ENGINEERING STANDARDS LIBRARY / DISTRIBUTED SYSTEMS

Durable Workflows, Queues, and Event-Driven Sagas Standard

Durable execution, message delivery semantics, idempotency, compensation, event ordering, and recoverable distributed workflows.

PERMANENT DOCUMENT ID

MYTHOS-DST-0001

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

Durable Workflows, Queues, and Event-Driven Sagas Standard

DOCUMENT ID
MYTHOS-DST-0001

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

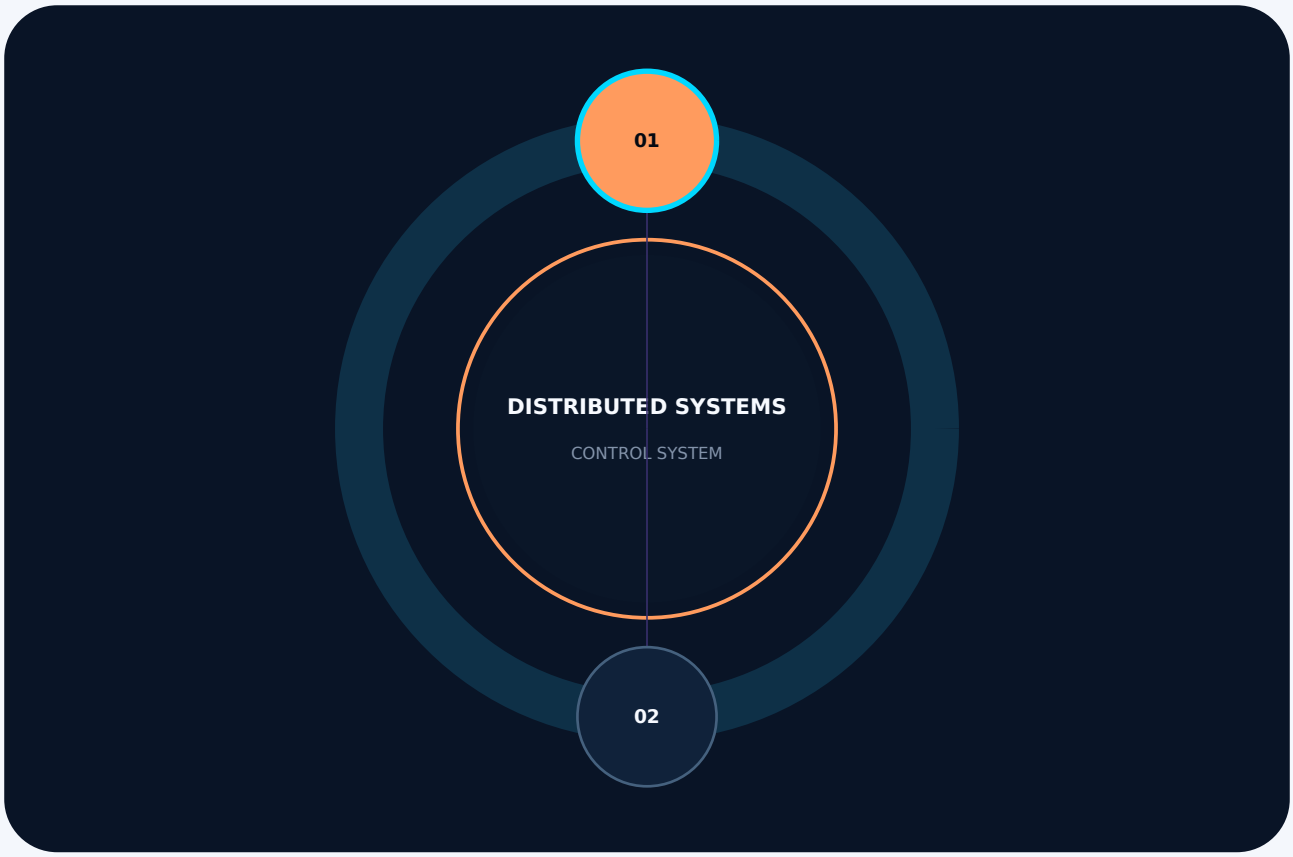
SCHEDULED REVIEW
2027-01-24

12	6	4	1
ATOMIC CRITERIA	ASSURANCE VIEWS	IMPLEMENTATION PROFILES	PERMANENT IDENTITY

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

DOMAIN CONTROL SYSTEM

MYTHOS-DST-0001 inside the distributed systems standard constellation

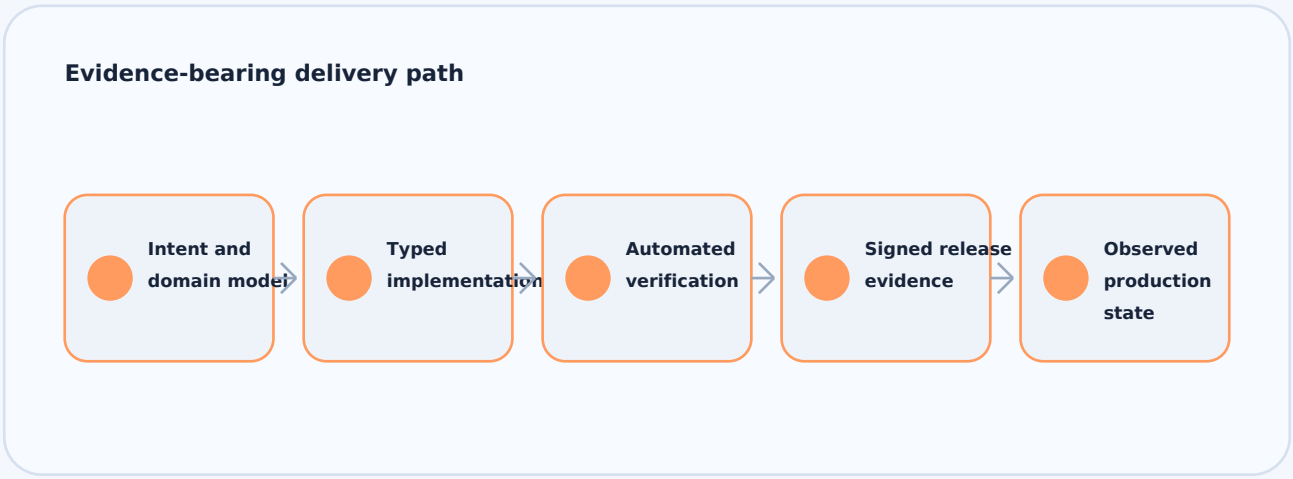


Connected assurance	Architecture, implementation, operations, evidence, and lifecycle are evaluated as one controlled system.
Specialization rule	Companion standards may add precision, but they cannot silently weaken this publication's requirements.

EXECUTIVE POSITION

Durable Workflows, Queues, and Event-Driven Sagas Standard

The architecture of distributed systems has failed.



CONTENTS

Publication navigation

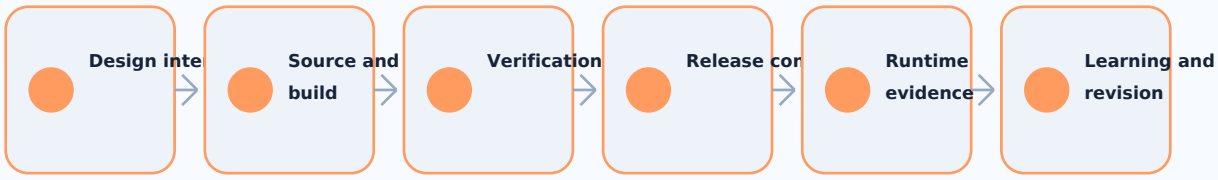
CONTROL TOPOLOGY

Assurance model and evaluation surface

The standard is assessed as an evidence-bearing system. Capability scores do not replace mandatory gates, implementation proof, or operational validation.

Capability family	Evaluation nodes
Durable Execution and Crash Recovery (Weight: 20%)	1
Event-Driven Delivery (Transactional Outbox) (Weight: 20%)	1
Saga and Distributed Transactions (Weight: 20%)	1
Idempotency and Exactly-Once (Weight: 15%)	1
Queue Management and Backpressure (Weight: 15%)	1
Poison Message and DLQ Handling (Weight: 10%)	1
Suite Composition	2
Additional Evaluation Dimension	4

Lifecycle control loop



EVALUATION INDEX

Criteria and evidence map

#	Capability family	Criterion
1	Durable Execution and Crash Recovery (Weight: 20%)	Workflow Resilience
2	Event-Driven Delivery (Transactional Outbox) (Weight: 20%)	Dual-Write Elimination
3	Saga and Distributed Transactions (Weight: 20%)	Cross-Service Consistency
4	Idempotency and Exactly-Once (Weight: 15%)	Message Processing Guarantees
5	Queue Management and Backpressure (Weight: 15%)	Load Shedding
6	Poison Message and DLQ Handling (Weight: 10%)	Fault Isolation
7	Suite Composition	MDSS-Durability
8	Suite Composition	MDSS-Consistency
9	Additional Evaluation Dimension	Workflow Dimensions
10	Additional Evaluation Dimension	The Distributed Systems Doctrine
11	Additional Evaluation Dimension	Scoring Model
12	Additional Evaluation Dimension	Suite Composition

NORMATIVE PUBLICATION

Requirements, evaluation methodology, and implementation guidance

The following body preserves the substantive source standard while normalizing publication status, identity, navigation, and visual presentation.

Executive Mandate

The architecture of distributed systems has failed.

Organizations attempt to coordinate complex, multi-step business logic and autonomous AI agent tasks using synchronous HTTP REST calls and in-memory state. When a downstream service times out, the upstream service is left in a zombie state. When a process crashes mid-execution, the workflow is lost forever. To solve data consistency across microservices, they attempt to implement Two-Phase Commits (2PC), which lock databases and guarantee cascading failures under network partitions.

This standard exists because:

1. Synchronous Coupling is a Distributed Monolith: If Service A cannot complete its task because Service B is offline, you do not have microservices; you have a distributed monolith. Inter-service communication MUST be decoupled via asynchronous message queues or durable orchestration runtimes.
2. In-Memory Workflows are Fragile: An agent or business workflow that relies on in-memory variables to track its progress through a 10-step process will lose all progress the moment its container is OOM-killed or the pod migrates. Execution MUST be durable, persisting state after every step, and resumable from exactly the point of failure.
3. Two-Phase Commits (2PC) are Prohibited: Distributed transactions that lock resources across multiple microservices do not scale and deadlock under network partitions. Systems MUST implement the Saga pattern, where a distributed transaction is broken into a sequence of local transactions, each with a deterministic compensating action to roll back on failure.
4. The Dual-Write Problem is Inevitable: Updating a database and then publishing an event to a message broker is a guaranteed data loss vector. If the broker is offline, the database update is committed, but the event is lost. Systems MUST implement the Transactional Outbox pattern, writing the state change and the event in the *same* local database transaction, with a separate process (CDC) reliably publishing the event.

This document establishes the Mythos Distributed Systems Standard (MDSS), a comprehensive, evidence-based methodology for evaluating distributed workflows against crash resilience, guaranteed delivery, and distributed transactional integrity.

Scope and Applicability

In Scope

This standard covers the evaluation of:

- Durable execution runtimes (Temporal, Restate, DBOS, Cadence)
- Message brokers and queues (Apache Kafka, NATS JetStream, RabbitMQ, SQS)
- Event-driven architecture and the Transactional Outbox pattern (CDC via Debezium)
- Saga patterns (Orchestration vs. Choreography) and compensating transactions
- Idempotency, exactly-once processing, and retry strategies (exponential backoff)
- Dead-letter queues (DLQs) and poison message isolation

Out of Scope

- General API contract design and gRPC (covered in MYTHOS-DST-0002)
- Event sourcing for state derivation (covered in MYTHOS-DAT-0004)
- General software design and DDD (covered in MYTHOS-SWE-0001)

Audience

- Distributed systems engineers and backend architects
- Platform engineers building event-driven infrastructure
- AI engineers building durable, multi-step agent loops
- SREs managing message broker fleets
- Standards bodies seeking a reference distributed systems methodology

Definitions and Taxonomy

Workflow Dimensions

Dimension	Definition
Durable Execution	A runtime paradigm where the execution state (variables, stack, program counter) is persisted after every step, allowing the process to survive crashes and resume seamlessly without re-executing side effects.
Saga	A sequence of local transactions where each transaction updates the database and publishes a message/event to trigger the next step. If a step fails, the saga executes a series of compensating transactions to rollback the preceding steps.
Transactional Outbox	A pattern that solves the dual-write problem by saving domain state changes and outgoing messages in the *same* database transaction, guaranteeing they are committed together, with a separate process reading the outbox to publish to a message broker.
Idempotency	The guarantee that processing the same message multiple times yields the same result as processing it once, achieved via unique message IDs and deduplication logic.
Compensating Transaction	An operation that semantically undoes the effects of a previously committed local transaction (e.g., issuing a refund to undo a payment), since a distributed transaction cannot be physically rolled back.

The Distributed Systems Doctrine

> A synchronous call is a chained failure. An in-memory workflow is a liability. A dual-write is guaranteed data loss. A distributed transaction without a compensating action is data corruption.

Evaluation Methodology

Scoring Model

Each capability is scored from 0 to 5.

Score	Meaning
0	Fails basic task; synchronous chains, in-memory state, 2PC, dual-writes
1	Basic message queues used, but lacks durability or idempotency
2	Functional async architecture, but lacks durable runtime or outbox pattern
3	Solid production performance; durable runtime, outbox pattern, orchestrated sagas, idempotency
4	Strong performance; exactly-once semantics, poisoned queue isolation, deterministic replay
5	Best-in-class; sets the standard for mathematically verified, zero-loss distributed systems

Weighting

Category	Weight	Rationale
Durable Execution & Crash Recovery	20%	In-memory workflows die on crash; state must be durable
Event-Driven Delivery (Outbox)	20%	Dual-writes guarantee eventual data loss
Saga & Distributed Transactions	20%	2PC deadlocks; Sagas with compensations are mandatory
Idempotency & Exactly-Once	15%	At-least-once delivery without idempotency causes double-charges
Queue Management & Backpressure	15%	Unbounded queues cause memory exhaustion and cascading failures
Poison Message & DLQ Handling	10%	A single bad message blocks the entire queue forever

Total: 100%

A system MUST score at least 3.0 in Durable Execution and Event-Driven Delivery to be considered for production use.

Evaluation Categories

Durable Execution and Crash Recovery (Weight: 20%)

Workflow Resilience

Can a multi-step workflow survive a process crash and resume exactly where it left off?

Score	Criteria
0	In-memory execution; state lost on crash; manual intervention required
1	Checkpoints saved periodically to a database, but gaps in state exist
2	Application-level retry logic with idempotency keys
3	Durable execution runtime (Temporal/Restate) persisting state after every step
4	Durable runtime with automatic deadlock detection and timeout recovery
5	Perfect durability: formally verified execution replay, zero loss of in-flight state, sub-second recovery time

Event-Driven Delivery (Transactional Outbox) (Weight: 20%)

Dual-Write Elimination

Is the system mathematically prevented from losing events during a database commit?

Score	Criteria
0	Database updated, then event published via HTTP/broker (Dual-write risk)
1	Best-effort publishing with a background retry queue
2	Transactional Outbox implemented in application code (polling the DB)
3	CDC (Change Data Capture) via Debezium reading DB transaction logs to publish events
4	Exactly-once delivery semantics enforced end-to-end (Kafka transactions)
5	Perfect delivery: formally verified atomic state-and-event commit, zero lost or duplicate events

Saga and Distributed Transactions (Weight: 20%)

Cross-Service Consistency

How are transactions spanning multiple microservices rolled back on failure?

Score	Criteria
0	Two-Phase Commits (2PC) locking databases across services
1	Manual rollback scripts or "hope it works" methodology
2	Choreographed Sagas (event-driven, but hard to track flow)
3	Orchestrated Sagas: Central durable runtime (Temporal) commands services and executes deterministic compensating actions on failure

Score	Criteria
4	Formal verification of the Saga state machine to prove it eventually reaches a consistent state
5	Perfect consistency: formally verified compensating logic, zero stuck states, mathematical proof of eventual consistency

Idempotency and Exactly-Once (Weight: 15%)

Message Processing Guarantees

Can the system process the same message multiple times without side effects?

Score	Criteria
0	At-most-once (fire and forget) or no deduplication
1	At-least-once delivery, but consumers are not idempotent (double-charges)
2	Basic deduplication using a database unique constraint
3	Explicit idempotency keys (e.g., Stripe-style) tracked in a deduplication table
4	Exactly-once processing semantics (consumer transactions tied to message ACKs)
5	Perfect idempotency: formally verified deduplication logic, zero side effects on replay, mathematical proof of exactly-once execution

Queue Management and Backpressure (Weight: 15%)

Load Shedding

How does the system handle a surge in messages that exceeds processing capacity?

Score	Criteria
0	Unbounded queues; memory exhaustion crashes the node
1	Fixed-size queues; messages dropped when full (silent data loss)
2	Basic auto-scaling of consumers, but lag occurs
3	Explicit backpressure mechanisms; upstream producers are throttled or rejected (HTTP 429)
4	Priority-based queueing (critical messages jump the line)
5	Perfect management: formally verified load shedding, zero OOM crashes, mathematical optimization of throughput vs. latency

Poison Message and DLQ Handling (Weight: 10%)

Fault Isolation

What happens when a message cannot be processed due to malformed data or a logic bug?

Score	Criteria
0	Infinite retry loop; the poison message blocks the entire queue forever
1	Manual operator intervention required to delete the message
2	Max-retry limit, but message is dropped/deleted after limit reached

Score	Criteria
3	Automated Dead-Letter Queue (DLQ): after N retries, message moved to DLQ for analysis, queue continues processing
4	Automated alerting on DLQ depth; automated replay mechanism once the logic bug is fixed
5	Perfect isolation: formally verified poison detection, zero queue blockages, cryptographic provenance of failed messages

The Mythos Distributed Systems Suite (MDSS)

Traditional backend benchmarks measure API throughput (RPS). The MDSS evaluates crash recovery, dual-write elimination, and saga consistency.

Suite Composition

MDSS-Durability

1. Kill Test: 100+ tests killing the executing process at every possible step in a 10-step saga, verifying the durable runtime resumes it without dropping state.
1. Dual-Write Injection: 50+ tests simulating a network partition between the database and the message broker, verifying the Outbox/CDC pattern prevents data loss.

MDSS-Consistency

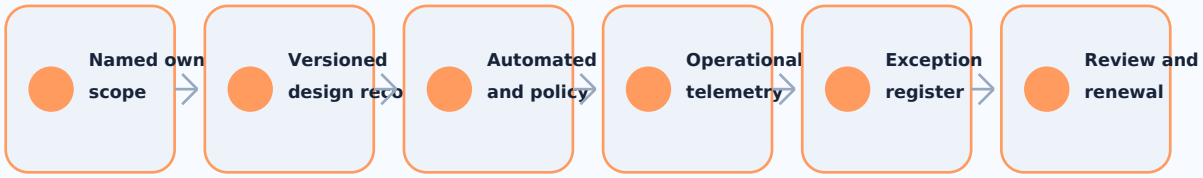
1. Saga Failure Test: 100+ tests forcing a failure at the final step of a distributed transaction, verifying all preceding services execute their compensating transactions correctly.
1. Poison Message Test: 50+ tests injecting a malformed JSON payload into the queue, verifying it is routed to the DLQ after 3 retries without blocking valid messages.

Execution Protocol

ASSURANCE AND ADOPTION

Implementation evidence and review gates

Required assurance chain



A passing score is not sufficient without current evidence. High-assurance adoption requires traceable design decisions, reproducible builds, independently reviewable tests, controlled secrets, runtime observability, failure exercises, and a dated exception register.

Assurance maturity spectrum

Current authority register

Authority	Version / snapshot	Applicability	Review by
TLA+ Foundation: TLA+ Language and Tools Documentation	Rolling documentation snapshot	High	2027-01-24
OpenAPI Initiative: OpenAPI Specification	3.2.0	High	2027-01-24
AsyncAPI Initiative: AsyncAPI Specification	3.1.0	High	2027-01-24
Kubernetes Project: Kubernetes Documentation	v1.36 current documentation	High	2026-10-22
OpenTelemetry Project: OpenTelemetry Documentation and Specification	Rolling specification snapshot; profiles public alpha and declarative configuration stable in 2026	High	2026-10-22
Cloud Native Computing Foundation: Cloud Native Landscape and Project Documentation	Rolling catalog snapshot	Moderate	2026-08-23

Research limitation. Official documentation establishes published interfaces and declared behavior. It does not prove the performance, security, interoperability, cost, or operational suitability of a specific implementation. Those claims require controlled testing and retained evidence.