CANONICAL LIVING OVERVIEW GUIDE

Durable Workflows, Queues, Streams, and Sagas

A distributed-execution guide covering durable workflows, activity execution, queues, brokers, event streams, delivery guarantees, timers, signals, idempotency, transactions, sagas, compensation, replay, testing, observability, and failure recovery.

PERMANENT ID
MYTHOS-FG-CLD-0006

PUBLICATION CLASS
FIELD GUIDE

VERSION / STATUS
1.0.0-candidate / CANDIDATE

PERMANENT PUBLICATION IDENTITY

Durable Workflows, Queues, Streams, and Sagas

Document ID
MYTHOS-FG-CLD-0006

Version
1.0.0-candidate

Status
Candidate Focused Field Guide

Review date
2026-10-26

Publication position
This is a living field guide. It is designed for frequent revision while retaining a stable permanent identity. Candidate status means the content is ready for structured use and review but is not represented as an external certification or authority publication.

LEVEL	CLASS	CADENCE	EVIDENCE
L1-L4	FIELD GUIDE	QUARTERLY	SOURCE-BOUND
Foundational through frontier	Practical and educational	Interim updates as needed	Limitations remain visible

Reader orientation

Dimension	Engineering position
Primary domain	MYTHOS-CLD - Cloud, Platform & Distributed Systems
Secondary domain	MYTHOS-DAT; MYTHOS-SWE; MYTHOS-SEC
Audience	Cloud, platform, integration, software, data, SRE, and agent-runtime engineers.
Prerequisites	Distributed systems, APIs, messaging, databases, transactions, reliability, and observability.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

Choose workflow, queue, stream, scheduler, and event architectures according to state and failure semantics.

Build durable executions that recover from process, worker, dependency, and infrastructure failure.

Engineer idempotency, ordering, retries, timeouts, transactions, and compensation explicitly.

Operate workflows and event systems with replay, backpressure, evidence, and safe migration.

Validate business outcomes rather than equating delivery acknowledgement with completion.

How to use this guide

Read the guide as a progressive system rather than a disconnected encyclopedia. Foundational material establishes the mental model; practitioner sections convert it into repeatable implementation and operations; advanced sections expose architecture, scale, security, and failure behavior; frontier sections describe technologies whose evidence or maturity is still evolving.

Suggested reading path
New practitioners should follow the chapters in order and complete the laboratories. Experienced engineers may begin with the architecture, failure, validation, or frontier sections, then use the related-document map to deepen a specific topic.

Learning and assurance model

Level	Reader outcome
L1 - Foundational	Terminology, first principles, component roles, packet or state flow, and simple working examples.
L2 - Practitioner	Implementation, configuration, automation, testing, troubleshooting, and operational evidence.
L3 - Advanced	Architecture, scale, concurrency, resilience, threat models, performance, and complex migrations.
L4 - Frontier	Emerging systems, research directions, technical uncertainty, readiness gates, and adoption signals.

Contents

CHAPTER 01

Durable execution and workflow state

Define workflow code as recoverable stateful computation rather than a long-running process.

Chapter operating position
Define workflow code as recoverable stateful computation rather than a long-running process.

1.1 Workflow histories and replay

Workflow histories and replay must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Persist decisions, events, timers, signals, activity outcomes, and deterministic execution history. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for workflow histories and replay.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating workflow histories and replay as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for workflow histories and replay.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

1.2 Workflow and activity boundaries

Workflow and activity boundaries must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Keep orchestration deterministic while isolating nondeterministic I/O in retryable activities. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for workflow and activity boundaries.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
workflow:
  id: order-042
  state: payment_pending
  history_version: 18
  activities:
    reserve_inventory:
      idempotency_key: order-042:reserve
      retry: bounded-exponential
    charge_payment:
      idempotency_key: order-042:charge
      timeout_seconds: 30
  compensation:
    on_shipping_failure: [refund_payment, release_inventory]
  completion:
    require: [order_state_committed, external_effects_verified]
```

Failure patterns

- Treating workflow and activity boundaries as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for workflow and activity boundaries.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

1.3 Timers, signals, updates, and queries

Timers, signals, updates, and queries must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Handle time, external input, state mutation, and read-only inspection without losing causality. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for timers, signals, updates, and queries.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating timers, signals, updates, and queries as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for timers, signals, updates, and queries.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

1.4 Versioning and migration

Versioning and migration must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Evolve long-lived workflows with compatible code paths, markers, and controlled retirement. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for versioning and migration.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating versioning and migration as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for versioning and migration.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 01 laboratory and review

Practical laboratory

Implement a durable order workflow, terminate its worker during three different steps, and prove replay and recovery.

Laboratory evidence package

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

Frontier extension

Explore durable agent missions whose plans, approvals, tools, and compensations survive model and harness replacement.

Review gate

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

CHAPTER 02

Queues, brokers, and consumer engineering

Select work distribution according to ownership, visibility, acknowledgement, and retry semantics.

Chapter operating position
Select work distribution according to ownership, visibility, acknowledgement, and retry semantics.

2.1 Queue and consumer models

Queue and consumer models must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Use pull or push delivery, consumer groups, visibility, acknowledgements, leases, and horizontal scale. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for queue and consumer models.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating queue and consumer models as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for queue and consumer models.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

2.2 Retention and dead letters

Retention and dead letters must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Define limits, age, work-queue deletion, poison-message isolation, redrive, and evidence. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for retention and dead letters.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
workflow:
  id: order-042
  state: payment_pending
  history_version: 18
  activities:
    reserve_inventory:
      idempotency_key: order-042:reserve
      retry: bounded-exponential
    charge_payment:
      idempotency_key: order-042:charge
      timeout_seconds: 30
  compensation:
    on_shipping_failure: [refund_payment, release_inventory]
  completion:
    require: [order_state_committed, external_effects_verified]
```

Failure patterns

- Treating retention and dead letters as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for retention and dead letters.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

2.3 Ordering and partitioning

Ordering and partitioning must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Choose keys, partitions, affinity, sequencing, and parallelism according to domain invariants. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for ordering and partitioning.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating ordering and partitioning as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for ordering and partitioning.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

2.4 Backpressure and admission

Backpressure and admission must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Bound producers, queues, retries, consumers, dependencies, memory, and storage. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for backpressure and admission.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating backpressure and admission as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for backpressure and admission.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 02 laboratory and review

Practical laboratory
Simulate duplicate, delayed, reordered, expired, and poisoned work items across multiple consumers.

Laboratory evidence package

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

Frontier extension

Explore globally scheduled work queues that combine agent, batch, GPU, and human work.

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

CHAPTER 03

Event streams and stateful processing

Treat ordered logs as durable facts that drive projections and continuous computation.

Chapter operating position
Treat ordered logs as durable facts that drive projections and continuous computation.

3.1 Topics, partitions, offsets, and retention

Topics, partitions, offsets, and retention must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Model event identity, ordering scope, replay horizon, compaction, and consumer progress. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for topics, partitions, offsets, and retention.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating topics, partitions, offsets, and retention as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for topics, partitions, offsets, and retention.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

3.2 Transactions and processing guarantees

Transactions and processing guarantees must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Use idempotent production, read-committed consumption, atomic offset updates, and exactly-once-v2 where supported. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for transactions and processing guarantees.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
workflow:
  id: order-042
  state: payment_pending
  history_version: 18
  activities:
    reserve_inventory:
      idempotency_key: order-042:reserve
      retry: bounded-exponential
    charge_payment:
      idempotency_key: order-042:charge
      timeout_seconds: 30
  compensation:
    on_shipping_failure: [refund_payment, release_inventory]
  completion:
    require: [order_state_committed, external_effects_verified]
```

Failure patterns

- Treating transactions and processing guarantees as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for transactions and processing guarantees.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

3.3 State stores and recovery

State stores and recovery must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Maintain local or remote state, changelogs, checkpoints, standby replicas, and restore. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for state stores and recovery.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating state stores and recovery as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for state stores and recovery.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

3.4 Stream-time and event-time

Stream-time and event-time must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Handle timestamps, watermarks, windows, late events, corrections, and deterministic aggregation. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for stream-time and event-time.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating stream-time and event-time as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for stream-time and event-time.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 03 laboratory and review

Practical laboratory
Build a stateful stream processor and inject consumer restart, late data, duplicate production, and partition movement.

Laboratory evidence package

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

Frontier extension

Explore verifiable streaming where every derived fact carries source offsets and transformation provenance.

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

CHAPTER 04

Event contracts and interoperability

Make events understandable across producers, transports, consumers, and time.

Chapter operating position
Make events understandable across producers, transports, consumers, and time.

4.1 CloudEvents envelope

CloudEvents envelope must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Use stable IDs, source, type, subject, time, schema, trace context, and content type. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for cloudevents envelope.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating cloudevents envelope as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for cloudevents envelope.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

4.2 Domain event semantics

Domain event semantics must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Define business meaning, invariants, identity, privacy, units, causality, and compatibility. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for domain event semantics.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
workflow:
  id: order-042
  state: payment_pending
  history_version: 18
  activities:
    reserve_inventory:
      idempotency_key: order-042:reserve
      retry: bounded-exponential
    charge_payment:
      idempotency_key: order-042:charge
      timeout_seconds: 30
  compensation:
    on_shipping_failure: [refund_payment, release_inventory]
  completion:
    require: [order_state_committed, external_effects_verified]
```

Failure patterns

- Treating domain event semantics as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for domain event semantics.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

4.3 Schema governance

Schema governance must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Version schemas, defaults, optionality, evolution, validation, ownership, and deprecation. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for schema governance.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating schema governance as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for schema governance.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

4.4 Tracing and correlation

Tracing and correlation must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Propagate workflow, event, message, transaction, tenant, and business identifiers. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for tracing and correlation.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating tracing and correlation as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for tracing and correlation.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 04 laboratory and review

Practical laboratory
Create one portable event contract and transport it over HTTP, Kafka-style streams, and NATS-style messaging.

Laboratory evidence package

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

Frontier extension

Explore semantic event registries that generate policies, tests, and observability conventions.

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

CHAPTER 05

Idempotency, retries, timeouts, and duplicate effects

Prevent delivery and execution uncertainty from creating incorrect external state.

Chapter operating position
Prevent delivery and execution uncertainty from creating incorrect external state.

5.1 Business idempotency keys

Business idempotency keys must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Choose stable operation identity, request scope, storage, expiry, conflict behavior, and result replay. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for business idempotency keys.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating business idempotency keys as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for business idempotency keys.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

5.2 Retry classification

Retry classification must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Separate transient, permanent, throttled, ambiguous, and unsafe-to-retry failures. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for retry classification.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
workflow:
  id: order-042
  state: payment_pending
  history_version: 18
  activities:
    reserve_inventory:
      idempotency_key: order-042:reserve
      retry: bounded-exponential
    charge_payment:
      idempotency_key: order-042:charge
      timeout_seconds: 30
  compensation:
    on_shipping_failure: [refund_payment, release_inventory]
  completion:
    require: [order_state_committed, external_effects_verified]
```

Failure patterns

- Treating retry classification as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for retry classification.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

5.3 Deadlines and cancellation

Deadlines and cancellation must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Propagate budgets, stop work, clean up resources, and prevent zombie effects. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for deadlines and cancellation.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating deadlines and cancellation as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for deadlines and cancellation.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

5.4 Inbox, outbox, and deduplication

Inbox, outbox, and deduplication must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Coordinate database state, event publication, consumer effects, and replay. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for inbox, outbox, and deduplication.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating inbox, outbox, and deduplication as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for inbox, outbox, and deduplication.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 05 laboratory and review

Practical laboratory
Implement an idempotent payment-like operation with ambiguous timeout, duplicate delivery, and safe result replay.

Laboratory evidence package

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

Frontier extension

Explore cryptographically receipted effects and cross-system idempotency ledgers.

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

CHAPTER 06

Sagas, compensation, and long-running transactions

Coordinate business state when one atomic transaction cannot span participants.

Chapter operating position
Coordinate business state when one atomic transaction cannot span participants.

6.1 Orchestration and choreography

Orchestration and choreography must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Compare central workflow state with event-driven participant coordination and failure visibility. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for orchestration and choreography.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating orchestration and choreography as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for orchestration and choreography.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

6.2 Compensating actions

Compensating actions must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Define reversibility, semantic undo, pivot steps, irreversibility, and human intervention. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for compensating actions.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
workflow:
  id: order-042
  state: payment_pending
  history_version: 18
  activities:
    reserve_inventory:
      idempotency_key: order-042:reserve
      retry: bounded-exponential
    charge_payment:
      idempotency_key: order-042:charge
      timeout_seconds: 30
  compensation:
    on_shipping_failure: [refund_payment, release_inventory]
  completion:
    require: [order_state_committed, external_effects_verified]
```

Failure patterns

- Treating compensating actions as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for compensating actions.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

6.3 Partial failure and reconciliation

Partial failure and reconciliation must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Detect incomplete steps, stale participants, orphan effects, and divergent views. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for partial failure and reconciliation.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating partial failure and reconciliation as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for partial failure and reconciliation.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

6.4 Human tasks and approvals

Human tasks and approvals must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Pause durably for decisions, deadlines, escalations, evidence, and alternate paths. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for human tasks and approvals.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating human tasks and approvals as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for human tasks and approvals.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 06 laboratory and review

Practical laboratory
Model an order saga with inventory, payment, shipping, timeout, cancellation, and nonreversible actions.

Laboratory evidence package

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

Frontier extension

Explore machine-generated sagas constrained by typed effects and verified compensation plans.

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

CHAPTER 07

Workflow testing, observability, and operations

Make durable systems reproducible and diagnosable across long time horizons.

Chapter operating position
Make durable systems reproducible and diagnosable across long time horizons.

7.1 Deterministic and time-skipping tests

Deterministic and time-skipping tests must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Test timers, retries, signals, activity failures, history replay, and versions quickly. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for deterministic and time-skipping tests.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating deterministic and time-skipping tests as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for deterministic and time-skipping tests.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

7.2 Operational telemetry

Operational telemetry must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Track workflow state, queue depth, lag, attempts, deadlines, history size, effects, and outcomes. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for operational telemetry.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
workflow:
  id: order-042
  state: payment_pending
  history_version: 18
  activities:
    reserve_inventory:
      idempotency_key: order-042:reserve
      retry: bounded-exponential
    charge_payment:
      idempotency_key: order-042:charge
      timeout_seconds: 30
  compensation:
    on_shipping_failure: [refund_payment, release_inventory]
  completion:
    require: [order_state_committed, external_effects_verified]
```

Failure patterns

- Treating operational telemetry as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for operational telemetry.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

7.3 Replay and repair

Replay and repair must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Reprocess safely, reset consumers, rebuild projections, patch state, and preserve audit evidence. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for replay and repair.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating replay and repair as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for replay and repair.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

7.4 Capacity and failure domains

Capacity and failure domains must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Plan partitions, workers, brokers, storage, regions, quotas, and dependency saturation. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for capacity and failure domains.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating capacity and failure domains as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for capacity and failure domains.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 07 laboratory and review

Practical laboratory
Run a chaos exercise involving broker loss, worker loss, dependency outage, replay, and projection rebuild.

Laboratory evidence package

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

Frontier extension

Explore self-healing workflow fabrics with independent repair agents and immutable history.

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

CHAPTER 08

Architecture selection and governance

Choose the simplest durable mechanism that satisfies domain semantics and operations.

Chapter operating position
Choose the simplest durable mechanism that satisfies domain semantics and operations.

8.1 Workflow versus queue versus stream

Workflow versus queue versus stream must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Compare state, duration, ordering, replay, coordination, throughput, latency, and ownership. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for workflow versus queue versus stream.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating workflow versus queue versus stream as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for workflow versus queue versus stream.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

8.2 Managed and self-hosted platforms

Managed and self-hosted platforms must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Assess control, sovereignty, upgrade, cost, integration, support, and exit. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for managed and self-hosted platforms.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Implementation sketch

```
workflow:
  id: order-042
  state: payment_pending
  history_version: 18
  activities:
    reserve_inventory:
      idempotency_key: order-042:reserve
      retry: bounded-exponential
    charge_payment:
      idempotency_key: order-042:charge
      timeout_seconds: 30
  compensation:
    on_shipping_failure: [refund_payment, release_inventory]
  completion:
    require: [order_state_committed, external_effects_verified]
```

Failure patterns

- Treating managed and self-hosted platforms as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for managed and self-hosted platforms.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

8.3 Evidence and acceptance

Evidence and acceptance must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Require business outcome, state reconciliation, failure tests, and recovery proof. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for evidence and acceptance.

Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.

Validate intended configuration, stored state, applied state, external effects, and user outcome separately.

Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.

Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.

Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

Treating evidence and acceptance as a feature without defining its state and failure semantics.

Equating acknowledgement, replication, snapshot, or successful export with correct business completion.

Using eventual consistency without exposing stale, pending, conflicting, or repair states.

Allowing retries, replay, synchronization, or restoration to duplicate external effects.

Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.

Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for evidence and acceptance.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

8.4 Lifecycle and retirement

Lifecycle and retirement must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Migrate histories, events, schemas, consumers, state stores, and retained evidence. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the request or event, admission, durable state, activity, queue or stream, external effect, compensation, evidence, and business outcome chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

Production implementation starts from a small known-good baseline. Add one queue, replica, index, collector processor, storage tier, synchronization peer, proof mechanism, or control at a time; preserve before-state evidence; test duplicates, stale state, partial failure, overload, and rollback; and retain a deterministic repair path.

Troubleshooting identifies the first divergence from the expected state machine. Inspect identity, time, sequence, version, cache, checkpoint, partition, retention, authorization, transformation, and external dependency boundaries. Closure requires a causal explanation, reproducibility, validated restoration or convergence, and visible residual limitations.

Engineering practices

- Define ownership, authority, state, dependencies, limits, and acceptance evidence for lifecycle and retirement.
- Use stable identities, typed schemas, idempotency, explicit consistency, and versioned lifecycle records.
- Validate intended configuration, stored state, applied state, external effects, and user outcome separately.
- Test nominal, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback conditions.
- Preserve source, event, trace, checkpoint, artifact, receipt, approval, and repair evidence.
- Scale with quotas, retention, backpressure, isolation, progressive rollout, and verified restore or convergence.

Failure patterns

- Treating lifecycle and retirement as a feature without defining its state and failure semantics.
- Equating acknowledgement, replication, snapshot, or successful export with correct business completion.
- Using eventual consistency without exposing stale, pending, conflicting, or repair states.
- Allowing retries, replay, synchronization, or restoration to duplicate external effects.
- Collecting evidence without stable identity, schema, time quality, provenance, or retention rules.
- Declaring resilience without representative recovery and end-to-end validation.

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for lifecycle and retirement.
Observe	Events, queues, streams, versions, replicas, indexes, telemetry, receipts, storage, effects, and user-visible state.
Test	Expected, prohibited, duplicate, stale, reordered, overloaded, partitioned, failed, recovery, and rollback cases.
Measure	Correctness, convergence, latency, throughput, durability, freshness, cost, resource use, and operator effort.
Reconcile	Intended and recorded state against actual data, infrastructure, external effect, evidence, and user outcome.

Chapter 08 laboratory and review

Practical laboratory
Create an architecture decision for a multi-day regulated process using at least three execution patterns.

Laboratory evidence package

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

Frontier extension

Develop a unified durable-execution control plane spanning workflows, events, queues, agents, and people.

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

Integrated learning roadmap

The guide is intended to be revisited. First establish fluency, then build a small implementation, operate it under failure, automate repeatable work, and finally evaluate emerging techniques against a stable baseline. Progress is demonstrated through evidence rather than reading completion alone.

Stage	Demonstrated capability
Foundation	Explain the system from first principles and trace its critical path without relying on product terminology.
Build	Implement the smallest representative system with explicit contracts, identities, telemetry, and tests.
Operate	Diagnose injected failures, recover safely, and preserve evidence of the causal chain.
Automate	Convert a proven manual workflow into bounded, idempotent, observable automation.
Architect	Design for scale, resilience, security, interoperability, and lifecycle change.
Explore	Evaluate frontier techniques using stated hypotheses, limitations, and adoption gates.

Source authority register

Source policy
Official standards, specifications, and project documentation remain with their issuing organizations. Mythos Systems provides original engineering interpretation, synthesis, implementation guidance, and relationship mapping. Source verification dates do not guarantee that an authority has not changed since review.

01. Temporal - Temporal Documentation - Workflow Execution
Role: Durable workflow state, replay, retries, timers, signals, and recovery.
Verified: 2026-07-26
Official source: https://docs.temporal.io/workflow-execution

02. Temporal - Activity Execution
Role: Activity retries, timeouts, heartbeats, cancellation, and failure handling.
Verified: 2026-07-26
Official source: https://docs.temporal.io/activity-execution

03. Apache Kafka - Apache Kafka Documentation
Role: Event streaming, durable topics, producers, consumers, transactions, and processing guarantees.
Verified: 2026-07-26
Official source: https://kafka.apache.org/documentation/

04. Apache Kafka - Kafka Streams
Role: Stateful stream processing, topology, state stores, recovery, and exactly-once-v2 semantics.
Verified: 2026-07-26
Official source: https://kafka.apache.org/documentation/streams/

05. NATS - JetStream Documentation
Role: Persistent streams, consumers, replay, retention, work queues, and replication.
Verified: 2026-07-26
Official source: https://docs.nats.io/nats-concepts/jetstream

06. CloudEvents - CloudEvents Specification
Role: Portable event envelope, attributes, bindings, and interoperability.
Verified: 2026-07-26
Official source: https://cloudevents.io/

07. CNCF Serverless Workflow - Serverless Workflow Specification
Role: Portable durable workflow and event-orchestration definitions.
Verified: 2026-07-26
Official source: https://serverlessworkflow.io/

Authority application and refresh triggers

Authority class	Required library action
Protocols and specifications	Recheck interoperability, security considerations, defaults, and migration guidance when a referenced specification changes.
Security and assurance frameworks	Update control mappings, evidence expectations, risk language, and assessment procedures after material framework revision.
Languages, runtimes, and projects	Re-run examples, tests, deprecation checks, and operational recommendations against supported releases.
Benchmarks and evaluations	Re-execute claims when workloads, methods, models, compilers, drivers, hardware, or scoring rules change.
Operational evidence	Incorporate validated incidents, failure tests, reader corrections, and measured implementation outcomes into the next revision.

Limitations, freshness, and revision control

Boundary	Statement
Publication limitation	This guide synthesizes broad engineering practice. It does not replace product documentation, legal advice, safety certification, or environment-specific design review.
Evidence limitation	Not every pattern is benchmarked in every environment. Examples illustrate engineering methods and must be validated against actual versions, workloads, hardware, policy, and failure conditions.
Freshness limitation	Vendor behavior, software libraries, AI models, security threats, and emerging standards can change quickly. Volatile claims require source revalidation before operational adoption.
Adoption limitation	Frontier content is not an implicit adoption recommendation. Adoption requires defined value, qualification gates, controlled proof, rollback, ownership, and residual-risk acceptance.
Status boundary	Candidate Focused Field Guide describes the publication, not certification of a product, implementation, engineer, or organization.

Revision record

Version	Revision statement
1.0.0-candidate	Initial candidate living-guide edition. Published 2026-07-26; scheduled review 2026-10-26.

Search and relationship metadata

Dimension	Engineering position
Primary domain	MYTHOS-CLD - Cloud, Platform & Distributed Systems
Secondary domain	MYTHOS-DAT; MYTHOS-SWE; MYTHOS-SEC
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	durable workflows, queues, streams, sagas, Temporal, Kafka, NATS JetStream, idempotency, event contracts, compensation
Canonical route	/library/field-guides/mythos-fg-cld-0006/

Living publication

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