPONENT 01

Microphone, camera, screen, sensors, device controls, and permissions

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 02

WebRTC/media transport, relay, QoS, jitter, encryption, and regional routing

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 03

STT, TTS, translation, diarization, vision, model routing, and grounding

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

CORE COMPONENTS - B

COMPONENTS 04-06

COMPONENT 04

Turn-taking, VAD, interruption, session memory, presence behavior, and tool invocation

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 05

Consent, disclosure, recording policy, synthetic-media marking, identity, moderation, and audit

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 06

Captioning, accessibility, experience telemetry, retention, export, incident and fallback

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

PART IV - DATA, CONTROL, AUTHORITY, AND EVIDENCE FLOW

FLOW CONTRACTS

1. Intent, service demand, policy, and environment constraints enter the architecture through explicit contracts.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

2. Authoritative identity, data, configuration, and dependency state are validated before execution or access.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

3. Control planes perform bounded orchestration while local enforcement preserves least privilege and safe behavior.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

4. Telemetry, tests, user outcomes, and incidents update evidence and trigger correction or rollback.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

5. Lifecycle governance manages versioning, capacity, exceptions, migration, and retirement.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

TRUST BOUNDARIES AND ENFORCEMENT

EVERY CROSSING REQUIRES AN EXPLICIT CONTRACT

TRUST BOUNDARY 01

Human or machine actor versus authorization service

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 02

Control plane versus workload, device, or enforcement point

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 03

Tenant or project versus shared platform

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 04

Local environment versus external provider or network

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 05

Mutable runtime state versus authoritative evidence

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 06

Normal operation versus emergency or break-glass authority

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

PART V - DEPLOYMENT PROFILES

PROFILE CHANGES SCALE, NOT ASSURANCE OUTCOMES

PROFILE 01 / PILOT OR LABORATORY PROFILE

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PROFILE 02 / DEPARTMENT OR BOUNDED PRODUCTION PROFILE

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PROFILE 03 / ENTERPRISE FEDERATED PROFILE

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PROFILE 04 / HIGH-ASSURANCE SOVEREIGN OR REGULATED PROFILE

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PART VI - OPERATING LIFECYCLE

FROM DISCOVERY THROUGH RETIREMENT

01 / Discover and classify demand

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

02 / Define service boundary and owners

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

03 / Model architecture and acceptance criteria

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

04 / Build isolated proof and challenge assumptions

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

05 / Pilot with stop conditions and rollback

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

06 / Scale through standard profiles and automation

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

07 / Operate, reassess, migrate, and retire

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

SECURITY, PRIVACY, SAFETY, AND RIGHTS

CROSS-CUTTING ASSURANCE

SECURITY

Threat model identities, trust boundaries, supply chain, control planes, data, protocols, privilege, abuse, detection, response, recovery, and residual risk.

PRIVACY

Purpose limitation, minimization, transparency, consent where required, access, correction, retention, deletion, sovereignty, metadata, and third-party processing.

SAFETY

Hazards, unsafe states, human intervention, physical consequence, dependency failure, automation limits, emergency stop, recovery sequence, and validation.

RIGHTS AND USABILITY

Accessibility, contestability, explanation, operator workload, affected-user communication, grievance, misuse prevention, and equitable performance.

SUPPLY CHAIN

Provenance, version, SBOM or component inventory, signature, vulnerability, support, update, provider, subcontractor, and exit.

EVIDENCE

Design decisions, tests, telemetry, incidents, approvals, exceptions, recovery exercises, source currency, and review findings remain linked to the architecture version.

FAILURE MODES AND RESILIENCE

DESIGN FOR SAFE DEGRADATION AND TRUSTWORTHY RECOVERY

FAILURE SCENARIO 01

Control-plane compromise or excessive privilege.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 02

Identity, policy, data, or configuration state becomes stale or inconsistent.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 03

A shared dependency creates correlated failure across tenants or domains.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 04

Automation succeeds technically but violates intended service, safety, or privacy outcomes.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 05

Recovery restores components without restoring trustworthy service state.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 06

Exit, migration, revocation, or deletion cannot be completed within the required time.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

PART VII - INTEGRATION AND DEPENDENCY MODEL

CONNECTED REFERENCE ARCHITECTURES AND STANDARDS

REFERENCE ARCHITECTURE DEPENDENCIES

- RA-019 - Sovereign Browser, Remote Access, and Web Automation
- RA-020 - Adaptive Learning, Cyber Range, and Workforce Assurance

MAPPED MYTHOS STANDARDS

- MYTHOS-MED-0001
- MYTHOS-AI-0008
- MYTHOS-ID-0001
- MYTHOS-WEB-0001

DEPENDENCY DOCTRINE

A dependency is not a blanket trust grant. Each connection requires an interface, identity, policy, data, availability, version, failure, recovery, and evidence contract. Correlated failure and circular authority must be analyzed across the full architecture graph.

PART VIII - IMPLEMENTATION AND ADOPTION ROADMAP

PHASED, EVIDENCE-DRIVEN TRANSITION

0-30 DAYS	Inventory purpose, owners, users, systems, data, dependencies, risks, current diagrams, incidents, recovery, and evidence gaps.
31-90 DAYS	Establish authoritative identity, configuration, policy, data, source, and architecture records; define target profile and acceptance tests.
3-6 MONTHS	Implement bounded foundations, isolation, telemetry, change control, backup, recovery, and representative pilot workflows.
6-12 MONTHS	Expand through standard patterns, automation, service ownership, SLOs, independent assessment, failure injection, and supplier-exit exercises.
ONGOING	Reconcile drift, review sources, reassess risk, measure outcomes, exercise recovery, renew exceptions, migrate dependencies, and retire obsolete paths.

ARCHITECTURE ACCEPTANCE CRITERIA - A

CRITERIA 01-09

AC-01	The architecture has a named service owner, technical owner, security owner, data owner, and recovery authority.
AC-02	The system boundary, external dependencies, retained responsibilities, and trust transitions are documented and reviewed.
AC-03	Identity, authorization, secrets, policy, and privileged operations are explicit and independently auditable.
AC-04	Desired state, runtime state, observed state, and evidence state are distinguishable and reconciled.
AC-05	Consequential changes require preconditions, approval policy, bounded execution, verification, and rollback.
AC-06	Telemetry supports service health, security detection, forensic reconstruction, and conformance evidence.
AC-07	Failure modes include safe degradation, dependency loss, corrupted state, operator error, and malicious action.
AC-08	Backup, restoration, reconstitution, and exit procedures are exercised against defined recovery objectives.
AC-09	Data classification, retention, privacy, sovereignty, and disposition controls follow the actual information flow.

ARCHITECTURE ACCEPTANCE CRITERIA - B

CRITERIA 10-18

AC-10	Supplier, model, protocol, platform, and software dependencies have provenance, version, support, and replacement records.
AC-11	Representative performance and capacity tests include saturation, contention, latency, and degraded conditions.
AC-12	Exceptions are time-bounded, visible, approved by accountable risk owners, and linked to compensating controls.
AC-13	Architecture diagrams and inventories remain synchronized with deployed components and interfaces.
AC-14	Mandatory trust, safety, privacy, recovery, and evidence gates cannot be bypassed by a lower assurance profile.
AC-15	External dependencies have contractual, technical, operational, data, and exit boundaries.
AC-16	Representative acceptance tests exercise normal, degraded, adversarial, recovery, and retirement scenarios.
AC-17	The architecture can explain why an action, decision, deployment, or access was permitted and what evidence supports it.
AC-18	A formal decision record names selected and rejected alternatives, residual risk, review date, and change triggers.

METRICS, EVIDENCE, AND DECISION GATES

MEASURE SERVICE OUTCOMES AND ASSURANCE

COVERAGE

Percent of components, interfaces, identities, data flows, dependencies, risks, and controls represented in current architecture evidence.

CHANGE QUALITY

Plan success, rollback, verification pass, escaped defect, drift, and emergency-change rates.

RELIABILITY

Availability, latency, saturation, dependency health, recovery time, recovery point, and safe-degradation success.

SECURITY

Privilege exposure, policy denial, identity compromise, vulnerable dependency, detection coverage, response, and evidence completeness.

HUMAN OUTCOME

Task success, operator workload, accessibility, override, contestability, training, and user-impact measures.

LIFECYCLE

Version currency, unsupported component, exception age, migration readiness, restore exercise, and retirement completeness.

Release or expansion gate: mandatory acceptance criteria pass; failed safety, privacy, identity, recovery, evidence, or legal gates block promotion regardless of aggregate score.

SOURCE AUTHORITY REGISTER

CURRENT PRIMARY AND OFFICIAL REFERENCES

SOURCE 01 **C2PA Technical Specification**

https://c2pa.org/specifications/

SOURCE 02 **NIST AI Risk Management Framework 1.0 (revision in progress as of publication date)**

https://www.nist.gov/itl/ai-risk-management-framework

SOURCE 03 **NIST SP 800-53 Rev. 5**

https://doi.org/10.6028/NIST.SP.800-53r5

SOURCE 04 **OpenTelemetry Specification**

https://opentelemetry.io/docs/specs/

Source currency note. Official frameworks, specifications, and implementation guidance evolve. Assessors must verify the current version, status, errata, applicability, and effective date at adoption and scheduled review. NIST AI RMF 1.0 was under revision at the publication date.

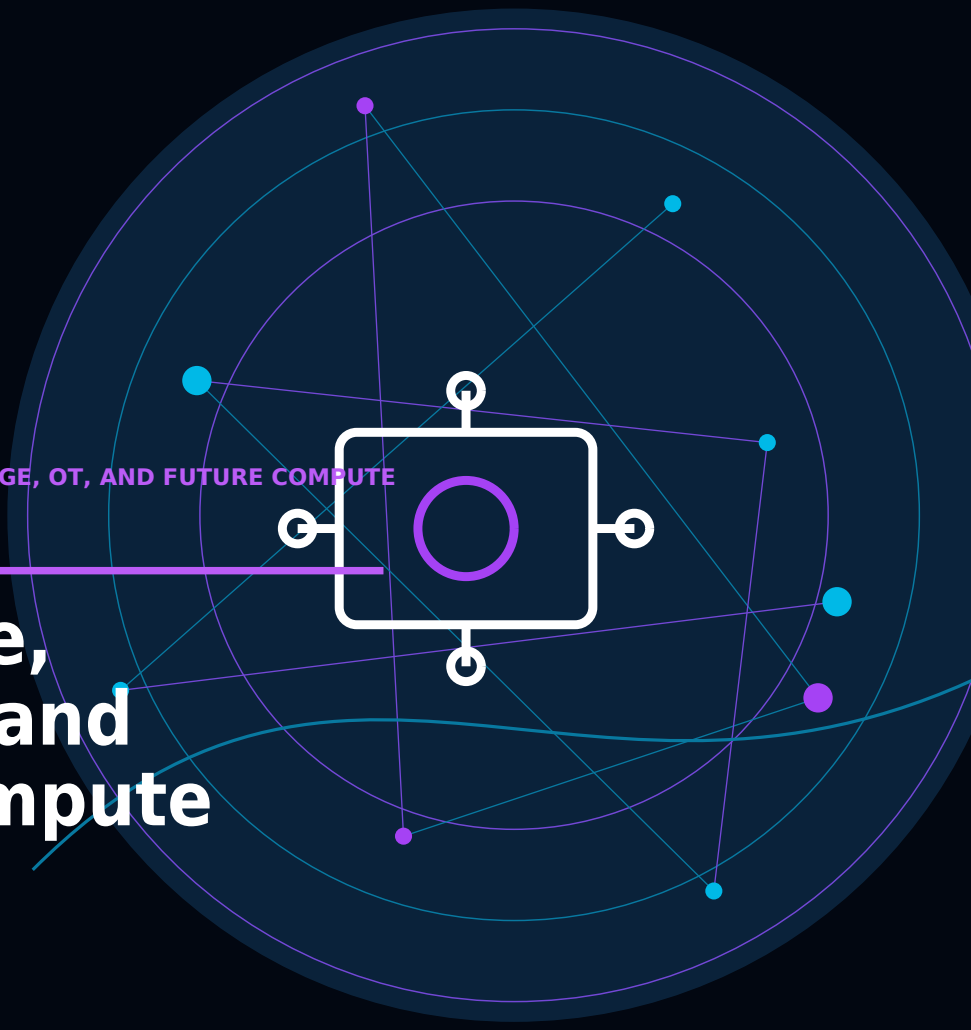
PUBLICATION CONTROL AND REVISION

PERMANENT IDENTITY / CONTROLLED EVOLUTION

PERMANENT DOCUMENT IDENTITY	RA-016
VERSION	1.0.0-candidate
PUBLICATION DATE	2026-07-24
SCHEDULED REVIEW	2027-01-24
STATUS	Candidate Reference Architecture
ARCHITECTURE SET	Future Domain Set
CLASSIFICATION	Public
PUBLISHER	Mythos Systems

REVISION RECORD

1.0.0-candidate (2026-07-24) - Permanent-publication candidate edition regenerated under the Mythos Systems architecture publication system. Production sequence metadata is excluded from public identity.



ENGINEERING STANDARDS LIBRARY / EDGE, OT, AND FUTURE COMPUTE

Quantum-Safe, Confidential, and Verifiable Compute

A compute trust architecture combining crypto-agility, post-quantum migration, confidential computing, attestation, measured boot, workload identity, verifiable artifacts, and controlled key release.

PERMANENT DOCUMENT ID

RA-017

PUBLICATION

Candidate Reference Architecture
Version 1.0.0-candidate

ASSURANCE PROFILE

Candidate Architecture
High Assurance

PERMANENT PUBLICATION IDENTITY

Quantum-Safe, Confidential, and Verifiable Compute

A compute trust architecture combining crypto-agility, post-quantum migration, confidential computing, attestation, measured boot, workload identity, verifiable artifacts, and controlled key release.

DOCUMENT ID
RA-017

VERSION
1.0.0-candidate

STATUS
Candidate Reference Architecture

ARCHITECTURE SET
Future Domain Set

DOMAIN
Edge, OT, and Future Compute

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

6	6	18	0
ARCHITECTURE LAYERS	CORE COMPONENTS	ACCEPTANCE CRITERIA	DEPENDENCIES

Architecture position. This future-domain candidate defines a stable architectural destination and assurance model. Product choices, scale, regulatory obligations, and implementation details remain environment-specific.

REFERENCE ARCHITECTURE TOPOLOGY

EDGE, OT, AND FUTURE COMPUTE

This future-domain candidate defines a stable architectural destination and assurance model. Product choices, scale, regulatory obligations, and implementation details remain environment-specific.

GOVERNED ARCHITECTURE FLOW

INTERPRETATION AND ASSURANCE

HOW TO APPLY THIS ARCHITECTURE

INFORMATIVE ARCHITECTURE

The architecture describes a deployable pattern and assurance model. Normative obligations remain in mapped standards, policy, law, contracts, and approved design decisions.

EVIDENCE OVER DIAGRAM

A diagram is not implementation evidence. Conformance requires inventories, configurations, tests, telemetry, incidents, recovery results, and accountable decisions.

PROFILE WITHOUT DILUTION

Profiles may change scale and technology, but cannot remove mandatory trust, safety, privacy, recovery, or evidence outcomes.

PRODUCT-NEUTRAL CORE

Named technologies are illustrative unless an organization explicitly adopts them. Interfaces, outcomes, and evidence remain primary.

BOUNDED CLAIMS

Performance, security, autonomy, availability, privacy, and compliance claims apply only to tested configurations and conditions.

CONTROLLED EVOLUTION

Architecture change preserves version, migration, rollback, source currency, exceptions, and residual-risk records.

Adoption requires environment-specific design, threat and hazard analysis, capacity engineering, implementation, verification, approval, operation, and periodic reassessment.

INTERACTIVE NAVIGATION

REFERENCE ARCHITECTURE CONTENTS

PART I - EXECUTIVE DIRECTION

OUTCOMES, RISKS, AND DECISIONS

ARCHITECTURE MANDATE

A compute trust architecture combining crypto-agility, post-quantum migration, confidential computing, attestation, measured boot, workload identity, verifiable artifacts, and controlled key release.

EXECUTIVE OUTCOMES

- Establish a deployable, product-neutral target architecture for quantum-safe, confidential, and verifiable compute.
- Make ownership, trust boundaries, data and control flows, failure handling, and evidence explicit.
- Support multiple implementation profiles without weakening mandatory assurance outcomes.
- Provide a governed adoption path from discovery through production operation and retirement.

DECISIONS REQUIRED

- Accountable service and risk ownership.
- Accepted architecture profile and consequence class.
- Authoritative identity, data, policy, configuration, and evidence systems.
- Mandatory approval, recovery, privacy, safety, and supplier-exit gates.
- Funding for operations, observability, validation, and lifecycle—not only implementation.

PART II - SCOPE, PRINCIPLES, AND SYSTEM CONTEXT

EDGE, OT, AND FUTURE COMPUTE

IN SCOPE

A compute trust architecture combining crypto-agility, post-quantum migration, confidential computing, attestation, measured boot, workload identity, verifiable artifacts, and controlled key release.

OUT OF SCOPE

Vendor selection, exact sizing, contractual obligations, detailed low-level design, product certification, regulatory approval, and proof that any named platform is implemented.

ARCHITECTURE INVARIANTS

Explicit ownership; least authority; separable desired, runtime, observed, and evidence state; bounded automation; tested recovery; product-neutral interfaces; source and claim currency.

ASSUMPTIONS

Organizations tailor the pattern through documented risk, environment, criticality, data, safety, regulatory, workforce, and supplier analysis.

SYSTEM CONTEXT AND RESPONSIBILITY MODEL

ACTORS, ENVIRONMENTS, AND EXTERNAL DEPENDENCIES

ACTORS AND RESPONSIBILITY

- Service consumers and affected users
- Product, platform, network, security, data, reliability, and support teams
- Human administrators, operators, reviewers, incident commanders, and risk owners
- Machine identities, agents, workloads, devices, controllers, and external services
- Suppliers, providers, carriers, integrators, assessors, and regulators where applicable

RESPONSIBILITY BOUNDARY

- The adopting organization retains accountability for purpose, risk, configuration, data, identity, operation, recovery, and evidence even where a provider performs underlying functions.
- Inherited controls are recorded with provider, scope, evidence, limitations, shared responsibility, incident path, and exit conditions.
- No component is considered trusted solely because it is internal, managed, signed, cloud-hosted, model-generated, or supplied by a recognized vendor.

ARCHITECTURE POSITION

This future-domain candidate defines a stable architectural destination and assurance model. Product choices, scale, regulatory obligations, and implementation details remain environment-specific.

PART III - LOGICAL ARCHITECTURE

LAYERS AND RESPONSIBILITIES

01

Cryptographic inventory and agility

Cryptographic inventory and agility owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

02

Hardware root of trust

Hardware root of trust owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

03

Measured boot and attestation

Measured boot and attestation owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

04

Confidential workload execution

Confidential workload execution owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

05

Verifiable software and data

Verifiable software and data owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

06

Key release, migration, recovery, and evidence

Key release, migration, recovery, and evidence owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

CORE COMPONENTS - A

COMPONENTS 01-03

COMPONENT 01

Algorithm, protocol, certificate, data-lifetime, and dependency inventory

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 02

TPM/HSM/secure element, firmware, boot chain, and platform identity

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 03

Reference values, attestation service, verifier policy, freshness, and revocation

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

CORE COMPONENTS - B

COMPONENTS 04-06

COMPONENT 04

TEE or confidential VM/container boundaries, memory protection, and side-channel controls

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 05

Signed artifacts, provenance, SBOM, data integrity, trusted time, and transparent evidence

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 06

Conditional key release, hybrid PQC, migration waves, break-glass, backup, and platform replacement

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

PART IV - DATA, CONTROL, AUTHORITY, AND EVIDENCE FLOW

FLOW CONTRACTS

1. Intent, service demand, policy, and environment constraints enter the architecture through explicit contracts.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

2. Authoritative identity, data, configuration, and dependency state are validated before execution or access.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

3. Control planes perform bounded orchestration while local enforcement preserves least privilege and safe behavior.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

4. Telemetry, tests, user outcomes, and incidents update evidence and trigger correction or rollback.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

5. Lifecycle governance manages versioning, capacity, exceptions, migration, and retirement.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

TRUST BOUNDARIES AND ENFORCEMENT

EVERY CROSSING REQUIRES AN EXPLICIT CONTRACT

TRUST BOUNDARY 01

Human or machine actor versus authorization service

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 02

Control plane versus workload, device, or enforcement point

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 03

Tenant or project versus shared platform

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 04

Local environment versus external provider or network

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 05

Mutable runtime state versus authoritative evidence

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 06

Normal operation versus emergency or break-glass authority

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

PART V - DEPLOYMENT PROFILES

PROFILE CHANGES SCALE, NOT ASSURANCE OUTCOMES

PROFILE 01 / PILOT OR LABORATORY PROFILE

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PROFILE 02 / DEPARTMENT OR BOUNDED PRODUCTION PROFILE

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PROFILE 03 / ENTERPRISE FEDERATED PROFILE

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PROFILE 04 / HIGH-ASSURANCE SOVEREIGN OR REGULATED PROFILE

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PART VI - OPERATING LIFECYCLE

FROM DISCOVERY THROUGH RETIREMENT

01 / Discover and classify demand

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

02 / Define service boundary and owners

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

03 / Model architecture and acceptance criteria

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

04 / Build isolated proof and challenge assumptions

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

05 / Pilot with stop conditions and rollback

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

06 / Scale through standard profiles and automation

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

07 / Operate, reassess, migrate, and retire

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

SECURITY, PRIVACY, SAFETY, AND RIGHTS

CROSS-CUTTING ASSURANCE

SECURITY

Threat model identities, trust boundaries, supply chain, control planes, data, protocols, privilege, abuse, detection, response, recovery, and residual risk.

PRIVACY

Purpose limitation, minimization, transparency, consent where required, access, correction, retention, deletion, sovereignty, metadata, and third-party processing.

SAFETY

Hazards, unsafe states, human intervention, physical consequence, dependency failure, automation limits, emergency stop, recovery sequence, and validation.

RIGHTS AND USABILITY

Accessibility, contestability, explanation, operator workload, affected-user communication, grievance, misuse prevention, and equitable performance.

SUPPLY CHAIN

Provenance, version, SBOM or component inventory, signature, vulnerability, support, update, provider, subcontractor, and exit.

EVIDENCE

Design decisions, tests, telemetry, incidents, approvals, exceptions, recovery exercises, source currency, and review findings remain linked to the architecture version.

FAILURE MODES AND RESILIENCE

DESIGN FOR SAFE DEGRADATION AND TRUSTWORTHY RECOVERY

FAILURE SCENARIO 01

Control-plane compromise or excessive privilege.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 02

Identity, policy, data, or configuration state becomes stale or inconsistent.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 03

A shared dependency creates correlated failure across tenants or domains.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 04

Automation succeeds technically but violates intended service, safety, or privacy outcomes.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 05

Recovery restores components without restoring trustworthy service state.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 06

Exit, migration, revocation, or deletion cannot be completed within the required time.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

PART VII - INTEGRATION AND DEPENDENCY MODEL

CONNECTED REFERENCE ARCHITECTURES AND STANDARDS

REFERENCE ARCHITECTURE DEPENDENCIES

- No mandatory architecture dependency. Interfaces to adjacent domains remain governed through explicit contracts.

MAPPED MYTHOS STANDARDS

- MYTHOS-EMT-0002
- MYTHOS-QTC-0004
- MYTHOS-HW-0002
- MYTHOS-ID-0003

DEPENDENCY DOCTRINE

A dependency is not a blanket trust grant. Each connection requires an interface, identity, policy, data, availability, version, failure, recovery, and evidence contract. Correlated failure and circular authority must be analyzed across the full architecture graph.

PART VIII - IMPLEMENTATION AND ADOPTION ROADMAP

PHASED, EVIDENCE-DRIVEN TRANSITION

0-30 DAYS	Inventory purpose, owners, users, systems, data, dependencies, risks, current diagrams, incidents, recovery, and evidence gaps.
31-90 DAYS	Establish authoritative identity, configuration, policy, data, source, and architecture records; define target profile and acceptance tests.
3-6 MONTHS	Implement bounded foundations, isolation, telemetry, change control, backup, recovery, and representative pilot workflows.
6-12 MONTHS	Expand through standard patterns, automation, service ownership, SLOs, independent assessment, failure injection, and supplier-exit exercises.
ONGOING	Reconcile drift, review sources, reassess risk, measure outcomes, exercise recovery, renew exceptions, migrate dependencies, and retire obsolete paths.

ARCHITECTURE ACCEPTANCE CRITERIA - A

CRITERIA 01-09

AC-01	The architecture has a named service owner, technical owner, security owner, data owner, and recovery authority.
AC-02	The system boundary, external dependencies, retained responsibilities, and trust transitions are documented and reviewed.
AC-03	Identity, authorization, secrets, policy, and privileged operations are explicit and independently auditable.
AC-04	Desired state, runtime state, observed state, and evidence state are distinguishable and reconciled.
AC-05	Consequential changes require preconditions, approval policy, bounded execution, verification, and rollback.
AC-06	Telemetry supports service health, security detection, forensic reconstruction, and conformance evidence.
AC-07	Failure modes include safe degradation, dependency loss, corrupted state, operator error, and malicious action.
AC-08	Backup, restoration, reconstitution, and exit procedures are exercised against defined recovery objectives.
AC-09	Data classification, retention, privacy, sovereignty, and disposition controls follow the actual information flow.

ARCHITECTURE ACCEPTANCE CRITERIA - B

CRITERIA 10-18

AC-10	Supplier, model, protocol, platform, and software dependencies have provenance, version, support, and replacement records.
AC-11	Representative performance and capacity tests include saturation, contention, latency, and degraded conditions.
AC-12	Exceptions are time-bounded, visible, approved by accountable risk owners, and linked to compensating controls.
AC-13	Architecture diagrams and inventories remain synchronized with deployed components and interfaces.
AC-14	Mandatory trust, safety, privacy, recovery, and evidence gates cannot be bypassed by a lower assurance profile.
AC-15	External dependencies have contractual, technical, operational, data, and exit boundaries.
AC-16	Representative acceptance tests exercise normal, degraded, adversarial, recovery, and retirement scenarios.
AC-17	The architecture can explain why an action, decision, deployment, or access was permitted and what evidence supports it.
AC-18	A formal decision record names selected and rejected alternatives, residual risk, review date, and change triggers.

METRICS, EVIDENCE, AND DECISION GATES

MEASURE SERVICE OUTCOMES AND ASSURANCE

COVERAGE

Percent of components, interfaces, identities, data flows, dependencies, risks, and controls represented in current architecture evidence.

CHANGE QUALITY

Plan success, rollback, verification pass, escaped defect, drift, and emergency-change rates.

RELIABILITY

Availability, latency, saturation, dependency health, recovery time, recovery point, and safe-degradation success.

SECURITY

Privilege exposure, policy denial, identity compromise, vulnerable dependency, detection coverage, response, and evidence completeness.

HUMAN OUTCOME

Task success, operator workload, accessibility, override, contestability, training, and user-impact measures.

LIFECYCLE

Version currency, unsupported component, exception age, migration readiness, restore exercise, and retirement completeness.

Release or expansion gate: mandatory acceptance criteria pass; failed safety, privacy, identity, recovery, evidence, or legal gates block promotion regardless of aggregate score.

SOURCE AUTHORITY REGISTER

CURRENT PRIMARY AND OFFICIAL REFERENCES

SOURCE 01 **NIST Post-Quantum Cryptography Project**

https://csrc.nist.gov/projects/post-quantum-cryptography

SOURCE 02 **NIST SP 800-53 Rev. 5**

https://doi.org/10.6028/NIST.SP.800-53r5

SOURCE 03 **SLSA Specification 1.2**

https://slsa.dev/spec/v1.2/

SOURCE 04 **NIST SP 800-207, Zero Trust Architecture**

https://csrc.nist.gov/pubs/sp/800/207/final

Source currency note. Official frameworks, specifications, and implementation guidance evolve. Assessors must verify the current version, status, errata, applicability, and effective date at adoption and scheduled review. NIST AI RMF 1.0 was under revision at the publication date.

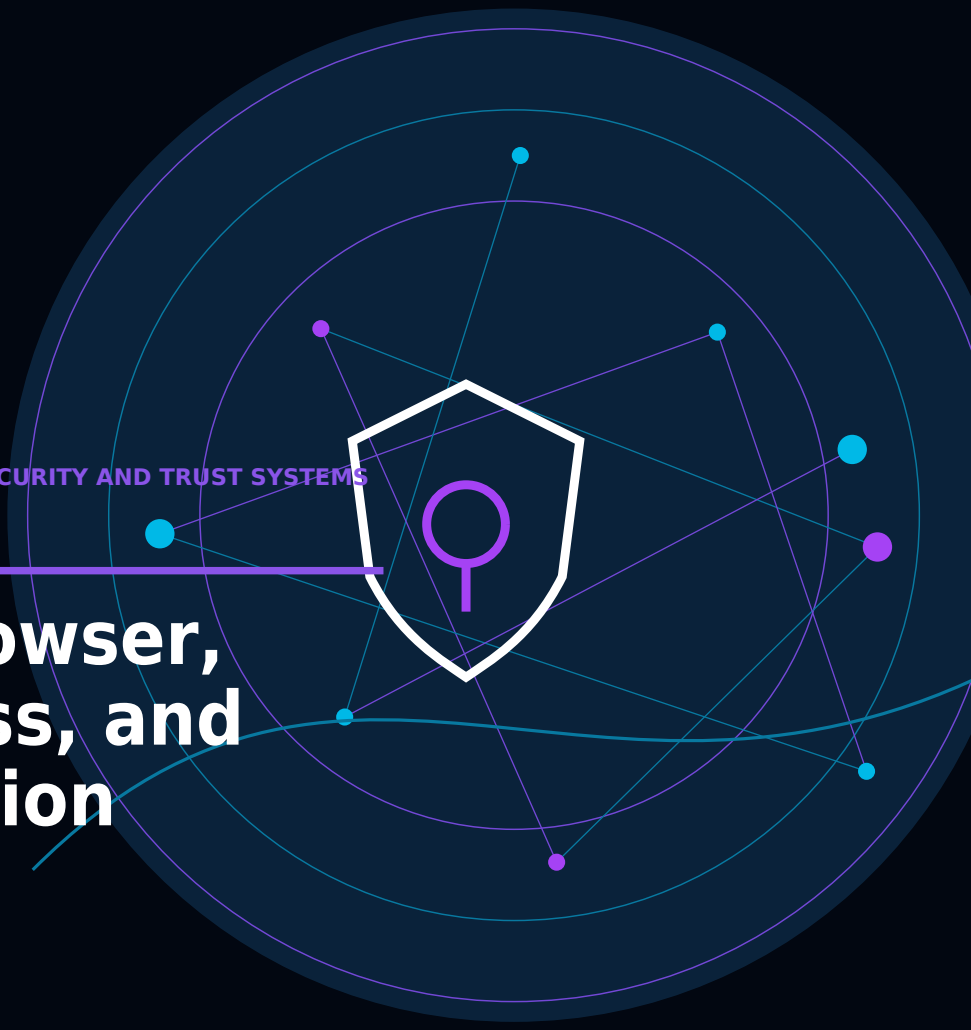
PUBLICATION CONTROL AND REVISION

PERMANENT IDENTITY / CONTROLLED EVOLUTION

PERMANENT DOCUMENT IDENTITY	RA-017
VERSION	1.0.0-candidate
PUBLICATION DATE	2026-07-24
SCHEDULED REVIEW	2027-01-24
STATUS	Candidate Reference Architecture
ARCHITECTURE SET	Future Domain Set
CLASSIFICATION	Public
PUBLISHER	Mythos Systems

REVISION RECORD

1.0.0-candidate (2026-07-24) - Permanent-publication candidate edition regenerated under the Mythos Systems architecture publication system. Production sequence metadata is excluded from public identity.



ENGINEERING STANDARDS LIBRARY / SECURITY AND TRUST SYSTEMS

Sovereign Browser, Remote Access, and Web Automation

A sovereign browser and remote-interaction architecture for isolated browsing, credential protection, remote access, web automation, session evidence, data boundaries, and controlled tool execution.

PERMANENT DOCUMENT ID

RA-019

PUBLICATION

Candidate Reference Architecture
Version 1.0.0-candidate

ASSURANCE PROFILE

Candidate Architecture
High Assurance

PERMANENT PUBLICATION IDENTITY

Sovereign Browser, Remote Access, and Web Automation

A sovereign browser and remote-interaction architecture for isolated browsing, credential protection, remote access, web automation, session evidence, data boundaries, and controlled tool execution.

DOCUMENT ID
RA-019

VERSION
1.0.0-candidate

STATUS
Candidate Reference Architecture

ARCHITECTURE SET
Future Domain Set

DOMAIN
Cybersecurity and Network Security

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

SCHEDULED REVIEW
2027-01-24

CLASSIFICATION
Public

6	6	18	0
ARCHITECTURE LAYERS	CORE COMPONENTS	ACCEPTANCE CRITERIA	DEPENDENCIES

Architecture position. This future-domain candidate defines a stable architectural destination and assurance model. Product choices, scale, regulatory obligations, and implementation details remain environment-specific.

REFERENCE ARCHITECTURE TOPOLOGY

CYBERSECURITY AND NETWORK SECURITY

This future-domain candidate defines a stable architectural destination and assurance model. Product choices, scale, regulatory obligations, and implementation details remain environment-specific.

01 Browser and workspace isolation

02 Identity, credentials, and session authority

03 Network and remote-access plane

04 Automation and agent tool plane

05 Data, download, upload, and clipboard policy

06 Observation, evidence, recovery, and privacy

GOVERNED ARCHITECTURE FLOW

INTERPRETATION AND ASSURANCE

HOW TO APPLY THIS ARCHITECTURE

INFORMATIVE ARCHITECTURE

The architecture describes a deployable pattern and assurance model. Normative obligations remain in mapped standards, policy, law, contracts, and approved design decisions.

EVIDENCE OVER DIAGRAM

A diagram is not implementation evidence. Conformance requires inventories, configurations, tests, telemetry, incidents, recovery results, and accountable decisions.

PROFILE WITHOUT DILUTION

Profiles may change scale and technology, but cannot remove mandatory trust, safety, privacy, recovery, or evidence outcomes.

PRODUCT-NEUTRAL CORE

Named technologies are illustrative unless an organization explicitly adopts them. Interfaces, outcomes, and evidence remain primary.

BOUNDED CLAIMS

Performance, security, autonomy, availability, privacy, and compliance claims apply only to tested configurations and conditions.

CONTROLLED EVOLUTION

Architecture change preserves version, migration, rollback, source currency, exceptions, and residual-risk records.

Adoption requires environment-specific design, threat and hazard analysis, capacity engineering, implementation, verification, approval, operation, and periodic reassessment.

INTERACTIVE NAVIGATION

REFERENCE ARCHITECTURE CONTENTS

PART I - EXECUTIVE DIRECTION

OUTCOMES, RISKS, AND DECISIONS

ARCHITECTURE MANDATE

A sovereign browser and remote-interaction architecture for isolated browsing, credential protection, remote access, web automation, session evidence, data boundaries, and controlled tool execution.

EXECUTIVE OUTCOMES

- Establish a deployable, product-neutral target architecture for sovereign browser, remote access, and web automation.
- Make ownership, trust boundaries, data and control flows, failure handling, and evidence explicit.
- Support multiple implementation profiles without weakening mandatory assurance outcomes.
- Provide a governed adoption path from discovery through production operation and retirement.

DECISIONS REQUIRED

- Accountable service and risk ownership.
- Accepted architecture profile and consequence class.
- Authoritative identity, data, policy, configuration, and evidence systems.
- Mandatory approval, recovery, privacy, safety, and supplier-exit gates.
- Funding for operations, observability, validation, and lifecycle—not only implementation.

PART II - SCOPE, PRINCIPLES, AND SYSTEM CONTEXT

CYBERSECURITY AND NETWORK SECURITY

IN SCOPE

A sovereign browser and remote-interaction architecture for isolated browsing, credential protection, remote access, web automation, session evidence, data boundaries, and controlled tool execution.

OUT OF SCOPE

Vendor selection, exact sizing, contractual obligations, detailed low-level design, product certification, regulatory approval, and proof that any named platform is implemented.

ARCHITECTURE INVARIANTS

Explicit ownership; least authority; separable desired, runtime, observed, and evidence state; bounded automation; tested recovery; product-neutral interfaces; source and claim currency.

ASSUMPTIONS

Organizations tailor the pattern through documented risk, environment, criticality, data, safety, regulatory, workforce, and supplier analysis.

SYSTEM CONTEXT AND RESPONSIBILITY MODEL

ACTORS, ENVIRONMENTS, AND EXTERNAL DEPENDENCIES

ACTORS AND RESPONSIBILITY

- Service consumers and affected users
- Product, platform, network, security, data, reliability, and support teams
- Human administrators, operators, reviewers, incident commanders, and risk owners
- Machine identities, agents, workloads, devices, controllers, and external services
- Suppliers, providers, carriers, integrators, assessors, and regulators where applicable

RESPONSIBILITY BOUNDARY

- The adopting organization retains accountability for purpose, risk, configuration, data, identity, operation, recovery, and evidence even where a provider performs underlying functions.
- Inherited controls are recorded with provider, scope, evidence, limitations, shared responsibility, incident path, and exit conditions.
- No component is considered trusted solely because it is internal, managed, signed, cloud-hosted, model-generated, or supplied by a recognized vendor.

ARCHITECTURE POSITION

This future-domain candidate defines a stable architectural destination and assurance model. Product choices, scale, regulatory obligations, and implementation details remain environment-specific.

PART III - LOGICAL ARCHITECTURE

LAYERS AND RESPONSIBILITIES

01

Browser and workspace isolation

Browser and workspace isolation owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

02

Identity, credentials, and session authority

Identity, credentials, and session authority owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

03

Network and remote-access plane

Network and remote-access plane owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

04

Automation and agent tool plane

Automation and agent tool plane owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

05

Data, download, upload, and clipboard policy

Data, download, upload, and clipboard policy owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

06

Observation, evidence, recovery, and privacy

Observation, evidence, recovery, and privacy owns a bounded set of responsibilities, interfaces, state, controls, failure behavior, telemetry, recovery procedures, and evidence. It does not silently absorb authority assigned to another layer.

CORE COMPONENTS - A

COMPONENTS 01-03

COMPONENT 01

Hardened browser profiles, containers/VMs, site isolation, extension policy, and updates

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 02

Passkeys, password vault, hardware keys, session broker, privilege, and step-up

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 03

ZTNA/VPN, proxy, DNS, egress, remote desktop/SSH, and destination policy

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

CORE COMPONENTS - B

COMPONENTS 04-06

COMPONENT 04

Browser automation, DOM/accessibility interfaces, tool broker, approvals, and anti-injection controls

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 05

Downloads, uploads, clipboard, printing, screenshots, secrets, DLP, and local storage

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

COMPONENT 06

Session logs, recordings where lawful, artifact capture, incident response, reset, and secure disposal

Required contract: owner, purpose, inputs, outputs, authority, data, dependencies, health, failure behavior, recovery, lifecycle, and evidence.

PART IV - DATA, CONTROL, AUTHORITY, AND EVIDENCE FLOW

FLOW CONTRACTS

1. Intent, service demand, policy, and environment constraints enter the architecture through explicit contracts.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

2. Authoritative identity, data, configuration, and dependency state are validated before execution or access.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

3. Control planes perform bounded orchestration while local enforcement preserves least privilege and safe behavior.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

4. Telemetry, tests, user outcomes, and incidents update evidence and trigger correction or rollback.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

5. Lifecycle governance manages versioning, capacity, exceptions, migration, and retirement.

   Evidence: request and actor identity, policy decision, versions, state transitions, outputs, tests, errors, timing, and accountable disposition.

TRUST BOUNDARIES AND ENFORCEMENT

EVERY CROSSING REQUIRES AN EXPLICIT CONTRACT

TRUST BOUNDARY 01

Human or machine actor versus authorization service

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 02

Control plane versus workload, device, or enforcement point

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 03

Tenant or project versus shared platform

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 04

Local environment versus external provider or network

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 05

Mutable runtime state versus authoritative evidence

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

TRUST BOUNDARY 06

Normal operation versus emergency or break-glass authority

Minimum contract: authenticated parties, authorized actions, data classification, protocol, encryption, validation, rate and resource limits, observability, failure behavior, revocation, and evidence.

PART V - DEPLOYMENT PROFILES

PROFILE CHANGES SCALE, NOT ASSURANCE OUTCOMES

PROFILE 01 / PILOT OR LABORATORY PROFILE

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PROFILE 02 / DEPARTMENT OR BOUNDED PRODUCTION PROFILE

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PROFILE 03 / ENTERPRISE FEDERATED PROFILE

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PROFILE 04 / HIGH-ASSURANCE SOVEREIGN OR REGULATED PROFILE

Use this profile to establish a bounded implementation with explicit scale, criticality, users, data, autonomy, external dependencies, recovery objectives, staffing, and validation depth. The profile may simplify technology or topology but must preserve identity, least authority, evidence, safe failure, recovery, lifecycle, and accountable risk acceptance.

PART VI - OPERATING LIFECYCLE

FROM DISCOVERY THROUGH RETIREMENT

01 / Discover and classify demand

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

02 / Define service boundary and owners

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

03 / Model architecture and acceptance criteria

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

04 / Build isolated proof and challenge assumptions

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

05 / Pilot with stop conditions and rollback

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

06 / Scale through standard profiles and automation

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

07 / Operate, reassess, migrate, and retire

Entry and exit criteria, owner, evidence, exceptions, dependencies, rollback, and next decision are recorded. No phase is complete solely because components were purchased, configured, or demonstrated.

SECURITY, PRIVACY, SAFETY, AND RIGHTS

CROSS-CUTTING ASSURANCE

SECURITY

Threat model identities, trust boundaries, supply chain, control planes, data, protocols, privilege, abuse, detection, response, recovery, and residual risk.

PRIVACY

Purpose limitation, minimization, transparency, consent where required, access, correction, retention, deletion, sovereignty, metadata, and third-party processing.

SAFETY

Hazards, unsafe states, human intervention, physical consequence, dependency failure, automation limits, emergency stop, recovery sequence, and validation.

RIGHTS AND USABILITY

Accessibility, contestability, explanation, operator workload, affected-user communication, grievance, misuse prevention, and equitable performance.

SUPPLY CHAIN

Provenance, version, SBOM or component inventory, signature, vulnerability, support, update, provider, subcontractor, and exit.

EVIDENCE

Design decisions, tests, telemetry, incidents, approvals, exceptions, recovery exercises, source currency, and review findings remain linked to the architecture version.

FAILURE MODES AND RESILIENCE

DESIGN FOR SAFE DEGRADATION AND TRUSTWORTHY RECOVERY

FAILURE SCENARIO 01

Control-plane compromise or excessive privilege.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 02

Identity, policy, data, or configuration state becomes stale or inconsistent.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 03

A shared dependency creates correlated failure across tenants or domains.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 04

Automation succeeds technically but violates intended service, safety, or privacy outcomes.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 05

Recovery restores components without restoring trustworthy service state.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

FAILURE SCENARIO 06

Exit, migration, revocation, or deletion cannot be completed within the required time.

Required response: detect, contain, preserve evidence, maintain or enter safe service, communicate, recover, verify, learn, and prevent recurrence.

PART VII - INTEGRATION AND DEPENDENCY MODEL

CONNECTED REFERENCE ARCHITECTURES AND STANDARDS

REFERENCE ARCHITECTURE DEPENDENCIES

- No mandatory architecture dependency. Interfaces to adjacent domains remain governed through explicit contracts.

MAPPED MYTHOS STANDARDS

- MYTHOS-WEB-0001
- MYTHOS-SEC-0003
- MYTHOS-ID-0001
- MYTHOS-END-0001

DEPENDENCY DOCTRINE

A dependency is not a blanket trust grant. Each connection requires an interface, identity, policy, data, availability, version, failure, recovery, and evidence contract. Correlated failure and circular authority must be analyzed across the full architecture graph.

PART VIII - IMPLEMENTATION AND ADOPTION ROADMAP

PHASED, EVIDENCE-DRIVEN TRANSITION

0-30 DAYS	Inventory purpose, owners, users, systems, data, dependencies, risks, current diagrams, incidents, recovery, and evidence gaps.
31-90 DAYS	Establish authoritative identity, configuration, policy, data, source, and architecture records; define target profile and acceptance tests.
3-6 MONTHS	Implement bounded foundations, isolation, telemetry, change control, backup, recovery, and representative pilot workflows.
6-12 MONTHS	Expand through standard patterns, automation, service ownership, SLOs, independent assessment, failure injection, and supplier-exit exercises.
ONGOING	Reconcile drift, review sources, reassess risk, measure outcomes, exercise recovery, renew exceptions, migrate dependencies, and retire obsolete paths.

ARCHITECTURE ACCEPTANCE CRITERIA - A

CRITERIA 01-09

AC-01	The architecture has a named service owner, technical owner, security owner, data owner, and recovery authority.
AC-02	The system boundary, external dependencies, retained responsibilities, and trust transitions are documented and reviewed.
AC-03	Identity, authorization, secrets, policy, and privileged operations are explicit and independently auditable.
AC-04	Desired state, runtime state, observed state, and evidence state are distinguishable and reconciled.
AC-05	Consequential changes require preconditions, approval policy, bounded execution, verification, and rollback.
AC-06	Telemetry supports service health, security detection, forensic reconstruction, and conformance evidence.
AC-07	Failure modes include safe degradation, dependency loss, corrupted state, operator error, and malicious action.
AC-08	Backup, restoration, reconstitution, and exit procedures are exercised against defined recovery objectives.
AC-09	Data classification, retention, privacy, sovereignty, and disposition controls follow the actual information flow.

ARCHITECTURE ACCEPTANCE CRITERIA - B

CRITERIA 10-18

AC-10	Supplier, model, protocol, platform, and software dependencies have provenance, version, support, and replacement records.
AC-11	Representative performance and capacity tests include saturation, contention, latency, and degraded conditions.
AC-12	Exceptions are time-bounded, visible, approved by accountable risk owners, and linked to compensating controls.
AC-13	Architecture diagrams and inventories remain synchronized with deployed components and interfaces.
AC-14	Mandatory trust, safety, privacy, recovery, and evidence gates cannot be bypassed by a lower assurance profile.
AC-15	External dependencies have contractual, technical, operational, data, and exit boundaries.
AC-16	Representative acceptance tests exercise normal, degraded, adversarial, recovery, and retirement scenarios.
AC-17	The architecture can explain why an action, decision, deployment, or access was permitted and what evidence supports it.
AC-18	A formal decision record names selected and rejected alternatives, residual risk, review date, and change triggers.

METRICS, EVIDENCE, AND DECISION GATES

MEASURE SERVICE OUTCOMES AND ASSURANCE

COVERAGE

Percent of components, interfaces, identities, data flows, dependencies, risks, and controls represented in current architecture evidence.

CHANGE QUALITY

Plan success, rollback, verification pass, escaped defect, drift, and emergency-change rates.

RELIABILITY

Availability, latency, saturation, dependency health, recovery time, recovery point, and safe-degradation success.

SECURITY

Privilege exposure, policy denial, identity compromise, vulnerable dependency, detection coverage, response, and evidence completeness.

HUMAN OUTCOME

Task success, operator workload, accessibility, override, contestability, training, and user-impact measures.

LIFECYCLE

Version currency, unsupported component, exception age, migration readiness, restore exercise, and retirement completeness.

Release or expansion gate: mandatory acceptance criteria pass; failed safety, privacy, identity, recovery, evidence, or legal gates block promotion regardless of aggregate score.

SOURCE AUTHORITY REGISTER

CURRENT PRIMARY AND OFFICIAL REFERENCES

SOURCE 01 **NIST SP 800-207, Zero Trust Architecture**

https://csrc.nist.gov/pubs/sp/800/207/final

SOURCE 02 **NIST SP 1800-35, Implementing a Zero Trust Architecture**

https://www.nccoe.nist.gov/projects/implementing-zero-trust-architecture

SOURCE 03 **W3C Web Authentication Level 3**

https://www.w3.org/TR/webauthn-3/

SOURCE 04 **CISA Secure by Design**

https://www.cisa.gov/resources-tools/resources/secure-by-design

Source currency note. Official frameworks, specifications, and implementation guidance evolve. Assessors must verify the current version, status, errata, applicability, and effective date at adoption and scheduled review. NIST AI RMF 1.0 was under revision at the publication date.

PUBLICATION CONTROL AND REVISION

PERMANENT IDENTITY / CONTROLLED EVOLUTION

PERMANENT DOCUMENT IDENTITY	RA-019
VERSION	1.0.0-candidate
PUBLICATION DATE	2026-07-24
SCHEDULED REVIEW	2027-01-24
STATUS	Candidate Reference Architecture
ARCHITECTURE SET	Future Domain Set
CLASSIFICATION	Public
PUBLISHER	Mythos Systems

REVISION RECORD

1.0.0-candidate (2026-07-24) - Permanent-publication candidate edition regenerated under the Mythos Systems architecture publication system. Production sequence metadata is excluded from public identity.