ENGINEERING STANDARDS LIBRARY / PLATFORM ENGINEERING

Internal Developer Platform, GitOps, and Policy-as-Code Standard

Golden paths, self-service platforms, GitOps reconciliation, policy enforcement, developer experience, and platform product management.

PERMANENT DOCUMENT ID

MYTHOS-PLT-0003

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

Internal Developer Platform, GitOps, and Policy-as-Code Standard

DOCUMENT ID
MYTHOS-PLT-0003

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

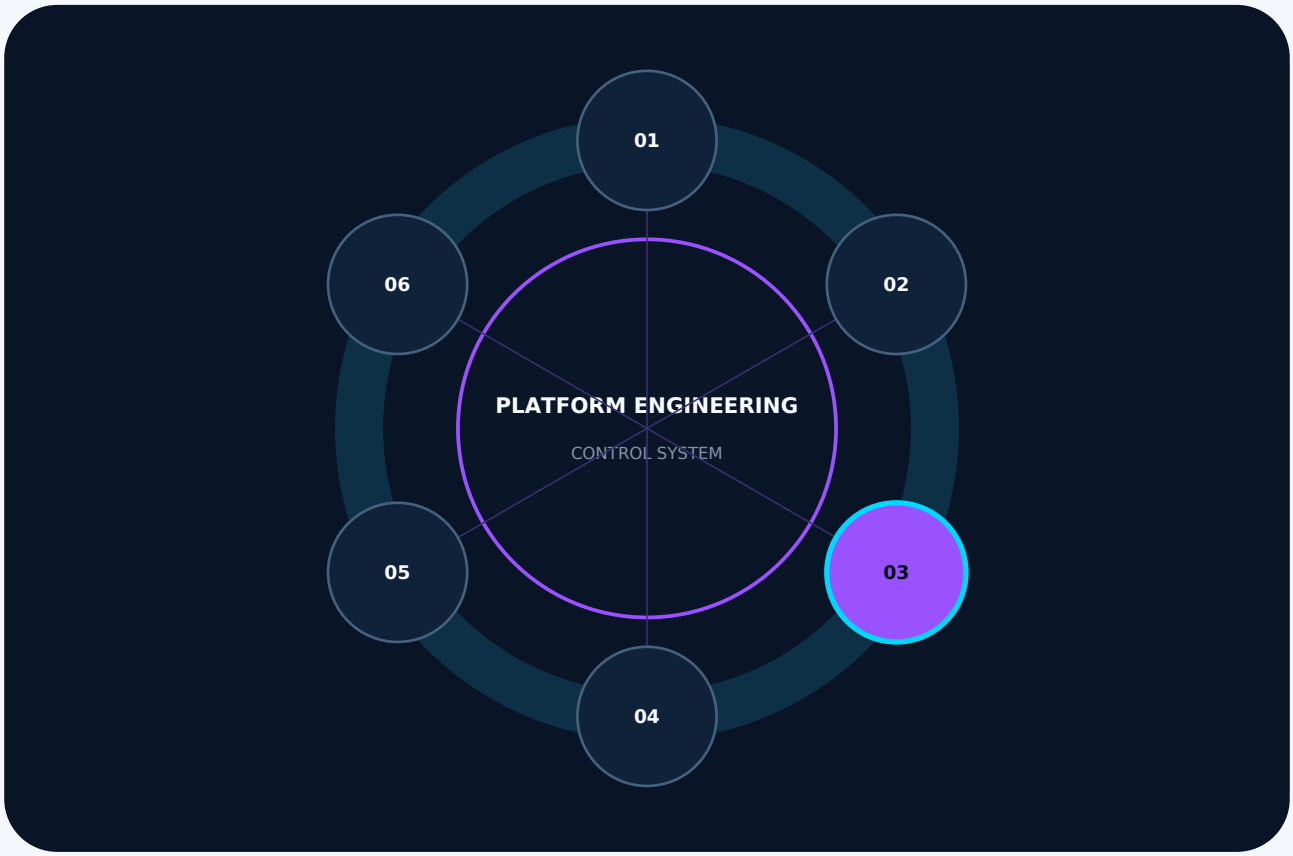
SCHEDULED REVIEW
2027-01-24

11	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-PLT-0003 inside the platform engineering standard constellation

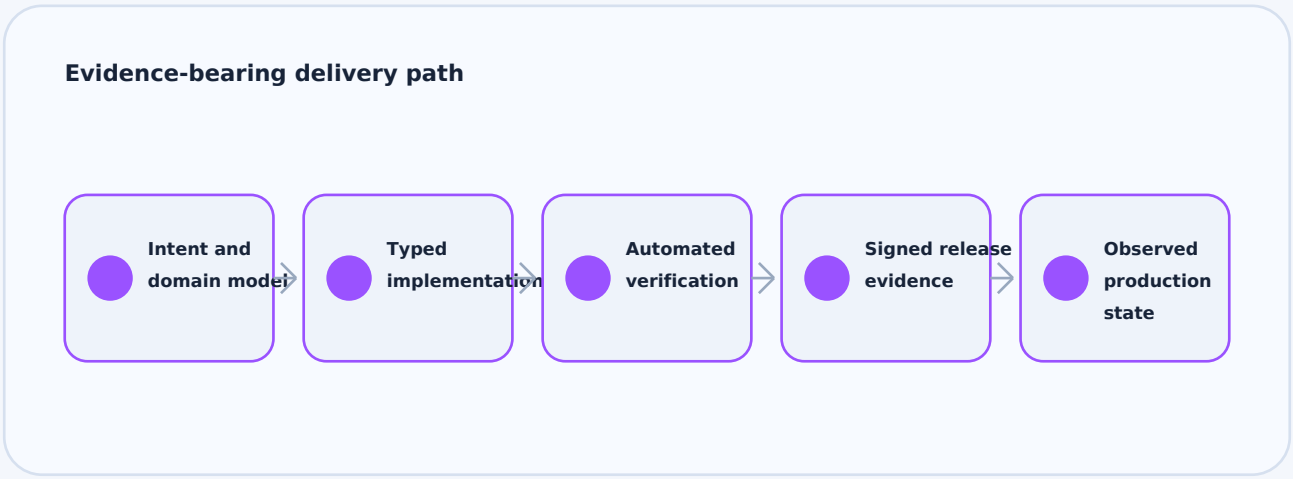


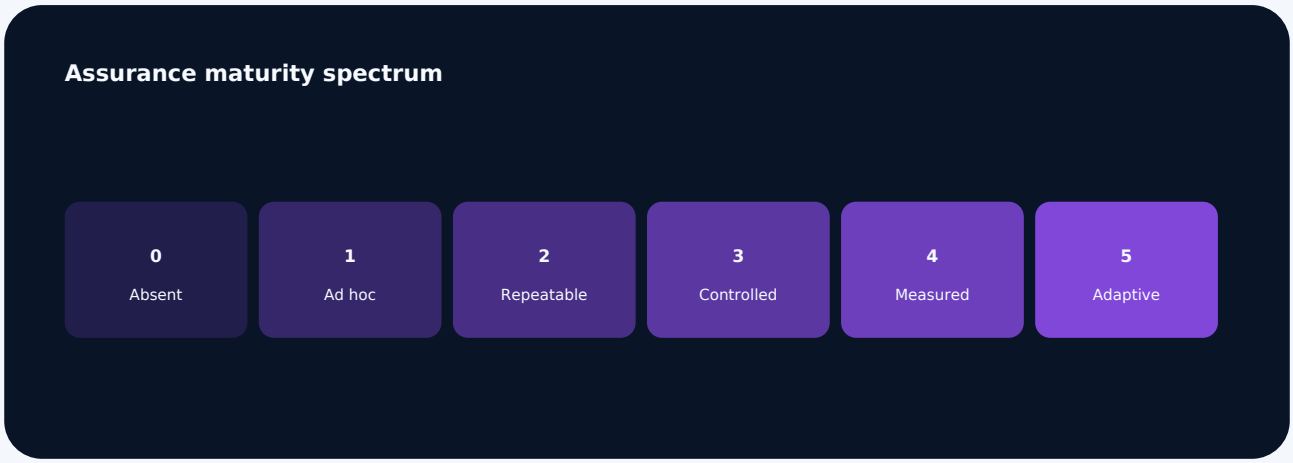
Connected assurance	Architecture, implementation, operations, evidence, and lifecycle are evaluated as one controlled system.
Specialization rule	Companion standards may add precision, but they cannot silently weaken this publication's requirements.

EXECUTIVE POSITION

Internal Developer Platform, GitOps, and Policy-as-Code Standard

The architecture of developer experience has failed.





CONTENTS

Publication navigation

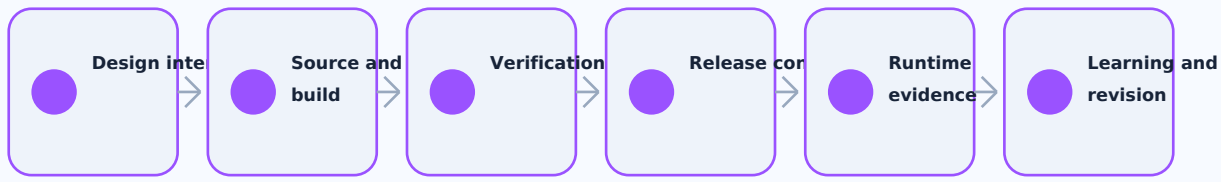
CONTROL TOPOLOGY

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
Golden Path Automation (Weight: 20%)	1
Self-Service Provisioning API (Weight: 20%)	1
Policy-as-Code (PaC) Guardrails (Weight: 20%)	1
Cognitive Load and Abstraction (Weight: 15%)	1
GitOps Reconciliation (Weight: 15%)	1
Platform Observability and DevEx (Weight: 10%)	1
Suite Composition	2
Additional Evaluation Dimension	3

Lifecycle control loop



EVALUATION INDEX

Criteria and evidence map

#	Capability family	Criterion
1	Golden Path Automation (Weight: 20%)	Scaffolding Rigor
2	Self-Service Provisioning API (Weight: 20%)	Infrastructure On-Demand
3	Policy-as-Code (PaC) Guardrails (Weight: 20%)	Automated Compliance
4	Cognitive Load and Abstraction (Weight: 15%)	Developer Experience
5	GitOps Reconciliation (Weight: 15%)	State Synchronization
6	Platform Observability and DevEx (Weight: 10%)	Platform Health
7	Suite Composition	MIDPS-Provisioning
8	Suite Composition	MIDPS-Policy
9	Additional Evaluation Dimension	Platform Dimensions
10	Additional Evaluation Dimension	The Platform Doctrine
11	Additional Evaluation Dimension	Scoring Model

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 developer experience has failed.

Organizations attempt to scale engineering teams by throwing Kubernetes and AWS over the wall to developers. When developers struggle with the immense complexity of configuring networking, storage, IAM, and CI/CD, the organization builds a static "Service Catalog" wiki. Developers must still write hundreds of lines of YAML, open a Jira ticket, and wait for a platform engineer to manually review and provision the infrastructure.

This standard exists because:

1. A Static Catalog is a Bookmark, Not a Platform: An IDP that merely lists available services and links to documentation is not a platform. A true IDP MUST provide "Golden Paths"—one-click, code-generated scaffolding that instantly provisions a fully compliant, production-ready environment (code repo, CI/CD, monitoring, IAM) via API.
2. Self-Service Without Guardrails is Chaos: Allowing developers to provision any resource they want via self-service guarantees cost overruns, security vulnerabilities, and architectural drift. Systems MUST enforce Policy-as-Code (PaC) as non-bypassable, automated guardrails during the provisioning process.
3. Ticket-Driven Provisioning is a Bottleneck: If a developer must wait three days for a platform team to manually merge a Terraform PR, the platform is a failure. Provisioning MUST be asynchronous, API-driven, and completed in minutes, not days.
4. Platform Abstraction Must Hide Complexity: Forcing developers to learn cloud-specific APIs (AWS VPCs, Subnets, Route Tables) violates the Domain-Driven Design principle of bounded contexts. The IDP MUST expose developer-friendly abstractions (e.g., "Postgres Database") and map them to infrastructure components via a control plane like Crossplane.

This document establishes the Mythos IDP Standard (MIDPS), a comprehensive, evidence-based methodology for evaluating Internal Developer Platforms against automation rigor, policy enforcement, and developer cognitive load reduction.

Scope and Applicability

In Scope

This standard covers the evaluation of:

1. Internal Developer Platforms (Spotify Backstage, Humanitec, Port, OpsLevel)

1. Golden Path engineering and code scaffolding (e.g., Yeoman, Copilot templates)
1. GitOps continuous reconciliation (ArgoCD, Flux)
1. Policy-as-Code engines (Open Policy Agent, Kyverno, HashiCorp Sentinel)
1. Kubernetes-native control planes (Crossplane) as IDP backends
1. Developer Experience (DevEx) metrics and platform observability

Out of Scope

1. General infrastructure provisioning rules (covered in MYTHOS-PLT-0002)
1. General multi-cloud architecture (covered in MYTHOS-CLD-0001)
1. General CI/CD supply chain security (covered in MYTHOS-PLT-0001)

Audience

1. Platform engineers and architects designing internal developer platforms
1. DevOps and SRE teams managing deployment pipelines
1. Security engineers designing shift-left compliance guardrails
1. Engineering managers measuring Developer Experience (DevEx)
1. Standards bodies seeking a reference platform engineering methodology

Definitions and Taxonomy

Platform Dimensions

Dimension	Definition
Golden Path	An opinionated, heavily automated, and fully supported architectural template that allows developers to build and deploy services quickly without needing to understand the underlying infrastructure complexity.
Policy-as-Code (PaC)	The practice of writing security, compliance, and operational rules in a domain-specific language (e.g., Rego), enforced automatically by the platform during the provisioning and deployment phases.
Cognitive Load	The amount of mental effort required by a developer to deploy and operate a service. The primary goal of an IDP is to minimize this load by abstracting away infrastructure complexity.
GitOps Reconciliation	An operational model where an in-cluster controller continuously pulls desired state from Git and applies it, ensuring the live environment strictly matches the declared source of truth.
Platform Orchestrator	A system (like Crossplane or Humanitec) that translates developer-friendly abstractions ("I need a database") into the actual cloud-specific API calls and configurations required to provision it.

The Platform Doctrine

> A wiki is not a platform. A Jira ticket is not a provisioning API. Self-service without policy is a breach. If the developer has to write infrastructure YAML, the abstraction has failed.

Evaluation Methodology

Scoring Model

Each capability is scored from 0 to 5.

Score	Meaning
0	Fails basic task; manual YAML, ticket-based provisioning, no policies, high cognitive load
1	Basic service catalog exists, but provisioning is still manual
2	Functional CI/CD templates, but lacks self-service provisioning or PaC
3	Solid production performance; Golden Paths, self-service API, PaC guardrails, GitOps
4	Strong performance; Crossplane abstractions, zero infrastructure YAML for developers, automated PaC
5	Best-in-class; sets the standard for mathematically verified, zero-friction developer platforms

Weighting

Category	Weight	Rationale
Golden Path Automation	20%	Manual scaffolding leads to inconsistency and security misconfigurations
Self-Service Provisioning API	20%	Ticket-driven provisioning destroys developer velocity
Policy-as-Code (PaC) Guardrails	20%	Self-service without automated security guardrails is reckless
Cognitive Load & Abstraction	15%	Forcing developers to learn cloud APIs breaks domain isolation
GitOps Reconciliation	15%	Drift between developer intent and production causes outages
Platform Observability & DevEx	10%	The platform itself must be observable and measured

Total: 100%

A system MUST score at least 3.0 in Golden Path Automation and Policy-as-Code to be considered for production use.

Evaluation Categories

Golden Path Automation (Weight: 20%)

Scaffolding Rigor

Can a developer instantiate a fully compliant, production-ready service in minutes?

Score	Criteria
0	Developers manually create repos, CI/CD files, and Dockerfiles from memory
1	Basic templates exist in Git, but require manual copy-paste and modification

Score	Criteria
2	IDP provides "Create Service" button, but only generates a basic repo
3	IDP scaffolding generates code, CI/CD, monitoring dashboards, and IAM roles automatically
4	Scaffolding dynamically adapts based on developer input (e.g., adding a DB triggers DB-provisioning IaC)
5	Perfect automation: formally verified Golden Paths, zero manual post-creation configuration, cryptographic attestation of template compliance

Self-Service Provisioning API (Weight: 20%)

Infrastructure On-Demand

How does a developer attach infrastructure (e.g., a database) to their service?

Score	Criteria
0	Open a Jira ticket; wait for DevOps to manually run Terraform
1	Submit a PR to a central Terraform repo; wait for review
2	Developer runs `terraform apply` from their service repo (requires cloud credentials)
3	Developer declares intent in their service manifest (e.g., `requires: postgres`); platform orchestrator provisions it asynchronously
4	Self-service portal allows dynamic resource scaling and environment cloning via API
5	Perfect self-service: formally verified provisioning bounds, zero platform team intervention, sub-minute API response times

Policy-as-Code (PaC) Guardrails (Weight: 20%)

Automated Compliance

Are security and compliance rules enforced automatically, without manual review?

Score	Criteria
0	No automated policies; relies on manual PR reviews
1	Basic CI checks (e.g., linting), but no infrastructure policy enforcement
2	Post-deployment scanning alerts on non-compliant resources (shift-right)
3	Shift-left PaC: OPA/Kyverno evaluates developer manifests during PR and blocks merges
4	Admission control: even if merged, the platform orchestrator refuses to provision non-compliant resources
5	Perfect enforcement: formally verified policy bounds, zero ability to bypass guardrails, mathematical proof of continuous compliance

Cognitive Load and Abstraction (Weight: 15%)

Developer Experience

Is the developer shielded from cloud-specific infrastructure complexity?

Score	Criteria
0	Developers must write AWS/Azure-specific Terraform or YAML
1	Internal modules exist, but developers must still understand networking and IAM
2	Platform exposes custom CRDs, but they leak cloud-specific concepts
3	Platform exposes pure domain abstractions (e.g., `PostgresCluster`, `RedisCache`); zero cloud API leakage
4	Developer UI requires zero YAML editing; entirely form-driven or API-driven
5	Perfect abstraction: formally verified domain isolation, zero cloud-specific cognitive load, seamless multi-cloud portability

GitOps Reconciliation (Weight: 15%)

State Synchronization

Does the platform continuously reconcile the live environment to the developer's declared intent?

Score	Criteria
0	CI/CD pipelines push changes; drift goes unnoticed
1	GitOps controllers (ArgoCD) manage Kubernetes state, but not underlying cloud infra
2	GitOps manages Kubernetes; separate Terraform pipeline manages cloud infra (disconnected)
3	Unified GitOps: ArgoCD/Flux reconciles both Kubernetes state and Crossplane cloud resources
4	Automated drift remediation: if cloud state is modified manually, platform reverts it to match Git
5	Perfect reconciliation: formally verified state parity, zero divergence between developer intent and production

Platform Observability and DevEx (Weight: 10%)

Platform Health

Is the IDP itself treated as a critical product with its own SLAs and metrics?

Score	Criteria
0	No visibility into platform health; developers don't know why provisioning fails
1	Basic logs for platform operators
2	Status page shows platform uptime
3	DORA metrics tracked per team; platform provides visibility into lead time and deployment frequency
4	Developer satisfaction (DevEx) surveys automated and tracked via platform metrics
5	Perfect observability: formally verified platform SLAs, real-time provisioning telemetry, mathematical proof of developer velocity

The Mythos IDP Suite (MIDPS)

Traditional benchmarks measure platform cost savings. The MIDPS evaluates provisioning latency, policy enforcement, and cognitive load reduction.

Suite Composition

MIDPS-Provisioning

1. Golden Path Timer: 100+ tests measuring the time from "Create Service" button click to a running, monitored staging environment, verifying it takes < 10 minutes.
1. Dependency Resolution Test: 50+ tests requesting a service with a database dependency, verifying the platform orchestrator provisions both without developer intervention.

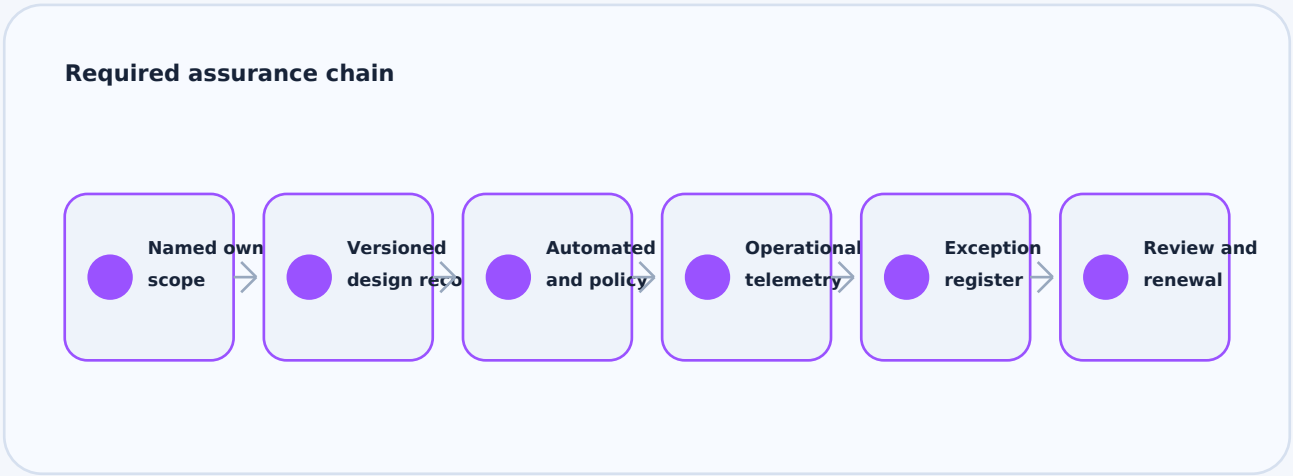
MIDPS-Policy

1. Non-Compliant Request Test: 100+ tests attempting to provision an public-facing S3 bucket via the developer API, verifying the PaC guardrails block the request instantly.
1. Drift Reversion Test: 50+ tests manually deleting a cloud resource out-of-band, verifying the GitOps controller automatically recreates it within 5 minutes.

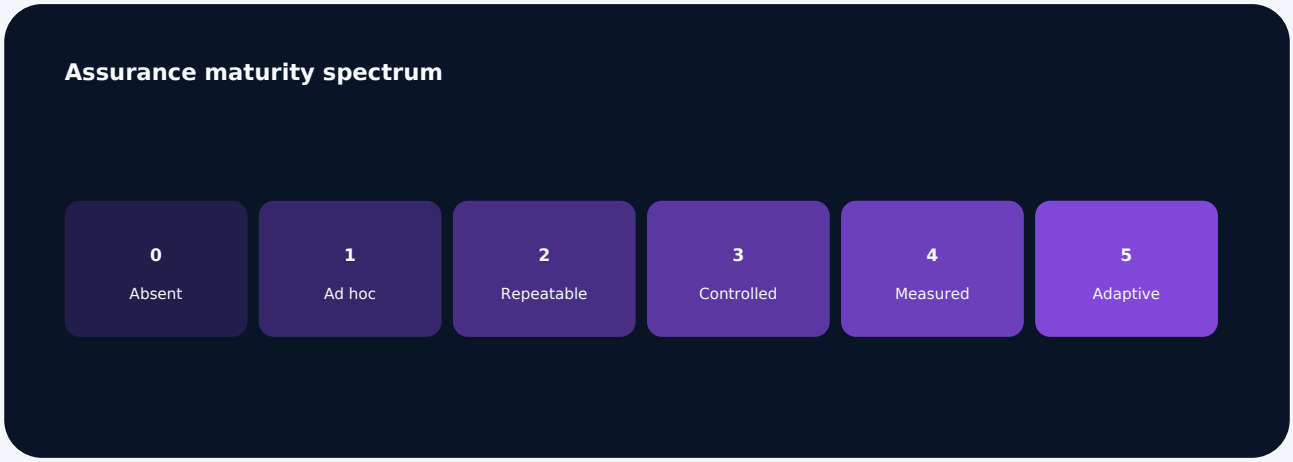
Execution Protocol

ASSURANCE AND ADOPTION

Implementation evidence and review gates



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.



Current authority register

Authority	Version / snapshot	Applicability	Review by
Kubernetes Project: Kubernetes Documentation	v1.36 current documentation	High	2026-10-22
SLSA Project: Supply-chain Levels for Software Artifacts Specification	1.2 current specification snapshot	Moderate	2026-10-22
SPDX Project: SPDX Specification	3.0.1 stable; 3.1-RC1 under review	High	2026-10-22
OWASP CycloneDX: CycloneDX Bill of Materials Standard	1.7	High	2027-01-24
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.