CANONICAL LIVING OVERVIEW GUIDE

AI and Agent Observability, Evidence, and Evaluation Telemetry

A complete observability and evidence guide for models and agents covering semantic telemetry, traces, mission graphs, model calls, tokens, context, retrieval, memory, tools, approvals, artifacts, evaluations, costs, privacy, replay, and assurance.

PERMANENT ID
MYTHOS-FG-AI-0013

PUBLICATION CLASS
FIELD GUIDE

VERSION / STATUS
1.0.0-candidate / CANDIDATE

PERMANENT PUBLICATION IDENTITY

AI and Agent Observability, Evidence, and Evaluation Telemetry

Document ID
MYTHOS-FG-AI-0013

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-AI - Artificial Intelligence & Agentic Systems
Secondary domain	MYTHOS-DAT; MYTHOS-SEC; MYTHOS-CLD
Audience	AI platform, observability, evaluation, security, SRE, agent-runtime, and governance teams.
Prerequisites	OpenTelemetry, distributed tracing, logs, metrics, AI systems, evaluation, privacy, and distributed systems.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

Define a canonical event and trace model spanning missions, agents, models, context, tools, memory, and outcomes.

Correlate operational telemetry with evaluation evidence, user experience, cost, and safety.

Preserve reproducible trajectories without indiscriminately logging sensitive prompts and content.

Detect model, context, tool, agent, and orchestration failures through causal evidence.

Use telemetry for release gates, incident response, audit, replay, and continuous improvement.

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

How to use this guide 3

Contents 3

Chapter 01 - Observability architecture and objectives 5

Chapter 01 laboratory and review 10

Chapter 02 - Mission, agent, and harness tracing 11

Chapter 02 laboratory and review 16

Chapter 03 - Model-call and token telemetry 17

Chapter 03 laboratory and review 22

Chapter 04 - Context, retrieval, memory, and grounding evidence 23

Chapter 04 laboratory and review 28

Chapter 05 - Tool, capability, and external-effect telemetry 29

Chapter 05 laboratory and review 34

Chapter 06 - Evaluation and quality telemetry 35

Chapter 06 laboratory and review 40

Chapter 07 - Privacy, security, and evidence integrity 41

Chapter 07 laboratory and review 46

Chapter 08 - Operations, replay, and assurance 47

Chapter 08 laboratory and review 52

Integrated learning roadmap 53

Source authority register 54

Limitations, freshness, and revision control 55

CHAPTER 01

Observability architecture and objectives

Define which questions telemetry must answer and who may access it.

Chapter operating position
Define which questions telemetry must answer and who may access it.

1.1 Operational and assurance questions

Operational and assurance questions must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Support latency, reliability, cost, quality, safety, debugging, audit, and incident decisions. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for operational and assurance questions.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating operational and assurance questions as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for operational and assurance questions.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

1.2 Telemetry planes and stores

Telemetry planes and stores must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Separate high-cardinality traces, metrics, events, artifacts, evaluation records, and immutable evidence. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for telemetry planes and stores.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
agent_span:
  trace_id: mission-042
  span: verifier-7
  links: [implementer-3, test-suite-12]
  attributes:
    agent.role: verifier
    model.route: local-verifier
    task.state: running
    context.version: ctx-91
    tool.calls: 4
    gen_ai.usage.input_tokens: 18420
    gen_ai.usage.output_tokens: 2200
  evidence:
    - artifact: hidden-test-report
    - claim: candidate-safe-to-canary
```

Failure patterns

Treating telemetry planes and stores as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for telemetry planes and stores.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

1.3 Identity and correlation

Identity and correlation must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Assign mission, task, agent, harness, model, request, session, tool, user, tenant, and artifact IDs. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for identity and correlation.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating identity and correlation as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for identity and correlation.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

1.4 Ownership and access

Ownership and access must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Define platform, application, evaluator, security, privacy, support, and user visibility. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for ownership and access.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating ownership and access as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for ownership and access.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 01 laboratory and review

Practical laboratory
Create an observability requirements matrix for an agentic application and identify prohibited content collection.

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 user-owned evidence vaults and federated telemetry.

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

Mission, agent, and harness tracing

Represent distributed agent work as a causal execution graph.

Chapter operating position
Represent distributed agent work as a causal execution graph.

2.1 Mission and task spans

Mission and task spans must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Record objective, parent, dependencies, state, start, end, outcome, owner, and critical path. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for mission and task spans.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating mission and task spans as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for mission and task spans.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

2.2 Agent and harness spans

Agent and harness spans must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Record role, runtime, model route, context version, capability set, resource placement, and health. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for agent and harness spans.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
agent_span:
  trace_id: mission-042
  span: verifier-7
  links: [implementer-3, test-suite-12]
  attributes:
    agent.role: verifier
    model.route: local-verifier
    task.state: running
    context.version: ctx-91
    tool.calls: 4
    gen_ai.usage.input_tokens: 18420
    gen_ai.usage.output_tokens: 2200
  evidence:
    - artifact: hidden-test-report
    - claim: candidate-safe-to-canary
```

Failure patterns

- Treating agent and harness spans as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for agent and harness spans.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

2.3 Handoffs and links

Handoffs and links must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Use span links or explicit edges for fan-out, fan-in, messages, shared artifacts, and retries. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for handoffs and links.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating handoffs and links as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for handoffs and links.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

2.4 State-transition events

State-transition events must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Capture queued, admitted, running, waiting, approved, compensating, failed, and completed transitions. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for state-transition events.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating state-transition events as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for state-transition events.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 02 laboratory and review

Practical laboratory
Instrument a four-agent mission and reconstruct its critical path, fan-out, retries, and fan-in from telemetry.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore graph-native trace backends for extreme agent parallelism.

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

Model-call and token telemetry

Measure model behavior and resource use without confusing provider metrics with outcomes.

Chapter operating position
Measure model behavior and resource use without confusing provider metrics with outcomes.

3.1 Request configuration

Request configuration must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Record provider, model, version, parameters, context length, output limit, tools, response format, and cache. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for request configuration.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating request configuration as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for request configuration.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

3.2 Latency decomposition

Latency decomposition must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Measure queue, network, prefill, first token, decode, tool wait, postprocessing, and end-to-end time. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for latency decomposition.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
agent_span:
  trace_id: mission-042
  span: verifier-7
  links: [implementer-3, test-suite-12]
  attributes:
    agent.role: verifier
    model.route: local-verifier
    task.state: running
    context.version: ctx-91
    tool.calls: 4
    gen_ai.usage.input_tokens: 18420
    gen_ai.usage.output_tokens: 2200
  evidence:
    - artifact: hidden-test-report
    - claim: candidate-safe-to-canary
```

Failure patterns

- Treating latency decomposition as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for latency decomposition.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

3.3 Token and cost accounting

Token and cost accounting must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Track input, output, reasoning, cached, multimodal, embedding, provider cost, and internal chargeback. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for token and cost accounting.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating token and cost accounting as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for token and cost accounting.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

3.4 Output and finish state

Output and finish state must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Record finish reason, validation, refusal, truncation, repair, retries, and user-visible completion. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for output and finish state.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating output and finish state as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for output and finish state.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 03 laboratory and review

Practical laboratory
Compare two model routes using task success, latency decomposition, token use, retries, and cost.

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 hardware energy attribution and model-call carbon accounting.

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

Context, retrieval, memory, and grounding evidence

Explain what information influenced the model and whether it was authorized.

Chapter operating position
Explain what information influenced the model and whether it was authorized.

4.1 Context assembly

Context assembly must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Record source IDs, versions, authority, freshness, selection score, token contribution, and redaction. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for context assembly.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating context assembly as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for context assembly.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

4.2 Retrieval and reranking

Retrieval and reranking must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Capture query, filters, candidates, rankings, permissions, citations, and cache behavior. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for retrieval and reranking.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
agent_span:
  trace_id: mission-042
  span: verifier-7
  links: [implementer-3, test-suite-12]
  attributes:
    agent.role: verifier
    model.route: local-verifier
    task.state: running
    context.version: ctx-91
    tool.calls: 4
    gen_ai.usage.input_tokens: 18420
    gen_ai.usage.output_tokens: 2200
  evidence:
    - artifact: hidden-test-report
    - claim: candidate-safe-to-canary
```

Failure patterns

- Treating retrieval and reranking as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for retrieval and reranking.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

4.3 Memory access

Memory access must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Record memory tier, key, purpose, consent, read, write, correction, expiry, and deletion. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for memory access.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating memory access as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for memory access.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

4.4 Grounding and claim support

Grounding and claim support must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Link output claims to sources, calculations, tool evidence, confidence, conflicts, and abstention. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for grounding and claim support.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating grounding and claim support as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for grounding and claim support.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 04 laboratory and review

Practical laboratory
Trace one answer from query through retrieval, reranking, context packing, output claims, and citation validation.

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 privacy-preserving influence tracing and proof-carrying grounded generation.

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

Tool, capability, and external-effect telemetry

Make model-initiated actions observable and reconstructable.

Chapter operating position
Make model-initiated actions observable and reconstructable.

5.1 Discovery and selection

Discovery and selection must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Record available capabilities, descriptions, versions, policy filters, selected tool, and alternatives. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for discovery and selection.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating discovery and selection as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for discovery and selection.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

5.2 Authorization and approval

Authorization and approval must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Record identity, scopes, policy, risk, user confirmation, approver, and expiration. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for authorization and approval.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
agent_span:
  trace_id: mission-042
  span: verifier-7
  links: [implementer-3, test-suite-12]
  attributes:
    agent.role: verifier
    model.route: local-verifier
    task.state: running
    context.version: ctx-91
    tool.calls: 4
    gen_ai.usage.input_tokens: 18420
    gen_ai.usage.output_tokens: 2200
  evidence:
    - artifact: hidden-test-report
    - claim: candidate-safe-to-canary
```

Failure patterns

- Treating authorization and approval as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for authorization and approval.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

5.3 Invocation and result

Invocation and result must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Record typed arguments, hashes or protected values, progress, duration, result, error, and evidence. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for invocation and result.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating invocation and result as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for invocation and result.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

5.4 Side effects and rollback

Side effects and rollback must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Record state before, intended change, observed effect, compensation, residual change, and validation. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for side effects and rollback.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating side effects and rollback as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for side effects and rollback.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 05 laboratory and review

Practical laboratory

Instrument a canary deployment tool and prove exactly who authorized it, what changed, and how rollback was verified.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore cryptographically signed action receipts and effect ledgers.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 06

Evaluation and quality telemetry

Connect live behavior to tests, benchmarks, and release decisions.

Chapter operating position
Connect live behavior to tests, benchmarks, and release decisions.

6.1 Evaluation run identity

Evaluation run identity must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Record suite, cases, dataset, graders, model, system, environment, seed, version, and artifacts. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for evaluation run identity.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating evaluation run identity as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for evaluation run identity.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

6.2 Per-case and slice results

Per-case and slice results must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Capture task success, critical failures, safety, grounding, calibration, latency, and cost. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for per-case and slice results.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
agent_span:
  trace_id: mission-042
  span: verifier-7
  links: [implementer-3, test-suite-12]
  attributes:
    agent.role: verifier
    model.route: local-verifier
    task.state: running
    context.version: ctx-91
    tool.calls: 4
    gen_ai.usage.input_tokens: 18420
    gen_ai.usage.output_tokens: 2200
  evidence:
    - artifact: hidden-test-report
    - claim: candidate-safe-to-canary
```

Failure patterns

- Treating per-case and slice results as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for per-case and slice results.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

6.3 Judge and human evidence

Judge and human evidence must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Record rubric, judge model, expert, disagreement, confidence, comments, and adjudication. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for judge and human evidence.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating judge and human evidence as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for judge and human evidence.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

6.4 Production feedback loop

Production feedback loop must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Promote incidents, user corrections, failed trajectories, and drift into regression suites. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for production feedback loop.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating production feedback loop as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for production feedback loop.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 06 laboratory and review

Practical laboratory
Join production traces to an evaluation regression and identify the change that introduced a critical failure.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore continuously generated counterfactual evaluations from production trajectories.

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

Privacy, security, and evidence integrity

Preserve useful evidence without creating a surveillance or secret-leakage system.

Chapter operating position
Preserve useful evidence without creating a surveillance or secret-leakage system.

7.1 Content collection policy

Content collection policy must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Choose metadata-only, sampled, redacted, encrypted, consented, or prohibited prompt and response logging. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for content collection policy.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating content collection policy as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for content collection policy.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

7.2 Sensitive data controls

Sensitive data controls must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Protect credentials, personal data, proprietary code, retrieved documents, model weights, and hidden policies. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for sensitive data controls.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
agent_span:
  trace_id: mission-042
  span: verifier-7
  links: [implementer-3, test-suite-12]
  attributes:
    agent.role: verifier
    model.route: local-verifier
    task.state: running
    context.version: ctx-91
    tool.calls: 4
    gen_ai.usage.input_tokens: 18420
    gen_ai.usage.output_tokens: 2200
  evidence:
    - artifact: hidden-test-report
    - claim: candidate-safe-to-canary
```

Failure patterns

- Treating sensitive data controls as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for sensitive data controls.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

7.3 Integrity and provenance

Integrity and provenance must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Hash artifacts, sign releases, preserve parser and schema versions, control write access, and detect tampering. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for integrity and provenance.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating integrity and provenance as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for integrity and provenance.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

7.4 Retention and deletion

Retention and deletion must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Apply purpose, tenant, jurisdiction, incident hold, user request, expiry, and derived-data cleanup. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for retention and deletion.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating retention and deletion as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for retention and deletion.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 07 laboratory and review

Practical laboratory

Design a telemetry policy for regulated data and test redaction, tenant isolation, deletion, and incident preservation.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore confidential observability and zero-knowledge compliance proofs.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 08

Operations, replay, and assurance

Use telemetry to diagnose, reproduce, and govern system behavior.

Chapter operating position
Use telemetry to diagnose, reproduce, and govern system behavior.

8.1 Dashboards and SLOs

Dashboards and SLOs must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Monitor availability, task success, latency, tokens, cost, tool errors, safety, and user outcomes. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for dashboards and slos.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

- Treating dashboards and slos as a feature rather than an end-to-end stateful system.
- Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.
- Trusting controller, agent, or proxy status without validating external state and user outcome.
- Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.
- Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.
- Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for dashboards and slos.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

8.2 Trajectory replay

Trajectory replay must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Reconstruct model, context, tool, state, and environment with explicit nondeterminism and version limits. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

- Define authority, ownership, desired state, dependencies, limits, and evidence for trajectory replay.
- Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.
- Separate control plane, execution plane, data plane, evidence plane, and human decision authority.
- Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.
- Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.
- Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Implementation sketch

```
agent_span:
  trace_id: mission-042
  span: verifier-7
  links: [implementer-3, test-suite-12]
  attributes:
    agent.role: verifier
    model.route: local-verifier
    task.state: running
    context.version: ctx-91
    tool.calls: 4
    gen_ai.usage.input_tokens: 18420
    gen_ai.usage.output_tokens: 2200
  evidence:
    - artifact: hidden-test-report
    - claim: candidate-safe-to-canary
```

Failure patterns

- Treating trajectory replay as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for trajectory replay.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

8.3 Incident response

Incident response must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Query affected missions, models, tools, sources, users, tenants, effects, and artifacts for containment. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for incident response.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating incident response as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for incident response.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

8.4 Assurance and audit

Assurance and audit must be engineered as a governed distributed-system mechanism, not a product label, agent persona, YAML object, or diagram. Produce evidence bundles for release, control validation, exceptions, disputes, and external review. The design should identify authoritative state, workload identities, typed contracts, resource and token budgets, trust boundaries, failure domains, telemetry, rollback, and acceptance evidence.

This guide traces the subject through the mission, task, harness, model call, context, retrieval, memory, tool effect, evaluation, and user outcome chain. A running agent, reconciled object, successful API call, healthy proxy, or completed function is not proof of the intended outcome. Desired state must be compared with applied state, external effects, negative tests, degraded behavior, causal evidence, cost, and user or mission results.

Production engineering begins with a minimal known-good control plane and explicit invariants. Add one worker class, model route, controller, policy, event source, gateway, or platform feature at a time; preserve before-state evidence; test authorized and prohibited behavior; inject realistic failures; and retain a tested compensation or rollback path.

Troubleshooting locates the first divergence from the expected state machine. Account for stale watches, leases, retries, eventual consistency, duplicated work, cache state, identity rotation, partial external effects, version skew, prompt injection, overload, network partitions, and missing telemetry. Closure requires causal explanation, reproducibility, validated recovery, and residual limitations.

Engineering practices

Define authority, ownership, desired state, dependencies, limits, and evidence for assurance and audit.

Use typed APIs, stable identities, explicit state machines, idempotency, and bounded effects.

Separate control plane, execution plane, data plane, evidence plane, and human decision authority.

Test nominal, prohibited, overloaded, partitioned, partially failed, recovery, upgrade, and rollback cases.

Preserve configuration, policy, model, source, trace, event, artifact, approval, and user-outcome evidence.

Scale through quotas, queues, backpressure, fairness, isolation, canaries, and deterministic fan-in.

Failure patterns

Treating assurance and audit as a feature rather than an end-to-end stateful system.

Scaling worker count or deployment scope before bounding queues, budgets, retries, and fan-in.

Trusting controller, agent, or proxy status without validating external state and user outcome.

Sharing credentials, workspaces, caches, or context across tenants and roles without explicit policy.

Allowing AI-generated plans, policies, or changes to bypass deterministic validation and human authority.

Declaring recovery after process restart without reconciling orphan effects, leases, artifacts, and evidence.

Validation and evidence

Method	Expected evidence
Examine	Requirements, APIs, identities, state machines, policies, versions, resources, and dependencies for assurance and audit.
Observe	Queue, task, controller, model, tool, route, proxy, event, resource, trace, artifact, and user-visible state.
Test	Expected, prohibited, malformed, duplicate, overloaded, partitioned, failed, recovery, upgrade, and rollback conditions.
Measure	Task success, reliability, latency, throughput, fairness, cost, token and compute use, blast radius, and operator burden.
Reconcile	Desired and planned state against actual cloud, cluster, agent, network, data, external-effect, and evidence state.

Chapter 08 laboratory and review

Practical laboratory
Replay a failed agent mission using retained versions and identify which observations cannot be reproduced.

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 evidence operating system that treats every agentic outcome as a verifiable claim graph.

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 telemetry APIs, SDKs, collectors, traces, metrics, and logs.
Verified: 2026-07-26
Official source: https://opentelemetry.io/

02. OpenTelemetry - OpenTelemetry Semantic Conventions 1.43.0
Role: Current semantic-convention baseline for interoperable telemetry.
Verified: 2026-07-26
Official source: https://opentelemetry.io/docs/specs/semconv/

03. OpenTelemetry - Generative AI Observability with OpenTelemetry
Role: Current GenAI observability guidance for model calls, tokens, content controls, tool calls, and results.
Verified: 2026-07-26
Official source: https://opentelemetry.io/blog/2026/genai-observability/

04. NIST - Artificial Intelligence Risk Management Framework 1.0
Role: Govern, Map, Measure, and Manage framework for trustworthy AI risk.
Verified: 2026-07-26
Official source: https://nvlpubs.nist.gov/nistpubs/ai/nist.ai.100-1.pdf

05. NIST - NIST AI 600-1 - Generative Artificial Intelligence Profile
Role: Generative-AI risk profile and recommended actions.
Verified: 2026-07-26
Official source: https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.600-1.pdf

06. Model Context Protocol - Model Context Protocol Specification 2025-11-25
Role: Stable protocol baseline for tool, resource, prompt, transport, authorization, lifecycle, and capability integration.
Verified: 2026-07-26
Official source: https://modelcontextprotocol.io/specification/2025-11-25

07. Agent2Agent Protocol - Agent2Agent Protocol Specification 1.0
Role: Open protocol for agent discovery, communication, task delegation, status, artifacts, and interoperability.
Verified: 2026-07-26
Official source: https://a2a-protocol.org/v1.0.0/

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-AI - Artificial Intelligence & Agentic Systems
Secondary domain	MYTHOS-DAT; MYTHOS-SEC; MYTHOS-CLD
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	AI observability, agent telemetry, OpenTelemetry, evidence, trajectory tracing, model telemetry, tool calls, evaluation telemetry, cost, replay
Canonical route	/library/field-guides/mythos-fg-ai-0013/

Living publication

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