



CANONICAL LIVING OVERVIEW GUIDE

Data Sovereignty, Privacy, Local-First Synchronization, and Egress

A data-control guide covering sovereignty, privacy engineering, data mapping, local-first architecture, offline-first state, CRDTs, synchronization, access control, key ownership, egress policy, deletion, cross-border flows, and user agency.

PERMANENT PUBLICATION IDENTITY

Data Sovereignty, Privacy, Local-First Synchronization, and Egress

Document ID
MYTHOS-FG-DAT-0007

Version
1.0.0-candidate

Status
Candidate Focused Field Guide

Review date
2026-10-26

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

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

Reader orientation

Dimension	Engineering position
Primary domain	MYTHOS-DAT - Data, State, Storage, Reliability & Evidence
Secondary domain	MYTHOS-SEC; MYTHOS-CLD; MYTHOS-UX
Audience	Data, privacy, security, cloud, local-first, product, platform, edge, and enterprise-architecture teams.
Prerequisites	Data architecture, distributed systems, privacy, cryptography, identity, synchronization, and cloud networking.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

- Translate sovereignty and privacy requirements into concrete data, identity, key, network, and operational controls.
- Build local-first systems that remain useful offline and treat local state as durable rather than disposable cache.
- Select synchronization and conflict models according to domain semantics and user expectations.
- Control egress across applications, APIs, telemetry, models, tools, users, and support processes.
- Provide correction, export, deletion, retention, transparency, and exit throughout the lifecycle.

How to use this guide

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

Suggested reading path

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

Learning and assurance model

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

Contents

How to use this guide	3
Contents	3
Chapter 01 - Sovereignty, privacy, and data-control objectives	5
Chapter 01 laboratory and review	10
Chapter 02 - Data inventory, classification, and flow mapping	11
Chapter 02 laboratory and review	16
Chapter 03 - Local-first architecture and offline operation	17
Chapter 03 laboratory and review	22
Chapter 04 - CRDTs, versions, and conflict semantics	23
Chapter 04 laboratory and review	28
Chapter 05 - Synchronization protocols and distributed storage	29
Chapter 05 laboratory and review	34

Chapter 06 - Identity, encryption, and key ownership	35
Chapter 06 laboratory and review	40
Chapter 07 - Egress architecture and policy enforcement	41
Chapter 07 laboratory and review	46
Chapter 08 - Lifecycle rights, governance, and exit	47
Chapter 08 laboratory and review	52
Integrated learning roadmap	53
Source authority register	54
Limitations, freshness, and revision control	55

CHAPTER 01

Sovereignty, privacy, and data-control objectives

Define the required control instead of treating geographic hosting as sufficient.

Chapter operating position

Define the required control instead of treating geographic hosting as sufficient.



1.1 Jurisdiction and legal exposure

Jurisdiction and legal exposure must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Identify individuals, organizations, laws, contracts, support, disclosure, and cross-border processing. 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control 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 jurisdiction and legal exposure.

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 jurisdiction and legal exposure 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 jurisdiction and legal exposure.
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 Technical sovereignty

Technical sovereignty must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Control storage, compute, identities, keys, networks, telemetry, software, updates, and administrative planes. 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control 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 technical sovereignty.

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

```
data_policy:
  data_class: restricted-user-work
  authoritative_copy: local-device
  sync:
    encrypted: end-to-end
    relay_can_read: false
    conflict_model: crdt-plus-domain-review
  egress:
    default: deny
    allowed: [approved-peer-sync]
  telemetry:
    content: prohibited
  deletion:
    propagate_to: [peers, indexes, caches, backups-on-expiry]
```

Failure patterns

Treating technical sovereignty 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 technical sovereignty.
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 Operational sovereignty

Operational sovereignty must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Maintain skills, ownership, incident authority, support, continuity, supplier alternatives, 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control 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 sovereignty.

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

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

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

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

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

Failure patterns

Treating operational sovereignty 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 sovereignty.
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 Privacy risk and individual impact

Privacy risk and individual impact must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Analyze data actions, visibility, linkability, exclusion, surveillance, loss of control, and harm. 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

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

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

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for privacy risk and individual impact.

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

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

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

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

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

Failure patterns

Treating privacy risk and individual impact as a feature without defining its state and failure semantics.

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

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

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

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

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

Validation and evidence

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

Chapter 01 laboratory and review

Practical laboratory

Create a sovereignty and privacy requirements matrix for a cloud-connected desktop application.

Laboratory evidence package

Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.

Versioned configuration or code with explanatory comments and reproducible setup instructions.

Nominal-path validation plus at least two injected failure or boundary conditions.

Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.

A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore user- and community-owned data infrastructures with verifiable governance.

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

Data inventory, classification, and flow mapping

Know what data exists, why it moves, and which authority permits each action.

Chapter operating position

Know what data exists, why it moves, and which authority permits each action.

2.1 Data elements and purpose

Data elements and purpose must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Record identity, content, sensitivity, origin, purpose, owner, subject, and lawful or policy basis. 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control 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 data elements and purpose.

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 data elements and purpose 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 data elements and purpose.
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 Data actions

Data actions must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Map collection, generation, inference, access, storage, transfer, sharing, retention, deletion, and disclosure. 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control 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 data 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

```
data_policy:
  data_class: restricted-user-work
  authoritative_copy: local-device
  sync:
    encrypted: end-to-end
    relay_can_read: false
    conflict_model: crdt-plus-domain-review
  egress:
    default: deny
    allowed: [approved-peer-sync]
  telemetry:
    content: prohibited
  deletion:
    propagate_to: [peers, indexes, caches, backups-on-expiry]
```

Failure patterns

Treating data 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 data 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.



2.3 Processing environment

Processing environment must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Identify device, site, region, cloud, model provider, third party, support, backup, and archive. 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control 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 processing environment.

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 processing environment 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 processing environment.
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 Derived and hidden data

Derived and hidden data must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Include embeddings, caches, logs, prompts, telemetry, indexes, backups, metadata, and learned parameters. 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control 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 derived and hidden data.

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 derived and hidden data 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 derived and hidden data.
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

Build a complete data-flow map and identify five derived or operational data sets omitted from the initial inventory.

Laboratory evidence package

Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.

Versioned configuration or code with explanatory comments and reproducible setup instructions.

Nominal-path validation plus at least two injected failure or boundary conditions.

Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.

A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore continuously generated data maps from runtime telemetry and policy enforcement.

Review gate

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

CHAPTER 03

Local-first architecture and offline operation

Make the local device a durable, capable participant rather than a thin cache.

Chapter operating position

Make the local device a durable, capable participant rather than a thin cache.



3.1 Local authoritative data

Local authoritative data must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Store user work, state, history, indexes, preferences, and capability locally with explicit 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

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

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

Engineering practices

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

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

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

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

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

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

Failure patterns

Treating local authoritative data as a feature without defining its state and failure semantics.

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

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

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

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

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

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for local authoritative data.
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 Offline functionality

Offline functionality must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Support create, read, update, search, automation, and recovery without network dependency. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control 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 offline functionality.

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

```
data_policy:
  data_class: restricted-user-work
  authoritative_copy: local-device
  sync:
    encrypted: end-to-end
    relay_can_read: false
    conflict_model: crdt-plus-domain-review
  egress:
    default: deny
    allowed: [approved-peer-sync]
  telemetry:
    content: prohibited
  deletion:
    propagate_to: [peers, indexes, caches, backups-on-expiry]
```

Failure patterns

Treating offline functionality 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 offline functionality.
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 Background synchronization

Background synchronization must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Exchange changes when available without blocking local work or silently overwriting state. 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control 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 background synchronization.

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 background synchronization 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 background synchronization.
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 Longevity and export

Longevity and export must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Keep data usable if servers, vendors, accounts, subscriptions, or networks disappear. 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control 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 longevity and export.

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 longevity and export 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 longevity and export.
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

Design an application that remains fully useful for thirty days offline and later synchronizes safely.

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 peer-to-peer local-first applications with no permanently trusted central server.

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

CRDTs, versions, and conflict semantics

Merge concurrent work only where the domain permits mathematically convergent behavior.

Chapter operating position

Merge concurrent work only where the domain permits mathematically convergent behavior.

4.1 Operation- and state-based replication

Operation- and state-based replication must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Understand changes, causal context, merge, convergence, delivery, 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control 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 operation- and state-based replication.

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 operation- and state-based replication 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 operation- and state-based replication.
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 Data-type semantics

Data-type semantics must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Use registers, counters, sets, maps, sequences, text, and custom domain conflict rules. 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control 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 data-type 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

```
data_policy:
  data_class: restricted-user-work
  authoritative_copy: local-device
  sync:
    encrypted: end-to-end
    relay_can_read: false
    conflict_model: crdt-plus-domain-review
  egress:
    default: deny
    allowed: [approved-peer-sync]
  telemetry:
    content: prohibited
  deletion:
    propagate_to: [peers, indexes, caches, backups-on-expiry]
```

Failure patterns

Treating data-type 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 data-type 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 Causality and version tracking

Causality and version tracking must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Apply vector clocks, version vectors, change hashes, actor identity, and branch 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control 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 causality and version tracking.

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 causality and version tracking 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 causality and version tracking.
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 Human-visible conflict and undo

Human-visible conflict and undo must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Expose concurrent edits, semantic conflicts, authorship, resolution, and collaborative undo. 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control 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-visible conflict and undo.

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-visible conflict and undo 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-visible conflict and undo.
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

Implement a small convergent replicated data type and test reordering, duplication, partition, and concurrent edits.

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 verified domain CRDTs and relationship-based local-first access control.

Review gate

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

CHAPTER 05

Synchronization protocols and distributed storage

Move only the necessary changes while preserving permissions and convergence.

Chapter operating position

Move only the necessary changes while preserving permissions and convergence.



5.1 Peer and relay topology

Peer and relay topology must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Use direct, local network, relay, cloud, store-and-forward, and multi-device synchronization. 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control 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 peer and relay topology.

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 peer and relay topology 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 peer and relay topology.
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 Change exchange

Change exchange must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Negotiate heads, missing changes, snapshots, chunks, compression, resume, and deduplication. 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

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

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

Engineering practices

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

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

```
data_policy:
  data_class: restricted-user-work
  authoritative_copy: local-device
  sync:
    encrypted: end-to-end
    relay_can_read: false
    conflict_model: crdt-plus-domain-review
  egress:
    default: deny
    allowed: [approved-peer-sync]
  telemetry:
    content: prohibited
  deletion:
    propagate_to: [peers, indexes, caches, backups-on-expiry]
```

Failure patterns

Treating change exchange as a feature without defining its state and failure semantics.

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

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

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

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

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

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for change exchange.
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 Access-aware sync

Access-aware sync must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Restrict documents, fields, changes, peers, roles, groups, purposes, and revoked participants. 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control 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 access-aware sync.

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 access-aware sync 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 access-aware sync.
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 Repair and garbage collection

Repair and garbage collection must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Detect missing history, compact safely, retain audit or undo, and recover corrupt replicas. 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control 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 repair and garbage collection.

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 repair and garbage collection 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 repair and garbage collection.
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

Simulate three devices syncing through partitions, lost messages, revoked access, and a stale backup restore.

Laboratory evidence package

Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.

Versioned configuration or code with explanatory comments and reproducible setup instructions.

Nominal-path validation plus at least two injected failure or boundary conditions.

Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.

A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore encrypted synchronization where relays cannot read data or infer social graphs.

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

Identity, encryption, and key ownership

Bind data control to identities and keys that remain operable through provider change.

Chapter operating position

Bind data control to identities and keys that remain operable through provider change.

6.1 User, device, and workload identity

User, device, and workload identity must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Use passkeys, hardware keys, device keys, workload identity, groups, and recovery. 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control 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 user, device, and workload identity.

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

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

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

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

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

Failure patterns

Treating user, device, and workload identity as a feature without defining its state and failure semantics.

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

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

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

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

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

Validation and evidence

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

6.2 Data encryption and envelopes

Data encryption and envelopes must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Separate data keys, wrapping keys, devices, accounts, groups, backups, and sharing. 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control 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 data encryption and envelopes.

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

```
data_policy:
  data_class: restricted-user-work
  authoritative_copy: local-device
  sync:
    encrypted: end-to-end
    relay_can_read: false
    conflict_model: crdt-plus-domain-review
  egress:
    default: deny
    allowed: [approved-peer-sync]
  telemetry:
    content: prohibited
  deletion:
    propagate_to: [peers, indexes, caches, backups-on-expiry]
```

Failure patterns

Treating data encryption and envelopes 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 data encryption and envelopes.
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 Rotation, revocation, and recovery

Rotation, revocation, and recovery must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Handle lost devices, removed users, compromised keys, offline peers, and historical data. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control 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 rotation, revocation, 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 rotation, revocation, 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 rotation, revocation, 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.

6.4 Provider and administrator access

Provider and administrator access must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Limit support, cloud operator, telemetry, backup, search, and emergency access. 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control 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 provider and administrator access.

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 provider and administrator access 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 provider and administrator access.
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

Design key management for local-first multi-device collaboration with lost device and member removal.

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 threshold user-controlled keys and post-quantum local-first collaboration.

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

Egress architecture and policy enforcement

Control every path by which data, metadata, or derived information can leave a boundary.

Chapter operating position

Control every path by which data, metadata, or derived information can leave a boundary.

7.1 Network and API egress

Network and API egress must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Allowlist destinations, protocols, methods, DNS, proxies, private links, and direct connections. 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

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

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

Engineering practices

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

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

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

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

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

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

Failure patterns

Treating network and api egress as a feature without defining its state and failure semantics.

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

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

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

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

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

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for network and api egress.
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 Application, model, and tool egress

Application, model, and tool egress must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Govern prompts, files, embeddings, tool arguments, generated outputs, plugins, and agents. 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

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

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

Engineering practices

Define ownership, authority, state, dependencies, limits, and acceptance evidence for application, model, and tool egress.

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

```
data_policy:
  data_class: restricted-user-work
  authoritative_copy: local-device
  sync:
    encrypted: end-to-end
    relay_can_read: false
    conflict_model: crdt-plus-domain-review
  egress:
    default: deny
    allowed: [approved-peer-sync]
  telemetry:
    content: prohibited
  deletion:
    propagate_to: [peers, indexes, caches, backups-on-expiry]
```

Failure patterns

Treating application, model, and tool egress as a feature without defining its state and failure semantics.

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

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

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

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

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

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for application, model, and tool egress.
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 Telemetry and support egress

Telemetry and support egress must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Minimize logs, crash reports, analytics, updates, license checks, and remote diagnostics. 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

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

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

Engineering practices

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

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

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

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

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

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

Failure patterns

Treating telemetry and support egress as a feature without defining its state and failure semantics.

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

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

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

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

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

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for telemetry and support egress.
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 Content inspection and receipts

Content inspection and receipts must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Classify, redact, tokenize, approve, record, and verify permitted transfers. 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control 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 content inspection and receipts.

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

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

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

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

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

Failure patterns

Treating content inspection and receipts as a feature without defining its state and failure semantics.

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

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

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

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

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

Validation and evidence

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

Chapter 07 laboratory and review

Practical laboratory

Red-team an application for hidden egress through telemetry, model calls, plugins, DNS, updates, and support bundles.

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 intent-bound egress capabilities and privacy-preserving policy proofs.

Review gate

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

CHAPTER 08

Lifecycle rights, governance, and exit

Keep user and organizational control effective after data has been processed and synchronized.

Chapter operating position

Keep user and organizational control effective after data has been processed and synchronized.



8.1 Transparency and user controls

Transparency and user controls must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Show storage, sync, sharing, recipients, models, telemetry, conflicts, and deletion state. 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control chain. A delivered message, successful query, completed backup, green dashboard, valid signature, or synchronized replica is not sufficient proof. Intended state must be reconciled with external effects, negative tests, degraded behavior, raw evidence, recovery, and the user or mission outcome.

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

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

Engineering practices

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

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

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

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

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

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

Failure patterns

Treating transparency and user controls as a feature without defining its state and failure semantics.

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

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

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

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

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

Validation and evidence

Method	Expected evidence
Examine	Architecture, authority, schemas, identities, state, retention, consistency, and dependencies for transparency and user controls.
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 Correction, deletion, and retention

Correction, deletion, and retention must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Propagate changes across replicas, indexes, caches, logs, backups, models, and evidence with documented limits. The design should name authoritative inputs, identities, versions, invariants, state transitions, consistency expectations, failure domains, resource limits, telemetry, recovery, and acceptance criteria.

This guide traces the subject through the data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control 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 correction, deletion, 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.

Implementation sketch

```
data_policy:
  data_class: restricted-user-work
  authoritative_copy: local-device
  sync:
    encrypted: end-to-end
    relay_can_read: false
    conflict_model: crdt-plus-domain-review
  egress:
    default: deny
    allowed: [approved-peer-sync]
  telemetry:
    content: prohibited
  deletion:
    propagate_to: [peers, indexes, caches, backups-on-expiry]
```

Failure patterns

Treating correction, deletion, 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 correction, deletion, 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.

8.3 Portability and interoperability

Portability and interoperability must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Export data, history, metadata, identities, relationships, and attachments in durable formats. 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control 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 portability and interoperability.

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 portability and interoperability 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 portability and interoperability.
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 Supplier and jurisdiction exit

Supplier and jurisdiction exit must be implemented as an observable state and evidence mechanism, not a product label or policy slogan. Migrate providers, regions, keys, software, support, and operational control through tested plans. 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 data purpose, local state, identity, key, synchronization, access, egress decision, retention, deletion, and user control 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 supplier and jurisdiction exit.

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

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

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

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

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

Failure patterns

Treating supplier and jurisdiction exit as a feature without defining its state and failure semantics.

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

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

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

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

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

Validation and evidence

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

Chapter 08 laboratory and review

Practical laboratory

Execute a user deletion and provider exit across local devices, cloud relays, search, telemetry, backups, and archives.

Laboratory evidence package

Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.

Versioned configuration or code with explanatory comments and reproducible setup instructions.

Nominal-path validation plus at least two injected failure or boundary conditions.

Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.

A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Develop sovereign local-first platforms where users can inspect and move the complete data and control state.

Review gate

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

Integrated learning roadmap

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

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

Source authority register

Source policy

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

01. NIST - Privacy Framework 1.1 Initial Public Draft

Role: Current draft privacy-risk framework for data processing, control, communication, protection, and governance.

Verified: 2026-07-26

Official source: <https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.40.ipd.pdf>

02. Ink & Switch - Local-First Software: You Own Your Data, in Spite of the Cloud

Role: Canonical local-first principles for offline operation, collaboration, longevity, privacy, and user control.

Verified: 2026-07-26

Official source: <https://www.inkandswitch.com/essay/local-first/>

03. Automerge - Automerge Documentation

Role: CRDT-based local-first data, offline editing, synchronization, repositories, and convergence.

Verified: 2026-07-26

Official source: <https://automerge.org/docs/hello/>

04. NIST - SP 800-207 - Zero Trust Architecture

Role: Resource-centric access, continuous authorization, policy enforcement, and distributed trust.

Verified: 2026-07-26

Official source: <https://csrc.nist.gov/pubs/sp/800/207/final>

05. NIST - SP 800-53 Rev. 5 Update 1

Role: Security and privacy controls for backup, recovery, audit, integrity, retention, and data protection.

Verified: 2026-07-26

Official source: <https://csrc.nist.gov/pubs/sp/800/53/r5/upd1/final>

06. OpenTelemetry - OpenTelemetry

Role: Vendor-neutral traces, metrics, logs, baggage, APIs, SDKs, and collectors.

Verified: 2026-07-26

Official source: <https://opentelemetry.io/>

Authority application and refresh triggers

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

Limitations, freshness, and revision control

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

Revision record

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

Search and relationship metadata

Dimension	Engineering position
Primary domain	MYTHOS-DAT - Data, State, Storage, Reliability & Evidence
Secondary domain	MYTHOS-SEC; MYTHOS-CLD; MYTHOS-UX
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	data sovereignty, privacy engineering, local first, CRDT, synchronization, egress, offline first, data control, Automerge, user agency
Canonical route	/library/field-guides/mythos-fg-dat-0007/

Living publication

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