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

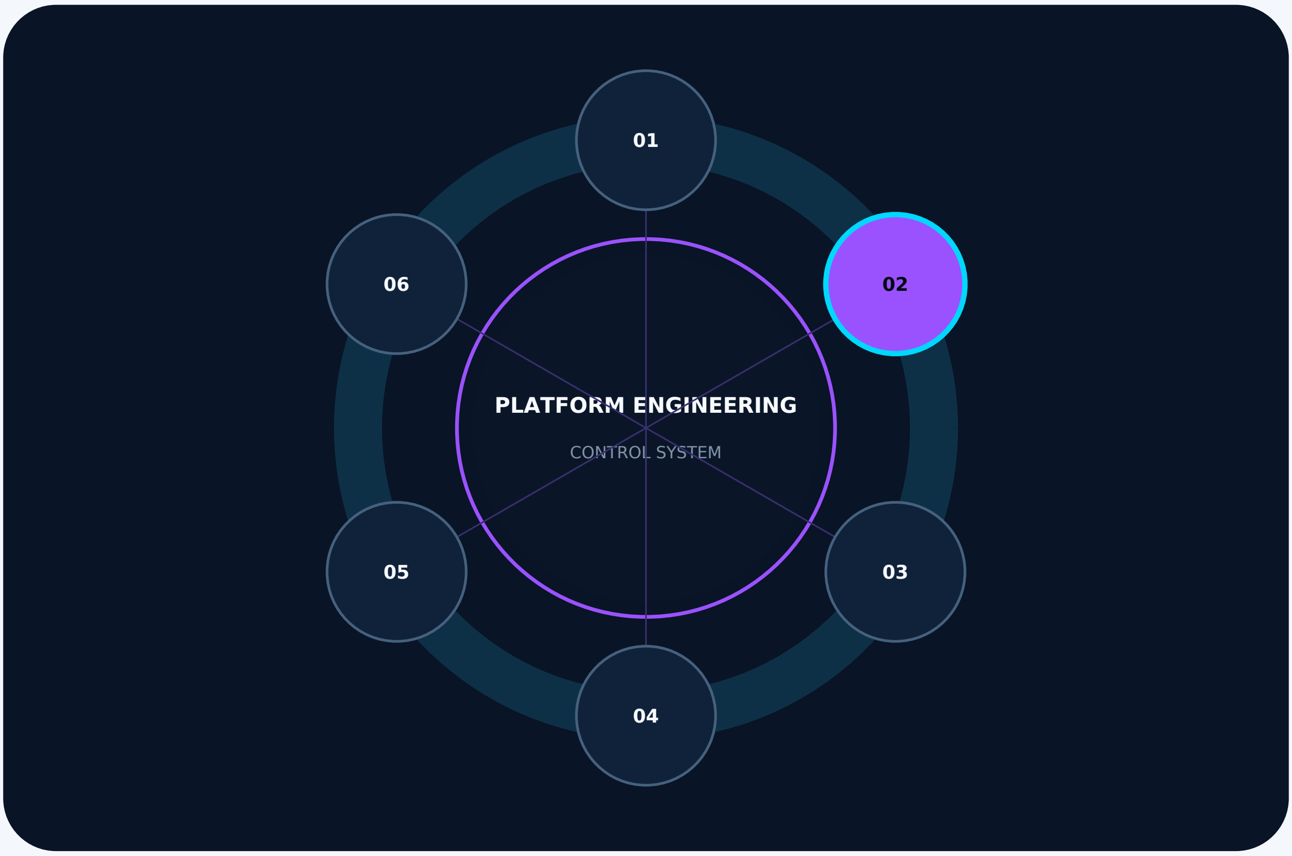
SCHEDULED REVIEW
2027-01-24

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

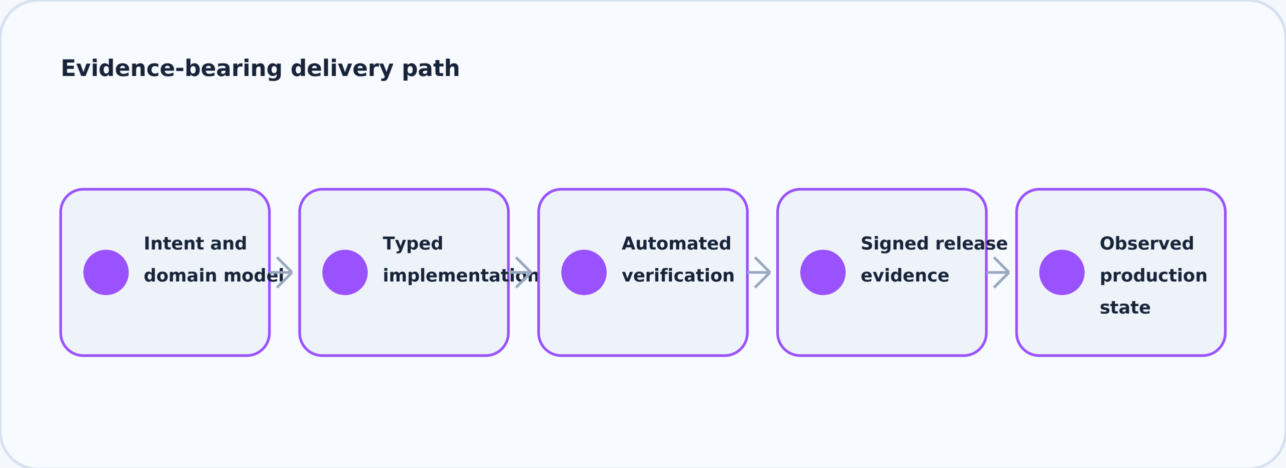


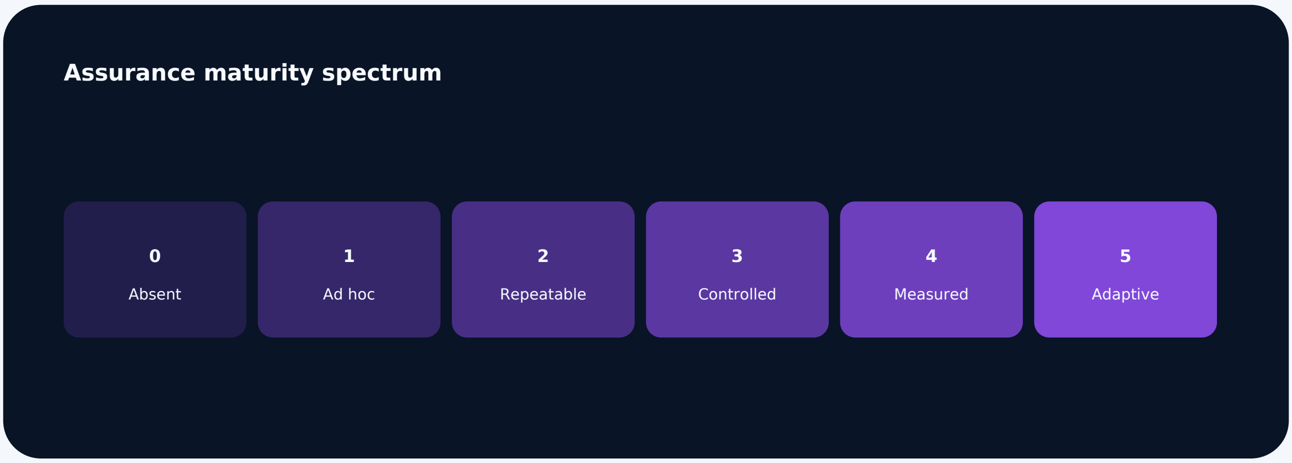
Connected assurance	Architecture, implementation, operations, evidence, and lifecycle are evaluated as one controlled system.
Specialization rule	Companion standards may add precision, but they cannot silently weaken this publication's requirements.

EXECUTIVE POSITION

Infrastructure and Configuration as Code Standard

The architecture of infrastructure provisioning has failed.





CONTENTS

Publication navigation

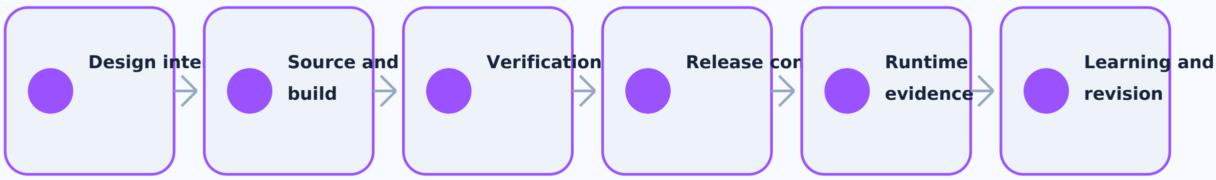
CONTROL TOPOLOGY

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

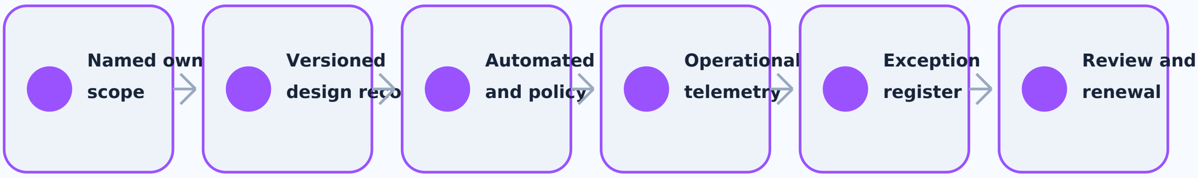
1. Malicious PR Test: 100+ tests submitting a PR that creates an open S3 bucket or weak IAM policy, verifying the Policy-as-Code engine blocks the PR from merging.
1. Secret Leak Test: 50+ tests attempting to store a database password in the Terraform code, verifying the pipeline blocks it and requires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

Execution Protocol

ASSURANCE AND ADOPTION

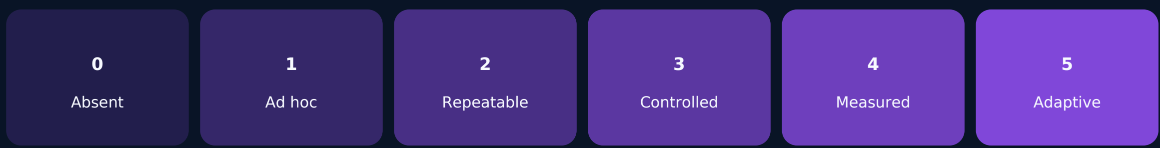
Implementation evidence and review gates

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.