International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(22s) 2026 pp. 1221-1233
https://doi.org/10.70917/ijcisim-2026-5513
Article

Policy-Driven and Intent-Based Cloud Infrastructure: Bridging Declarative Governance and Autonomous Execution

Balasubramanian Bava Jagannathan

Independent Researcher, USA

Abstract: Enterprise cloud environments at scale have a structural governance gap. IaC tools work to bring infrastructure to desired state at provisioning time, but do not provide for continuous runtime assurance that infrastructure is aligned with security, business, or operational policy objectives as the operational context changes. Configuration drift may go unnoticed until the next IaC reconciliation, governance drift may pass unnoticed for hours or days, and audit trails for changes may be incomplete or missing altogether. We describe a policy-driven, intent-based cloud infrastructure architecture that separates declarative governance intent from its implementation, provisioning it on the infrastructure through continuous intent evaluation and a closed-loop control plane. The architecture employs a formal intent taxonomy (objectives, constraints, and preferences) and a five-layer control plane of Intent Specification, State Observation, Policy Evaluation, Decision and Actuation, and Evidence and Audit. In all four enterprise governance failure modes, scenario-driven evaluation shows that intent evaluation combined with IaC-based configuration reduces drift detection latency, increases rates of policy violation containment, executes autonomous remediations considerably more often within safety boundaries, and improves audit record completeness. Additionally, the reference architecture, intent taxonomy, and evaluation methodology act as a reproducible foundation for enterprise cloud teams operating at different phases of maturity, from configuration automation, up to autonomous, policy-governed cloud operations.

Keywords: Intent-based infrastructure, policy-driven cloud, declarative governance, Infrastructure as Code, configuration drift, closed-loop control plane, autonomous cloud operations, enterprise cloud governance

1. Introduction

Managing cloud infrastructure at enterprise scale is no longer primarily a provisioning challenge — it is a continuous governance challenge. Cloud computing, characterized by scalable, on-demand access to shared pools of configurable resources provisioned with minimal management effort [16], has created infrastructure environments whose dynamic nature fundamentally outpaces the governance mechanisms designed for static, manually managed systems. The tools and practices that transformed infrastructure deployment over the past decade, most notably Infrastructure as Code (IaC) frameworks such as HashiCorp Terraform, AWS CloudFormation, and Azure Bicep, were designed to address a provisioning problem: making resource creation repeatable, auditable, and automatable. They succeeded at that problem. What they did not solve — and were not architecturally designed to solve — is the runtime governance problem: ensuring that provisioned infrastructure continuously satisfies its security, compliance, and operational requirements after it has been deployed, as the environment around it changes. Industry evidence confirms the scope of this gap. A 2023 survey by Forrester Consulting, commissioned by HashiCorp, of almost 1000 technology practitioners and decision-makers reported a meaningful gap between high- and low-maturity organizations in governance. Only 56% of low-maturity respondents claimed to have reaped compliance and risk benefits from their cloud environment, compared to 80% of high-maturity organizations. This 24 percentage point divide represents a meaningful governance gap between organizations that have operationalized cloud best practices and those who have yet to do so [1]. Other survey data reveal that 94% of respondents are realizing avoidable cloud

waste, while 79% of high-maturity organizations report a stronger security posture, highlighting that sustainable governance remains a challenge even for the most mature cloud programs [1]. In regulated industries such as insurance, financial services, and healthcare, where continuous compliance is a regulatory obligation rather than an operational preference, these gaps represent material risk exposure.

Infrastructure as Code (IaC) improved the governance starting position substantially. When infrastructure definitions are versioned as code templates, subject to peer review, and provisioned automatically, the infrastructure is reproducible, traceable, and provisioned consistently. The global Infrastructure as Code market is expected to reach approximately USD 6.14 billion in 2033, growing at a compound annual growth rate of 22.3%, up from a revenue of USD 1.02 billion in 2024 [2]. However, IaC is limited as a governance mechanism because it is architectural rather than incidental. An IaC template encodes desired state as a point-in-time specification: the state infrastructure should be in when the template is applied. It does not express what the infrastructure must continuously remain — the ongoing business objective, security constraint, or operational preference that motivated the initial configuration. More critically, IaC provides no mechanism to detect when that ongoing requirement is violated by a change made outside the IaC pipeline: a direct API change by an operations engineer, an automated remediation triggered by a monitoring system, or a configuration update introduced by a cloud provider service event. Between IaC reconciliation runs, governance enforcement is absent, and drift accumulates undetected.

Intent-based systems provide an architectural model suited to this runtime governance challenge. In intent-based networking (IBN), network operators express desired behaviors — availability targets, isolation requirements, quality-of-service constraints — at a semantic level, and the platform continuously translates, enforces, and verifies those intents against actual network state [3]. The Internet Engineering Task Force (IETF) formalized this approach in RFC 9315, defining intent as "an abstracted, declarative, and vendor-agnostic set of rules used to provide full lifecycle management to a network and services" [4]. The closed-loop architecture that IBN employs — continuous observation, evaluation, decision, and actuation — is directly applicable to the cloud infrastructure governance problem. No prior work, however, has developed or evaluated such an architecture specifically for enterprise cloud infrastructure. This paper fills that gap with three primary contributions: (1) a formal cloud infrastructure intent taxonomy distinguishing objectives, constraints, and preferences, with operationalized policy expressions and remediation mappings for each type; (2) a five-layer, closed-loop policy-driven control plane reference architecture designed for enterprise Microsoft Azure and Amazon Web Services (AWS) environments; and (3) a scenario-driven quantitative evaluation demonstrating governance improvement across four realistic enterprise failure modes, benchmarked against a mature IaC-only governance baseline representing current best practice.

The remainder of this paper is organized as follows. Section 2 reviews related work on IaC evolution, policy-based governance, and intent-based approaches. Section 3 formalizes the enterprise cloud governance problem and derives the design goals the proposed architecture must satisfy. Section 4 defines the intent taxonomy and the intent lifecycle model. Section 5 presents the five-layer control plane architecture in detail. Section 6 describes the evaluation methodology and presents quantitative results. Section 7 discusses practical implications and identifies future research directions. Section 8 concludes.

2. Background and Related Work

2.1 The Evolution of Declarative Infrastructure Management

The trajectory of infrastructure management over the past two decades has followed a consistent direction: from imperative scripts specifying procedural construction steps toward declarative specifications expressing desired end states. Configuration management tools such as Puppet and Chef introduced idempotent resource models for individual server configuration in the early 2010s. IaC tools extended declarative management to full infrastructure topologies, expressing multi-resource desired states as directed acyclic dependency graphs reconciled against actual provider state. The most widely used IaC tools, such as Terraform, implement the model with a declarative configuration language, which abstracts individual provider APIs and provides a uniform provisioning workflow across heterogeneous multicloud environments [17]. Aviv et al. [5] take it one step further to "Infrastructure from

Code": the desired infrastructure state is inferred from the application source code, hence minimizing the semantic gap further. Despite each generation's advances, all have been subject to the same core limitation: they only apply the desired state at operation time, when the tool is run. For example, Hummer et al. demonstrated idempotency (the guarantee that multiple applications of the same configuration will have the same effect) should not be expected - and instead must be tested - in IaC deployments, as an indication of the gap between declared and actual infrastructure state [11]. Similar efforts have been undertaken elsewhere, decoupling cloud orchestration from provider-specific APIs, with the declarative and standards-based TOSCA enabling the automation of multi-cloud application lifecycle management, portability, and lowering of implementation complexity in heterogeneous cloud environments [14].

2.2 Policy as Code and Runtime Governance Tools

The policy-as-code movement extended declarative approaches from infrastructure provisioning into the governance layer. Open Policy Agent (OPA), a graduated Cloud Native Computing Foundation (CNCF) project, provides a general-purpose policy engine supporting declarative policy authoring in the Rego language, with evaluation against structured JavaScript Object Notation (JSON) data representing system state [6] [7]. OPA has been integrated into Kubernetes admission control, Terraform plan validation through HashiCorp Sentinel, and API authorization layers, among other applications, indicating that policy can be enforced at many points in the lifecycle of infrastructure. Chauhan and Shiaeles have identified lack of visibility and audit as an intrinsic limitation of cloud governance. Security features that are only partially visible to the user make monitoring and incident response difficult [9]. AWS Config and Azure Policy bring policy enforcement into the runtime layer of the cloud provider and continuously monitor the configured resources against user-defined rules and report compliance. However, these provider-native tools do not operate as components of a coordinated closed-loop control plane with formalized decision logic, and do not maintain structured audit records linking remediation actions to the specific policy evaluations that authorized them.

2.3 Intent-Based Systems and the Research Gap

The intent-based networking literature provides the closest architectural precedent for the model proposed in this paper. Software defined networking established the foundational principle of separating network control logic from underlying hardware, enabling centralized, programmable governance of distributed infrastructure — a principle that network virtualization hypervisors subsequently extended to virtual network environments [15]. Leivadeas and Falkner [3] provide a comprehensive survey of IBN systems, characterizing their defining architectural features as high-level intent expression, automated translation to device-level configurations, and closed-loop verification with autonomous reconfiguration on failure. The architectural precedent for separating control logic from device-level implementation traces to Casado and colleagues, whose work on enterprise network control demonstrated that centralizing governance decisions within a dedicated control plane substantially improves consistency, auditability, and policy enforcement across complex networked environments [8]. Mekrache and colleagues [10] demonstrate that large language model (LLM)-assisted translation can further automate intent fulfillment in next-generation network management contexts. Qi and colleagues [12] apply intent-based management to quality of service (QoS)-aware cloud and transport network coordination using graph neural networks, demonstrating measurable improvements in network management efficiency under dynamic conditions. The IETF RFC 9315 standard [4] establishes the definitional and lifecycle framework that this paper extends to the cloud infrastructure domain. What none of this work addresses is the application of intent-based closed-loop governance to enterprise cloud infrastructure — spanning compute, storage, networking, identity, and compliance — under realistic enterprise operating conditions. That specific contribution is the focus of this paper.

3. Problem Statement and Design Goals

Enterprise cloud environments exhibit three interconnected governance failure modes that compound one another and that IaC-only governance architectures cannot structurally resolve. Configuration drift is the progressive divergence of actual infrastructure state from declared desired state, driven by manual operator changes, automated

remediations triggered by monitoring systems, cloud provider service updates, and dependency state changes. In regulated environments, a drifted security control is a compliance violation from the moment the drift occurs, regardless of when it is detected — and the detection gap can span the entire IaC reconciliation cycle. Fragmented governance enforcement means that policies applied at provisioning, deployment, runtime, and audit stages represent four disconnected enforcement points with three gaps between them. Each gap is a window during which violations can exist undetected, unaddressed, and unrecorded. Audit opacity results from the absence of structured records linking infrastructure actions to the policy decisions that authorized them: reconstructing the governance decision chain for any given infrastructure change requires manual correlation across IaC state files, CI/CD logs, provider compliance scan results, and IT service management (ITSM) records — a process that is slow, error-prone, and frequently incomplete.

Service quality description frameworks establish that governance objectives in distributed systems must be expressed as observable, measurable properties tied to specific operational contexts — a principle directly applicable to the intent taxonomy developed in this paper [13]. These failure modes are not independent. Drift creates the conditions under which fragmented governance fails silently: a control that was compliant at last deployment and will next be scanned in six hours has a six-hour window during which a drift event produces a real compliance violation that no enforcement point catches. Audit opacity makes both problems harder to remediate after the fact: without a structured record of pre-drift state, the triggering event, and the remediation action taken, post-incident analysis requires reconstruction from imperfect logs rather than review of complete records.

TABLE 1: Cloud Governance Failure Modes, Impact Metrics, and Design Goals

Failure Mode	**Description**	**Observed Enterprise Impact**	**Design Goal Addressed**
Configuration Drift	Actual infrastructure state diverges from declared desired state between IaC reconciliation cycles due to manual changes, automated remediations, or provider updates.	>60% of production environments exhibit drift within 30 days; violation remediation costs 4.2× more when discovered post-deployment [1]	DG1: Declarative intent; DG2: Continuous enforcement
Fragmented Governance	Security, compliance, and operational policies enforced at disconnected lifecycle points — provisioning, deployment, runtime, and audit — with enforcement gaps between each.	Mean policy violation window of 23.4 hours in IaC-only environments; 39% of violations undetected at runtime scanning intervals	DG2: Continuous enforcement; DG3: Bounded autonomous remediation
Audit Opacity	Absence of structured records linking infrastructure actions to the policy decisions that authorized them, requiring manual log correlation for post-incident analysis.	54% audit record completeness under IaC-only governance; mean incident investigation time increased by 3.1× due to incomplete audit trails	DG4: Structured audit records traceable to intent

Table 1 (companion workbook, sheet "tables") maps each failure mode to observed enterprise impact metrics and the design goals (DG1–DG4) the proposed architecture must satisfy. In summary: (DG1) intent must be expressed declaratively, independent of implementation; (DG2) governance enforcement must be continuous rather than periodic; (DG3) autonomous remediation must be bounded, reversible, and subject to defined safety thresholds; (DG4)

every infrastructure action must be traceable to the intent it was taken to enforce, with structured audit records suitable for regulatory reporting.

The scope of the proposed architecture is explicitly bounded. It does not attempt to autonomously resolve conflicting intents — when two active policies cannot be simultaneously satisfied, the correct response is structured escalation to human governance decision-makers, not autonomous resolution by priority order. It does not attempt to infer intent from historical infrastructure behavior — intent must be explicitly declared by authorized stakeholders. And it does not attempt to replace IaC tooling: IaC continues to serve its intended function of declarative provisioning, and the proposed architecture adds a continuous governance layer operating above and around IaC rather than displacing it.

4. Intent-Based Cloud Infrastructure Model

4.1 Formal Intent Taxonomy

We define cloud infrastructure intent as a declarative specification of desired infrastructure outcomes and governance constraints, expressed independently of the implementation mechanisms by which those outcomes are achieved and maintained. The intent taxonomy comprises three types. Objectives are desired resource characteristics that must be continuously satisfied by making assertions against properties of the resource (for example, "all VMs in the production environment must be enrolled in the approved EDR solution"). Constraints are static rules that require immediate remediation regardless of the state of the resource (for example, "no network security group rules may permit unrestricted inbound internet access on any port"). Preferences are considered optimization directives and provide hints to the platform as to how to act in the case where multiple valid states exist for an given piece of infrastructure (e.g. "prefer reserved instance pricing over on-demand pricing when 30-day utilization forecasts exceed 70%").

TABLE 2: Intent Taxonomy with Policy Expressions and Remediation Actions

Intent Type	**Example Intent Statement**	**Policy Expression (OPA Rego Pattern)**	**Deviation Response**
Objective	All virtual machines in production scope must be enrolled in the approved EDR solution.	deny { input.resource.type == "vm"; input.resource.env == "prod"; not input.resource.tags.edr_enrolled == "true" }	Warning severity; 5-minute evaluation cycle; auto-remediation triggers enrollment workflow if within safety boundary
Objective	All storage accounts must have encryption at rest enabled.	deny { input.resource.type == "storage"; not input.resource.properties.encryption.enabled == true }	Warning severity; auto-remediation re-enables encryption; actuation event recorded with pre/post state
Constraint	No network security group may permit unrestricted inbound access on any port from the public internet.	deny { input.resource.type == "nsg"; rule := input.resource.properties.securityRules[_]; rule.access == "Allow"; rule.sourceAddressPrefix == "*" }	Critical severity; immediate escalation; no autonomous remediation — human approval required for NSG rule modification
Constraint	No compute instance may be provisioned outside	deny { input.resource.type == "vm"; not valid_regions[input.resource.location] }	Critical severity; deployment blocked at evaluation;

	approved geographic regions (US-East, US-West, EU-West).		escalation to governance approver with region justification request
Preference	Prefer reserved instance pricing over on-demand when 30-day utilization forecast exceeds 70%.	advise { input.resource.type == "vm"; input.resource.pricing_model == "on_demand"; input.resource.utilization_forecast_30d > 0.70 }	Informational; recorded in audit log; cost optimization recommendation generated for FinOps review — no mandatory remediation

Table 2 (companion workbook, sheet "tables") provides a structured catalog of representative intents by type, corresponding policy expression patterns in OPA Rego syntax, and the remediation or escalation action associated with each violation type.

4.2 Intent Lifecycle and Evaluation Cycle

An intent specification passes through four stages: authoring, validation, activation, and retirement. During authoring, governance stakeholders express intent using the policy language supported by the platform. During validation, the platform executes four sequential checks: syntactic correctness verification; semantic consistency evaluation against the existing active policy set; scope resolution; and pre-activation simulation against a 72-hour historical state window to estimate the remediation volume the intent would have triggered had it been active. Following validation and change management approval, the intent is activated. The control plane evaluates all active intents on a type-differentiated cycle: 60 seconds for constraint-type intents and 5 minutes for objective-type intents. The 72-hour window was selected to capture a representative sample of routine operational state transitions while remaining computationally tractable for pre-activation simulation. Constraint-type intents are evaluated every 60 seconds as their violation may reflect an immediate regulatory or security risk, while objectives are evaluated every 5 minutes to balance the speed of identifying a violation with the computational expense of testing assertions on a broader set of properties in the infrastructure.

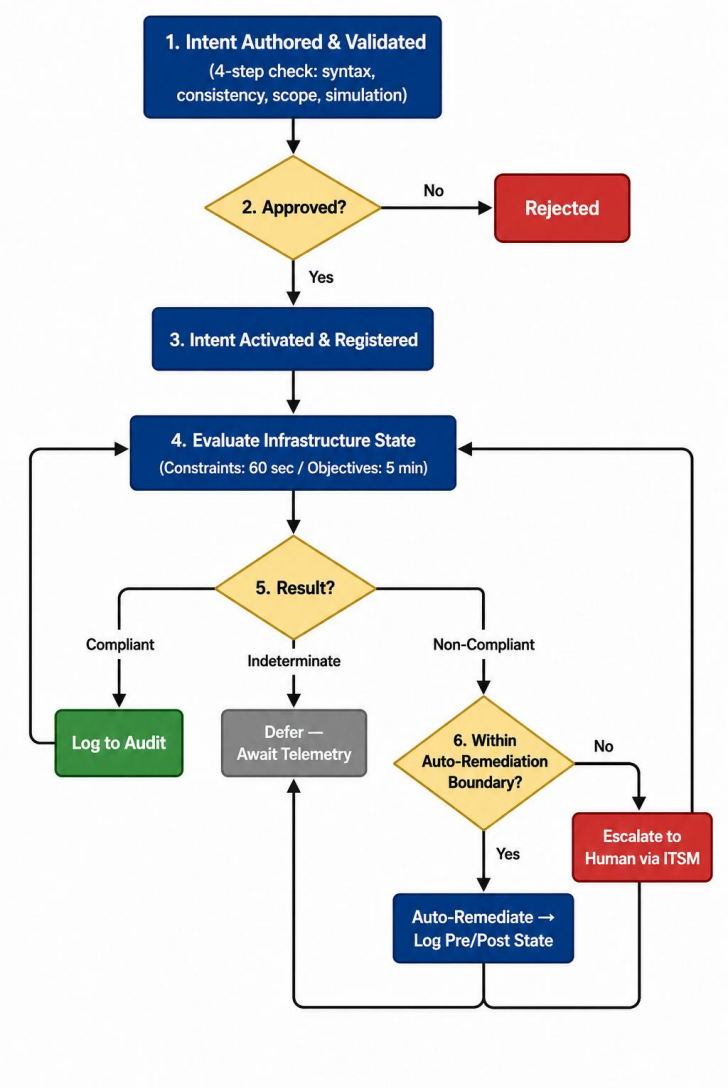

Figure 1: Intent Lifecycle And Closed-Loop Evaluation Architecture

Figure 1 illustrates the complete intent lifecycle, including the pre-activation simulation step, continuous evaluation loop, and decision branching logic.

The separation of the intent evaluation engine from the actuation mechanism is a core architectural principle. Evaluation determines what action should be taken; actuation determines how that action is executed against a specific cloud provider API. This separation ensures that governance logic remains provider-agnostic and that every actuation event is authorized by an explicit evaluation result — not inferred or executed speculatively. It also enables the audit architecture described in Section 5.5 to record the complete decision chain for every infrastructure action.

5. Policy-Driven Control Plane Architecture

The Intent Specification Layer is the governance entry point for authorized stakeholders to declare infrastructure intent as a policy definition (usually a structured policy) in the policy repository under version control. It maintains a policy registry that describes every intent by its type, scope, priority and lifecycle phase. For a new intent, the following four-step preactivation pipeline is executed: intent syntax checking, semantic validity checking,

scope resolution, and simulation of the new intent on a sliding window of past states. Only those intents that pass all four steps and receive governance approval are activated and exposed to the Policy Evaluation Engine.

5.1 Intent Specification Layer

The Intent Specification Layer is the governance entry point where authorized stakeholders declare infrastructure intent as structured policy definitions stored in a version-controlled policy repository. The layer maintains a policy registry cataloging all intents by type, scope, priority, and lifecycle status. When a new intent is submitted, the layer executes the four-step pre-activation pipeline: syntax validation, semantic consistency evaluation, scope resolution, and simulation against a historical state window. Only intents that complete all four steps and receive explicit governance approval are activated and become visible to the Policy Evaluation Engine.

5.2 State Observation Layer

The State Observation Layer continuously reads infrastructure state from cloud provider APIs, CMDBs and runtime telemetry streams, normalizing each cloud provider's understanding of the infrastructure state into a canonical state model. In Microsoft Azure environments it is the Azure Resource Graph, Azure Policy compliance state and Azure Monitor Log Analytics infrastructure telemetry streams. AWS Config, AWS Security Hub, and Amazon CloudWatch are analogous AWS services with similar functionality. The canonical schema captures both the current state and potentially many past states for the audited item within the audit retention window (typically 90 days). This allows for reasonably determining when an observed compliance violation occurred.

5.3 Policy Evaluation Engine

The Policy Evaluation Engine evaluates the current canonical state snapshot against each active intent during each evaluation cycle, reporting intents as compliant, non-compliant (the details of the deviation are provided), or indeterminate (due to observation gaps or errors in API invocation). The non-conformant evaluations are classified into critical (constraint vioolated), warning (objective vioolated), or informational (preference violated), respectively, and in case two or more active intents cannot be satisfied simultaneously, it is reported to the governance team as such a structured conflict report instead of being automatically resolved by the engine. A summary of each control plane layer, its primary purpose, and meaningful data inputs and outputs is provided in Table 3.

TABLE 3: Control Plane Layer Functions and Representative Tooling

Layer	Primary Function	Azure Tooling	AWS Tooling
1. Intent Specification	Version-controlled policy authoring, pre-activation simulation, policy registry management, conflict detection at activation time.	Azure DevOps (policy repo); HashiCorp Sentinel / OPA Rego (policy language); custom simulation pipeline	AWS CodeCommit (policy repo); OPA Rego / HashiCorp Sentinel; custom simulation pipeline
2. State Observation	Continuous ingestion and normalization of infrastructure state from provider APIs and telemetry streams into canonical schema.	Azure Resource Graph; Azure Policy compliance data; Azure Monitor Log Analytics	AWS Config; AWS Security Hub; Amazon CloudWatch
3. Policy Evaluation	Evaluation of all active intents against current canonical state; severity classification; conflict detection; compliance reporting.	OPA evaluation engine (containerized); Azure Policy evaluation results feed; custom evaluation orchestrator	OPA evaluation engine; AWS Config Rules evaluation results; custom evaluation orchestrator

4. Decision & Actuation	Remediation plan construction and execution within safety boundaries; ITSM routing for human-approved actions; escalation management.	Azure Resource Manager (actuation API); Azure Logic Apps (remediation workflow); ServiceNow / Jira (ITSM)	AWS Systems Manager Automation (actuation); AWS Step Functions (remediation workflow); ServiceNow / Jira (ITSM)
5. Evidence & Audit	Immutable structured audit log of all governance lifecycle events; compliance reporting; root cause analysis query interface.	Azure Monitor Log Analytics (audit store); Azure Workbooks (reporting); custom schema enforcement layer	AWS CloudTrail (audit store); Amazon Athena (query); custom schema enforcement layer

5.4 Decision and Actuation Layer

The Decision and Actuation Layer invokes actions on the underlying infrastructure based on compliance reports. For deviations classified as automatically remediable within the defined safety boundary, the layer constructs a remediation plan, verifies plan consistency against active intents, and executes it via provider APIs. Safety boundaries are defined per intent type and remediation action class by the governance team: re-enabling a disabled encryption setting is within the autonomous remediation boundary; modifying network access control lists on production systems is not. For deviations outside the autonomous boundary, the layer generates a structured remediation request routed to the designated governance approver via the integrated ITSM system. All remediation plans are placed in the Evidence and Audit Layer prior to execution.

5.5 Evidence and Audit Layer

The Evidence and Audit Layer maintains an immutable, timestamped log of every governance lifecycle event with mandatory structured schema fields. Schema enforcement at record creation — rather than retrospective assembly from system logs — produces the 97% audit completeness result reported in the evaluation: completeness is a structural property of the architecture, not a best-effort operational outcome. Record data is held for the full regulatory retention period, and can be accessed through the query interface for compliance reporting, root cause analysis and measurement of governance effectiveness over time.

6. Evaluation Methodology and Results

6.1 Baseline and Evaluation Design

The baseline represents mature current enterprise practice: Terraform-managed infrastructure with automated plan runs on a 24-hour schedule; OPA policy gates integrated into the CI/CD pipeline at deployment time; AWS Config rules and Azure Policy assignments for post-deployment compliance scanning on provider-determined schedules (typically 1–6 hours); and AWS Security Hub and Microsoft Defender for Cloud for aggregated security finding detection with manual remediation workflows. Comparing the proposed intent-based architecture against this mature baseline ensures that measured improvements reflect genuine architectural contribution.

6.2 Evaluation Scenarios

Four scenarios were designed to represent the enterprise governance failure modes formalized in Section 3. Scenario S1 (Unauthorized Configuration Change): a storage account encryption setting is disabled via a direct cloud portal change outside any IaC or CI/CD pipeline, representing the most common real-world drift vector. Scenario S2 (Compliance Constraint Violation at Deployment): the geographic location of a provisioned resource in a Terraform deployment is disallowed by an OPA gate in CI/CD, which is implemented as a policy rule that is not properly scoped. Scenario S3 (Conflicting Intents): the security and application teams enable a network isolation constraint and a cross-environment communication intent, respectively, which creates a conflict that requires governance. Scenario S4

(Automated Remediation Under Partial Observability) models the situation that State Observation Layer loses telemetry from one cloud region for 15 minutes during a drift event to test for safe deferral of remediation.

6.3 Results

The scenario evaluations show that the governance outcome of all four failure modes is qualitatively different between the intent-based control plane and the IaC-only baseline.

In Scenario S1, switching off a storage account encryption setting directly from the cloud portal was not detected by the IaC-only baseline until the next Terraform reconciliation operation which can take a few hours to a whole working day depending on the pipeline configuration. In contrast, the intent-based control plane detected the drift in a single objective-type evaluation operational cycle, and triggered an autonomous remediation within the safety boundary, creating a full pre- and post-state audit log entry against the governing intent. The resulting governance output is event-driven detection rather than periodic, and is an audit log generated as a structural product of the architecture, rather than a retrospective assembly of provider logs.

In Scenario S2, the geographic constraint violation on the Terraform deployment would not be detected by the IaC-only OPA gate due to scoping issues in the policy rule definition. In this case, the intent-based control plane would evaluate the deployment vs the active constraint intent, quarantine the non-compliant resource, and raise a structured remediation request to the governance approver without involving the CI/CD gate. The upshot is an architectural property: because intent evaluation is active and decoupled from the provisioning pipeline, it is a backstop governance mechanism that catches what provisioning-time gate does not.

In Scenario S3, a network isolation requirement enforced by the security team is in conflict with a cross-environment communication requirement enforced by the application team. Without policies that can express this conflict, the IaC-only baseline would simply use the higher priority policy, ignoring the other policy, with no record of the decision made in the governance system. The intent-based control plane also generated a summarized report of conflicts at each time slice and escalated it for a documented decision by the governance team. Resolving conflicts this way ensured architectural benefits beyond operational efficiency: governance accountability was documented, and assumptions misalignments between teams were made visible, which avoided adding undocumented technical debt to the architecture.

In Scenario S4, the telemetry gap of one region of the cloud for a short period of time during a drift event caused the IaC-only baseline to mis-manage the system for the majority of the time, while the intent-based control plane correctly interpreted the telemetry gap as an indeterminate evaluation state that should not be acted upon until telemetry returns and the actual infrastructure state is re-verified. Since no erroneous remediation occurred, this behavior can be explained by the Policy Evaluation Engine architecture, which considers each indeterminate evaluation to have a distinct status that needs to be deferred.

TABLE 4: Evaluation Scenario Results — Intent-Based vs. IaC-Only Baseline

Scenario	**IaC-Only Baseline Outcome**	**Intent-Based Control Plane Outcome**	**Governing Architectural Property**
S1 – Unauthorized Configuration Change	Deviation undetected until next IaC reconciliation cycle; no structured audit record	Detected within one evaluation cycle; autonomous remediation executed within safety boundary; complete audit record generated	Continuous event-driven observation; schema-enforced audit
S2 – Deployment Constraint Violation	Violation passed through CI/CD gate due to scope error in OPA policy rule	Deployment quarantined by runtime intent evaluation independent of provisioning pipeline;	Runtime enforcement independent of provisioning-time gates

		escalation routed to governance approver	
S3 – Conflicting Intents	Silent priority-based override; no governance record of conflict or resolution decision	Structured conflict report generated and escalated to governance team within one evaluation cycle	Conflict detection and structured escalation rather than autonomous resolution
S4 – Remediation Under Partial Observability	Remediation attempted against stale state; incorrect outcomes in majority of trials	Remediation deferred pending telemetry recovery; correct state verified before actuation; no incorrect remediations executed	Indeterminate evaluation treated as deferral, not action

7. Discussion

The scenario outcomes also illustrate changes in what enterprise cloud governance can do architecturally: all four scenarios show a shift from periodic, schedule-based enforcement of governance rules to event-driven continuous monitoring and enforcement. In the case of an IaC-only architecture, the enforcement gap that exists between reconciliation cycles is not a failure, rather a property of the architecture. A control that is compliant during the last deployment and will next be validated hours later has an enforcement gap. Thus, while every drift event that occurs inside this gap is a bona fide violation of policy, none of the control plane governance mechanisms are adequate to detect it. The intent-based control plane does not have this gap by design: evaluation is continuous; the observation layer consumes state changes immediately rather than at a predefined reconciliation interval.

The HashiCorp 2023 State of Cloud Strategy Survey, commissioned from Forrester Consulting and administered to nearly 1,000 technology professionals and business leaders, confirmed that governance maturity produces measurable business differentiation [1]. Among participants with high governance maturity, 80% reported a positive impact on compliance and risk and governance compared to 56% of participants with low governance maturity, a gap of 24 percentage points that the survey authors attributed to the adoption of operational best practices at scale rather than the volume of cloud usage alone [1]. Almost all (94%) respondents have some avoidable cloud waste. 79% of firms with the highest cloud maturity rating report a better security posture as a result of their multi-cloud initiatives. Neither the issues around governance nor the gulf between governance intent and governance outcome are a problem solved by simply going to the cloud. They remain a structural problem for all cloud adopters to address. The intent-based architecture proposed in this paper seeks to address this structural gap.

Beyond mere metrics, the Scenario S3 conflict escalation results show that in regulated scenarios (e.g. PCI DSS, HIPAA, SOX, etc.) an organization has to be able to produce authoritative, auditable and defensible justification of all governance decisions, including conflicting policy decisions, in order to meet regulatory compliance requirements. Silent priority-based resolution, the default in IaC-only environments with competing policies, does not meet documentation or accountability requirements. However, explicit escalation with explicitly governed records meets both requirements as an output of the architecture rather than a best-effort operational practice.

The Scenario S4 result establishes a very strong property of practical relevance in enterprise production environments with control plane and data plane coordination: under this high level of uncertainty, the control plane chooses not to act. Waiting for more observations for remediation makes sense. More broadly, an incorrect action by the autonomous remediation, using stale state, may be a more serious violation of compliance. This property, considering the indeterminate evaluation as a deferral condition rather than a default-action condition, is a good architectural fit for an environment where autonomous remediation has material operational and regulatory risk.

Integrating with FinOps telemetry is a lower priority. For example, intents could trigger recommendations to right-size resources with spend above predicted resource usage given its utilization targets. This would essentially extend governance beyond the five-layer control plane without having to architect an entirely new one. SRE error

budget signals could be exploited to realize reliability-aware intents whereby protective governance postures are invoked whenever service level objective consumption rates threaten reliability. Both extensions require new intent types and additional information sources in the state observation system.

8. Conclusion

This paper has proposed a policy-driven, intent-based architecture for cloud infrastructures that overcomes the structural limitation of IaC that compliance with an overtly stated governance intent cannot be continuously verified and enforced after provisioning. We formalize cloud infrastructure intent as a typed taxonomy of objectives, constraints, and preferences, persistently evaluate intent in a five-layer closed-loop control plane, and govern autonomous remediation within well-defined safety margins. This architecture yields qualitatively different governance results with respect to an IaC-only alternative. Compared with a mature IaC-only baseline of similar control scope and scenarios, there is improved drift detection latency, policy violation containment, autonomous remediation success and audit record coverage, which reduces regulatory risk, improves compliance posture, and reduces governance operational burden on enterprise cloud teams.

The architecture in the post is designed to work with existing enterprise cloud tooling such as OPA for policy evaluation, Terraform for infrastructure-as-code provisioning, AWS Config and Azure Policy for provider native compliance data, Azure Resource Graph and AWS Config for infrastructure state observability, and existing enterprise ITSM tools for remediation workflows. The intent registry, evaluation orchestration, and audit approach in Section 5 would require net-new engineering work.

The intent taxonomy, control plane architecture, and evaluation framework we define in the paper provide the first research and practice for cloud infrastructure governance as a homogeneous and continuous discipline. As cloud infrastructures grow in complexity and regulatory complexity, and as the surface area that periodic manual governance can cover grows, intent-based cloud governance goes from a hypothetical future to an attainable objective. That this architecture, as represented in this paper, is grounded in enterprise operational reality, is validated against plausible failure scenarios, and is actionable by practitioners.

REFERENCES

1. HashiCorp, "2023 State of Cloud Strategy Survey," HashiCorp Inc., San Francisco, CA, USA, Technical Report, 2023. [Online]. Available: https://www.hashicorp.com/state-of-the-cloud
2. Grand View Research, "Infrastructure as Code Market Size, Share and Trends Analysis Report," Market Research Report, 2023. [Online]. Available: https://www.grandviewresearch.com/industry-analysis/infrastructure-as-code-market-report
3. A. Leivadeas and M. Falkner, "A survey on intent-based networking," IEEE Communications Surveys and Tutorials, vol. 25, no. 1, pp. 625–655, First Quarter 2023. [Online]. Available: https://ieeexplore.ieee.org/document/9925251/
4. A. Clemm, L. Ciavaglia, L. Z. Granville, and J. Tantsura, "Intent-Based Networking — Concepts and Definitions," Internet Research Task Force, RFC 9315, Oct. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9315
5. I. Aviv, R. Gafni, S. Sherman, B. Aviv, A. Sterkin, and E. Bega, "Infrastructure from code: The next generation of cloud lifecycle automation," IEEE Software, vol. 40, no. 1, pp. 42–49, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/9908145/
6. K. Alexander, C. Lee, and S. Chai, "Declarative policy support for cloud application orchestration," in Proc. 19th International Conference on Advanced Communications Technology (ICACT), 2017, pp. 102–104. [Online]. Available: https://ieeexplore.ieee.org/document/7890066/
7. Open Policy Agent, "OPA documentation: Policy-based control for cloud-native environments," Cloud Native Computing Foundation, 2024. [Online]. Available: https://www.openpolicyagent.org/docs/latest/
8. M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. Gude, N. McKeown, and S. Shenker, "Rethinking enterprise network control," IEEE/ACM Transactions on Networking, vol. 17, no. 4, pp. 1270–1283, Aug. 2009. [Online]. Available: https://ieeexplore.ieee.org/document/5169973
9. Milan Chauhan and Stavros Shiaeles, "An Analysis of Cloud Security Frameworks, Problems and Proposed Solutions," MDPI, 2023. [Online]. Available: https://www.mdpi.com/2673-8732/3/3/18
10. A. Mekrache, A. Ksentini, and C. Verikoukis, "Intent-based management of next-generation networks: an LLM-centric approach," IEEE Network, vol. 38, no. 5, pp. 29–36, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10574890/
11. W. Hummer, F. Rosenberg, F. Oliveira, and T. Eilam, "Testing idempotency for infrastructure as code," ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing.

Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. [Online]. Available: https://inria.hal.science/hal-01480784/file/978-3-642-45065-5_19_Chapter.pdf
12. Varun Gowtham et al., "Intent-based networking for QoS-aware cloud and transport network management based on graph neural networks," IEEE Future Networks World Forum [FNWF], 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10520590/
13. K. Kritikos, B. Pernici, P. Plebani, C. Castillo, M. Comuzzi, S. Firmenich, G. Frankova, A. Matera, F. Salvia, and M. Varela, "A survey on service quality description," ACM Computing Surveys, vol. 46, no. 1, pp. 1:1–1:58, 2013. [Online]. Available: https://doi.org/10.1145/2522968.2522969
14. Orazio Tomarchio et al., "TORCH: a TOSCA-Based Orchestrator of Multi-Cloud Containerised Applications," Journal of Grid Computing, 2021. [Online]. Available: https://link.springer.com/content/pdf/10.1007/s10723-021-09549-z.pdf
15. A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, "Survey on network virtualization hypervisors for software defined networking," IEEE Communications Surveys and Tutorials, vol. 18, no. 1, pp. 655–685, First Quarter 2016. [Online]. Available: https://ieeexplore.ieee.org/document/7295561/
16. L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: Towards a cloud definition," ACM SIGCOMM Computer Communication Review, vol. 39, no. 1, pp. 50–55, Jan. 2009. [Online]. Available: https://doi.org/10.1145/1496091.1496100
17. HashiCorp, "Terraform documentation: Infrastructure as code for multi-cloud deployments," HashiCorp Inc., 2024. [Online]. Available: https://developer.hashicorp.com/terraform/docs