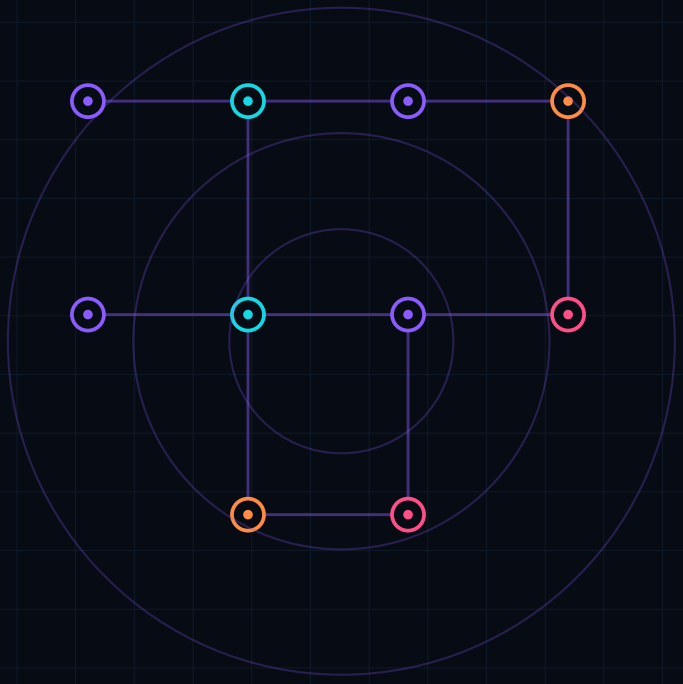
MYTHOS SYSTEMS

ENGINEERING STANDARDS LIBRARY / PORTFOLIO EDITION

Platform Engineering Standards Portfolio

Release engineering, supply chain, infrastructure as code, developer platforms, lifecycle, and governed extension ecosystems.



PERMANENT DOCUMENT ID
MYTHOS-PLT-PORT-0001

PUBLICATION
Candidate Portfolio Edition

VERSION
1.0.0-candidate

PERMANENT PUBLICATION IDENTITY

Platform Engineering Standards Portfolio

DOCUMENT ID
MYTHOS-PLT-PORT-0001

VERSION
1.0.0-candidate

STATUS
Candidate Portfolio Edition

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-09-17

CLASSIFICATION
Public

6	71	6	1
STANDARDS	CRITERIA	ATOMIC SOURCES	PERMANENT IDENTITY

Portfolio doctrine. This generated reader view does not replace its atomic source publications. Permanent IDs, evidence boundaries, assessment scope, freshness, and revision history remain authoritative at the atomic publication level.

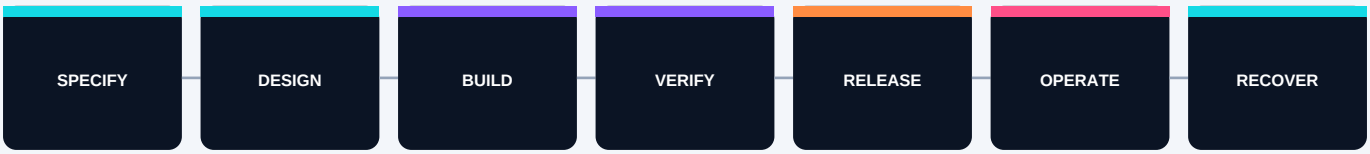
Publication domains

- DevSecOps and supply-chain assurance
- Infrastructure and configuration as code
- Internal developer platforms and GitOps
- Managed platform evaluation
- Lifecycle and extension architecture

PORTFOLIO READING MAP

From code to verified recovery

A visual navigation model for the software, platform, state, and reliability lifecycle.



STATE & DATA

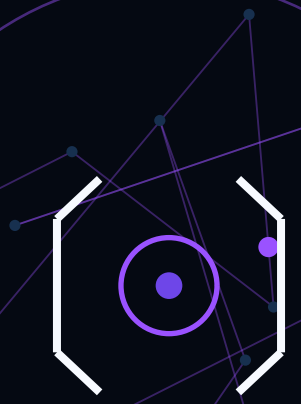
contracts -> events -> durable state -> replicas -> storage -> evidence

SERVICE TOPOLOGY

API -> service mesh -> queues -> workers -> dependencies -> users

RELIABILITY

SLI -> SLO -> error budget -> capacity -> failure test -> recovery proof



ENGINEERING STANDARDS LIBRARY / PLATFORM ENGINEERING

DevSecOps, Release Engineering, and Supply Chain Standard

Reproducible builds, artifact provenance, controlled promotion, software supply-chain assurance, and evidence-bearing releases.

PERMANENT DOCUMENT ID

MYTHOS-PLT-0001

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

DevSecOps, Release Engineering, and Supply Chain Standard

DOCUMENT ID
MYTHOS-PLT-0001

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

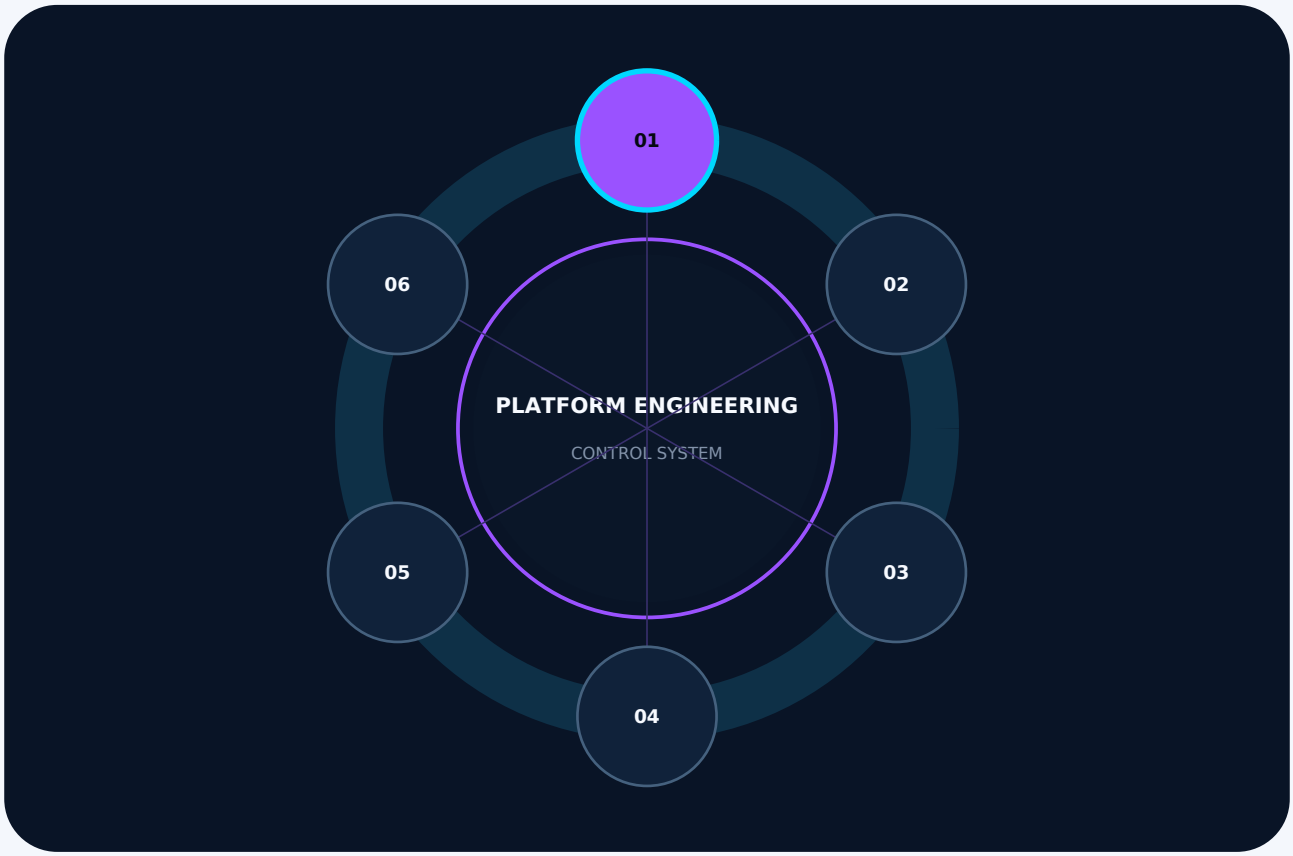
SCHEDULED REVIEW
2027-01-24

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-PLT-0001 inside the platform engineering standard constellation



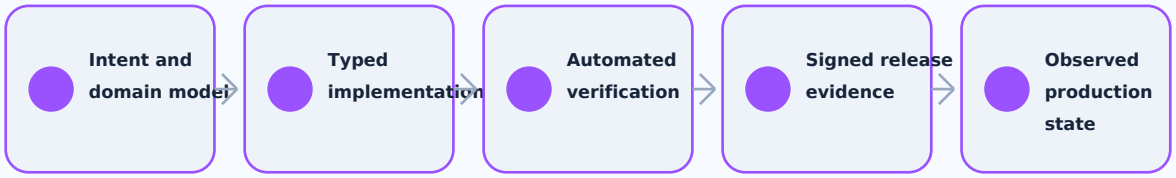
Connected assurance	Architecture, implementation, operations, evidence, and lifecycle are evaluated as one controlled system.
Specialization rule	Companion standards may add precision, but they cannot silently weaken this publication's requirements.

EXECUTIVE POSITION

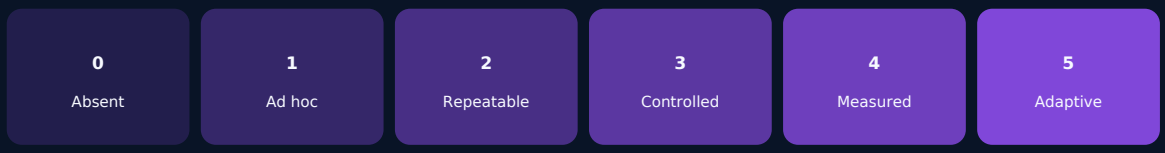
DevSecOps, Release Engineering, and Supply Chain Standard

The architecture of software delivery has failed.

Evidence-bearing delivery path



Assurance maturity spectrum



CONTENTS

Publication navigation

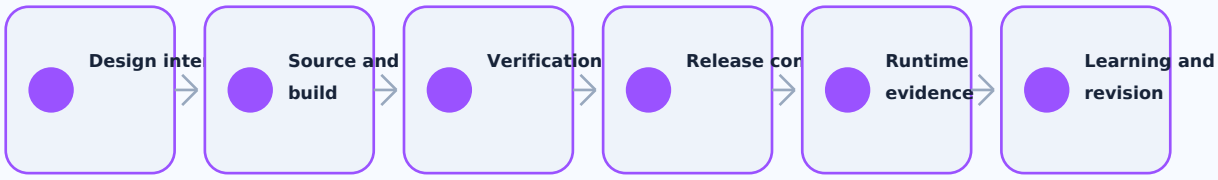
CONTROL TOPOLOGY

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
Build Isolation and Runner Security (Weight: 20%)	1
Provenance and SLSA Compliance (Weight: 20%)	1
Dependency Management and Confusion (Weight: 15%)	1
SBOM/AIBOM Generation and Verification (Weight: 15%)	1
Artifact Signing and Transparency (Weight: 15%)	1
Shift-Left Security and Policy Gates (Weight: 15%)	1
Suite Composition	2
Additional Evaluation Dimension	4

Lifecycle control loop



EVALUATION INDEX

Criteria and evidence map

#	Capability family	Criterion
1	Build Isolation and Runner Security (Weight: 20%)	Build Environment Integrity
2	Provenance and SLSA Compliance (Weight: 20%)	Artifact Integrity
3	Dependency Management and Confusion (Weight: 15%)	Resolution Security
4	SBOM/AIBOM Generation and Verification (Weight: 15%)	Software Transparency
5	Artifact Signing and Transparency (Weight: 15%)	Trust Verification
6	Shift-Left Security and Policy Gates (Weight: 15%)	Continuous Enforcement
7	Suite Composition	MSCS-Isolation
8	Suite Composition	MSCS-Provenance
9	Additional Evaluation Dimension	Supply Chain Dimensions
10	Additional Evaluation Dimension	The DevSecOps 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 software delivery has failed.

Organizations build critical infrastructure using CI/CD pipelines that pull arbitrary dependencies from the public internet at build time. They use long-lived, static credentials on shared build runners. They treat the build server as a trusted environment, and when a sophisticated attacker (e.g., SolarWinds, `xz-utils`) injects malicious code during the compilation phase, the resulting artifacts are signed and deployed blindly across the enterprise.

This standard exists because:

1. The Build Server is the Crown Jewel: If an attacker compromises the CI/CD runner, they compromise every artifact it produces. Build environments MUST be ephemeral, isolated (hermetic), and possess zero outbound internet access during the compilation step.
2. Provenance Must Be Cryptographic, Not Implicit: Trusting a binary because "it came from our Jenkins server" is a fatal flaw. Every artifact MUST carry cryptographically signed, in-toto attestations detailing exactly how it was built, what sources were used, and what dependencies were resolved (SLSA Level 3+).
3. Public Registries are Hostile Territory: Package managers that fall back to public registries (npm, PyPI) for internal package names are vulnerable to Dependency Confusion attacks. Builds MUST resolve dependencies exclusively from verified, internal, private mirrors.
4. Security is Not a Final Gate: Scanning a compiled binary for CVEs right before deployment is too late. Security and policy enforcement MUST be shifted left, occurring at the PR stage and continuously during the build process, blocking non-compliant code from ever reaching the registry.
5. AI Models are Supply Chain Artifacts: Downloading unverified `.gguf` or `.safetensors` files from HuggingFace is the equivalent of running `curl | bash`. AI model weights MUST be treated as untrusted binaries, requiring SBOMs (AIBOMs), vulnerability scanning, and cryptographic signing.

This document establishes the Mythos Supply Chain Standard (MSCS), a comprehensive, evidence-based methodology for evaluating DevSecOps and release engineering pipelines against hermetic isolation, provenance verification, and zero-trust dependency management.

Scope and Applicability

In Scope

This standard covers the evaluation of:

1. CI/CD pipelines and release engineering (GitHub Actions, GitLab CI, Jenkins, Tekton)
1. Supply-chain Levels for Software Artifacts (SLSA) framework compliance
1. Software Bill of Materials (SBOM) and AI Bill of Materials (AIBOM) generation
1. Cryptographic artifact signing and transparency logs (Sigstore, Cosign, Rekor)
1. Hermetic builds and dependency confusion prevention
1. Ephemeral runner architecture and OIDC federation (zero static secrets)

Out of Scope

1. General infrastructure-as-code (covered in MYTHOS-PLT-0002)
1. General container runtime security (covered in MYTHOS-SEC-0012)
1. Developer identity and passkeys (covered in MYTHOS-ID-0001)

Audience

1. DevSecOps and platform engineers building CI/CD infrastructure
1. Security architects evaluating supply chain threats
1. Release engineers managing artifact registries
1. Open-source maintainers securing public packages
1. Standards bodies seeking a reference supply chain methodology

Definitions and Taxonomy

Supply Chain Dimensions

Dimension	Definition
SLSA (Supply-chain Levels for Software Artifacts)	A security framework providing a graduated set of requirements ensuring the integrity of software artifacts throughout the software supply chain.
Hermetic Build	A build process that is completely isolated from the internet and external state, fetching source code and dependencies only from verified, internal, cached mirrors.
Provenance Attestation	A cryptographically signed metadata file (in-toto format) generated by the build platform, attesting to the exact source, build parameters, and dependencies used to create an artifact.
Dependency Confusion	A supply chain attack where a malicious package with the same name as an internal private package is published to a public registry, tricking the build into downloading the malicious code.
SBOM/AIBOM	Software/AI Bill of Materials. A formal, machine-readable inventory of software components, dependencies, and (for AI) model weights, training data provenance, and licenses.

The DevSecOps Doctrine

> A build server with internet access is a compromised server. A binary without provenance is a Trojan horse. A public package is a stranger. If the dependency resolution is not isolated, the supply chain is poisoned.

Evaluation Methodology

Scoring Model

Each capability is scored from 0 to 5.

Score	Meaning
0	Fails basic task; shared runners, static secrets, internet access during build, no SBOMs
1	Basic CI/CD with automated tests, but no supply chain controls
2	Functional scanning, but relies on implicit trust of the build server
3	Solid production performance; hermetic builds, SLSA L3, Sigstore signing, internal registry mirrors
4	Strong performance; reproducible builds, policy-as-code gates, AIBOMs for AI models
5	Best-in-class; sets the standard for mathematically verified, zero-trust software delivery

Weighting

Category	Weight	Rationale
Build Isolation & Runner Security	20%	Shared runners with internet access are the primary attack vector
Provenance & SLSA Compliance	20%	Implicit trust in binaries allows compromised artifacts to deploy
Dependency Management & Confusion	15%	Public registry fallbacks guarantee eventual code injection
SBOM/AIBOM Generation & Verification	15%	You cannot secure what you do not inventory
Artifact Signing & Transparency	15%	Unsigned artifacts can be tampered with in the registry
Shift-Left Security & Policy Gates	15%	Scanning at deployment is too late; must block at PR/Build

Total: 100%

A system MUST score at least 3.0 in Build Isolation and Provenance to be considered for production use.

Evaluation Categories

Build Isolation and Runner Security (Weight: 20%)

Build Environment Integrity

Is the CI/CD pipeline treated as a highly privileged, untrusted environment?

Score	Criteria
0	Shared, persistent build runners; long-lived SSH keys; outbound internet access
1	Private runners, but run as root with internet access
2	Ephemeral runners spun up per job, but still possess outbound internet access
3	Hermetic builds: runners have zero outbound internet; dependencies fetched from internal mirrors
4	Ephemeral runners with OIDC federation (zero static secrets); strict seccomp/AppArmor profiles
5	Perfect isolation: formally verified build boundaries, zero network egress capability, cryptographic attestation of runner OS state

Provenance and SLSA Compliance (Weight: 20%)

Artifact Integrity

Can the organization cryptographically prove how, where, and from what an artifact was built?

Score	Criteria
0	No provenance; binaries trusted based on "which Jenkins job built it"
1	Basic build logs retained, but easily forgeable
2	SLSA Level 1-2: Provenance generated, but not cryptographically signed
3	SLSA Level 3: Signed in-toto provenance bound to the artifact; build platform isolates jobs
4	SLSA Level 4: Reproducible builds verified by two independent platforms; provenance strictly enforced at deployment
5	Perfect integrity: formally verified build pipeline, zero ability to deploy artifacts without valid provenance, mathematical proof of source-to-binary equivalence

Dependency Management and Confusion (Weight: 15%)

Resolution Security

Is the build system protected against public registry poisoning?

Score	Criteria
0	Build pulls dependencies directly from public internet (npm, PyPI)
1	Private registry used, but falls back to public if package not found
2	Public fallback disabled for internal namespaces, but no vulnerability scanning
3	Strict scoped registries: all builds resolve *only* from internal, cached mirrors; zero public access

Score	Criteria
4	Continuous vulnerability scanning of the internal mirror; auto-quarantine of compromised packages
5	Perfect management: formally verified dependency graphs, zero typosquatting potential, cryptographic hashing of all resolved dependencies

SBOM/AIBOM Generation and Verification (Weight: 15%)

Software Transparency

Is there a machine-readable inventory of every component, including AI models?

Score	Criteria
0	No inventory; dependencies tracked manually in READMEs
1	Basic SBOM generated for release artifacts, but not verified
2	SBOM (CycloneDX/SPDX) generated in CI; stored alongside artifact
3	AIBOM generated for AI models (tracking weights, training data licenses); SBOMs verified at deployment gate
4	Dependency drift detection: build fails if SBOM changes without PR approval
5	Perfect transparency: formally verified component inventory, zero undocumented dependencies, cryptographic binding of SBOM to artifact hash

Artifact Signing and Transparency (Weight: 15%)

Trust Verification

Are artifacts cryptographically signed and logged publicly?

Score	Criteria
0	Unsigned artifacts deployed directly to production
1	Artifacts signed with long-lived GPG keys
2	Keyless signing via Sigstore/Cosign using OIDC identity
3	Transparency log (Rekor) integration for all signed artifacts
4	Runtime (Kubernetes/Host) refuses to execute artifacts lacking a valid Rekor entry
5	Perfect trust: formally verified signature propagation, zero ability to deploy unsigned code, cryptographic proof of audit trail

Shift-Left Security and Policy Gates (Weight: 15%)

Continuous Enforcement

Is security a blocking gate throughout the development lifecycle?

Score	Criteria
0	Security scanned manually right before release
1	Basic SAST/DAST running in CI, but non-blocking
2	Scanning blocks PRs on critical vulnerabilities

Score	Criteria
3	Policy-as-code (OPA) evaluating SBOMs, licenses, and SLSA provenance in CI
4	Developer IDE integration preventing insecure code from ever being committed
5	Perfect enforcement: formally verified policy bounds, zero ability to bypass security gates, mathematical proof of release readiness

The Mythos Supply Chain Suite (MSCS)

Traditional DevOps benchmarks measure build throughput (builds per minute). The MSCS evaluates hermetic isolation, provenance verification, and dependency confusion resistance.

Suite Composition

MSCS-Isolation

1. Egress Test: 100+ tests attempting to `curl` external endpoints during the build script, verifying the hermetic runner blocks the connection.
1. Secret Persistence Test: 50+ tests checking for static credentials in the runner environment, verifying OIDC federation is strictly enforced.

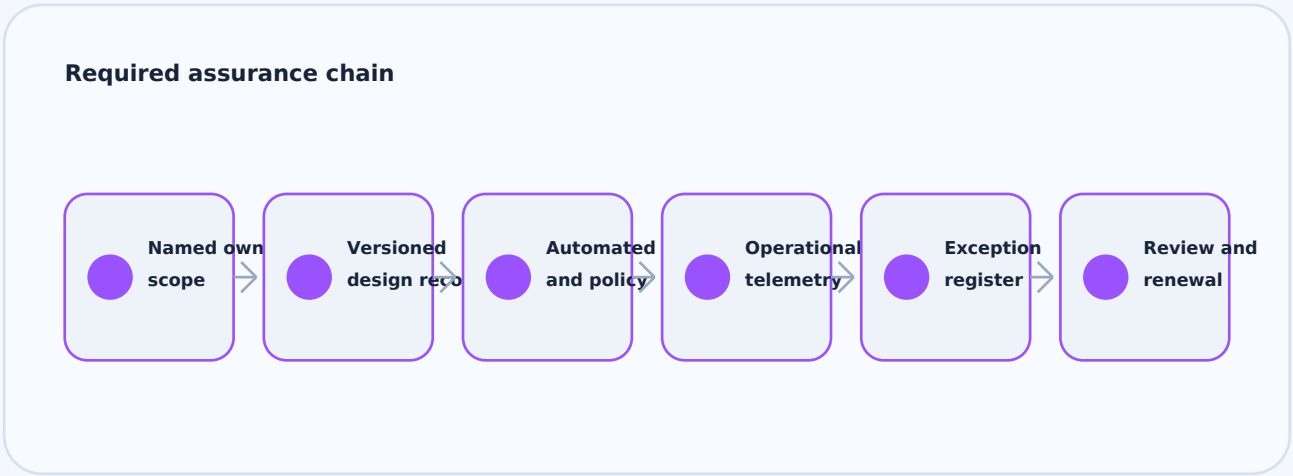
MSCS-Provenance

1. Tampered Artifact Deployment: 100+ tests attempting to deploy an artifact with invalid or missing SLSA provenance, verifying the Kubernetes admission controller blocks it.
1. Dependency Confusion Test: 50+ tests publishing a malicious package with an internal name to a public registry, verifying the build resolver ignores it.

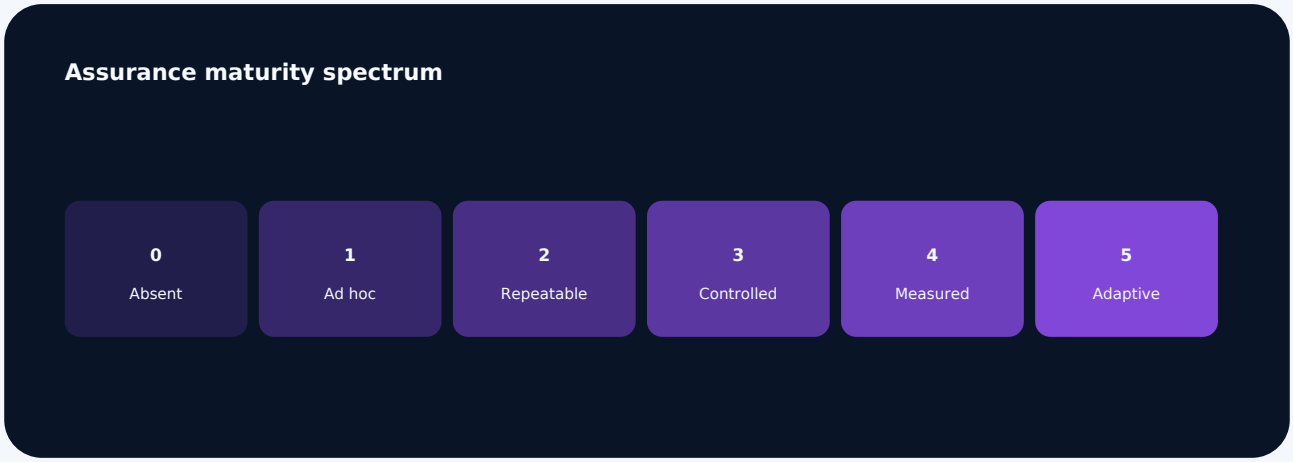
Execution Protocol

ASSURANCE AND ADOPTION

Implementation evidence and review gates



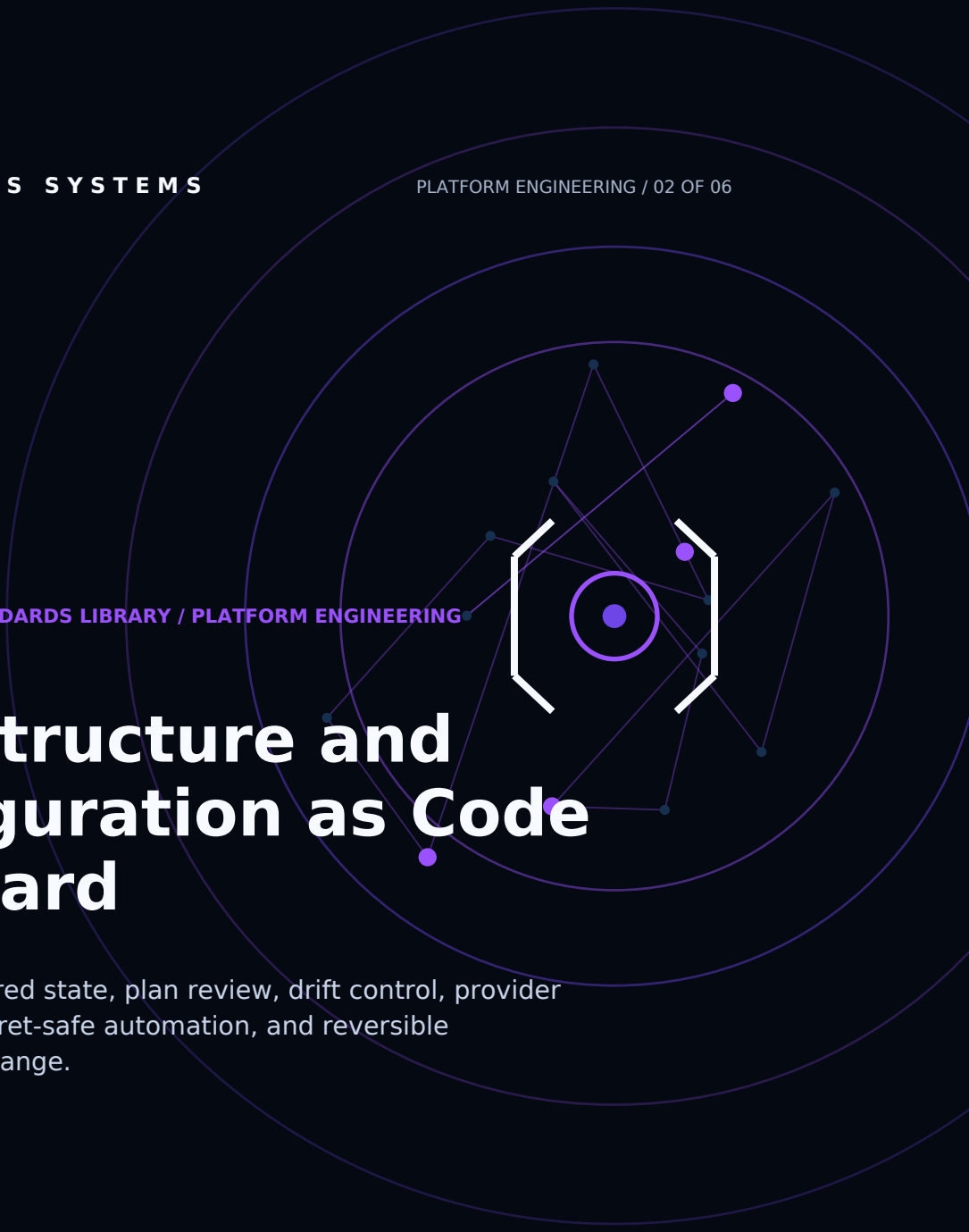
A passing score is not sufficient without current evidence. High-assurance adoption requires traceable design decisions, reproducible builds, independently reviewable tests, controlled secrets, runtime observability, failure exercises, and a dated exception register.



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.



ENGINEERING STANDARDS LIBRARY / PLATFORM ENGINEERING

Infrastructure and Configuration as Code Standard

Declarative desired state, plan review, drift control, provider governance, secret-safe automation, and reversible infrastructure change.

PERMANENT DOCUMENT ID

MYTHOS-PLT-0002

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

Infrastructure and Configuration as Code Standard

DOCUMENT ID
MYTHOS-PLT-0002

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

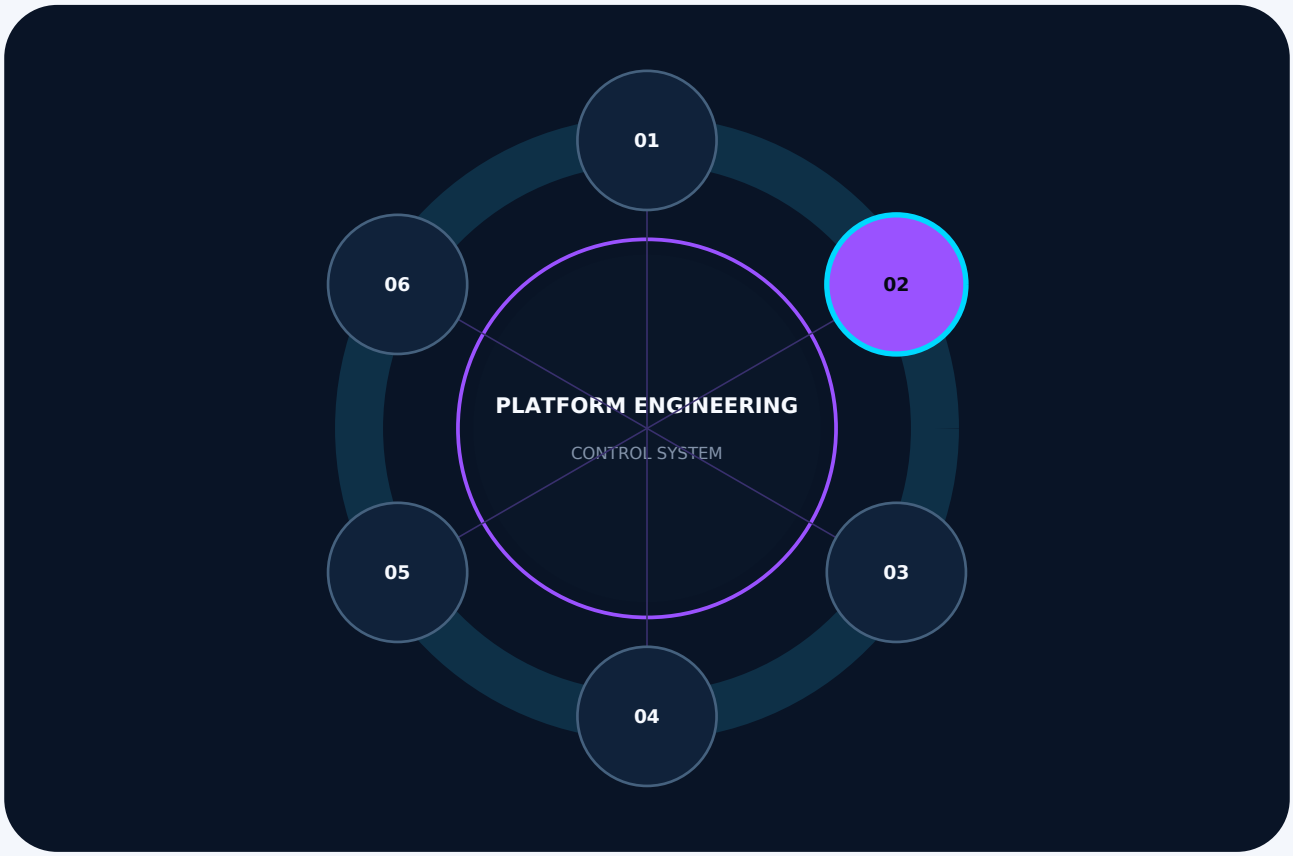
SCHEDULED REVIEW
2027-01-24

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-PLT-0002 inside the platform engineering standard constellation



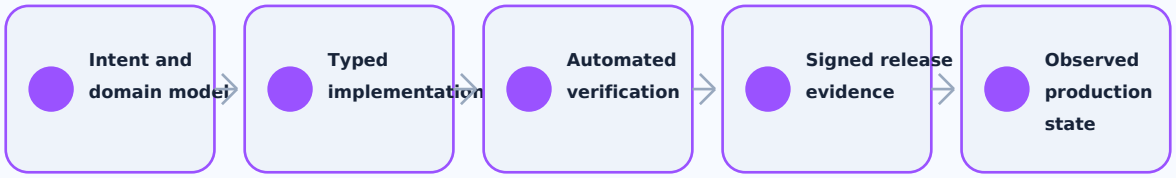
Connected assurance	Architecture, implementation, operations, evidence, and lifecycle are evaluated as one controlled system.
Specialization rule	Companion standards may add precision, but they cannot silently weaken this publication's requirements.

EXECUTIVE POSITION

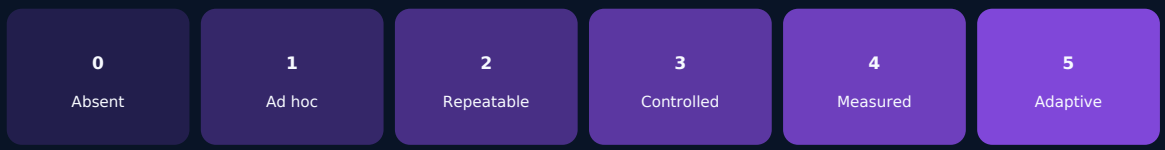
Infrastructure and Configuration as Code Standard

The architecture of infrastructure provisioning has failed.

Evidence-bearing delivery path



Assurance maturity spectrum



CONTENTS

Publication navigation

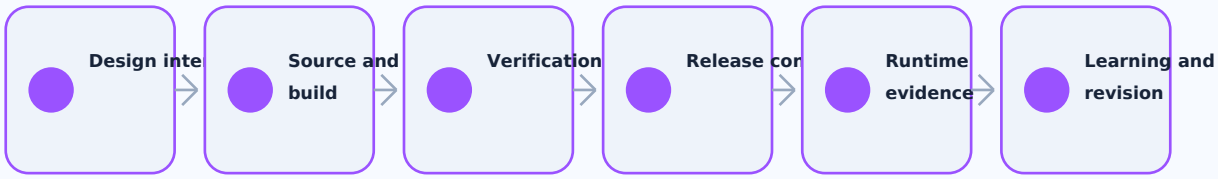
CONTROL TOPOLOGY

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
Declarative Paradigm and Abstraction (Weight: 20%)	1
GitOps and Pull-Based Reconciliation (Weight: 20%)	1
Drift Detection and Auto-Remediation (Weight: 20%)	1
State Backend Security and Integrity (Weight: 15%)	1
Policy-as-Code (Shift-Left Infra) (Weight: 15%)	1
Identity and Secret Management (Weight: 10%)	1
Suite Composition	2
Additional Evaluation Dimension	4

Lifecycle control loop



EVALUATION INDEX

Criteria and evidence map

#	Capability family	Criterion
1	Declarative Paradigm and Abstraction (Weight: 20%)	Infrastructure Modeling
2	GitOps and Pull-Based Reconciliation (Weight: 20%)	Execution Model
3	Drift Detection and Auto-Remediation (Weight: 20%)	State Integrity
4	State Backend Security and Integrity (Weight: 15%)	State Protection
5	Policy-as-Code (Shift-Left Infra) (Weight: 15%)	Compliance Enforcement
6	Identity and Secret Management (Weight: 10%)	Credential Handling
7	Suite Composition	MICAS-Drift
8	Suite Composition	MICAS-Policy
9	Additional Evaluation Dimension	IaC Dimensions
10	Additional Evaluation Dimension	The IaC 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 infrastructure provisioning has failed.

Organizations attempt to manage complex, multi-cloud environments using a mix of imperative Python scripts, manual UI configurations ("ClickOps"), and CI/CD pipelines that blindly `terraform apply` on push. When an engineer manually changes a firewall rule in the AWS console to fix an outage, the IaC state file becomes a lie. The next pipeline run either fails due to state conflict or silently overwrites the manual fix, causing a recurring outage.

This standard exists because:

1. ClickOps is Untrackable Sabotage: A manual change made in a cloud provider's web console is unversioned, unpeer-reviewed, and unrepeatable. It bypasses all security and architecture gates. Production infrastructure MUST be governed exclusively by version-controlled code.
2. Imperative Scripts are Fragile: A script that says "create a VPC, then create a subnet" will fail catastrophically if run twice or if the VPC already exists. Infrastructure MUST be declarative; the code defines the *desired state*, and the tool figures out how to converge reality to match it.
3. Push-Based CI/CD is Not GitOps: Triggering a pipeline to push infrastructure changes means the pipeline possesses long-lived cloud credentials, and state drift goes unnoticed until the next push. Systems MUST utilize pull-based reconciliation, where an in-cluster controller continuously compares the live state to the Git repository and auto-remediates drift.
4. Configuration Drift Must Be Auto-Remediated: Alerting on drift is insufficient; if an engineer manually changes a configuration, the system MUST automatically revert it to the declared source of truth. Otherwise, drift accumulates and causes unpredictable failures.
5. State is the Crown Jewel: The Terraform state file (or equivalent) contains highly sensitive secrets (database passwords, private keys) and dictates the entire infrastructure topology. State MUST be stored remotely, encrypted, access-controlled, and locked.

This document establishes the Mythos IaC Standard (MICAS), a comprehensive, evidence-based methodology for evaluating infrastructure automation against declarative rigor, continuous reconciliation, and zero-trust state management.

Scope and Applicability

In Scope

This standard covers the evaluation of:

1. Declarative Infrastructure as Code (Terraform, Pulumi, OpenTofu)
1. Kubernetes-native control planes (Crossplane, ACK)
1. Configuration Management / Ansible (Agentless, declarative state)
1. GitOps pull-based reconciliation (ArgoCD, Flux)
1. State backend security (remote state, locking, encryption)
1. Drift detection and automated remediation
1. Policy-as-Code for infrastructure (OPA Gatekeeper, HashiCorp Sentinel)

Out of Scope

1. General software supply chain security (covered in MYTHOS-PLT-0001)
1. General multi-cloud abstraction design (covered in MYTHOS-CLD-0001)
1. CI/CD pipeline security for application code (covered in MYTHOS-PLT-0001)

Audience

1. Platform engineers and DevOps architects
1. Cloud architects designing scalable infrastructure
1. Security engineers evaluating infrastructure attack surfaces
1. Site Reliability Engineers (SREs) managing drift and outages
1. Standards bodies seeking a reference IaC methodology

Definitions and Taxonomy

IaC Dimensions

Dimension	Definition
ClickOps	The anti-pattern of manually configuring infrastructure via a cloud provider's web GUI, resulting in unversioned, unrepeatable state.
Declarative State	An architectural paradigm where the code defines the *what* (desired end state), and the tool determines the *how* (API calls to converge reality).
GitOps (Pull-Based)	An operational model where a controller running inside the target environment continuously pulls the desired state from a Git repository and applies it, rather than a CI pipeline pushing changes.
Drift Reconciliation	The process of detecting when live infrastructure deviates from the declared code, and automatically remediating the live infrastructure to match the code.
Policy-as-Code	The practice of writing security and compliance rules (e.g., "no S3 buckets can be public") in code, enforced automatically during the IaC planning phase before any infrastructure is created.

The IaC Doctrine

> A manual UI change is sabotage. An imperative script is a liability. A state file without a lock is corruption. If the system does not auto-remediate drift, the code is a suggestion, not the truth.

Evaluation Methodology

Scoring Model

Each capability is scored from 0 to 5.

Score	Meaning
0	Fails basic task; ClickOps, imperative scripts, local state, no drift detection
1	Basic Terraform used, but via local execution and push-based CI
2	Functional remote state, but lacks Policy-as-Code or auto-remediation
3	Solid production performance; declarative, GitOps pull-model, strict policies, auto-remediation
4	Strong performance; Crossplane/K8s-native control plane, signed state, zero ClickOps capability
5	Best-in-class; sets the standard for mathematically verified, autonomous infrastructure convergence

Weighting

Category	Weight	Rationale
Declarative Paradigm & Abstraction	20%	Imperative scripts cannot handle complex dependencies
GitOps & Pull-Based Reconciliation	20%	Push-based CI leaves infrastructure blind to drift between runs
Drift Detection & Auto-Remediation	20%	Manual fixes cause outages when pipelines overwrite them
State Backend Security & Integrity	15%	State files contain secrets and dictate topology; corruption is fatal
Policy-as-Code (Shift-Left Infra)	15%	Waiting to scan infrastructure after it is built is too late
Identity & Secret Management	10%	IaC runners require cloud credentials; static keys are a liability

Total: 100%

A system MUST score at least 3.0 in GitOps Reconciliation and Drift Auto-Remediation to be considered for production use.

Evaluation Categories

Declarative Paradigm and Abstraction (Weight: 20%)

Infrastructure Modeling

Is infrastructure defined declaratively, allowing the tool to resolve dependencies?

Score	Criteria
0	Infrastructure provisioned via imperative bash/Python scripts

Score	Criteria
1	Basic Terraform used, but heavily relies on `local-exec` to run imperative commands
2	Purely declarative Terraform/Pulumi, but massive monolithic state files
3	Declarative code broken into reusable, versioned modules; no imperative hacks
4	Kubernetes-native control plane (Crossplane) treating cloud resources as Custom Resources
5	Perfect abstraction: formally verified dependency graphs, zero state collisions, mathematical proof of convergence

GitOps and Pull-Based Reconciliation (Weight: 20%)

Execution Model

How are infrastructure changes applied to the cloud?

Score	Criteria
0	Engineers run `terraform apply` from their local laptops
1	CI/CD pipeline triggers on Git push and runs `terraform apply` (Push-based)
2	Dedicated CI runner with OIDC federation, but still push-based
3	GitOps pull-based model: In-cluster controller (ArgoCD/Flux) pulls manifest and applies
4	Controller continuously reconciles live state to Git state every X minutes
5	Perfect reconciliation: formally verified convergence loops, zero credential exposure to CI, sub-minute reconciliation bounds

Drift Detection and Auto-Remediation (Weight: 20%)

State Integrity

What happens when an engineer makes a manual change via the console?

Score	Criteria
0	Manual changes are not detected; next pipeline run fails or overwrites silently
1	Drift detection runs nightly, alerts sent to Slack
2	Drift detection runs in CI, blocking new PRs until drift is resolved
3	Automated drift remediation: controller detects manual change and reverts it to match Git within minutes
4	Root-cause analysis attached to drift alerts (identifying who/what made the out-of-band change)
5	Perfect integrity: formally verified state parity, zero tolerance for manual changes, mathematical proof of continuous alignment

State Backend Security and Integrity (Weight: 15%)

State Protection

Is the Terraform/Pulumi state file protected against tampering and exfiltration?

Score	Criteria
0	State file stored locally on a laptop or in a public Git repo
1	State stored in remote backend (S3), but no locking mechanism
2	Remote backend with state locking (DynamoDB) and basic encryption
3	State encrypted via KMS; strict IAM policies limiting access to the GitOps controller only
4	State file tamper-evidence: cryptographic hashing of state objects; signed state versions
5	Perfect security: formally verified state isolation, zero plaintext secrets in state (all injected dynamically at apply time), mathematical proof of state integrity

Policy-as-Code (Shift-Left Infra) (Weight: 15%)

Compliance Enforcement

are security and compliance rules enforced before infrastructure is created?

Score	Criteria
0	No policy checks; engineers can create public S3 buckets or 0.0.0.0/0 security groups
1	Manual architectural reviews required before merge
2	Post-deployment scanning (e.g., Cloud Custodian) alerts on non-compliant resources
3	Policy-as-Code (OPA/Sentinel) integrated into CI; `terraform plan` is evaluated, and PRs are blocked if violations exist
4	Admission control: GitOps controller refuses to apply manifests that violate policy, even if merged to Git
5	Perfect enforcement: formally verified policy bounds, zero ability to provision non-compliant infrastructure, mathematical proof of continuous compliance

Identity and Secret Management (Weight: 10%)

Credential Handling

how does the IaC pipeline authenticate to the cloud provider?

Score	Criteria
0	Long-lived AWS access keys stored as CI/CD environment variables
1	Dedicated IAM user for the pipeline, keys rotated manually
2	OIDC federation: pipeline assumes a short-lived role using OIDC tokens (zero static keys)
3	Workload Identity (SPIFFE) bound to the GitOps controller running in-cluster
4	Just-in-Time privilege elevation: controller requests elevated rights only during `apply`, drops to read-only during `plan`
5	Perfect identity: formally verified credential boundaries, zero standing cloud privileges, cryptographic attestation of IaC execution

The Mythos IaC Suite (MICAS)

Traditional benchmarks measure `terraform apply` execution time. The MICAS evaluates drift tolerance, policy enforcement, and state security.

Suite Composition

MICAS-Drift

1. Console Injection: 100+ tests simulating an engineer manually changing a security group rule in the AWS console, verifying the GitOps controller detects and auto-remediates the change within 5 minutes.
1. State Corruption Test: 50+ tests attempting to manually edit the remote state file, verifying the system detects the tampering and refuses to apply.

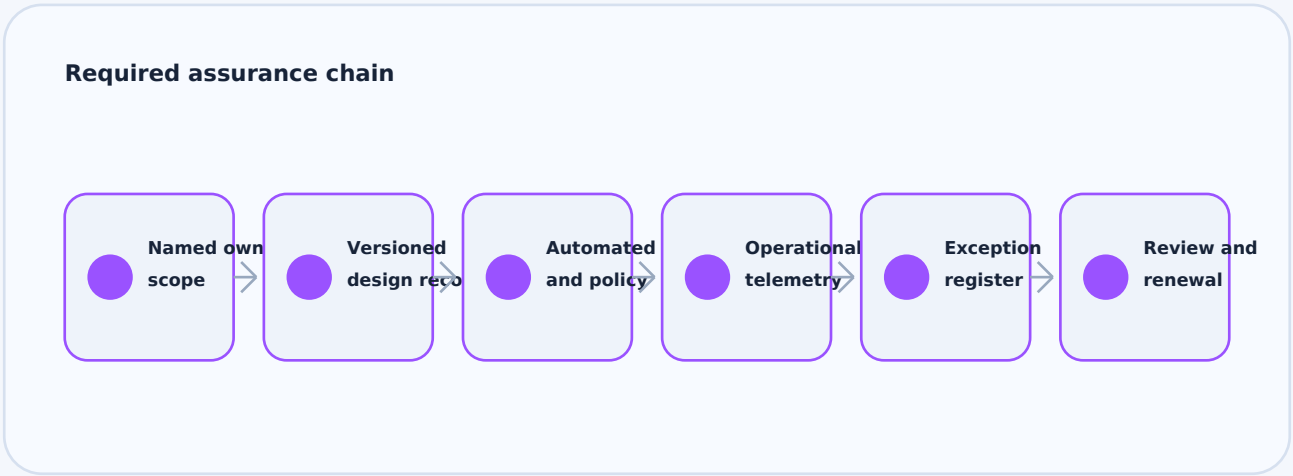
MICAS-Policy

1. Malicious PR Test: 100+ tests submitting a PR that creates an open S3 bucket or weak IAM policy, verifying the Policy-as-Code engine blocks the PR from merging.
1. Secret Leak Test: 50+ tests attempting to store a database password in the Terraform code, verifying the pipeline blocks it and requires dynamic secret injection.

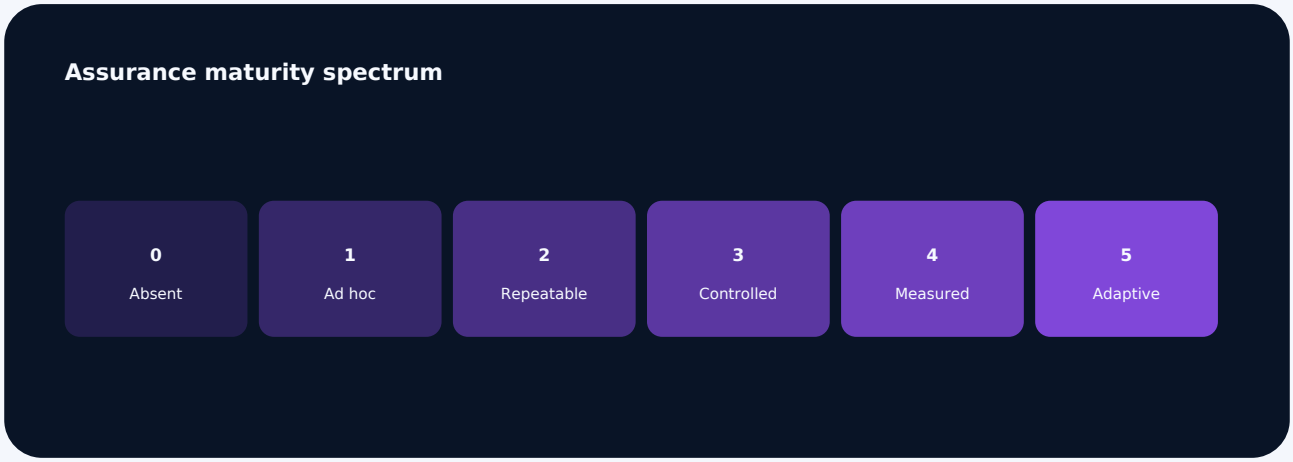
Execution Protocol

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.

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

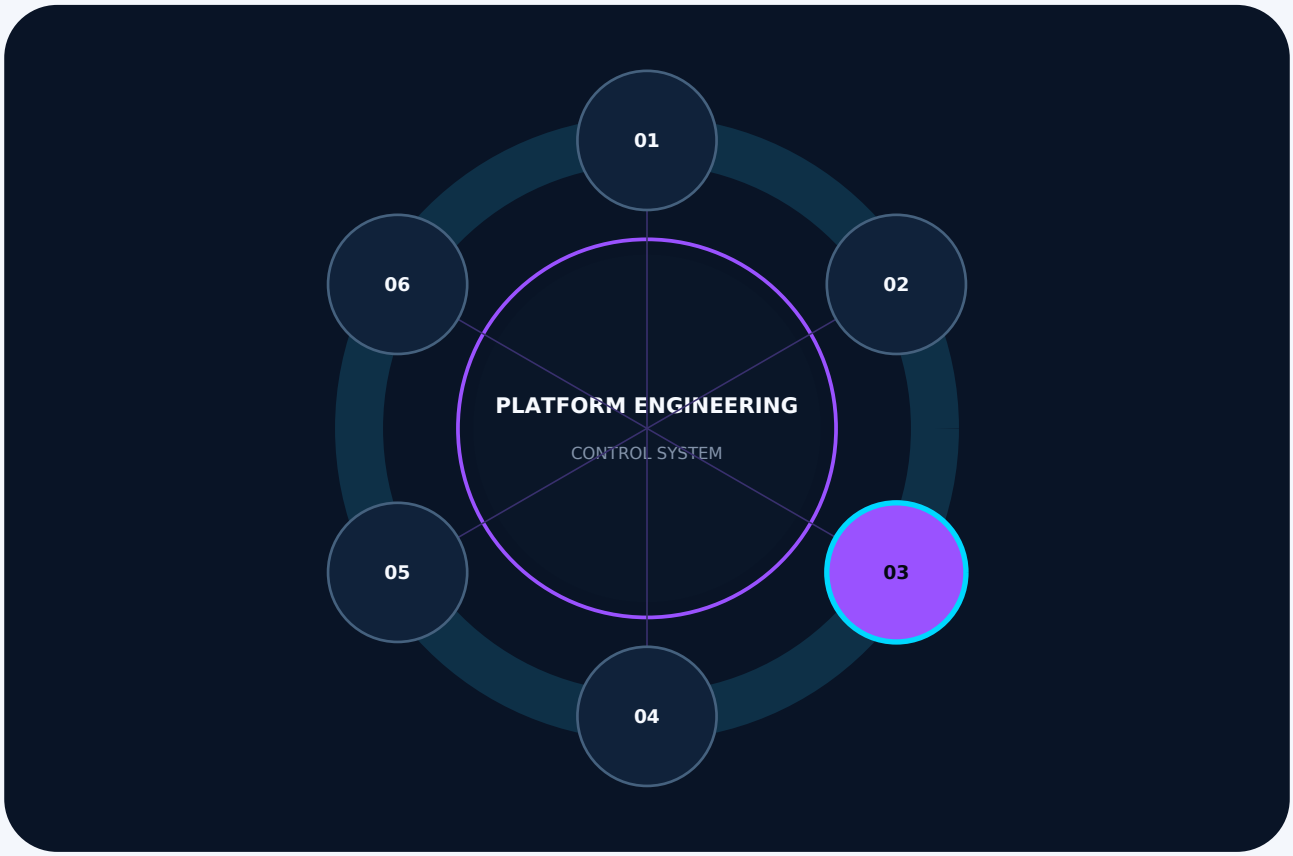
SCHEDULED REVIEW
2027-01-24

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

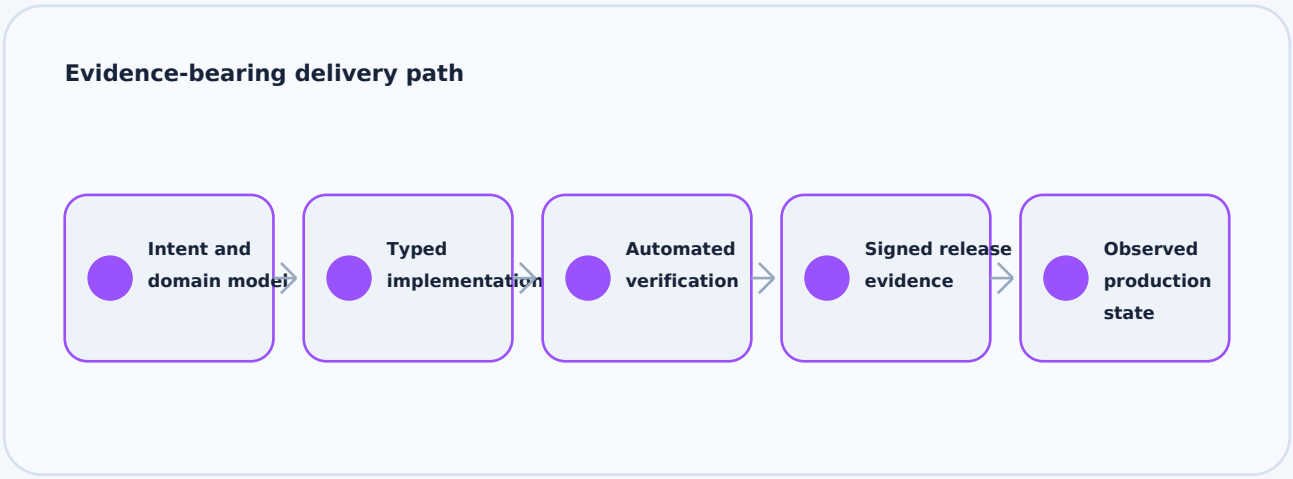


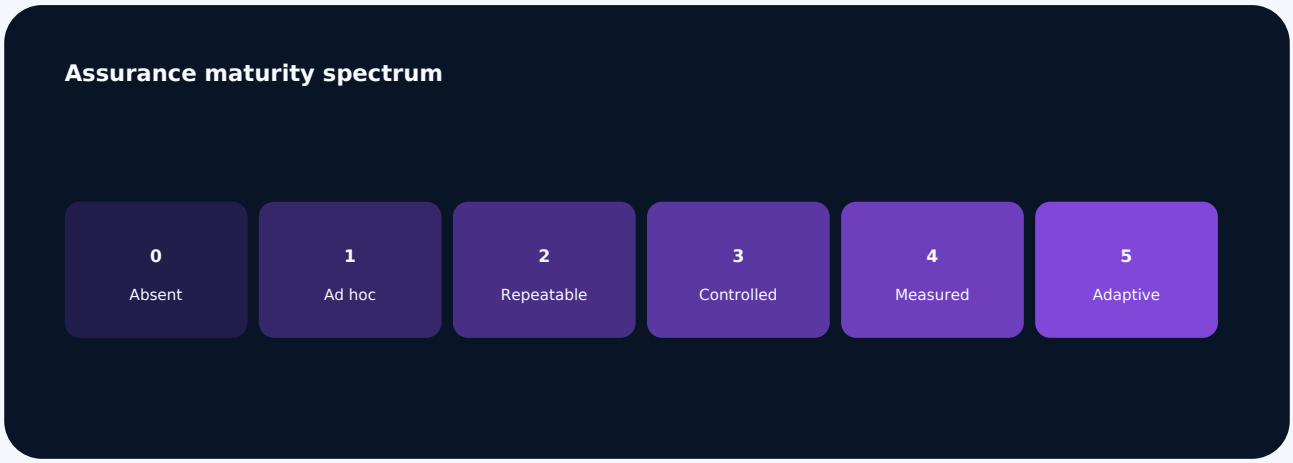
Connected assurance	Architecture, implementation, operations, evidence, and lifecycle are evaluated as one controlled system.
Specialization rule	Companion standards may add precision, but they cannot silently weaken this publication's requirements.

EXECUTIVE POSITION

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

The architecture of developer experience has failed.





CONTENTS

Publication navigation

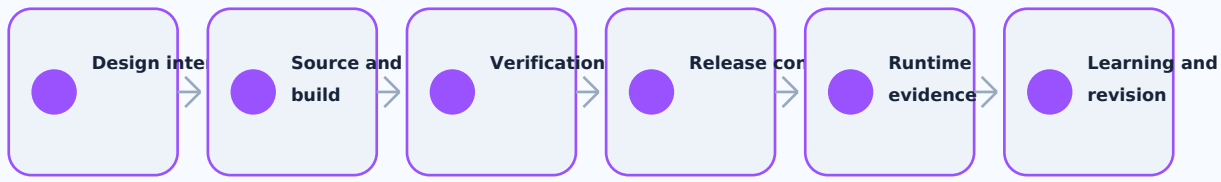
CONTROL TOPOLOGY

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.

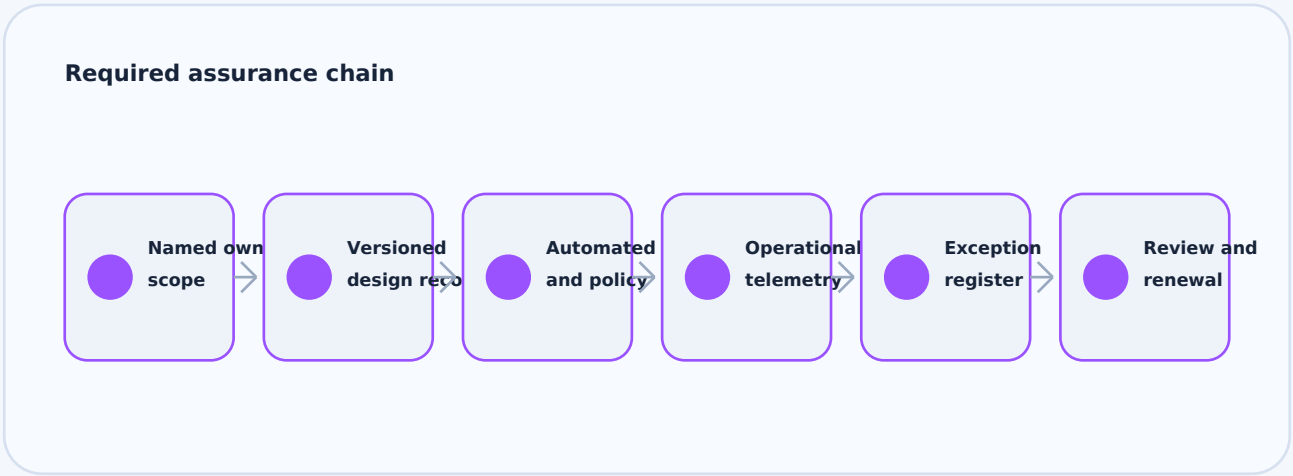
MIDPS-Policy

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.

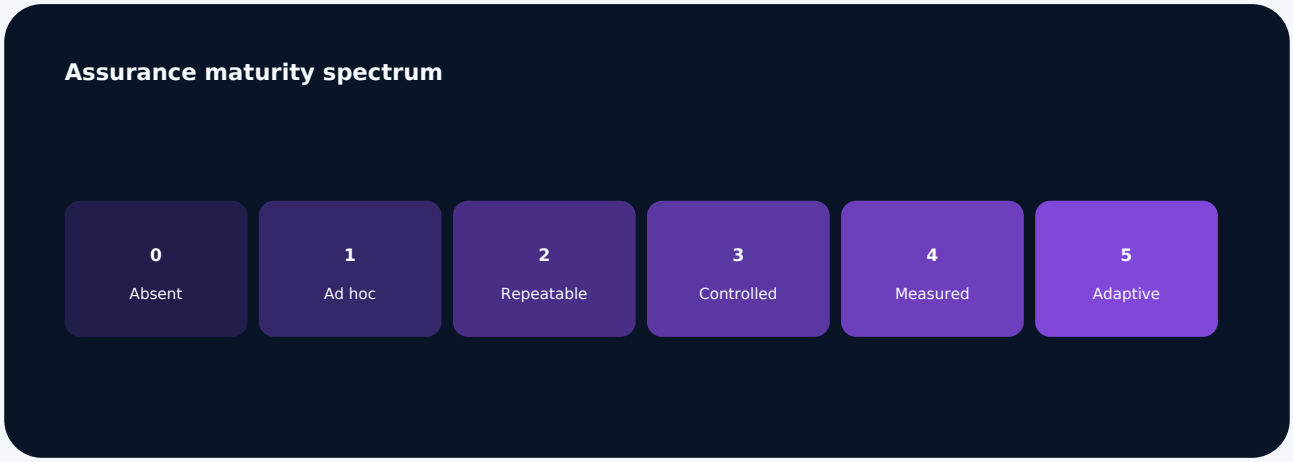
Execution Protocol

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.

ENGINEERING STANDARDS LIBRARY / PLATFORM ENGINEERING

Managed Platform and Agentic Service Evaluation Standard

Evidence-driven selection and governance of managed platforms, hosted runtimes, agent services, and external control planes.

PERMANENT DOCUMENT ID

MYTHOS-PLT-0004

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

Managed Platform and Agentic Service Evaluation Standard

DOCUMENT ID
MYTHOS-PLT-0004

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

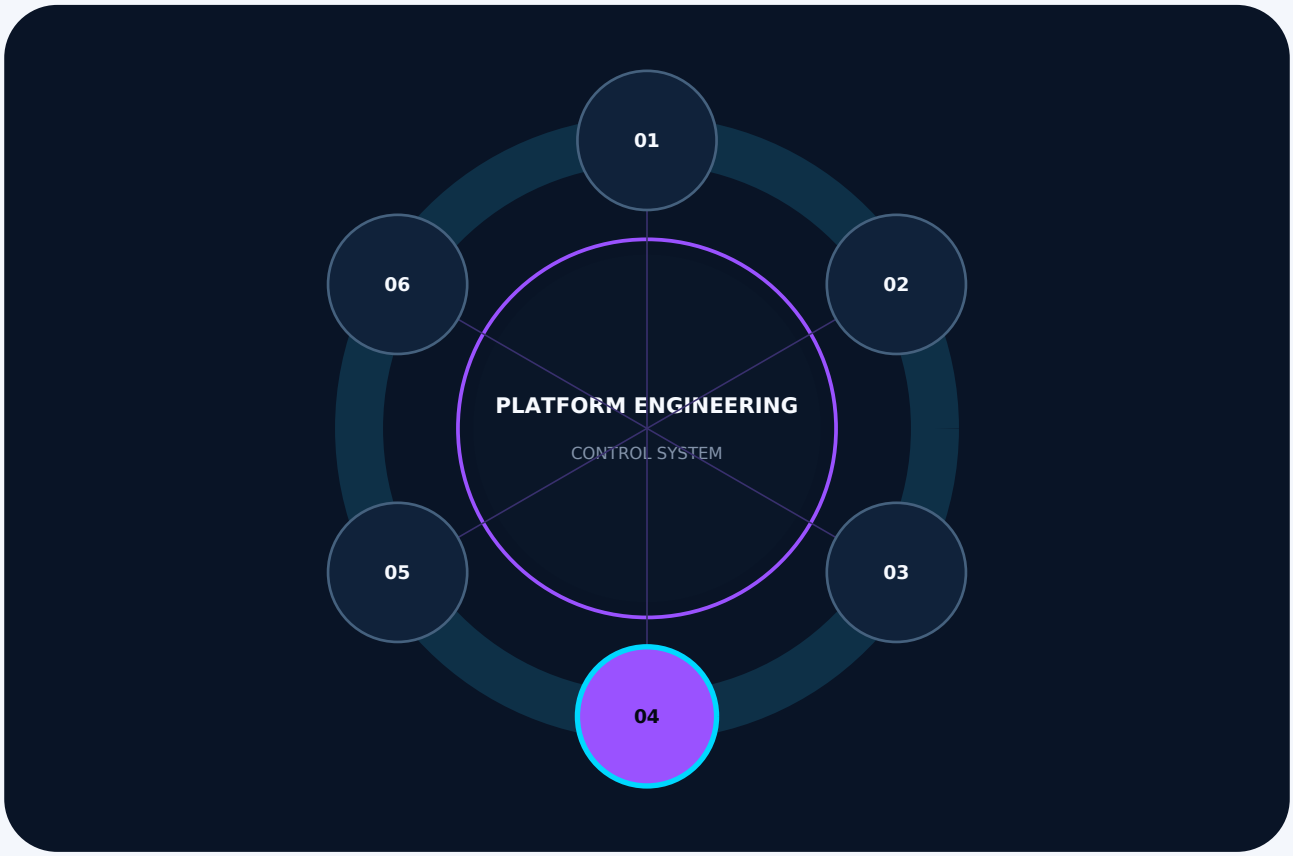
SCHEDULED REVIEW
2027-01-24

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-PLT-0004 inside the platform engineering standard constellation



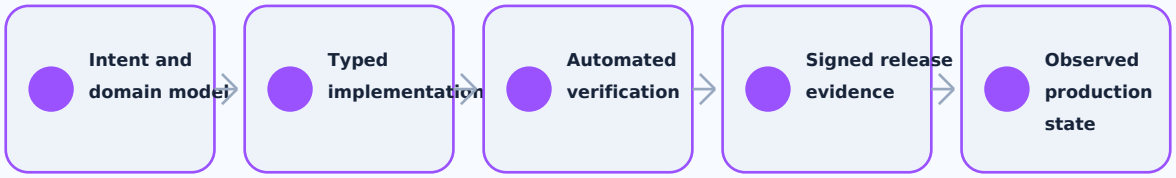
Connected assurance	Architecture, implementation, operations, evidence, and lifecycle are evaluated as one controlled system.
Specialization rule	Companion standards may add precision, but they cannot silently weaken this publication's requirements.

EXECUTIVE POSITION

Managed Platform and Agentic Service Evaluation Standard

The architecture of managed platform adoption has failed.

Evidence-bearing delivery path



Assurance maturity spectrum

0	1	2	3	4	5
Absent	Ad hoc	Repeatable	Controlled	Measured	Adaptive

CONTENTS

Publication navigation

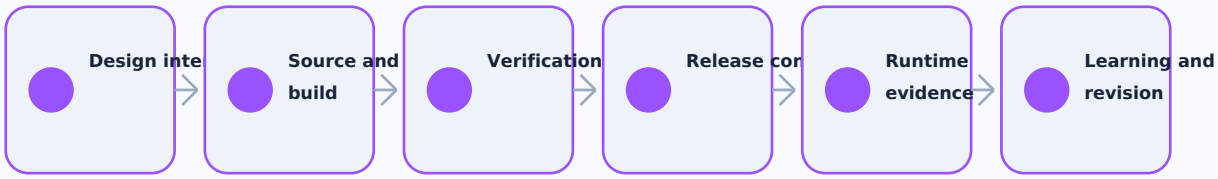
CONTROL TOPOLOGY

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
Vendor Lock-in and Escape Hatches (Weight: 20%)	1
Authority and Tool Execution (Weight: 20%)	1
Data Sovereignty and Egress Control (Weight: 15%)	1
State and Memory Portability (Weight: 15%)	1
Observability and Evidence Export (Weight: 15%)	1
Operational Control and CI/CD (Weight: 15%)	1
Suite Composition	2
Additional Evaluation Dimension	4

Lifecycle control loop



EVALUATION INDEX

Criteria and evidence map

#	Capability family	Criterion
1	Vendor Lock-in and Escape Hatches (Weight: 20%)	Architectural Portability
2	Authority and Tool Execution (Weight: 20%)	Zero-Trust Boundary
3	Data Sovereignty and Egress Control (Weight: 15%)	Prompt & Context Protection
4	State and Memory Portability (Weight: 15%)	Agent Continuity
5	Observability and Evidence Export (Weight: 15%)	Telemetry Independence
6	Operational Control and CI/CD (Weight: 15%)	Lifecycle Management
7	Suite Composition	MMPS-LockIn
8	Suite Composition	MMPS-Authority
9	Additional Evaluation Dimension	Managed Platform Dimensions
10	Additional Evaluation Dimension	The Managed Platform 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 managed platform adoption has failed.

Organizations, desperate to capitalize on the AI hype cycle, rush to adopt proprietary managed platforms like AWS Bedrock AgentCore or Itential. They build complex, mission-critical autonomous workflows directly into the vendor's control plane using proprietary SDKs and visual drag-and-drop builders. When the vendor changes the pricing model, deprecates a feature, or suffers a multi-region outage, the organization realizes its core business logic is held hostage.

This standard exists because:

1. Time-to-Value is Not Architecture: A managed platform that lets you build an agent in 10 minutes is a prototype, not a production strategy. Coupling your core intellectual property (agent logic, memory, tool definitions) to a single vendor's proprietary API is architectural surrender.
2. Managed Platforms are Black Boxes: Vendors claim "enterprise security," but if your agent's reasoning loop, prompt history, and tool execution logs live entirely inside the vendor's observability stack, you have zero verifiable evidence of what the agent actually did.
3. Authority Must Not Be Delegated to the Vendor: A managed platform that executes agent actions (e.g., modifying infrastructure, calling APIs) using the vendor's internal service roles is a zero-trust nightmare. The customer MUST retain absolute cryptographic control over tool execution and capability grants, even if the vendor hosts the reasoning loop.
4. Agent State Must Be Portable: If an agent's memory, session state, and checkpoint history cannot be exported in an open standard (e.g., OpenTelemetry, standard JSON/Protobuf) and migrated to a self-hosted runtime, the organization does not own the agent; the vendor does.

This document establishes the Mythos Managed Platform Standard (MMPS), a comprehensive, evidence-based methodology for evaluating managed agentic and automation platforms against lock-in risk, authority separation, and operational sovereignty.

Scope and Applicability

In Scope

This standard covers the evaluation of:

1. Managed AI agent platforms (AWS Bedrock AgentCore, Azure AI Foundry, Google Agent Builder)
1. Enterprise orchestration/automation platforms (Itential, Ansible Automation Platform, Workato)

1. SaaS-based workflow and integration engines
1. Vendor lock-in metrics and escape-hatch architecture
1. Authority separation in managed runtimes (who executes the tool?)
1. Data sovereignty and prompt egress controls in SaaS platforms

Out of Scope

1. Open-source, self-hosted agentic frameworks (covered in MYTHOS-AI-0001)
1. General multi-cloud infrastructure design (covered in MYTHOS-CLD-0001)
1. General identity and access management (covered in MYTHOS-ID-0001)

Audience

1. Principal engineers and architects evaluating SaaS/PaaS AI solutions
1. Security teams evaluating third-party agent risk
1. Procurement and vendor management teams
1. CTOs and engineering leaders making multi-year platform commitments
1. Standards bodies seeking a reference vendor evaluation methodology

Definitions and Taxonomy

Managed Platform Dimensions

Dimension	Definition
Escape Hatch	A documented, tested architectural strategy for migrating an agent's logic, state, and memory off a managed platform to a self-hosted or alternative runtime without a complete rewrite.
Authority Boundary	The architectural line determining who executes a tool. In a zero-trust model, the managed platform reasons; the customer's independent control plane executes.
Vendor Lock-in (Cognitive)	The reliance on proprietary visual builders (e.g., drag-and-drop nodes) that cannot be version-controlled, peer-reviewed, or exported as declarative code.
State Portability	The ability to export an agent's temporal memory, session context, and execution history in an open, machine-readable format.
Observability Egress	The capability to stream telemetry, prompts, and tool execution logs out of the vendor's platform into the customer's tamper-evident ledger in real-time.

The Managed Platform Doctrine

> A drag-and-drop builder is a prototype, not an architecture. A reasoning loop you cannot inspect is a black box. An agent you cannot migrate is a hostage. If the vendor executes the tools, you have surrendered your zero-trust boundary.

Evaluation Methodology

Scoring Model

Each capability is scored from 0 to 5.

Score	Meaning
0	Fails basic task; 100% proprietary lock-in, vendor executes tools, no data egress
1	Basic API access exists, but state/logic is trapped in the vendor cloud
2	Functional platform, but lacks portable state or independent authority enforcement
3	Solid production performance; declarative code, portable state, observability egress, customer-bound execution
4	Strong performance; multi-region failover, zero-knowledge tool execution, formally tested escape hatches
5	Best-in-class; sets the standard for sovereign, portable, zero-trust managed agentic platforms

Weighting

Category	Weight	Rationale
Vendor Lock-in & Escape Hatches	20%	Core IP trapped in a SaaS guarantees multi-year architectural debt
Authority & Tool Execution	20%	Vendors executing tools with their own credentials bypasses zero-trust
Data Sovereignty & Egress Control	15%	Prompts and context sent to vendor APIs must be strictly governed
State & Memory Portability	15%	Agent memory must be exportable to prevent platform hostage situations
Observability & Evidence Export	15%	Vendor UIs are not evidence; telemetry must stream to customer ledgers
Operational Control & CI/CD	15%	Visual builders break GitOps and Infrastructure-as-Code paradigms

Total: 100%

A system MUST score at least 3.0 in Vendor Lock-in and Authority/Tool Execution to be considered for production use.

Evaluation Categories

Vendor Lock-in and Escape Hatches (Weight: 20%)

Architectural Portability

Can the organization migrate off this platform without a total rewrite?

Score	Criteria
0	Logic built entirely in proprietary visual UI; no code export; logic trapped in SaaS
1	Logic can be exported as JSON, but is tightly coupled to vendor-specific schema

Score	Criteria
2	SDKs provided for code-first development, but runtime depends on vendor APIs
3	Open-standard protocols (MCP, A2A) used; agent logic is declarative and runnable on alternative runtimes
4	Documented and tested escape hatch: migration to open-source runtime (e.g., LangGraph) verified via CI/CD
5	Perfect portability: formally verified logic abstraction, zero vendor-specific runtime dependencies, sub-day migration capability

Authority and Tool Execution (Weight: 20%)

Zero-Trust Boundary

Who executes the consequential actions (tools) proposed by the agent?

Score	Criteria
0	Vendor platform executes tools directly using vendor-managed IAM roles
1	Customer provides static API keys to the vendor platform for tool execution
2	Vendor platform calls customer webhooks, but lacks cryptographic payload signing
3	Strict separation: Vendor reasons, customer's independent control plane (e.g., PANTHEON Aegis) executes and validates
4	Customer-bound capability grants: vendor cannot invoke a tool without a cryptographically attenuated, short-lived token from the customer
5	Perfect authority: formally verified zero-trust boundary, zero vendor ability to execute actions independently, cryptographic non-repudiation of all tool calls

Data Sovereignty and Egress Control (Weight: 15%)

Prompt & Context Protection

Is enterprise data protected from the vendor's training pipelines and internal logging?

Score	Criteria
0	Prompts and RAG context ingested by vendor may be used for model training; no egress controls
1	Vendor promises not to train on data (PDF policy), but architecture does not enforce it
2	Data residency options available (e.g., regional endpoints), but no customer-managed encryption keys (CMK)
3	CMK encryption enforced; zero data retention policies configured at the API level
4	Context-aware DLP intercepts and redacts PII/IP before prompts ever reach the managed platform
5	Perfect sovereignty: formally verified zero-knowledge execution (FHE/TEE), cryptographic proof that vendor cannot inspect payload, zero training data exfiltration risk

State and Memory Portability (Weight: 15%)

Agent Continuity

If the organization leaves the platform, does the agent retain its memory?

Score	Criteria
0	Agent memory and session state locked in vendor proprietary database
1	Basic session history export available via API (manual download)
2	Short-term memory exportable, but long-term temporal knowledge graph is trapped
3	Full state export in open standard (e.g., JSON/Protobuf); memory can be synced to customer DB in real-time
4	Bidirectional state sync: customer's local-first memory (e.g., Mnemosyne) is the source of truth; vendor pulls from it
5	Perfect portability: formally verified state synchronization, zero memory lock-in, seamless continuity across heterogeneous runtimes

Observability and Evidence Export (Weight: 15%)

Telemetry Independence

Does the organization rely on the vendor's dashboards, or does it own the evidence?

Score	Criteria
0	Logs and traces only viewable in the vendor's web UI
1	Basic log export to S3/Blob available (nightly batches)
2	Webhooks for major events, but lacks full cognitive tracing
3	Native OpenTelemetry (OTel) export with GenAI semantic conventions streaming to customer's SIEM/observability stack
4	Tamper-evident evidence bundles (hash-chained) generated for high-risk actions, exported instantly
5	Perfect observability: formally verified telemetry completeness, zero reliance on vendor UI for audit, cryptographic proof of cognitive trace integrity

Operational Control and CI/CD (Weight: 15%)

Lifecycle Management

Can the agent be deployed, rolled back, and version-controlled using standard DevSecOps practices?

Score	Criteria
0	Agent updates require manual clicks in the vendor UI
1	CLI tools exist, but lack declarative state management
2	API-driven updates, but no GitOps reconciliation
3	Fully declarative (IaC/Terraform/CloudFormation); agent definitions version-controlled in Git
4	CI/CD pipeline enforces policy-as-code (OPA) on agent configurations before vendor API accepts changes

Score	Criteria
5	Perfect control: formally verified CI/CD boundaries, zero manual UI intervention, mathematical proof of configuration parity

The Mythos Managed Platform Suite (MMPS)

Traditional vendor evaluations measure feature checklists. The MMPS evaluates lock-in risk, authority boundaries, and exit strategies.

Suite Composition

MMPS-LockIn

1. Export & Re-Host Test: 100+ tests exporting an agent's logic, state, and memory from the managed platform and deploying it to an open-source runtime (e.g., LangGraph), verifying functionality parity.
1. UI Dependency Audit: 50+ tests verifying that no critical operational task (rollback, scaling, secret rotation) requires access to the vendor's web UI.

MMPS-Authority

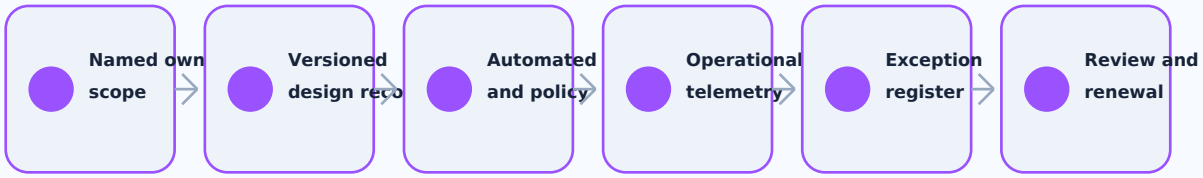
1. Tool Execution Test: 100+ tests simulating a vendor platform compromise, verifying the attacker cannot execute customer-side tools without the customer's ephemeral capability grant.
1. Egress DLP Test: 50+ tests injecting PII into the agent's context, verifying the data sovereignty layer redacts it before it hits the vendor's inference API.

Execution Protocol

ASSURANCE AND ADOPTION

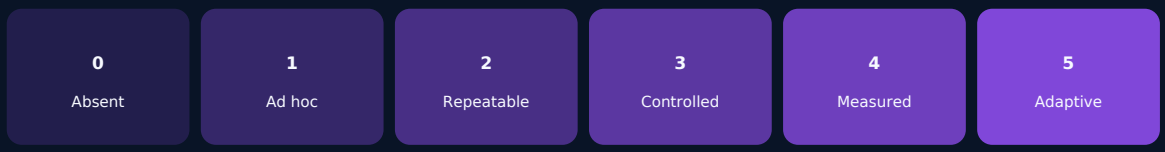
Implementation evidence and review gates

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.

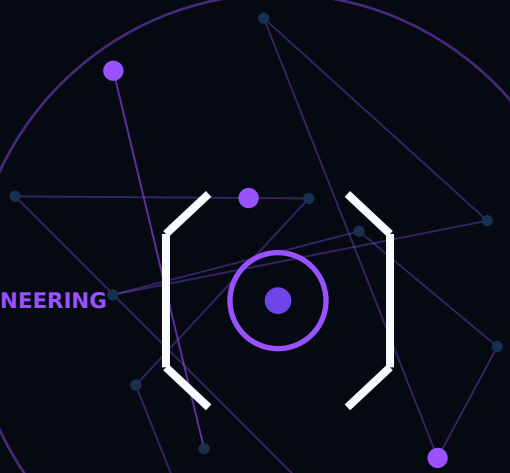
Assurance maturity spectrum



Current authority register

Authority	Version / snapshot	Applicability	Review by
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.



ENGINEERING STANDARDS LIBRARY / PLATFORM ENGINEERING

Hardware and Software Lifecycle and Refresh Standard

Lifecycle intelligence, supportability, refresh planning, compatibility management, technical debt, and controlled retirement.

PERMANENT DOCUMENT ID

MYTHOS-PLT-0005

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

Hardware and Software Lifecycle and Refresh Standard

DOCUMENT ID
MYTHOS-PLT-0005

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

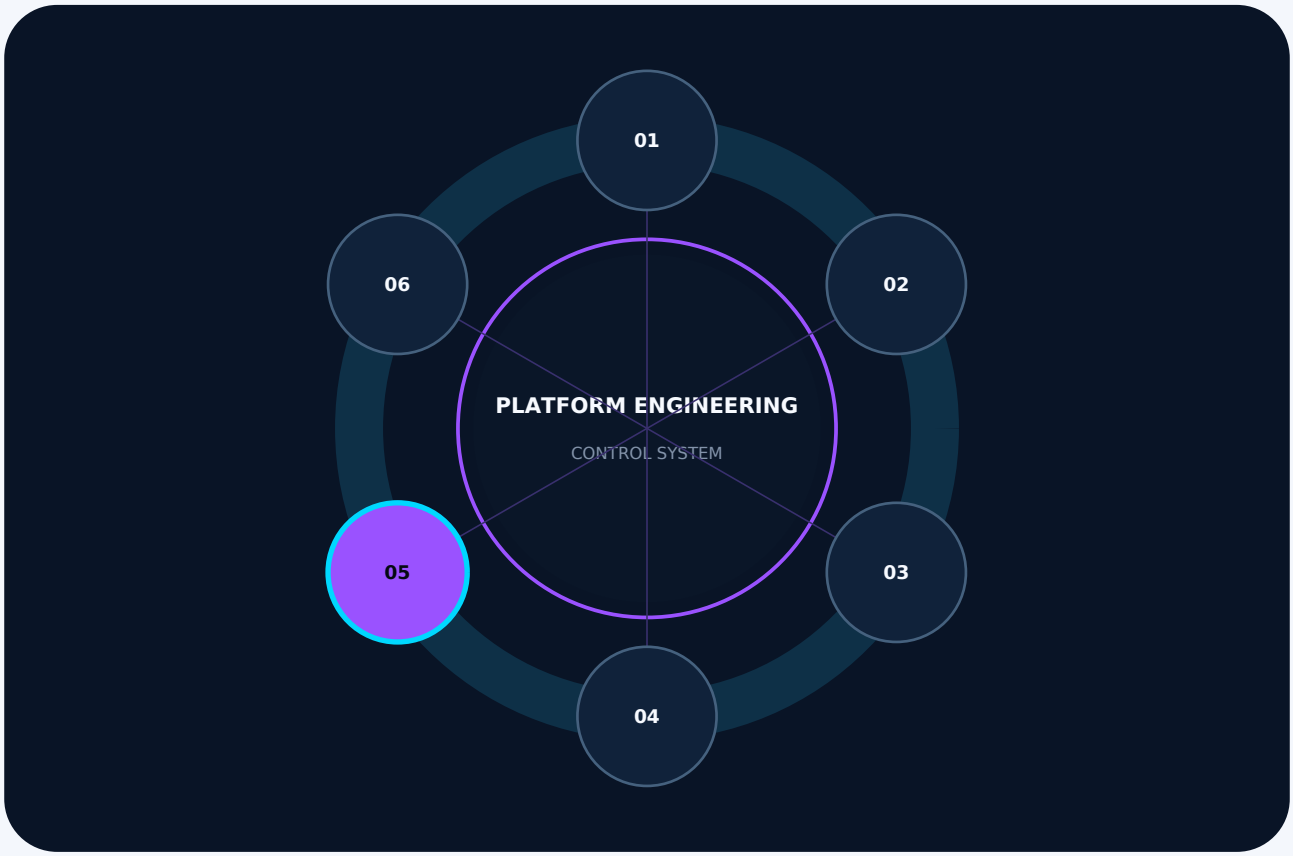
SCHEDULED REVIEW
2027-01-24

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-PLT-0005 inside the platform engineering standard constellation



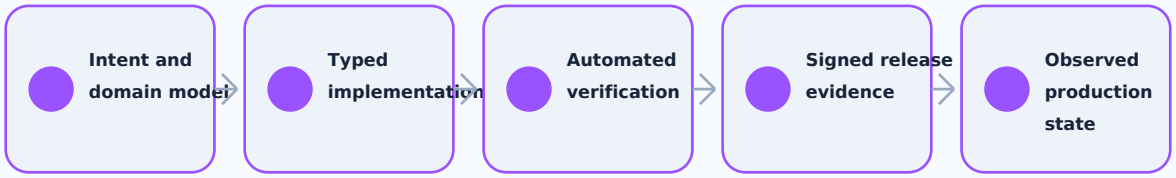
Connected assurance	Architecture, implementation, operations, evidence, and lifecycle are evaluated as one controlled system.
Specialization rule	Companion standards may add precision, but they cannot silently weaken this publication's requirements.

EXECUTIVE POSITION

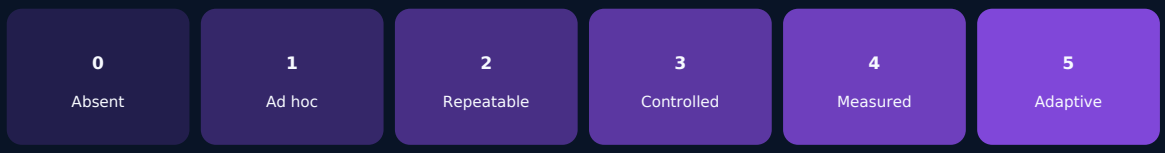
Hardware and Software Lifecycle and Refresh Standard

The architecture of asset and lifecycle management has failed.

Evidence-bearing delivery path



Assurance maturity spectrum



CONTENTS

Publication navigation

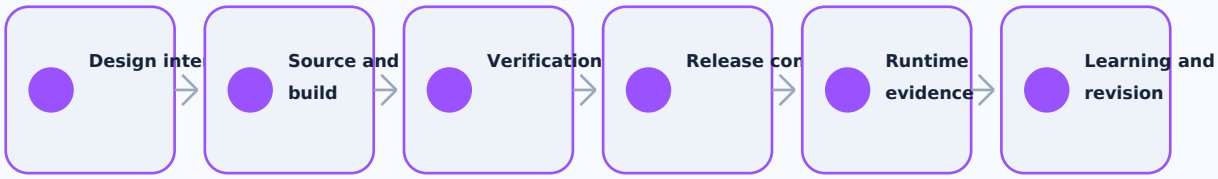
CONTROL TOPOLOGY

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
EOL/EOS Tracking and Automation (Weight: 20%)	1
Hardware Refresh Cadences (Weight: 15%)	1
Software Dependency Pruning (Weight: 15%)	1
Cabling Structure and Management (Weight: 20%)	1
Secure Disposal and Sanitization (Weight: 15%)	1
Asset Inventory (DCIM) as Code (Weight: 15%)	1
Suite Composition	2
Additional Evaluation Dimension	4

Lifecycle control loop



EVALUATION INDEX

Criteria and evidence map

#	Capability family	Criterion
1	EOL/EOS Tracking and Automation (Weight: 20%)	Lifecycle Visibility
2	Hardware Refresh Cadences (Weight: 15%)	Proactive Replacement
3	Software Dependency Pruning (Weight: 15%)	Legacy Software Eradication
4	Cabling Structure and Management (Weight: 20%)	Physical Plant Rigor
5	Secure Disposal and Sanitization (Weight: 15%)	Data Destruction
6	Asset Inventory (DCIM) as Code (Weight: 15%)	Infrastructure as Data
7	Suite Composition	MLCS-Lifecycle
8	Suite Composition	MLCS-Physical
9	Additional Evaluation Dimension	Lifecycle Dimensions
10	Additional Evaluation Dimension	The Lifecycle 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 asset and lifecycle management has failed.

Organizations run production-critical infrastructure on 7-year-old routers and operating systems that haven't received a security patch in years. They track thousands of physical servers and software licenses using static Excel spreadsheets that are obsolete the moment they are saved. In the datacenter, they bind cables with plastic zip ties, creating physically rigid bundles that guarantee a single cable change will disrupt the entire link, all while having no idea what a randomly colored fiber cable actually connects.

This standard exists because:

1. EOL/EOS is a Critical Security Vulnerability: Running hardware or software past its End-of-Life (EOL) or End-of-Support (EOS) date is an unconditional security failure. Vendors stop releasing patches, meaning zero-day vulnerabilities will remain permanently unpatched. Systems MUST be retired or isolated before they hit EOL.
2. Spreadsheets are Not Asset Management: If an organization cannot programmatically query the exact firmware version, warranty status, and location of a piece of hardware via an API, they do not own their infrastructure; they are hostage to it. Asset management MUST be an automated, immutable, API-driven Source of Truth (e.g., DCIM/IPAM).
3. Zip Ties are Architectural Sabotage: Plastic zip ties damage fiber optics, restrict copper airflow, and create rigid bundles that make maintenance impossible without causing widespread physical disruption. Cabling MUST be bound exclusively with hook-and-loop (Velcro) and managed via structured, color-coded pathways.
4. Refresh Cycles Must Be Proactive, Not Reactive: Waiting for a server to catch fire or a switch to panic before replacing it is break-fix theater. Organizations MUST implement scheduled, rolling refresh cadences based on MTBF (Mean Time Between Failures), thermal degradation curves, and technological obsolescence.
5. Data Sanitization Must Be Cryptographic: Throwing a hard drive in the trash or running a single `dd` pass is a data breach waiting to happen. Hardware retirement MUST include cryptographic erasure (destroying the encryption keys) or physically verified degaussing/shredding.

This document establishes the Mythos Lifecycle Standard (MLCS), a comprehensive, evidence-based methodology for evaluating hardware and software lifecycle processes against automation rigor, physical hygiene, and zero-legacy compliance.

Scope and Applicability

In Scope

This standard covers the evaluation of:

1. Hardware End-of-Life (EOL) and End-of-Support (EOS) tracking
1. IT Asset Management (ITAM) and DCIM automation
1. Software dependency lifecycle and SBOM pruning
1. Technology refresh cadences and capital expenditure (CapEx) planning
1. Structured cabling standards (TIA-606-C, color-coding, hook-and-loop)
1. Secure data sanitization and hardware disposal

Out of Scope

1. General datacenter cooling and power topology (covered in MYTHOS-NET-0009)
1. General CI/CD supply chain security (covered in MYTHOS-PLT-0001)
1. General vulnerability scoring (CVEs) (covered in MYTHOS-SEC-0008)

Audience

1. Datacenter facility managers and mechanical engineers
1. IT asset managers and procurement teams
1. Network engineers and cable plant technicians
1. Platform engineers managing software dependency lifecycles
1. Security teams evaluating end-of-life risk and data sanitization
1. Standards bodies seeking a reference lifecycle methodology

Definitions and Taxonomy

Lifecycle Dimensions

Dimension	Definition
EOL (End-of-Life)	The date after which a vendor will no longer manufacture or sell a piece of hardware or software version.
EOS (End-of-Support)	The date after which a vendor will no longer provide security patches, bug fixes, or technical support for a product. Running EOS software is a critical security failure.
DCIM (Datacenter Infrastructure Management)	Software tools that monitor, measure, and manage datacenter resources (power, cooling, space, connectivity) and track hardware lifecycles in real-time.
Cryptographic Erasure	The process of rendering data permanently unreadable by securely destroying the encryption keys used to encrypt the data, rather than attempting to overwrite the physical disk.
Structured Cabling	A standardized architecture for telecommunications cabling, using dedicated pathways, patch panels, strict color-coding, and hook-and-loop fasteners to ensure modularity.

The Lifecycle Doctrine

> A spreadsheet is not an asset database. A zip tie is a physical chokehold. An EOS switch is a compromised switch. If the hardware cannot be cryptographically erased before disposal, the data is already breached.

Evaluation Methodology

Scoring Model

Each capability is scored from 0 to 5.

Score	Meaning
0	Fails basic task; spreadsheets, zip ties, running EOS hardware, no disposal policy
1	Basic asset tracking, but manual EOL checks and disorganized cabling
2	Functional DCIM, but reactive refresh cycles and inconsistent cable colors
3	Solid production performance; automated EOL alerts, scheduled refreshes, TIA-606-C cabling, crypto-erasure
4	Strong performance; zero-landfill disposal, predictive refresh modeling, zero zip-ties
5	Best-in-class; sets the standard for mathematically verified, zero-legacy lifecycle management

Weighting

Category	Weight	Rationale
EOL/EOS Tracking & Automation	20%	Running unsupported hardware/software is an unconditional security failure
Hardware Refresh Cadences	15%	Reactive break-fix causes catastrophic outages and unplanned spending
Software Dependency Pruning	15%	Bloated software portfolios accumulate massive technical debt and vulnerabilities
Cabling Structure & Management	20%	Zip ties and unlabeled cables cause massive MTTR during physical outages
Secure Disposal & Sanitization	15%	Improperly disposed drives guarantee data exfiltration
Asset Inventory (DCIM) as Code	15%	If assets aren't API-driven, they cannot be automated or reconciled

Total: 100%

A system MUST score at least 3.0 in EOL/EOS Tracking and Cabling Structure to be considered for production use.

Evaluation Categories

EOL/EOS Tracking and Automation (Weight: 20%)

Lifecycle Visibility

Does the organization have real-time, automated visibility into the support status of all assets?

Score	Criteria
0	Manual tracking via spreadsheets; no one knows what is EOS until it breaks
1	Basic ITAM tool used, but EOL dates must be checked manually on vendor websites
2	EOL dates imported into ITAM, but alerts are not automated
3	Automated EOL/EOS alerts triggered 6-12 months before expiration; integrated with procurement
4	Continuous API sync with vendor support portals; live firmware compliance tracking
5	Perfect tracking: formally verified asset registries, zero EOS assets in production, mathematical proof of continuous support coverage

Hardware Refresh Cadences (Weight: 15%)

Proactive Replacement

Is hardware replaced proactively based on obsolescence and reliability curves?

Score	Criteria
0	Break-fix only; hardware runs until catastrophic failure
1	Ad-hoc replacements when performance becomes a bottleneck
2	Standard 5-7 year refresh cycle, but execution is delayed and unplanned
3	Scheduled rolling refresh cycles based on MTBF and technological obsolescence (e.g., 3-4 years for compute, 5 for network)
4	Predictive refresh modeling using thermal and power degradation data from DCIM
5	Perfect cadence: formally verified lifecycle bounds, zero unplanned hardware failures, seamless CapEx integration

Software Dependency Pruning (Weight: 15%)

Legacy Software Eradication

Is the software portfolio continuously pruned of deprecated and unused components?

Score	Criteria
0	Legacy frameworks and OS versions kept running indefinitely ("we're afraid to touch it")
1	Periodic audits of installed software, but no enforcement of removal
2	SBOM (Software Bill of Materials) generated for all apps, but EOL dependencies remain
3	Automated CI/CD checks block the deployment of software with EOL dependencies

Score	Criteria
4	Continuous dependency scanning auto-quarantines apps running deprecated libraries
5	Perfect pruning: formally verified software registries, zero EOL dependencies in production, mathematical proof of minimal attack surface

Cabling Structure and Management (Weight: 20%)

Physical Plant Rigor

Is the cabling plant strictly organized, labeled, and safely bound?

Score	Criteria
0	"Spaghetti" cabling; plastic zip ties used; unlabeled fibers
1	Basic patch panels used, but inconsistent labeling and zip ties remain
2	TIA-606-C compliant labels at both ends; hook-and-loop (Velcro) straps used exclusively
3	Strict color-coding enforced (e.g., Red = Security, Blue = Copper, Orange = Internal Fiber, Yellow = OOB); zero zip ties on the premises
4	Cable paths documented in DCIM; fiber optic OTDR traces mapped; locking clips on critical fiber
5	Perfect structure: formally verified cable topologies, zero undocumented strands, automated patch-port validation via smart patch panels

Secure Disposal and Sanitization (Weight: 15%)

Data Destruction

How is data irrecoverably destroyed before hardware leaves the facility?

Score	Criteria
0	Drives thrown in the trash or sold on eBay with a quick `format`
1	Single-pass overwrite (e.g., `dd if=/dev/zero`), but no verification
2	Multi-pass wipe (DoD 5220.22-M) with cryptographic verification
3	Cryptographic Erasure: drives are encrypted at rest (AES-256); disposal = destroying the KEK (Key Encryption Key)
4	Physical degaussing and shredding of SSDs/HDDs with video evidence and certificate of destruction
5	Perfect sanitization: formally verified zero-recoverability, zero-landfill hardware recycling policy, cryptographic attestation of destruction

Asset Inventory (DCIM) as Code (Weight: 15%)

Infrastructure as Data

Is the physical asset inventory treated as an API-driven, declarative source of truth?

Score	Criteria
0	Asset data lives in a spreadsheet on a file share

Score	Criteria
1	Dedicated DCIM/ITAM platform used, but data entry is manual
2	DCIM auto-discovers network gear via SNMP, but compute/space/power tracked manually
3	API-first DCIM: all provisioning scripts automatically update asset location, rack elevation, and power draw
4	Real-time reconciliation: DCIM detects physical changes (e.g., server moved) via RFID/sensors and alerts on drift
5	Perfect inventory: formally verified asset parity, zero manual data entry, cryptographic attestation of physical topology

The Mythos Lifecycle Suite (MLCS)

Traditional facility audits measure "did we buy new stuff." The MLCS evaluates EOL risk exposure, physical cable traceability, and data sanitization guarantees.

Suite Composition

MLCS-Lifecycle

1. EOL Audit: 100+ tests querying the asset registry for any hardware or software past its EOS date, verifying zero critical assets are unsupported.
1. Software Prune Test: 50+ tests attempting to deploy a container with an EOL base image (e.g., Ubuntu 18.04), verifying the CI/CD pipeline blocks it.

MLCS-Physical

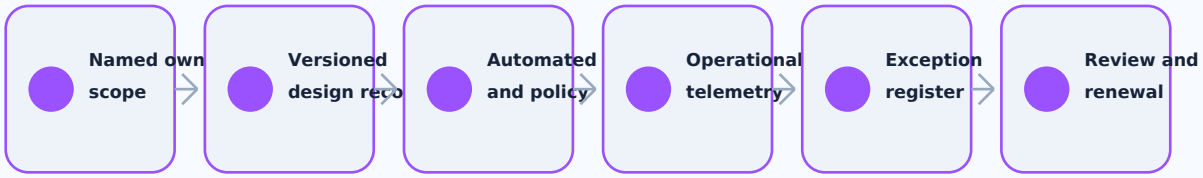
1. Cable Audit: 100+ visual and DCIM inspections verifying zero zip ties are present and all cables match the strict color-code matrix.
1. Blind Trace Test: 50+ tests requiring a technician to trace a specific port to its origin, verifying the DCIM and labeling system allows instant identification.
1. Sanitization Verification: 50+ tests attempting to recover data from retired drives, verifying cryptographic erasure or physical shredding was successful.

Execution Protocol

ASSURANCE AND ADOPTION

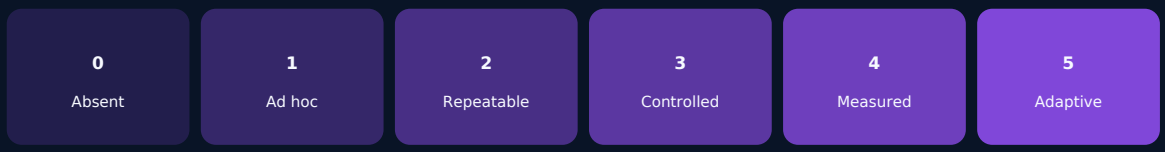
Implementation evidence and review gates

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
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.

ENGINEERING STANDARDS LIBRARY / PLATFORM ENGINEERING

Digital Marketplace and Extension Architecture Standard

Governed extension discovery, trust tiers, package integrity, compatibility, monetization boundaries, and marketplace operations.

PERMANENT DOCUMENT ID

MYTHOS-PLT-0006

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

Digital Marketplace and Extension Architecture Standard

DOCUMENT ID
MYTHOS-PLT-0006

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

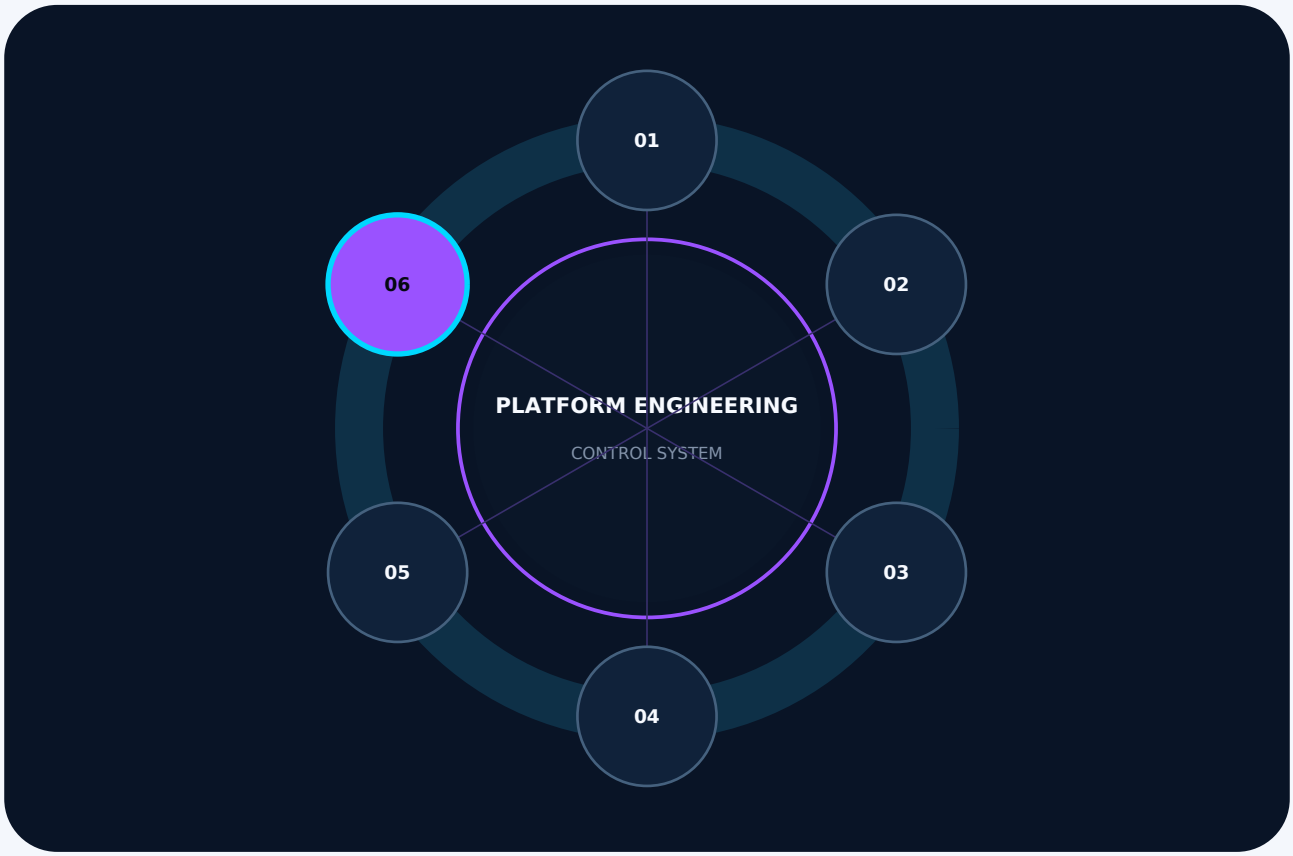
SCHEDULED REVIEW
2027-01-24

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-PLT-0006 inside the platform engineering standard constellation



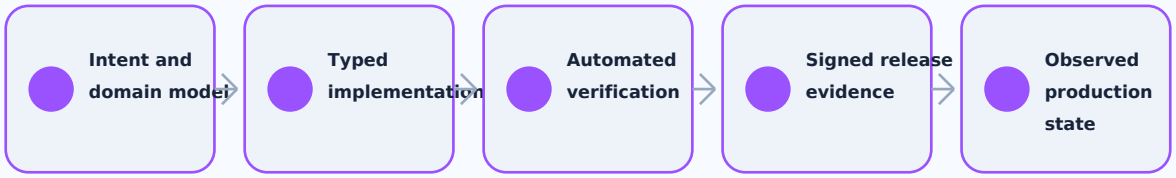
Connected assurance	Architecture, implementation, operations, evidence, and lifecycle are evaluated as one controlled system.
Specialization rule	Companion standards may add precision, but they cannot silently weaken this publication's requirements.

EXECUTIVE POSITION

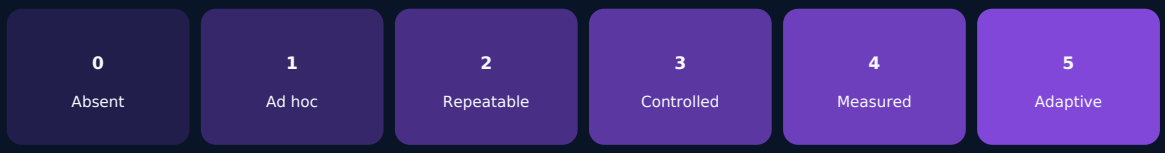
Digital Marketplace and Extension Architecture Standard

The architecture of digital marketplaces and extension ecosystems has failed.

Evidence-bearing delivery path



Assurance maturity spectrum



CONTENTS

Publication navigation

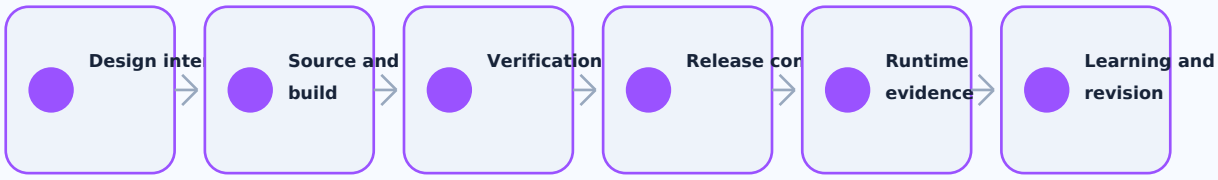
CONTROL TOPOLOGY

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
Sandboxed Execution and Isolation (Weight: 20%)	1
Capability-Based Authorization (Weight: 20%)	1
Supply Chain and Transparency Logs (Weight: 15%)	1
Revocation and Kill Switches (Weight: 15%)	1
DRM and Licensing Resilience (Weight: 15%)	1
Developer Experience (DX) and APIs (Weight: 15%)	1
Suite Composition	2
Additional Evaluation Dimension	4

Lifecycle control loop



EVALUATION INDEX

Criteria and evidence map

#	Capability family	Criterion
1	Sandboxed Execution and Isolation (Weight: 20%)	Boundary Integrity
2	Capability-Based Authorization (Weight: 20%)	Permission Scoping
3	Supply Chain and Transparency Logs (Weight: 15%)	Digital Good Provenance
4	Revocation and Kill Switches (Weight: 15%)	Threat Neutralization
5	DRM and Licensing Resilience (Weight: 15%)	Ownership Verification
6	Developer Experience (DX) and APIs (Weight: 15%)	Publishing Pipeline
7	Suite Composition	MMS-Sandbox
8	Suite Composition	MMS-SupplyChain
9	Additional Evaluation Dimension	Marketplace Dimensions
10	Additional Evaluation Dimension	The Marketplace 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 digital marketplaces and extension ecosystems has failed.

Organations build platform stores (like Chrome Extensions or app stores) that rely on "review theater" rather than architectural isolation. They allow third-party developers to upload native code, unverified AI tools, and opaque model weights. When an extension goes rogue—or is purchased by a malicious actor—it silently exfiltrates user data, injects prompt payloads into the host AI, or crashes the system because it was granted blanket permissions at install time.

This standard exists because:

1. Review Theater is Not Security: Manual vetting of marketplace code cannot catch obfuscated malware or zero-day exploits. Extensions, bolt-on features, and MCP servers MUST be executed inside hardware-backed or WebAssembly (WASM) sandboxes with strict memory isolation.
2. Blanket Permissions are an Attack Vector: An avatar pack or a UI theme should not have access to the file system or network. An MCP tool should only be able to read specific directories. Systems MUST enforce deny-by-default, capability-based security where extensions request granular, cryptographically verifiable scopes.
3. Digital Goods Require Supply Chain Provenance: A malicious voice model or a poisoned MCP server distributed through the PANTHEON marketplace can backdoor the entire infrastructure. All digital goods MUST be signed (Sigstore/Cosign), carry an AIBOM/SBOM, and be recorded in a transparency log (Rekor) before publication.
4. Centralized DRM is a Single Point of Failure: If the marketplace licensing server goes down, users lose access to their purchased avatars and tools. Digital Rights Management (DRM) and licensing MUST be decentralized, utilizing zero-knowledge proofs (ZKPs) or offline cryptographic tokens to verify ownership without phoning home.

This document establishes the Mythos Marketplace Standard (MMS), a comprehensive, evidence-based methodology for evaluating digital marketplace architectures against sandboxed execution, capability scoping, and supply chain integrity.

Scope and Applicability

In Scope

This standard covers the evaluation of:

1. Digital marketplace architectures (PANTHEON, Chrome Web Store, VS Code Marketplace)

1. Extension and plugin execution models (WASM, microVMs, native processes)
1. MCP (Model Context Protocol) server and agentic tool distribution
1. Digital Rights Management (DRM) and offline licensing (ZKPs, token-bound ownership)
1. Supply chain security for digital goods (avatars, voices, themes, code)
1. Revocation, kill switches, and transparency logs (Rekor)

Out of Scope

1. General AI agent framework evaluation (covered in MYTHOS-AI-0001)
1. General WebAssembly runtime evaluation (covered in MYTHOS-EMT-0001)
1. General identity and PKI (covered in MYTHOS-ID-0001/0002)

Audience

1. Platform engineers building marketplaces and extension hosts (PANTHEON)
1. Security engineers evaluating third-party code and supply chains
1. AI engineers building MCP tool registries and agentic capability boundaries
1. UX/UI designers distributing themes and avatars
1. Standards bodies seeking a reference marketplace security methodology

Definitions and Taxonomy

Marketplace Dimensions

Dimension	Definition
Sandboxed Extension	A bolt-on feature, MCP tool, or theme that executes inside a WebAssembly (WASM) or microVM boundary, strictly isolated from the host application's memory and system resources.
Capability Grant	A cryptographically signed, non-forgeable token granting an extension access to a specific resource (e.g., `read:./project/`, `fetch:api.weather.com`) for a limited time.
Transparency Log (Rekor)	An append-only cryptographic ledger that records the publication of digital goods (extensions, avatars, voices), providing verifiable proof of what was published and when.
Zero-Knowledge DRM	A licensing model where the user proves ownership of a digital good (e.g., a premium avatar) via a Zero-Knowledge Proof, without revealing their identity or requiring a centralized server check.
Kill Switch	A cryptographic revocation mechanism embedded in the host platform that instantly disables a distributed extension or digital good if it is found to be malicious.

The Marketplace Doctrine

> A vetted extension is still a threat. A UI theme with network access is a spy. A marketplace without a transparency log is a malware distribution engine. If the permissions are not granular, the user is compromised.

Evaluation Methodology

Scoring Model

Each capability is scored from 0 to 5.

Score	Meaning
0	Fails basic task; native code execution, blanket permissions, no signing, centralized DRM
1	Basic review process, but no architectural sandboxing or capability scoping
2	Functional extensions, but lacks WASM isolation or supply chain attestations
3	Solid production performance; WASM sandboxed, capability grants, signed goods, transparency logs
4	Strong performance; ZKP offline DRM, automated kill switches, formally verified isolation
5	Best-in-class; sets the standard for mathematically proven, zero-trust digital distribution

Weighting

Category	Weight	Rationale
Sandboxed Execution & Isolation	20%	Native extensions guarantee host compromise eventually
Capability-Based Authorization	20%	Blanket permissions allow themes/tools to exfiltrate data
Supply Chain & Transparency Logs	15%	Unsigned digital goods allow supply chain poisoning
Revocation & Kill Switches	15%	Malicious extensions must be instantly neutralized globally
DRM & Licensing Resilience	15%	Centralized licensing servers create single points of failure
Developer Experience (DX) & APIs	15%	If publishing is hard, developers will bypass the marketplace

Total: 100%

A system MUST score at least 3.0 in Sandboxed Execution and Capability Authorization to be considered for production use.

Evaluation Categories

Sandboxed Execution and Isolation (Weight: 20%)

Boundary Integrity

How are third-party extensions, tools, and avatars executed?

Score	Criteria
0	Native executables or direct script injection into host process (critical failure)
1	Process isolation, but extensions share host filesystem/network access

Score	Criteria
2	Containerized execution, but heavy startup latency
3	WebAssembly (WASM) Component Model used for all extensions; strict memory isolation
4	Heterogeneous execution: WASM for logic, microVMs (Firecracker) for heavy MCP tools
5	Perfect isolation: formally verified sandbox boundaries, zero host attack surface, mathematical proof of memory protection

Capability-Based Authorization (Weight: 20%)

Permission Scoping

Are extensions restricted to least-privilege access?

Score	Criteria
0	Extensions run with the full privileges of the host application
1	Static manifest of permissions requested at install time (e.g., "Access all files")
2	Basic sandboxing, but network access is all-or-nothing
3	Deny-by-default; extensions must request granular capability grants (e.g., `fetch:api.github.com`) via the host's policy engine
4	Capabilities are time-boxed and attenuated; an MCP tool cannot delegate more access than it possesses
5	Perfect authorization: formally verified capability bounds, zero ability to bypass scopes, cryptographic non-repudiation of resource access

Supply Chain and Transparency Logs (Weight: 15%)

Digital Good Provenance

Are avatars, voices, and MCP tools verified before distribution?

Score	Criteria
0	Unverified files uploaded directly to marketplace CDN
1	Manual malware scan before publication
2	Artifacts signed by the developer, but marketplace does not verify signatures
3	Mandatory Sigstore/Cosign signing; SBOM/AIBOM generated for all goods
4	All publications recorded in a transparency log (Rekor); marketplace refuses to serve unsigned goods
5	Perfect provenance: formally verified supply chain, in-toto attestations bound to artifacts, zero ability to distribute poisoned goods

Revocation and Kill Switches (Weight: 15%)

Threat Neutralization

Can a malicious extension be instantly disabled globally?

Score	Criteria
0	No revocation mechanism; relies on users manually uninstalling
1	Marketplace page removed, but extension continues to function if installed
2	Centralized blocklist checked at startup, but fails offline
3	Cryptographic revocation (CRL/OCSP) checked before execution; cached for offline use
4	Hard kill switch: host platform queries Rekor transparency log and instantly neutralizes malicious extensions at runtime
5	Perfect neutralization: formally verified revocation propagation, zero ability for revoked code to execute, sub-second global kill capability

DRM and Licensing Resilience (Weight: 15%)

Ownership Verification

How is user ownership of premium digital goods verified?

Score	Criteria
0	No DRM; goods are freely copyable or tied to a static, easily bypassed license key
1	Centralized licensing server checked on every launch (fails offline)
2	Online activation with offline grace period, but server outage blocks new activations
3	Token-bound licensing: ownership bound to the user's cryptographic identity (e.g., Passkey/SPIFFE)
4	Zero-Knowledge DRM: user proves ownership of the marketplace token via ZKP without revealing identity or phoning home
5	Perfect resilience: formally verified licensing bounds, zero central server dependency, cryptographic proof of ownership

Developer Experience (DX) and APIs (Weight: 15%)

Publishing Pipeline

Is the publishing pipeline secure, automated, and developer-friendly?

Score	Criteria
0	Manual email submission of zip files to marketplace admins
1	Basic web portal upload, but manual signing required by the developer
2	CLI tool for publishing, but lacks CI/CD integration
3	Keyless signing via OIDC in CI/CD; automated publishing via API
4	Automated policy-as-code evaluation of extension behavior before publication
5	Perfect pipeline: formally verified publishing bounds, zero friction for developers, mathematical proof of artifact integrity before public listing

The Mythos Marketplace Suite (MMS)

Traditional marketplace benchmarks measure download speed and catalog size. The MMS evaluates sandbox escape resistance, capability enforcement, and supply chain integrity.

Suite Composition

MMS-Sandbox

1. Escape Attempt: 100+ tests executing malicious WASM modules designed to exploit memory bounds, spectre side-channels, and host API abuses, verifying the host remains uncompromised.
1. Capability Violation: 50+ tests attempting to read local files or make network calls from a UI theme or avatar pack, verifying the capability engine blocks the action.

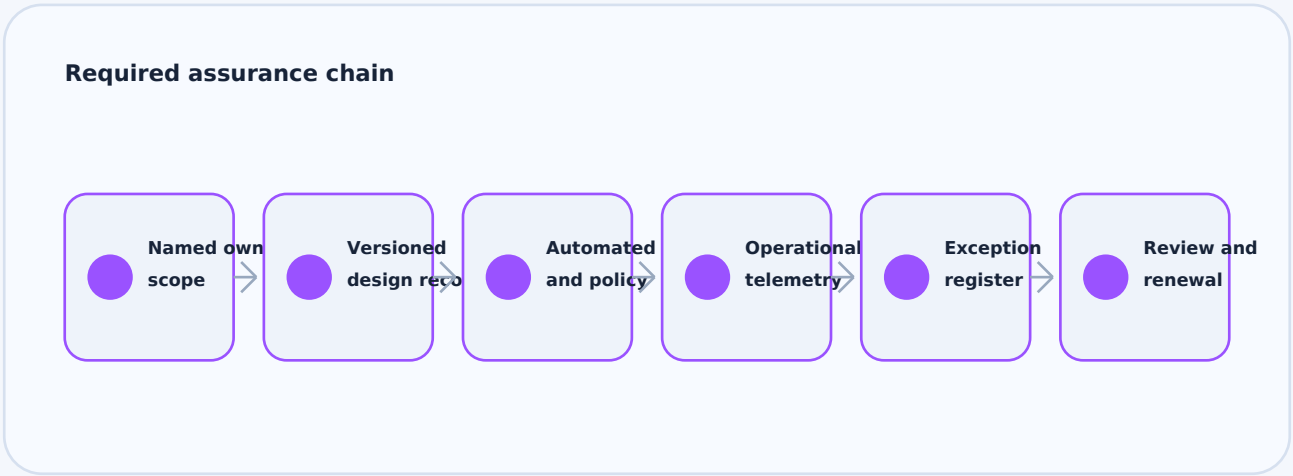
MMS-SupplyChain

1. Poisoned Good Test: 100+ tests attempting to upload an unsigned MCP tool or a voice model lacking an AIBOM, verifying the marketplace API rejects it.
1. Revocation Test: 50+ tests marking a published extension as compromised, verifying the host platforms instantly disable it for all users via the transparency log.

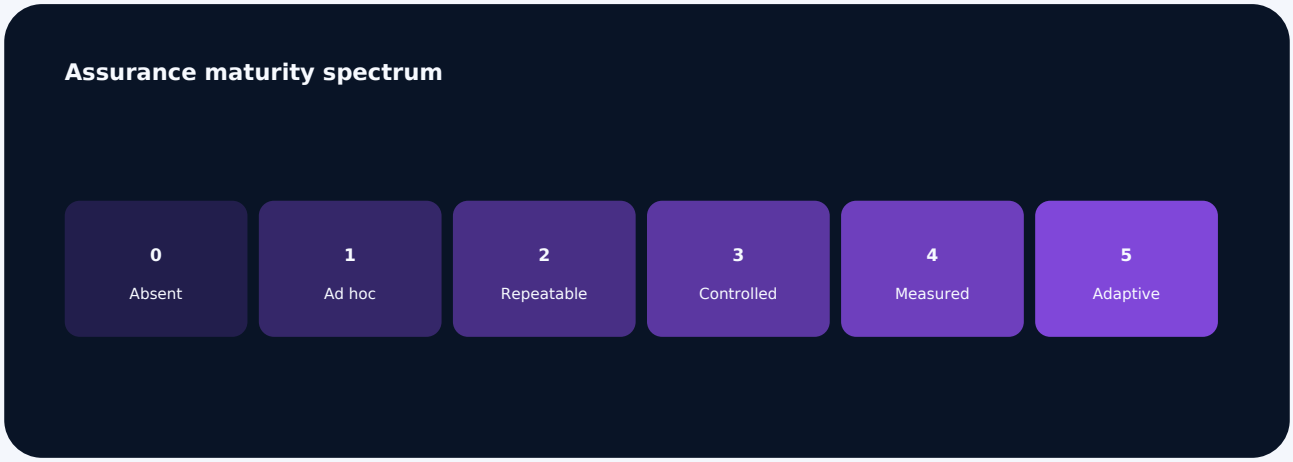
Execution Protocol

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.