CANONICAL LIVING OVERVIEW GUIDE

OpenTelemetry, Semantic Conventions, and Observability Pipelines

An observability-engineering guide covering telemetry design, resources, traces, metrics, logs, events, baggage, semantic conventions, OTLP, collector pipelines, sampling, enrichment, routing, storage, privacy, reliability, and governance.

PERMANENT ID
MYTHOS-FG-DAT-0004

PUBLICATION CLASS
FIELD GUIDE

VERSION / STATUS
1.0.0-candidate / CANDIDATE

PERMANENT PUBLICATION IDENTITY

OpenTelemetry, Semantic Conventions, and Observability Pipelines

Document ID
MYTHOS-FG-DAT-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-DAT - Data, State, Storage, Reliability & Evidence
Secondary domain	MYTHOS-CLD; MYTHOS-SWE; MYTHOS-SEC
Audience	Software, platform, SRE, security, data, network, and AI observability engineers.
Prerequisites	Distributed systems, application instrumentation, networking, schemas, data pipelines, and operations.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

Design telemetry from operational and assurance questions rather than indiscriminate collection.

Use stable semantic conventions, resources, identifiers, units, and schemas across teams.

Engineer resilient collector pipelines with queues, backpressure, routing, privacy, and failure handling.

Correlate traces, metrics, logs, events, profiles, artifacts, and user outcomes.

Govern telemetry cost, cardinality, retention, access, evolution, and evidence quality.

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

Observability strategy and telemetry contracts

Define the questions, decisions, and evidence the system must support.

Chapter operating position
Define the questions, decisions, and evidence the system must support.

1.1 Operational questions

Operational questions must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Support availability, latency, errors, saturation, dependencies, releases, security, cost, and user 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 questions.
- 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 operational questions 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 questions.
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 Signal and event model

Signal and event model must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Choose traces, metrics, logs, events, profiles, exemplars, and artifacts according to use. 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 signal and event model.
- 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

```
telemetry_contract:
  resource: service.name
  operation: order.process
  trace:
    required_attributes: [service.name, deployment.environment.name, enduser.tenant.id]
  metrics:
    - name: order.processing.duration
      unit: s
      type: histogram
  logs:
    event_name: order.processing.failed
  privacy:
    prohibited_attributes: [customer.email, payment.token]
  schema_version: semconv-1.43.0-plus-mythos-1
```

Failure patterns

- Treating signal and event model 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 signal and event model.
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 Telemetry contract

Telemetry contract must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Specify names, attributes, units, identity, timestamps, privacy, sampling, retention, and owners. 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 telemetry contract.

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 telemetry contract 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 telemetry contract.
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 Source and evidence quality

Source and evidence quality must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Record instrumentation version, confidence, gaps, collection path, transformation, and 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 source and evidence quality.
- 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 source and evidence quality 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 source and evidence quality.
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 telemetry contract for one critical user journey and reject fields that cannot support a decision.

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 observability schemas generated from executable architecture and state models.

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

Resources, identity, and correlation

Attach every signal to stable entities and causal context.

Chapter operating position
Attach every signal to stable entities and causal context.

2.1 Resource model

Resource model must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Describe service, instance, host, container, pod, cloud, device, process, deployment, and tenant. 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 resource model.
- 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 resource model 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 resource model.
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 Trace and span context

Trace and span context must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Propagate trace IDs, span IDs, flags, state, and links across synchronous and asynchronous 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 trace and span context.
- 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

```
telemetry_contract:
  resource: service.name
  operation: order.process
  trace:
    required_attributes: [service.name, deployment.environment.name, enduser.tenant.id]
  metrics:
    - name: order.processing.duration
      unit: s
      type: histogram
  logs:
    event_name: order.processing.failed
  privacy:
    prohibited_attributes: [customer.email, payment.token]
  schema_version: semconv-1.43.0-plus-mythos-1
```

Failure patterns

- Treating trace and span context 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 trace and span context.
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 Baggage and business context

Baggage and business context must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Carry limited correlation attributes without treating baggage as secure authorization. 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 baggage and business context.

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 baggage and business context 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 baggage and business context.
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 Clock and timestamp quality

Clock and timestamp quality must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Handle wall clocks, monotonic time, skew, precision, event time, receive time, and ordering. 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 clock and timestamp quality.
- 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 clock and timestamp quality 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 clock and timestamp quality.
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
Trace a transaction across HTTP, queue, worker, database, and external API while preserving entity identity.

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 signed context propagation for high-assurance distributed 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

Traces and distributed causality

Represent execution paths, dependencies, concurrency, and failure.

Chapter operating position
Represent execution paths, dependencies, concurrency, and failure.

3.1 Span boundaries and naming

Span boundaries and naming must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Model server, client, producer, consumer, internal, long-running, and batch operations. 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 span boundaries and naming.
- 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 span boundaries and naming 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 span boundaries and naming.
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 Attributes, events, links, and status

Attributes, events, links, and status must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Capture stable dimensions, important state changes, fan-out, retries, errors, 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 attributes, events, links, and status.
- 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

```
telemetry_contract:
  resource: service.name
  operation: order.process
  trace:
    required_attributes: [service.name, deployment.environment.name, enduser.tenant.id]
  metrics:
    - name: order.processing.duration
      unit: s
      type: histogram
  logs:
    event_name: order.processing.failed
  privacy:
    prohibited_attributes: [customer.email, payment.token]
  schema_version: semconv-1.43.0-plus-mythos-1
```

Failure patterns

- Treating attributes, events, links, and status 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 attributes, events, links, and status.
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 Asynchronous and batch tracing

Asynchronous and batch tracing must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Link messages, workflows, streams, jobs, agents, and delayed processing without false parentage. 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 asynchronous and batch tracing.
- 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 asynchronous and batch tracing 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 asynchronous and batch tracing.
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 Sampling and completeness

Sampling and completeness must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Use head, tail, priority, probabilistic, and rule-based sampling with known evidence 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 sampling and completeness.
- 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 sampling and completeness 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 sampling and completeness.
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
Instrument a fan-out workflow and reconstruct critical path, retries, and partial failure from spans and links.

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 graph-native trace analysis for agent, data, and infrastructure 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 04

Metrics, cardinality, and SLO telemetry

Measure populations and time without creating unbounded dimensionality.

Chapter operating position
Measure populations and time without creating unbounded dimensionality.

4.1 Instrument selection

Instrument selection must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Use counters, histograms, gauges, observable instruments, and events 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 instrument selection.
- 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 instrument selection 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 instrument selection.
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 Attributes and cardinality

Attributes and cardinality must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Budget dimensions, prevent unique identifiers, aggregate intentionally, and preserve drill-down through exemplars. 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 attributes and cardinality.
- 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

```
telemetry_contract:
  resource: service.name
  operation: order.process
  trace:
    required_attributes: [service.name, deployment.environment.name, enduser.tenant.id]
  metrics:
    - name: order.processing.duration
      unit: s
      type: histogram
  logs:
    event_name: order.processing.failed
  privacy:
    prohibited_attributes: [customer.email, payment.token]
  schema_version: semconv-1.43.0-plus-mythos-1
```

Failure patterns

- Treating attributes and cardinality 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 attributes and cardinality.
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 Histograms and distributions

Histograms and distributions must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Choose buckets or exponential histograms, units, aggregation temporality, and percentiles. 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 histograms and distributions.

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 histograms and distributions 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 histograms and distributions.
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 SLIs and derived metrics

SLIs and derived metrics must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Build availability, latency, correctness, freshness, and durability indicators from reliable sources. 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 slis and derived metrics.
- 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 slis and derived metrics 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 slis and derived metrics.
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
Design a metric schema and test cardinality under tenants, endpoints, errors, versions, and unique IDs.

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 negotiated telemetry detail based on incident and error-budget 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 05

Logs, events, and structured diagnostics

Make discrete records searchable, correlated, and semantically stable.

Chapter operating position
Make discrete records searchable, correlated, and semantically stable.

5.1 Structured log records

Structured log records must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Use timestamp, severity, body, event name, resource, scope, trace context, and typed attributes. 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 structured log records.
- 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 structured log records 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 structured log records.
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 Event semantic conventions

Event semantic conventions must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Define named events, schemas, units, lifecycle, and development or stable status. 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 event semantic conventions.
- 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

```
telemetry_contract:
  resource: service.name
  operation: order.process
  trace:
    required_attributes: [service.name, deployment.environment.name, enduser.tenant.id]
  metrics:
    - name: order.processing.duration
      unit: s
      type: histogram
  logs:
    event_name: order.processing.failed
  privacy:
    prohibited_attributes: [customer.email, payment.token]
  schema_version: semconv-1.43.0-plus-mythos-1
```

Failure patterns

- Treating event semantic conventions 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 event semantic conventions.
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 Security and audit records

Security and audit records must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Separate operational logs from privileged, tamper-evident, retention-controlled 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 security and audit records.

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 security and audit records 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 security and audit records.
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 Diagnostic context

Diagnostic context must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Capture errors, stack, configuration, decisions, external state, and remediation without secrets. 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 diagnostic context.
- 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 diagnostic context 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 diagnostic context.
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
Convert unstructured logs into typed events and prove trace correlation and privacy redaction.

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 schema-validated event generation compiled into application code and collectors.

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

OTLP and collector pipeline engineering

Move, process, and route signals reliably across environments.

Chapter operating position
Move, process, and route signals reliably across environments.

6.1 Receivers and exporters

Receivers and exporters must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Use OTLP, agents, gateways, files, protocols, vendor endpoints, authentication, and TLS. 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 receivers and exporters.
- 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 receivers and exporters 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 receivers and exporters.
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 Processors and connectors

Processors and connectors must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Batch, memory-limit, filter, transform, enrich, sample, route, redact, and derive signals. 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 processors and connectors.
- 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

```
telemetry_contract:
  resource: service.name
  operation: order.process
  trace:
    required_attributes: [service.name, deployment.environment.name, enduser.tenant.id]
  metrics:
    - name: order.processing.duration
      unit: s
      type: histogram
  logs:
    event_name: order.processing.failed
  privacy:
    prohibited_attributes: [customer.email, payment.token]
  schema_version: semconv-1.43.0-plus-mythos-1
```

Failure patterns

- Treating processors and connectors 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 processors and connectors.
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 Queues, retries, and backpressure

Queues, retries, and backpressure must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Bound memory and disk, handle partial success, retryable failure, overload, and 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 queues, retries, and backpressure.

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 queues, retries, and backpressure 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 queues, retries, and backpressure.
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 Deployment patterns

Deployment patterns must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Compare SDK direct export, node agents, sidecars, gateways, regional tiers, edge, and sovereign collectors. 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 deployment patterns.
- 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 deployment patterns 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 deployment patterns.
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
Build a two-tier collector pipeline and inject exporter outage, malformed data, overload, and regional disconnection.

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 telemetry pipelines whose transforms and losses are machine-auditable.

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

Storage, query, retention, and cost

Deliver useful analysis while controlling scale and access.

Chapter operating position
Deliver useful analysis while controlling scale and access.

7.1 Backend selection

Backend selection must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Match trace, metric, log, profile, event, and evidence stores to query and retention. 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 backend selection.
- 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 backend selection 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 backend selection.
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 Indexing and query

Indexing and query must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Design fields, labels, search, joins, exemplars, service maps, and cross-signal correlation. 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 indexing and query.
- 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

```
telemetry_contract:
  resource: service.name
  operation: order.process
  trace:
    required_attributes: [service.name, deployment.environment.name, enduser.tenant.id]
  metrics:
    - name: order.processing.duration
      unit: s
      type: histogram
  logs:
    event_name: order.processing.failed
  privacy:
    prohibited_attributes: [customer.email, payment.token]
  schema_version: semconv-1.43.0-plus-mythos-1
```

Failure patterns

- Treating indexing and query 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 indexing and query.
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 Retention and tiering

Retention and tiering must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Keep high-resolution, aggregate, incident, audit, and archival data according to 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 tiering.

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 tiering 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 tiering.
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 Cost and capacity

Cost and capacity must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Measure ingest, egress, storage, query, cardinality, sampling, and operator effort. 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 cost and capacity.
- 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 cost and capacity 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 cost and capacity.
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
Create a retention and sampling plan that meets SLO, security, debugging, and cost requirements.

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 federated query across local, regional, cloud, and evidence stores.

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

Governance, privacy, quality, and evolution

Keep telemetry trustworthy as systems and conventions change.

Chapter operating position
Keep telemetry trustworthy as systems and conventions change.

8.1 Semantic-convention lifecycle

Semantic-convention lifecycle must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Track stable, release-candidate, development, deprecated, and custom attributes explicitly. 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 semantic-convention 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.

Failure patterns

- Treating semantic-convention 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 semantic-convention 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.

8.2 Privacy and content policy

Privacy and content policy must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Control personal data, prompts, code, payloads, secrets, tenant content, and user consent. 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 content 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.

Implementation sketch

```
telemetry_contract:
  resource: service.name
  operation: order.process
  trace:
    required_attributes: [service.name, deployment.environment.name, enduser.tenant.id]
  metrics:
    - name: order.processing.duration
      unit: s
      type: histogram
  logs:
    event_name: order.processing.failed
  privacy:
    prohibited_attributes: [customer.email, payment.token]
  schema_version: semconv-1.43.0-plus-mythos-1
```

Failure patterns

- Treating privacy and content 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 privacy and content 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.

8.3 Quality and conformance

Quality and conformance must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Validate schemas, units, required fields, cardinality, timestamps, loss, and source coverage. 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 quality and conformance.

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 quality and conformance 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 quality and conformance.
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 Change and migration

Change and migration must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Version instrumentation, collectors, conventions, backends, dashboards, alerts, and retained 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 instrumented operation, resource, context, signal, collector, processing, export, storage, query, and operational decision 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 change 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 change 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 change 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 08 laboratory and review

Practical laboratory
Create a semantic-convention conformance gate and migrate one telemetry namespace without breaking SLOs.

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 an observability control plane that reconciles instrumentation, pipelines, evidence, and operational questions.

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. OpenTelemetry - OpenTelemetry
Role: Vendor-neutral traces, metrics, logs, baggage, APIs, SDKs, and collectors.
Verified: 2026-07-26
Official source: https://opentelemetry.io/

02. OpenTelemetry - Semantic Conventions 1.43.0
Role: Current common attribute, span, metric, log, event, and resource conventions.
Verified: 2026-07-26
Official source: https://opentelemetry.io/docs/specs/semconv/

03. OpenTelemetry - OpenTelemetry Protocol Specification
Role: OTLP transport, request, response, partial success, retry, and signal semantics.
Verified: 2026-07-26
Official source: https://opentelemetry.io/docs/specs/otlp/

04. OpenTelemetry - OpenTelemetry Collector
Role: Receivers, processors, exporters, connectors, pipelines, queues, and deployment patterns.
Verified: 2026-07-26
Official source: https://opentelemetry.io/docs/collector/

05. Google - Site Reliability Engineering
Role: SRE principles, SLOs, monitoring, toil, automation, release, and operations.
Verified: 2026-07-26
Official source: https://sre.google/sre-book/table-of-contents/

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-DAT - Data, State, Storage, Reliability & Evidence
Secondary domain	MYTHOS-CLD; MYTHOS-SWE; MYTHOS-SEC
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	OpenTelemetry, semantic conventions, OTLP, collector, traces, metrics, logs, observability pipeline, sampling, telemetry governance
Canonical route	/library/field-guides/mythos-fg-dat-0004/

Living publication

Corrections, expanded laboratories, improved diagrams, authority changes, and new engineering evidence should revise this permanent publication rather than create disconnected replacements.