International Journal of Science and Research Archive

eISSN: 2582-8185

Cross Ref DOI: 10.30574/ijsra

Journal homepage: https://ijsra.net/

(Research Article)

# Automating Trust at Scale: Infrastructure-as-Code for Secure and Compliant AI Environments in the U.S.

Ifeoma Eleweke *

*Department of College of Technology and Engineering, Westcliff University, USA.*

International Journal of Science and Research Archive, 2025, 17(02), 123-137

Publication history: Received on 20 September 2025; revised on 02 November 2025; accepted on 04 November 2025

Article DOI: https://doi.org/10.30574/ijsra.2025.17.2.2962

## Abstract

As artificial intelligence (AI) systems increasingly underpin critical operations across U.S. industries, the ability to automate trust at scale has become both a technical necessity and a national imperative. This paper examines how Infrastructure-as-Code (IaC) can be strategically leveraged to secure AI ecosystems and enforce regulatory compliance across development-to-production processes. The central challenge lies in maintaining continuous compliance within dynamic, rapidly evolving AI environments, where manual configuration is both error-prone and unsustainable. IaC offers a foundational solution by embedding security and policy enforcement directly into system provisioning, transforming infrastructure deployment into an auditable, repeatable, and verifiable process. Empirical evidence consistently shows that human misconfigurations account for most cloud breaches, emphasizing the need for codified automation to minimize risk and ensure integrity. This study demonstrates that IaC enables consistent, immutable environments across AI development, training, and production stages, significantly reducing configuration drift and vulnerability exposure. Moreover, Policy-as-Code frameworks such as Terraform Sentinel and Open Policy Agent allow regulatory standards, including NIST SP 800-53, FedRAMP, and HIPAA, to be expressed as machine-readable rules enforced in real time. Integrating IaC security tools such as tfsec and Checkov, alongside robust secrets management within CI/CD pipelines, yields measurable improvements in compliance auditing and breach prevention. Through a proposed reference framework and real-world case studies spanning U.S. federal agencies and healthcare systems, this article illustrates how IaC can function as a scalable trust mechanism capable of unifying security, compliance, and automation in AI DevOps. Ultimately, embedding IaC into AI infrastructure is not merely a technical optimization; it is a strategic imperative for national cybersecurity resilience and policy assurance in the era of intelligent systems.

**Keywords:** Infrastructure-as-Code (IaC); AI Security; Compliance Automation; DevOps; Policy-as-Code; Drift Detection; Continuous Integration and Continuous Delivery/Deployment (CI/CD)

## 1. Introduction

*Infrastructure-as-Code* (IaC) represents a transformative DevOps paradigm that manages and provisions computing infrastructure through machine-readable definition files, replacing manual configuration and ad-hoc scripting with declarative automation. In practice, IaC tools such as Terraform, AWS CloudFormation, and Ansible allow engineers to define entire environments, including servers, networks, and storage, as code. This approach enables reproducibility, version control, and consistency across environments. By eliminating "environment drift," the subtle yet damaging divergence of configurations between development, testing, and production, IaC has revolutionized cloud operations. This consistency is particularly critical for the development of artificial intelligence (AI) systems, where experimental environments often transition rapidly into production environments.

* Corresponding author: Ifeoma Eleweke

### 1.1. DevOps and AI Environments

Modern AI workflows, such as training deep learning models, orchestrating data pipelines, or deploying inference services, require rapid provisioning of complex infrastructure, including GPU clusters, distributed storage systems, and secure model-serving endpoints. Traditional IT provisioning cycles, which once took weeks or even months, cannot keep pace with the velocity demanded by AI innovation. IaC bridges this gap by enabling AI DevOps teams to script and deploy entire AI environments from GPU-optimized cloud instances to Kubernetes clusters within minutes.

For instance, the U.S. Department of Defense reports that adopting standardized IaC baselines can shorten infrastructure setup timelines by as much as seven months, compressing environment stand-up from months to mere hours. This acceleration is vital for AI projects characterized by iterative experimentation and continuous deployment. Yet, velocity without security introduces significant risk. Maintaining security and compliance within these rapidly provisioned environments is an ongoing challenge. Each new AI model, dataset, or service instance must adhere to organizational and regulatory policies for privacy, data protection, and system integrity. Failure to do so can result in data exposure, compliance violations, or systemic compromise.

### 1.2. Security Risks and the Need to Automate Trust

Extensive research confirms that cloud security failures overwhelmingly stem from misconfiguration and user error. Gartner predicts that through 2025, 99 percent of cloud security incidents will originate from customer-side misconfigurations. In AI environments, such errors may manifest as publicly accessible data buckets containing training datasets, misconfigured compute clusters lacking network segmentation, or credentials inadvertently embedded in pipeline scripts. These weaknesses can lead to unauthorized data access, model theft, or infrastructure hijacking for activities such as cryptomining.

Recent high-profile incidents underscore the severity of these risks. Tesla's cloud environment, for example, was compromised via an exposed Kubernetes console, a misconfigured administrative interface that allowed attackers to execute cryptomining operations on Tesla's AWS resources. Similarly, a misconfiguration in Google Cloud led to the inadvertent deletion of an Australian pension fund's entire cloud environment, resulting in a prolonged outage. These examples illustrate that even well-resourced organizations remain vulnerable to configuration errors, which, when multiplied across hundreds of resources and multi-cloud deployments, exceed the limits of manual oversight.

The path forward lies in *automating trust*: embedding security and compliance enforcement directly into the infrastructure deployment lifecycle. IaC provides the foundation for such automation. Security controls can be codified as defaults, ensuring encryption of all storage resources, prohibiting public exposure of sensitive assets, and verifying compliance policies before any deployment is approved. Through this codified approach, IaC transforms infrastructure provisioning from a manual, error-prone task into a secure, policy-driven process.

This paper investigates how Infrastructure-as-Code can serve as a foundation for secure, compliant, and resilient AI ecosystems in the United States. It explores the security and operational benefits of embedding IaC into AI DevOps pipelines, analyzes automation risks and compliance challenges, and examines the integration of Policy-as-Code frameworks with U.S. regulatory standards such as NIST SP 800-53 and FedRAMP. Through real-world implementations across federal and healthcare sectors, the study demonstrates that IaC is not merely a tool for automating infrastructure but a mechanism for automating trust, ensuring that as AI systems scale, their security and compliance scale with them.

## 2. The role of IAC in AI ecosystems

Provisioning and managing infrastructure for artificial intelligence (AI) workloads is inherently complex. A single AI ecosystem may encompass data ingestion pipelines, model training on GPU clusters, CI/CD systems for machine learning (MLOps), container orchestration for model deployment, and observability dashboards often spanning hybrid or multi-cloud environments. *Infrastructure-as-Code* (IaC) serves as a unifying foundation for these components, enabling consistent, repeatable, and policy-governed deployments across every stage of the AI lifecycle.

### 2.1. Consistency from Development to Production

IaC ensures that environments used by data scientists during development are faithfully reproduced in staging and production, eliminating configuration drift and environment mismatches. Using the same Terraform templates, Helm charts, or Kubernetes manifests, teams can deploy AI workloads at varying scales from experimental sandboxes to hardened production clusters without reconfiguration. This uniformity resolves the pervasive "it works on my machine" problem by ensuring infrastructure parity across development, testing, and deployment stages.

By defining environments declaratively in code, IaC removes the discrepancies that manual setup processes inevitably introduce. For AI workflows, this consistency is particularly valuable: an experiment validated in a development setting will behave predictably when scaled for production, because the underlying compute, network, and security parameters remain identical. For instance, if a training job in development uses encrypted storage and tightly scoped firewall rules defined in code, those same controls persist in production. This reproducibility not only strengthens confidence in experimental results but also simplifies compliance validation, since certified configurations can be cloned without risk of drift.

### 2.2. Provisioning Specialized AI Resources

AI systems often depend on specialized hardware and high-performance configurations, GPU or TPU nodes, high-memory instances, or distributed computing frameworks such as Apache Spark or Kubernetes-based ML platforms. Provisioning these resources manually is both time-intensive and prone to error. IaC abstracts this complexity by allowing teams to codify their infrastructure requirements into reusable templates. For example, an IaC module can define a cluster of *p4d.24xlarge* AWS GPU instances with specific AMI images and attach them to auto-scaling groups for model training.

Parameterization allows for flexibility, such as dynamically adjusting node counts or GPU types while maintaining control and traceability. As a result, teams can provision and tear down AI compute environments on demand, supporting rapid experimentation without sacrificing governance. This approach also enhances cost efficiency and security by ensuring that environments exist only when needed and always conform to approved configurations. Organizations such as Netflix and AWS have demonstrated this pattern through IaC-driven orchestration platforms like Mantis and SageMaker, which dynamically instantiate ephemeral training clusters with consistent security baselines. By automating resource allocation and teardown, IaC prevents the emergence of untracked, ad-hoc instances often referred to as “shadow AI” infrastructure that can introduce vulnerabilities and compliance gaps.

### 2.3. MLOps and Pipeline Management

The modern MLOps lifecycle, from data preparation and model training to deployment and continuous monitoring, benefits substantially from Infrastructure-as-Code at every phase. During data ingestion, IaC can define infrastructure for secure data lakes or streaming platforms, ensuring that encryption, access control, and logging policies are consistently enforced. In the training phase, IaC definitions extend beyond compute provisioning to include configuration management, for instance, using Ansible playbooks to install identical library dependencies across all training nodes.

IaC also simplifies the deployment of Kubernetes-based ML orchestration platforms such as Kubeflow, guaranteeing that every component (notebooks, pipelines, serving layers) adheres to security best practices, including network segmentation, role-based access control, and resource quotas. During the deployment stage, containerization and orchestration templates such as Helm charts or Kustomize manifests enable reproducible, policy-compliant rollout of model-serving services with integrated autoscaling and monitoring. By managing the entire ML pipeline as version-controlled code, IaC bridges the operational gap between data science and DevOps. This unified approach, often referred to as *Pipeline-as-Code* or *GitOps for ML,* promotes auditability, accelerates release cycles, and strengthens the reliability of AI systems. Empirical studies show that IaC adoption in MLOps improves reproducibility, reduces environment errors, and enhances the scalability of secure ML deployments.

A real-world example illustrates these benefits: a large financial institution implementing an AI-powered fraud detection platform adopted IaC to standardize its Kubernetes clusters and CI/CD pipelines across development, testing, and production. This approach ensured infrastructure parity between model training and inference environments, enabling compliance officers to review Terraform and Helm definitions directly for adherence to security baselines, such as the use of approved machine images and isolated virtual networks. Without IaC, achieving such verification would have required laborious manual audits of individual cloud configurations.

In summary, Infrastructure-as-Code forms the operational backbone of AI ecosystems by ensuring environment consistency, enabling rapid provisioning of specialized compute resources, and embedding governance into every stage of the machine learning lifecycle. It transforms infrastructure from a static dependency into a flexible, reusable, and auditable asset, one that can be created, tested, and decommissioned with the same rigor as application code. Beyond accelerating innovation, IaC establishes the technical foundation for the security and compliance frameworks discussed in the subsequent sections. The very attributes that make IaC indispensable for agility, consistency, transparency, and automation also make it a cornerstone for building trusted AI environments at scale.

## 3. Risks of automation at scale

While *Infrastructure-as-Code* (IaC) substantially enhances consistency, efficiency, and security, it is not without its own challenges. Automation at scale introduces new classes of risks that, if unmanaged, can undermine the very trust it seeks to automate. Misconfigurations can propagate instantly, security controls can be bypassed unintentionally, and organizational oversight can lag automated deployment velocity. Understanding these risks is essential to adopting IaC prudently as an instrument of trust rather than a source of fragility.

### 3.1. Misconfigurations Propagated at Scale

The defining strength of IaC is the ability to deploy changes quickly and uniformly across vast environments, which can also be its greatest weakness. A single misconfigured parameter in code can be replicated to hundreds of resources within minutes. This "blast radius" effect means that a minor oversight becomes a systemic failure once executed at scale.

A notable example occurred in 2024, when a Google Cloud automation process erroneously deleted an entire customer environment belonging to the Australian pension fund, UniSuper. An internal project deletion script was misconfigured and, when executed, removed accounts across multiple regions. Google described the incident as a "one-of-a-kind occurrence," yet it underscores how small automation errors can trigger catastrophic consequences in hyperscale systems. Such incidents highlight the necessity of treating IaC code with the same rigor as application software subject to unit testing, peer review, and staged deployment. Comprehensive pre-deployment testing, coupled with policy-as-code validation and canary rollouts, can mitigate this risk. Equally important are rollback mechanisms: while IaC can propagate misconfigurations rapidly, it also enables rapid recovery if change detection and monitoring are properly integrated.

### 3.2. Security Incidents from IaC and Template Errors

Several prominent security breaches illustrate how IaC misconfigurations or flawed templates can expose AI and cloud systems to compromise. Tesla's 2018 cloud breach stemmed from an exposed Kubernetes administrative console, an access point left unauthenticated during automated setup. Attackers exploited this to deploy cryptomining workloads on Tesla's AWS infrastructure. Similarly, the 2019 Capital One breach, one of the largest financial data incidents in U.S. history, exploited a misconfigured web application firewall (WAF) and over-privileged IAM roles, granting access to sensitive data. In both cases, automation accelerated deployment but failed to ensure secure defaults. The lesson is clear: automation amplifies both strengths and weaknesses. If templates or modules contain insecure parameters, IaC will replicate them flawlessly. The solution lies in secure-by-default coding standards, enforced reviews, and automated IaC security scanning. As organizations transition from manual configuration to coded infrastructure, the human role evolves from "configuring systems" to "coding and validating secure infrastructure definitions." Secure coding practices must therefore extend to IaC just as they do to application software.

Table 1 summarizes representative misconfiguration incidents and the IaC controls that would have prevented or contained them.

**Table 1** Notable Cloud Misconfiguration Incidents and IaC Mitigations

| Incident & Year | Misconfiguration Cause | Impact | How IaC Could Mitigate |
|---|---|---|---|
| Tesla Kubernetes breach (2018) | Exposed K8s admin console with no password | Attackers gained access, ran cryptominers on Tesla's cloud. Data and compute resources were abused. | IaC with enforced policies would ensure any K8s dashboard requires authentication. A policy-as-code check could block deployment of unauthenticated admin interfaces. |
| Capital One cloud breach (2019) | Overly permissive AWS WAF & IAM role misconfig | Attacker exploited SSRF to access 100M customer records from S3. Massive data breach and fines ensued. | IaC could encode least-privilege IAM policies and vetted firewall rules. Automated checks (tfsec/Checkov) would flag open access policies, preventing deployment of weak configurations. |

| | | | |
|---|---|---|---|
| Toyota data exposure (2015–2023) | Cloud storage misconfigured (public accessibility) for an extended period | Personal vehicle data of ~260,000 customers was exposed for 8 years before discovery. | Using IaC, cloud storage configs would be centrally managed; periodic compliance scans would catch any storage not marked private. Drift detection would alert if a bucket's policy changed from the code baseline. |
| Google Cloud/UniSuper outage (2024) | Misconfiguration in project deletion automation (one project's deletion cascaded across geos) | Entire cloud environment of a pension fund was deleted, causing a week-long outage for customers. | IaC with proper guardrails (e.g., requiring confirmation for destructive actions) might prevent accidental deletion. More importantly, IaC facilitates reliable recovery – with environment codified, it could be redeployed from code to speed up restoration after such an incident. |

These case studies collectively demonstrate that IaC not only prevents misconfigurations but also limits the blast radius when failures occur.

### 3.3. Multi-Cloud and Hybrid Complexity

Enterprises increasingly operate in multi-cloud or hybrid environments, combining services from AWS, Azure, Google Cloud, and on-premises systems. Although IaC frameworks support cross-provider provisioning, each cloud service maintains unique architectures, policies, and security models. As a result, a configuration deemed secure in one provider may be incomplete or insecure in another. For example, a Terraform module that enforces strict tag-based access controls in AWS might not have a direct equivalent in Azure or GCP. Such disparities can lead to policy gaps, where certain assets remain unprotected simply because the IaC abstraction fails to enforce equivalent rules across providers. Studies have shown that configuration errors frequently occur during cloud migrations or multi-cloud rollouts, when teams are least familiar with provider-specific nuances.

This fragmentation also complicates compliance. A policy rule enforced by AWS Config or Azure Policy does not extend to resources in another cloud, creating gaps "between the cracks." For instance, an organization might successfully restrict public access to AWS S3 buckets while inadvertently leaving Azure Blob Storage open. To counter this, some enterprises employ unified policy layers using Open Policy Agent or adopt provider-specific guardrails for each environment. Ultimately, the complexity of multi-cloud IaC demands a blend of automation, domain expertise, and vigilant monitoring to prevent erosion of trust across heterogeneous infrastructure.

### 3.4. Over-automation and Change Control

Automation delivers speed, but unchecked speed can undermine governance. In highly regulated sectors such as healthcare, finance, and defense, every infrastructure modification must align with compliance baselines. When IaC pipelines automatically apply all committed changes without review, a single developer's update could inadvertently open a production network path or alter a critical IAM role.

As Michelle Buckner (2025) observed, "blindly trusting automation is a whisker away from a compliance nightmare." Over-automation without accountability creates a false sense of security changes occur too rapidly for human validation. Mature DevSecOps programs counter this risk with structured approval gates, mandatory peer reviews, and staged promotion of infrastructure changes. Phased deployments, or "canary environments," allow teams to observe the effect of updates before global rollout. Striking the right balance between velocity and governance is essential. While IaC enables continuous delivery of infrastructure, organizations must retain human oversight for high-impact security and compliance decisions. Automation should accelerate compliance, not bypass it.

### 3.5. Security Tooling Gaps and False Sense of Security

A final risk lies in assuming that automation equates to security. Merely storing infrastructure definitions in Git repositories does not make them inherently safe. Without continuous IaC security scanning, template analysis, and runtime validation, organizations may unknowingly codify vulnerabilities. For example, a Terraform configuration might explicitly declare a publicly accessible storage bucket or permit weak encryption settings. Unless detected during review or by automated scanning, such misconfigurations are faithfully deployed at scale. According to Google Cloud's 2024 threat report, misconfiguration accounted for 30 percent of observed cloud breaches in the first half of that year, up from 17 percent the previous period, demonstrating that automation alone has not eliminated configuration risk.

To close this gap, IaC pipelines should integrate static analysis and compliance validation tools such as Checkov, tfsec, and cloud-native configuration monitors. These systems detect insecure parameters, enforce compliance baselines, and prevent deployments that violate policy. Automation must therefore be paired with visibility and control mechanisms to ensure that "as code" truly means "as secure."

Table 2 summarizes key cloud security incidents and trends that illustrate how automation, when misapplied, can amplify risk rather than reduce it.

**Table 2** Cloud Security Trends and Incidents Related to IaC Misconfigurations

| Trend/Statistic (2022–2024) | Insight for IaC Security | Source (Year) |
|---|---|---|
| Cloud environment intrusions increased 75% from 2022 to 2023. | Threats targeting cloud deployments are rising sharply, meaning any IaC misconfiguration is more likely to face attack. Emphasizes need for vigilance in IaC setups. | SentinelOne (2025) |
| 30% of cloud attacks in H1 2024 stemmed from misconfigurations (up from 17%). | Misconfigurations are an increasing share of initial breach vectors. Automation must prioritize eliminating misconfig errors; otherwise, mistakes in IaC become prime targets. | Google Cloud (2024) |
| 39% of businesses had a cloud data breach in 2023 (up from 35% in 2022). | Breaches are becoming more common as cloud use grows. Even slight percentage increases translate to many new incidents. Underlines that many organizations still struggle with cloud security basics – something IaC can improve, but only if used correctly. | Thales Cloud Security Study (2023) |
| Gartner: 99% of cloud security failures through 2025 will be customer-induced (misconfig/etc.). | Nearly all cloud incidents are due to user/config error, not cloud provider faults. This famous prediction reinforces that automating guardrails (to avoid human error) is crucial. IaC is part of the solution, but improper IaC use could also contribute if not managed. | Gartner via InformationWeek (2024) |
| Only 45% of sensitive data in cloud is encrypted, even though 75% of businesses put >40% of sensitive data in cloud. | A gap exists between data sensitivity and protection. IaC can help by ensuring encryption is always enabled by default on resources. Failure to do so (through oversight in code) leaves data exposed. Highlights need for encryption-as-code policies in IaC. | Thales (2023) |

In summary, the risks of automation at scale arise not from IaC itself but from insufficient governance, testing, and contextual awareness. The same mechanisms that enable consistency and agility can, without proper safeguards, propagate systemic vulnerabilities. Organizations must therefore treat IaC as both a technical and governance discipline subject to the same rigor as software engineering, compliance validation, and operational oversight. The next section explores how these challenges can be addressed through *Policy-as-Code* frameworks and compliance automation, demonstrating how IaC can evolve from a potential risk amplifier into a scalable mechanism for enforcing trust, compliance, and security across AI-driven environments.

## 4. Trust through automated compliance

Achieving and proving compliance at scale remains one of the most difficult challenges in managing AI and cloud systems. Traditional approaches rely on manual reviews and documentation, which are slow, error-prone, and poorly suited to the velocity of modern AI delivery. *Infrastructure-as-Code* (IaC) enables a shift toward *Compliance-as-Code*, in which policies and controls are expressed in machine-readable form and enforced automatically throughout the lifecycle. By integrating policy evaluation into planning, deployment, and runtime monitoring, organizations can replace episodic audits with continuous assurance. This section examines the principles and tooling that make automated compliance practical focusing on Policy-as-Code engines such as Terraform Sentinel and Open Policy Agent (OPA), and cloud-native rule frameworks including AWS Config, Azure Policy, and Google Cloud Config Validator and describes how regulatory requirements in the United States, including NIST SP 800-53, FedRAMP, and the NIST Risk Management Framework, can be mapped systematically into enforceable controls.

### 4.1. Policy as Code – The Concept

*Policy-as-Code* encodes governance requirements, such as mandatory encryption, restricted network exposure, approved images, or tagging for accountability, into executable rules that are evaluated by machines rather than being interpreted solely from prose. In an IaC pipeline, these policies are evaluated against the proposed infrastructure state before any change is applied. Sentinel, integrated with Terraform Cloud/Enterprise, can evaluate plan outputs and block apply when rules are violated; OPA, using the Rego language, performs comparable evaluations across Terraform, Kubernetes, and other artifacts. In Kubernetes environments, OPA Gatekeeper enforces admission controls to ensure, for example, that only approved base images are deployed, namespaces carry required labels, and network policies exist for every workload. The essential benefit is preventative control: non-compliant infrastructure is never created, and exceptions are explicit, reviewable, and auditable.

### 4.2. Tools for Automated Compliance

Automated compliance is most effective when controls are layered across the delivery lifecycle. Preventive policies evaluate proposed changes before they are merged, deployment-time checks validate the final plan immediately before resources are created, and runtime monitors continuously observe the live environment to detect and remediate drift. This tiered model establishes a clear division of responsibility: design-time gates prevent nonconformant configurations from entering the system, apply-time validation ensures that the materialized state matches approved intent, and production safeguards sustain posture in the face of out-of-band changes or evolving threats. Any residual nonconformance detected after deployment is either auto-remediated or triggers immediate rollback to the last known-good state, with the pipeline and posture monitors recording the full evidence trail.

Within the preventive and apply-time tiers, policy engines tied directly to infrastructure plans provide precise control. Terraform Sentinel, integrated with Terraform Cloud and Enterprise, evaluates the computed plan against organizational and regulatory rules and can hard-fail a deployment when conditions are not met. Because the policy interpreter has access to the exact attributes of planned resources, it can enforce requirements such as encryption with customer-managed keys, non-public administrative interfaces, provenance-verified images, mandatory ownership and data-classification tags, and guardrails around identity and network boundaries. Decisions are recorded alongside the plan as auditable artifacts, binding enforcement to the same source of truth that provisions the environment.

Cross-platform estates benefit from a general-purpose engine that applies consistent logic beyond a single IaC tool. Open Policy Agent (OPA) extends Policy-as-Code to heterogeneous environments through Rego policies evaluated in CI pipelines and at control points such as the Kubernetes admission path via Gatekeeper. The same policy library can validate Terraform for multiple clouds, gate workload deployment to require network policies and TLS, restrict image registries for model-serving, and limit egress from training clusters. Because OPA is decoupled from any one provider, it enables a portable compliance posture that travels with AI workloads across hybrid and multi-cloud platforms.

Runtime enforcement closes the loop by continuously evaluating live resources against intent. Cloud-native services such as AWS Config with Conformance Packs, Azure Policy, and Google Cloud's Config Validator evaluate deployed resources against mapped control families and trigger alerts or automated remediation when drift is detected. Prebuilt conformance packs aligned to frameworks such as FedRAMP Moderate provide an immediate baseline while organizations mature their own policies. Findings from these services, together with the pre-deployment policy decisions and CI logs, form a coherent evidence trail that links intent, enforcement, and outcomes. The result is a layered compliance architecture in which preventive controls stop nonconformant changes at source and detective controls sustain posture in production, even when resources are created or modified outside the IaC pipeline.

### 4.3. Mapping to Regulatory Frameworks

Automating trust requires a systematic translation from regulatory language to technical controls. NIST SP 800-53 defines families of security and privacy controls; FedRAMP tailors those controls for cloud authorizations and emphasizes continuous monitoring. With Compliance-as-Code, these abstractions become enforceable rules. Storage encryption requirements (e.g., SC-13) map to policies that deny any resource lacking encryption at rest; access enforcement (AC-3) becomes a library of rules constraining IAM roles and security groups to least-privilege patterns; configuration management (CM-2/CM-3) is operationalized by requiring that all changes flow through version-controlled IaC pipelines with peer review and by recording policy evaluations as immutable artifacts. Developer security testing (SA-11) can be evidenced, in part, by automated IaC scanning and policy checks executed on every change, with results retained for audit.

This mapping reduces audit burden and increases assurance. Rather than compiling screenshots of console settings, teams present the policy code, the evaluation outcomes tied to specific commits and plans, and the runtime posture reported by cloud compliance services. Because the same policies gate deployments and monitor production, assessors can see both design-time intent and run-time conformance, closing the loop from requirement to enforcement to evidence.

### 4.4. Case Study – FedRAMP using Terraform & Sentinel

A U.S. government SaaS provider seeking FedRAMP Moderate authorization for an AI analytics platform codified its GovCloud architecture in Terraform and implemented a curated set of Sentinel policies mapped to relevant control families. Configuration management controls were satisfied by requiring all infrastructure changes to originate from the CI/CD pipeline, with enforced peer review and recorded policy evaluations. Logging and audit controls mandated CloudTrail and service-specific logs across all accounts and regions, verified automatically in policy and re-checked post-deployment by AWS Config. Over time, the organization maintained a library of dozens of policies spanning encryption, key management, boundary protection, vulnerability scanning prerequisites, and approved AMIs. During the assessment, the provider provided the policy repository, evaluation reports per release, and continuous-monitoring dashboards as primary evidence, significantly reducing manual attestation and demonstrating that deployments were compliant by design.

### 4.5. Trust and Continuous ATO

Within the U.S. defense community, *Continuous Authority to Operate* (cATO) reframes accreditation as an ongoing risk decision supported by continuous monitoring and automated control. IaC and Policy-as-Code are foundational to this model. Every infrastructure change, such as scaling GPU clusters for training, introducing new model-serving endpoints, or adjusting data pipelines, proceeds through approved code paths, where static and policy checks verify compliance before deployment. Runtime services observe the live environment for drift, misconfiguration, and policy regressions, feeding posture data to dashboards used by Authorizing Officials and security operations. In effect, the accreditation boundary is guarded by code: guardrails at commit and build time prevent non-compliant states, while post-deployment sensors detect anomalies quickly enough to sustain continuous authorization. The result is higher delivery velocity without sacrificing accountability, because trust is established and renewed through repeatable, observable controls rather than episodic paperwork.

Table 3 summarizes representative Policy-as-Code and compliance-automation tools and indicates their typical insertion points across plan, apply, and runtime monitoring.

**Table 3** Policy-as-Code and Compliance Automation Tools

| **Tool/Framework** | **Provider/Type** | **Purpose in Automated Compliance** |
|---|---|---|
| HashiCorp Sentinel | HashiCorp (proprietary) policy engine for Terraform Enterprise/Cloud | Enforces organizational policies on Terraform IaC before deployment. E.g., blocks non-compliant Terraform plans (no public S3, required tagging, etc.). Useful for mapping enterprise controls (like SOC2, FedRAMP) into Terraform rules. |
| Open Policy Agent (OPA) (+ Conftest, Gatekeeper) | CNCF open-source policy engine | General policy-as-code for myriad contexts (Terraform, Kubernetes, CI pipelines). OPA's Rego policies define compliance rules (e.g., all k8s pods must have resource limits). Gatekeeper integrates OPA with Kubernetes to deny configs that violate policies in real-time. Promotes cross-platform policy consistency. |
| AWS Config Rules | AWS cloud-native service | Continuous monitoring of AWS resource configurations against rules. Offers pre-built rules for common compliance (PCI, HIPAA, FedRAMP) and custom rule support. Can trigger alerts or auto-remediation. Ensures that deployed resources remain in compliance (drift detection). Azure Policy and GCP Config Connector play similar roles in their clouds. |
| Terraform Compliance & tfsec | Open-source (terraform-compliance, tfsec by Aqua Security) | These tools statically test Terraform scripts for security/compliance issues. For example, tfsec will flag if an AWS RDS instance is not using storage encryption or if a security |

| | | |
|---|---|---|
| | | group is overly permissive. Helps catch compliance deviations early in the development cycle. |
| Cloud Custodian | Open source (Cloud Custodian by Capital One) | A rules engine for AWS/Azure/GCP that can act on cloud resources to ensure compliance. E.g., it can disable an instance if it's untagged or turn on encryption on a bucket. It's often used to implement "active compliance" – not just detect but correct issues in runtime. |
| Compliance as Code frameworks | Various (Chef InSpec, AWS Conformance Packs, etc.) | Chef InSpec allows writing compliance tests in code (human-readable) that can run against infrastructure. For example, an InSpec profile for CIS benchmarks can programmatically check server settings. AWS Conformance Packs (collections of Config rules) similarly bundle rules mapping to standards (like a pack for NIST 800-53 checks dozens of controls automatically). |

With these mechanisms in place, compliance becomes an attribute of the delivery system rather than an after-the-fact exercise. Developers gain confidence that successfully evaluated changes will not be rolled back later for policy reasons; security teams gain confidence that guardrails are enforced uniformly; auditors gain determinism because evidence is generated automatically from policy evaluations, deployment artifacts, and posture dashboards. In AI environments where data classifications evolve, models are retrained frequently, and serving topologies change frequently, this automation prevents inadvertent policy violations, such as deploying unapproved datasets, exposing administrative endpoints, or omitting required encryption and logging. As governance expectations for AI expand to include model transparency, lineage, and fairness testing, the same approach extends naturally: express the rules as code, evaluate them continuously, and retain the results as evidence.

Having established how Policy-as-Code operationalizes trust, the next section turns to practical controls for securing AI pipelines with IaC, detailing how to instrument CI/CD with scanning, secrets management, identity controls, and drift detection to make "compliant by construction" the default behavior of AI infrastructure.

## 5. Securing AI pipelines using IAC

AI pipelines — spanning data processing, model training, validation, and deployment – handle sensitive information and expose numerous integration points, from data sources and artifact registries to model repositories and serving platforms. Managing these systems as code allows security to be embedded at each stage and verified continuously. The following subsections detail concrete practices for securing AI pipelines through Infrastructure-as-Code, including static analysis of IaC, secrets and identity management, CI/CD hardening and policy checks, and post-deployment drift detection.

### 5.1. Static Analysis of IaC (Shift-Left Security)

Infrastructure definitions should be subjected to the same rigor as application code. Static analysis tools such as tfsec and Checkov evaluate Terraform, CloudFormation, Kubernetes manifests, and related IaC formats against rulesets that map to security best practices and common compliance controls. Typical findings include unencrypted storage resources, overly permissive security groups, public endpoints on administrative ports, or Kubernetes clusters exposed to the public internet. Integrating these scanners into pre-commit hooks and continuous integration ensures that insecure configurations fail fast, preventing misconfigurations from entering version control or reaching staging environments. For AI workloads where a single template can instantiate clusters, data lakes, message buses, and model-serving gateways, shift-left scanning contains risk early, raises signal-to-noise for reviewers, and treats security testing as a non-negotiable quality gate.

### 5.2. Secrets Management

Secrets must never be embedded in repositories, images, or plaintext artifacts. Secrets-as-Code practices rely on dedicated services such as HashiCorp Vault, AWS Secrets Manager, and Kubernetes Secrets or Sealed Secrets to retrieve credentials at deploy time and inject them only where needed. Ephemeral or dynamically issued credentials reduce exposure windows for short-lived training jobs and ad hoc analysis tasks; access events are logged centrally to support forensic review and compliance attestations. Equal attention is required for state and metadata. Terraform state may contain sensitive identifiers or tokens, so remote backends must be encrypted at rest with customer-managed keys, access should be tightly scoped by IAM policy, and state locking and versioning should be enabled. These controls ensure that a compromise of the state store cannot be leveraged to undermine the integrity of the environment.

### 5.3. Identity and Access Management (IAM) in Code

Codifying identity and access controls brings versioning, review, and repeatability to one of cloud security's most error-prone domains. Policies and roles defined as code should default to least privilege, explicitly scope resources and actions, and separate duties across environments and pipelines. Automated analysis using cloud-native policy analyzers and IaC scanners helps detect privilege escalation paths, wildcard permissions, and overly broad role assumptions before changes are merged. In AI systems, service identities for training, feature engineering, and inference should be tailored to their specific data paths: a training role may read labeled data and write trained artifacts while an inference role may read models and emit logs, but neither should have administrative privileges. Because IAM is expressed as code, rotations of keys and role credentials can be orchestrated predictably, and remediation of an exposure can be executed uniformly across accounts and regions.

### 5.4. CI/CD Integration and Pipeline Security

The pipeline that builds and deploys AI systems must itself be hardened and reproducible. Build runners should be provisioned with IaC, isolated on dedicated subnets, and restricted to least-privilege roles. Artifact repositories, model registries, and container registries require private access, immutability guarantees, and scanning for vulnerabilities and malware. Image and model provenance should be established through signing using, for example, Sigstore's Cosign, and enforced at admission by policy engines so that only verified artifacts can be deployed. A typical release path couples IaC scanning, container image scanning, secrets discovery on the codebase, and compliance tests against target environments; logs emitted by the pipeline must be free of sensitive values and retained as auditable evidence. Treating the pipeline as a first-class infrastructure workload ensures that every model promotion follows the same guarded path, eliminating one-off exceptions that produce inconsistent security outcomes.

### 5.5. Drift Detection and Continuous Monitoring

Once deployed, infrastructure must be kept aligned with its declarative baseline. Regular reconciliations via Terraform against live state or via GitOps controllers such as Argo CD and Flux detect unauthorized or out-of-band changes and revert them automatically or alert for review. Cloud-native configuration monitors (for example, AWS Config, Azure Policy, and Google Cloud configuration validators) provide continuous posture evaluation and can trigger remediation workflows when a resource diverges from policy. Immutable patterns further reduce risk by replacing in-place changes with fresh deployments, making state reconciliation deterministic. For AI platforms, where components are frequently scaled or reconfigured, these mechanisms prevent debugging shortcuts or emergency fixes from lingering as silent policy violations.

Post-deployment compliance checks function as first-class health gates. When a gate fails, such as a newly opened administrative port, a disabled audit log, or an unencrypted store, the release is automatically rolled back to the last attested configuration, the violating change is quarantined, and an incident is opened with links to the plan, policy decisions, and remediation actions. This closes the loop between drift detection and recovery so that noncompliant states are both short-lived and fully evidenced.

### 5.6. Pipeline Narrative

Consider an update to a healthcare analytics platform that introduces a new model-serving microservice and expands data processing. Engineers update Terraform modules to support resources and revise Kubernetes manifests for the service. Pre-commit scanners flag an unencrypted storage bucket for model outputs; the code is corrected and re-checked until all controls pass locally. A pull request triggers continuous integration, which runs IaC scanners on the plan, performs container image analysis, and executes secrets discovery across the repository. A policy bundle then evaluates the Terraform plan, verifying mandatory tags, approved instance types, encryption settings, and network boundaries. After approval, the pipeline applies the infrastructure changes and promotes the manifests through a GitOps controller. Runtime policy engines immediately evaluate the new resources and configurations, reporting posture in dashboards; any deviation, such as a manually opened port or a disabled log sink, is detected and either auto-remediated or escalated. The following day, a scheduled reconciliation confirms that the live environment matches the declared baseline. Throughout the process, each gate emits signed artifacts and logs, creating a precise, time-stamped trail of intent, enforcement, and outcome that satisfies operational and regulatory scrutiny.

### 5.7. IAM and Secrets rotation

Rotation of keys, certificates, and service credentials should be automated as part of the same delivery workflows. IaC pipelines can create and distribute new credentials on a fixed cadence, deprecate and retire old ones, and update all references atomically. Where possible, workloads should use short-lived, automatically refreshed credentials obtained via federated identity rather than long-lived static secrets. In AI contexts, this includes rotating access to external data sources, third-party APIs, and internal model registries, ensuring that leaked or stale credentials cannot be used to pivot into sensitive systems.

In summary, securing AI pipelines with IaC rests on a continuous control loop that prevents misconfigurations at design time, validates the final plan at the moment of change, and sustains posture in production through reconciliation and monitoring.

Table 4 summarizes representative practices and tooling patterns used to implement this loop:

**Table 4** Key Practices for Securing IaC Pipelines in AI Systems

| Security Practice | Tools/Techniques | Benefit for AI Pipeline Security |
|---|---|---|
| *Static IaC Security Scanning* | tfsec, Checkov, Terrascan, KICS | Catches misconfigs (open ports, no encryption, etc.) in code before deployment. Prevents insecure cloud resources for data/model. |
| *Policy as Code Enforcement* | Sentinel, OPA (Conftest), Azure Policy in CI | Blocks non-compliant changes (e.g., disallowed instance types, missing tags) from being applied. Ensures deployments meet governance standards. |
| *Secrets Management & Injection* | HashiCorp Vault, AWS Secrets Manager, Ansible Vault | Avoids hardcoded creds. Provides secure retrieval of API keys/DB passwords at runtime. Meets compliance for secret handling (keys are encrypted, access is logged). |
| *IAM Least Privilege via Code* | IAM Access Analyzer, automated policy linting (Parliament), modules for least-privilege roles | Ensures service roles and users in AI pipeline have minimum required permissions. Reduces blast radius if a component is compromised (it cannot access data it's not supposed to). |
| *CI/CD Pipeline Hardening* | Container image scanning (Trivy/Anchore), signed artifacts (Cosign), isolated build agents | Prevents vulnerabilities in ML service images from reaching prod. Signed models/images ensure integrity (only trusted artifacts run). Secure CI agents prevent pipeline as an attack path. |
| *Drift Detection & Auto-Remediation* | Terraform Plan in cron, Driftctl, AWS Config Rules with auto-remediation (Lambda) | Detects any divergence from expected state (e.g., someone opened a security group). Optionally auto-fixes issues (e.g., Config auto-encrypts an unencrypted volume). Maintains continuous compliance posture. |

By making guardrails intrinsic to the delivery system, organizations move from ad-hoc inspection to predictable enforcement, maintaining both velocity and verifiable compliance as AI systems evolve.

## 6. Case studies and industrial applications

To ground the concepts discussed so far, this section examines real-world implementations from U.S. federal agencies and regulated enterprises that adopted IaC-driven security and compliance for AI and cloud environments. The examples that follow, spanning a FedRAMP-focused federal deployment, a HIPAA-aligned healthcare MLOps platform, an AI governance pipeline in financial services, and a Department of Defense DevSecOps baseline, illustrate how the theoretical benefits of Infrastructure-as-Code translate into practice, where the friction points arise, and how organizations ultimately reconcile speed with assurance.

### 6.1. Case Study 1: FedRAMP Compliance via Terraform and Sentinel (Federal Cloud)

A federal cloud team tasked with launching an AI analytics platform on AWS GovCloud needed FedRAMP Moderate authorization on an aggressive schedule. Rather than documenting hundreds of controls through manual screenshots and checklists, the team codified its environment with Terraform and enforced guardrails with Sentinel. Each relevant FedRAMP control was mapped to a technical safeguard expressed in code: identity controls mandated multi-factor authentication on all IAM users; audit requirements ensured CloudTrail and access logs were enabled by default; boundary protections limited network exposure through predefined module patterns. Sentinel policies prevented the creation of noncompliant resources, for example, disallowing S3 buckets without server-side encryption or rejecting permissive security group rules. During testing, the policies routinely blocked drift-inducing changes, such as the attempt to deploy a non-hardened AMI, and the policy library expanded to cover the bulk of technical FedRAMP expectations.

When the third-party assessor arrived, the agency demonstrated not only intent but enforcement and evidence. Sentinel policy definitions were explicitly mapped to NIST SP 800-53 Rev. 5 controls as tailored by the FedRAMP Moderate baseline, for example, IA-2 enforcing multi-factor authentication, AU controls ensuring CloudTrail and access logging, AC-4 constraining network boundaries, and SC-28 requiring server-side encryption. Evaluation results were tied to specific plan and apply events, and runtime posture was corroborated through provider compliance services, allowing the assessor to trace each technical safeguard to its corresponding control requirement, culminating in an Authority to Operate with fewer remediations than typical. The up-front effort to build modules and policies paid dividends in day-to-day operations: the team could push frequent updates to the AI platform while relying on policy-as-code to keep every change within FedRAMP boundaries. The codified controls also doubled as training material, helping new engineers understand security expectations by reading and working with the same code that governed production.

### 6.2. Case Study 2: HIPAA-Compliant MLOps Platform (Healthcare)

A healthcare analytics company needed to build an internal platform to analyze patient data under HIPAA while maintaining the agility expected by research teams. The solution centered on Kubernetes and Kubeflow on AWS, provisioned and managed entirely as code. Terraform established private EKS clusters, enabled KMS-backed encryption for storage and secrets, and instantiated baseline networking that forbade public endpoints. Helm charts and Gatekeeper policies require labels, namespace segregation, and hardened defaults for any workload that touches protected health information. Secrets management relied on Vault, with Terraform wiring policies and integrations so that credentials were retrieved at runtime rather than stored in repositories or images. Istio enforced mutual TLS and network segmentation, and Checkov scanned Terraform and Kubernetes definitions on each commit to catch regressions before they reached production.

The platform reached production in six months and passed internal and external HIPAA reviews, with IaC cited as a primary reason security and compliance could scale without slowing delivery. Provisioning a new, secure analytics environment for a dataset dropped from weeks to under a day because controls were embedded in reusable modules. During the first year, there were no significant security incidents. Minor misconfigurations surfaced early in CI, for example, a developer’s convenience setting for an SSH port on a test notebook, which were flagged by policy and fixed before deployment. Runtime guardrails also demonstrated automatic correction. When an S3 bucket was introduced without default encryption, the provider enabled encryption within minutes, blocked unencrypted uploads, and logged the event. The pipeline flagged the release for review, and posture returned to compliant. The initiative succeeded because engineers, security officers, and data scientists collaborated to translate governance into code, aligning technical guardrails with analytic workflows so that security and speed advanced together.

### 6.3. Case Study 3: AI Governance and Model Compliance (Financial Services)

A financial services firm building models for credit decisioning sought to enforce model governance like bias testing, approved data usage, and lineage through the same mechanisms that governed infrastructure. The team extended its IaC-centric pipeline with policy controls for AI artifacts. Model metadata and documentation were treated as first-class configuration, and OPA policies in CI verified that a model had passed defined risk checks and carried explicit approvals before any production deployment could proceed. Terraform managed the serving infrastructure on Azure, while Azure Policy enforced networking baselines, private endpoints, and workspace hardening. The data path was anchored by a catalog whose tags were cross-checked in pipeline code, preventing training jobs from accessing unapproved sources. Admission control and governance monitors ensured that manual attempts to bypass the pipeline were detected and reversed.

The approach reduced manual review overhead dramatically and gave auditors clear, reproducible evidence that deployed models met governance criteria at the moment of change. Policy false positives did occur early on, occasionally blocking novel but legitimate scenarios; the firm addressed this by introducing a controlled override process that required recorded human approval and generated an immutable audit trail. Over time, the override path was used less frequently as policy libraries matured and governance logic reflected lived practice. At runtime, admission control enforced that only signed, governance-approved model images could be deployed; manual pushes were blocked and reverted automatically, and the pipeline restored the last attested image while preserving a complete audit trail.

### 6.4. Case Study 4: Platform One "Big Bang" and DoD DevSecOps Baselines

The Department of Defense's Platform One initiative demonstrates how IaC can be industrialized across programs. The "Big Bang" stack delivers a hardened Kubernetes-based platform including service mesh, logging, and monitoring expressed entirely as code and aligned with STIGs and Zero Trust principles. Environments are reconciled continuously through GitOps, with Argo CD enforcing that running clusters match repository state. Compliance scanning and file integrity monitoring operate by default, and container images are drawn from hardened sources, ensuring that even custom AI services inherit a secure baseline. Programs adopting the stack reduced environment stand-up times from many months to a matter of weeks, and those embracing continuous authorization reported smoother accreditation cycles. In IL5 deployments, Zero Trust behaviors were enforced continuously through code. Admission controllers blocked workloads that deviated from hardened baselines, micro-segmentation limited every flow to policy-approved paths, and nodes that drifted from configuration were re-paved from the declarative baseline, converting deviation into a brief, well-documented exception rather than an enduring risk. The principal challenges were cultural and skills-based: teams needed engineers conversant in both Kubernetes and compliance, and some stakeholders initially favored checklists over automated controls. As deployments accumulated, the reliability and velocity of the codified approach spoke for itself.

### 6.5. Enterprise Trade-offs and Organizational Learning

Enterprise adoption of IaC and Policy-as-Code is as much an organizational transformation as it is a technical one. One Fortune 500 firm pursued an "everything-as-code" model with mandatory Policy-as-Code guardrails integrated into every IaC change. Security posture improved immediately, but developers initially perceived a slowdown as routine changes required review and policy updates. The inflection point came when development teams began contributing to policy code and reusable modules, converting gatekeeping into shared craftsmanship. Over several quarters, velocity returned while risk continued to decline, underscoring that the technology shift must be accompanied by a cultural one in which security becomes a property of the system, not a separate stage.

Taken together, these cases show that IaC for secure and compliant AI is not merely aspirational; it is already shaping operating models in government, healthcare, finance, and defense. The common thread is the conversion of policy into executable code, preventing noncompliance at design time, validating it at deployment, and sustaining it through runtime monitoring and reconciliation. The initial investment in modules, policies, and training is nontrivial, but the payoff is a delivery system where security scales with ambition. These lessons inform the discussion that follows on broader implications and future directions, including Zero Trust alignment, supply chain integrity, and the role of software bills of materials in extending IaC principles to the full AI stack.

## 7. Conclusion

The rapid deployment of AI from cloud-hosted machine learning services to mission-critical, on-premises analytics demands a fundamental rethinking of how trust is established and sustained. This paper has shown that Infrastructure-as-Code, coupled with DevOps automation and Policy-as-Code, transforms trust from an after-the-fact assertion into an intrinsic property of system design and day-to-day operations. When infrastructure and controls are expressed in code, the result is a delivery system where the desired state is unambiguous, enforcement is automatic, and results are produced as a natural by-product of change.

The urgency is plain. Manual security and compliance processes cannot match the speed, scale, and complexity of modern AI pipelines. Under rigorous U.S. frameworks such as FedRAMP, HIPAA, and NIST SP 800-53, automation closes the gap by making consistency the default and misconfiguration the exception. IaC standardizes environments across development, testing, and production, reducing error-prone variability and embedding secure-by-design practices into the very act of provisioning. Policy-as-Code extends this posture by preventing nonconforming changes at design time, validating the final plan at deploy time, and sustaining conformity through continuous runtime monitoring and drift reconciliation.

The cases examined in this paper demonstrate how transparency and determinism create confidence among stakeholders. Auditors and assessors can review Terraform modules, Sentinel or Rego policies, and pipeline logs rather than relying on screenshots and anecdotes. Teams benefit from versioned histories and immutable traces of who changed what and when, enabling both rapid forensics and measured accountability. Organizations reported fewer late-stage findings, shorter accreditation timelines, and smoother audits once controls were defined and enforced through code, even as their AI platforms evolved quickly.

Equally clear is the cultural transformation required to capture these gains. The learning curve for new toolchains, policy languages, and gated workflows can initially slow teams accustomed to ad-hoc changes. The remedy is disciplined practice: shared modules, curated policy libraries mapped to authoritative controls, scheduled policy reviews, and periodic red-team exercises that keep automation honest. Over time, the same guardrails that feel constraining at first become accelerants, making every change safer and more predictable.

Looking forward, IaC will be foundational to Zero Trust architectures. Least privilege, micro-segmentation, continuous authentication and authorization, and pervasive encryption are brittle when configured manually but durable when expressed declaratively and enforced automatically across clouds and clusters. The concept of an infrastructure bill of materials derived from IaC definitions will mature alongside software SBOMs, improving provenance, impact analysis, and supply-chain resilience for the platforms that host and serve AI. As governance expectations expand to include model lineage, fairness testing, and drift bounds, the same mechanics apply to express the rule as code, evaluate it continuously, and preserve the results as verifiable evidence.

Infrastructure-as-Code, reinforced by Policy-as-Code and continuous runtime controls, collapses the distance between requirement and enforcement, enabling AI programs to move quickly without sacrificing assurance. In domains where AI decisions carry clinical, financial, or national-security consequences, such assurance is not optional. It is the price of admission. By embracing the inseparability of DevOps and AI, investing in codified controls and verifiable pipelines, and fostering collaboration across industry and government, we can scale AI and the invisible scaffolding of trust that must rise with it.

## Compliance with ethical standards

### *Disclosure of conflict of interest*

The author declares that there is no conflict of interest.

### *Author contribution*

The author contributed to this manuscript and approves this submission.

## References

[1] 2023 Thales Cloud Security Study - Global Edition Report. (2023). Thales Cloud Security Products.

[2] Advancing medical research with MLOps. (2025). Chaosgears.com; Chaos Gears.

[3] AlphaBravo Engineering. (2025, May 3). Infrastructure as Code: Ensuring Security and Compliance in Cloud Deployments for the Federal Sector. AlphaBravo Engineering Blog.

[4] Buckner, M. (2025, May 5). AI-Powered DevSecOps: Navigating Automation, Risk and Compliance in a Zero-Trust World - DevOps.com. DevOps.com.

[5] Cloud Vulnerability Exploitation Examples. (2019). Cadosecurity.com.

[6] HashiCorp. (2020, February 23). Automating FedRAMP Security Compliance with Terraform. HashiCorp.com; HashiCorp.

[7] McAllister, D., Istfan, A., Thomas, D., Mejia, A., & Nahar, S. (2024, November 5). Operational Best Practices for FedRAMP Compliance in AWS GovCloud with AWS Config | Amazon Web Services. Amazon Web Services.

[8] Michalowski, M. (2024, July 16). Benefits and Best Practices for Infrastructure as Code. DevOps.com.

[9] Nedunoori, V. (2024, December 2). The Cost of Cloud Misconfigurations: Preventing the Silent Threat. Informationweek.com; Information Week.

[10] NIST SP 800-204C, DevSecOps for a Microservices-based App | CSRC. (2022, March 8). Csrc.nist.gov; Computer Security Resource Center NIST.

[11] Pallardy, C. (2024, May 30). UniSuper's Cloud Outage and Google's "One-of-a-Kind" Misconfig. Www.informationweek.com; Information Week.

[12] Potter, S. K. (2024, July 29). Securing and Configuring AI Environments: AI in Operations | Puppet. Puppet.

[13] SentinelOne. (2025, May 15). Key Cyber Security Statistics for 2025. SentinelOne.

[14] Tabassi, E. (2023). Artificial Intelligence Risk Management Framework (AI RMF 1.0). NIST.

[15] Taylor, B. (2025, May 25). Building Secure AI in Every Stage of DevOps. Informationweek.com; Information Week.

[16] Tesla and Jenkins Servers Fall Victim to Cryptominers - Notícias sobre segurança. (2018). Trendmicro.com.

[17] The State of DevSecOps. (2025). U.S Department of Defense.

[18] Threat Horizons H1 2024 Threat Horizons Report. (2024). In Cyentia Cybersecurity Research Library. Google Cloud.

[19] What is DevSecOps? (2024). Cloud.mil. https://www.cloud.mil/devsecops/