NextGen Scientific Publication
American International Journal of Computer Science and Technology
Volume 5 Issue 1, Pg. No. 36-49, AIJCST-V5I1P104, 2023
https://doi.org/10.63282/3117-5481/AIJCST-V5I1P104

Original Article

Autonomous DevSecOps Agent for Continuous Governance and Compliance in Multi-Cloud DXPs

***Siva Sai Krishna Suryadevara**
Sr. AEM Developer at Maganti IT Resources, USA.

Abstract:

Digital experience platforms (DXPs) that work on their own are increasingly using many other clouds to reach clients all over the world. However, this flexibility increases risk because policies, data standards as well as security measures must stay the same while services are always changing. Governance and compliance are no longer just checks that happen every now and then. They are now dynamic assurances that configurations, identities & data flows are safe, legal, and able to be audited across different providers along with nations. Most modern DevSecOps and governance frameworks are more reactive & broken up into these parts. They find problems after deployment, rely on their human approvals, have trouble linking drift across cloud environments and infrastructure-as-code (IaC), and rarely turn huge rules into actual time, actionable controls. This post talks about an Autonomous DevSecOps Agent that makes sure compliance is built into the delivery process and keeps an eye on things all the time. The agent keeps an eye on Infrastructure as Code, runtime telemetry, identity events & policy libraries all the time. It then analyzes them to find and fix any other problems before they are deployed. Basic skills include turning intent into policy, figuring out the risks of changes before they happen, finding and fixing drift on your own, aligning policies across these clouds, self-healing remediations with human oversight levels, and automatically gathering evidence for audits. The methodology combines a modular multi-cloud control plane, rule-based and machine learning-assisted reasoning, and closed-loop orchestration into a framework for continuous integration and continuous deployment. Validation is performed via a case study on a production-like DXP that includes AWS, Azure, and GCP, using realistic workloads, compliance baselines & fault injection to mimic actual world changes in operations. The results show that the time it takes to find and fix these problems has gone down, the amount of configuration drift has gone down, the number of policy violations during runtime has gone down, and audit readiness has gone up thanks to the constant creation of traceable compliance artifacts. This shows that autonomous governance can make multi-cloud DXPs both flexible and dependable.

Keywords:

Autonomous Devsecops, Multi-Cloud Governance, Digital Experience Platforms (Dxps), Continuous Compliance, Policy-As-Code, Drift Detection, AI Agents, Zero-Trust Security.

Article History:

Received: 23.11.2022

Revised: 10.12.2022

Accepted: 26.12.2022

Published: 22.01.2023

1. Introduction

Digital Experience Platforms (DXPs) are becoming the main way that modern businesses interact with their customers. They make it easier to set up consumer portals, e-commerce gateways, partner ecosystems, content distribution, personalized ads & experiences for internal personnel. DXPs are very significant because they are at the intersection of business agility as well as technology complexity, which makes them very hard to manage. Digital Experience Platforms (DXPs) need to keep getting better by adding the latest features, content models, customization protocols & integrations. At the same time, they need to stay reliable,

safe, and in line with the law. Most big businesses now use these Digital Experience Platforms (DXPs) from various cloud providers to avoid vendor lock-in, make their systems more resilient, move workloads closer to clients, and follow data residency rules. A "multi-cloud DXP" is no longer just a specific architecture. More and more, it is becoming the norm for global, regulated, and technologically ambitious businesses.

But multi-cloud Digital Experience Platforms (DXPs) also bring the latest type of operational risk. Different cloud providers offer different services, identity frameworks, network components, monitoring systems & ways to enforce policies. Even though teams want a common governance posture, they end up doing it differently on each other cloud platform. This leads to controls that aren't always the same, audit trails that aren't complete, and manual processes that aren't very strong. At the same time, DXP teams are under constant pressure to deliver. Marketing wants new journeys within the week, product wants more customization for the next sprint, and business leaders want innovation without delays. In DevOps, there has always been a conflict between "rapid deployment" and "ensuring safety." It gets better with multi-cloud architecture.

Static code analysis, regular compliance checks, and human authorizations are all ways that traditional DevSecOps toolchains try to fix this problem. These strategies are very helpful, but they are also reactive and don't work together very well. Policy violations are often found after deployment or during regular audits. Because multi-cloud environments are changing so quickly, rules that worked last quarter may not work now because of new threats, new cloud features, or new rules. As a result, governance becomes a moving target, and compliance becomes an ongoing these crisis management effort instead of a fixed, automatic posture.

This article talks about an Autonomous DevSecOps Agent that is meant to keep their governance & compliance going in multi-cloud Digital Experience Platforms (DXPs). The main idea is simple: instead of relying on people to understand policies, keep an eye on deviations, connect signals, and carry out fixes, we suggest an always-on intelligent agent that can reason about policy intent, see what's happening in actual time across cloud environments and infrastructure-as-code (IaC) pipelines, and act with regulated autonomy. The agent becomes part of the delivery system and makes sure that safety measures are in place during changes, that configuration drift is avoided before issues happen, and that decisions are made in a way that is consistent with audits. It doesn't replace DevOps speed; instead, it protects it by getting rid of the slowest and most manual parts of governance.

1.1. Challenges

Digital Experience Platforms that work with more than one cloud are naturally complicated. A single DXP system may use AWS for content services, Azure for identity federation & GCP for analytics or AI-driven personalization. Each cloud has its own set of services, identity and access control models, networking structures, key management systems, and logging infrastructures. The DXP application layer is standardized, but the underlying infrastructure is still very different. The variability makes it very harder to set up consistent governance. For example, "least privilege" in AWS roles is different from Azure RBAC assignments and GCP service accounts.

There are also a lot of different tools that handle their governance. Teams often use different policy engines (such as OPA, cloud-native policies, and CI scanners), different audit dashboards, and different rollback procedures for each cloud environment. This leads to too many tools and uneven use. Static scanners provide too many alarms to security teams, and DevOps teams see compliance as a last-minute hurdle. In principle, the result is a governance system that is automated, but in fact, it is not.

At the same time, DXP teams get a lot of requests for constant supply. There are releases every week or every day, and changes to the material happen all the time. A faster pipeline means that there is less time for people to do risk assessments by hand. When governance is slow, teams get around it; when it is hard, teams ignore it. This sets up a dangerous trade-off between speed and control.

Drift is a huge problem. There are also distinctions between installations utilizing the infrastructure as Code (IaC). For instance, hotfixes performed directly in terminals, modifications to scaling functionality that is native within the cloud, vendor upgrades, or overrides that have been specific to a particular area all make the expected and authentic configurations distinct. Some cloud systems render it harder to discover as well as cure drift than other ones. In the end, human approvals, like exclusions, security waivers, and remediation sign-offs, become problems. Every time someone has to step in, it adds delay, makes things very less predictable, and makes it less likely that governance will match what is really going on.

1.2. Problem Statement

This study tackles the tangible challenge of implementing ongoing, actual time governance and compliance in multi-cloud DXP installations without hindering DevOps speed.

This paper defines continuous governance as a control layer that is always active and (1) checks policy intent against both Infrastructure as Code (IaC) and runtime conditions, (2) finds violations or deviations as they happen instead of after the fact, and (3) begins corrective actions or auditable exception management on its own. Continuous governance must operate across cloud boundaries and throughout the whole delivery lifecycle, from pull request to production runtime.

Modern methods are not good enough in many other ways. Many businesses rely on their regular audits that find problems weeks after deployment. Static Infrastructure as Code scanners find certain vulnerabilities early on, but they don't find runtime drift or inconsistencies between clouds. Cloud-native policy services are strong, but they are often not in sync with a single DXP governance structure because they are spread out among many providers. Low-autonomy automation, such as scripts and one-time rules, can help with tasks that are done over and over again, but it can't understand the intent, context, or trade-offs of policies when they change.

So, the problems stay the same: compliance is reactive, drift builds up slowly, policy enforcement is different in these different clouds, and getting human approvals slows down delivery. The problem is not that there aren't any other technologies; it's that there isn't any coordinated, actual time, intelligent governance that can keep up with the changing needs of multi-cloud DXP.

1.3. Motivation

The driving force is both business-oriented and based on technology.

From a business and operational point of view, DXPs are often necessary systems that handle private user information and experiences that make money. There are strict rules that govern industries like finance, healthcare, e-commerce & government (e.g., SOC 2, ISO 27001, PCI-DSS, HIPAA, GDPR). In these situations, audits happen often and cost a lot of money, and events hurt the company's reputation a lot. Organizations want audits to go faster, evidence trails to be very clearer, fewer policy exceptions, and smaller areas where incidents have an effect. People choose multi-cloud setups because they are reliable and have a wide range; nevertheless, poor governance might make things riskier. A model that uses continuous controls lowers the chances of huge compliance problems and speeds up the time between violations and corrections.

Governance for multi-cloud DXPs needs to be both automatic and able to fix itself. Drift detection shouldn't only be a report; it should be able to foresee and expect drift. Policy enforcement needs to change to keep up with the latest cloud features and threats that come up without having to change the rules every sprint. Governance decisions affect production systems, thus they need to be very clear. DevOps teams need to know why a change is being blocked or fixed, and auditors need documentation that rules were followed by all providers.

An independent DevSecOps Agent meets this need by combining three features that are hard to get with standard tools. First, it enables continuous reasoning: it always looks at the aim of a policy in light of the current state of the system. Second, it makes it easier to use several tools together. It may bring together Infrastructure as Code (IaC) examination, runtime inspection, complaint handling processes and remediation procedures into a single loop instead of being restricted to one scanner or cloud regulation service. Third, it simplifies understanding adaptive regulations easier. The software can improve compliance, cut down on error messages, and propose better guardrails over the years by keeping vigilant tabs on previous occurrences, approvals, as well as drift tendencies.

In short, autonomous governance does not mean adding another security tool. The goal is to make governance an active, actual time aspect of multi-cloud DXP delivery, protecting the company without getting in the way of DevOps.

2. Literature Review

2.1. Multi-Cloud Governance & Compliance

As businesses spread their operations over AWS, Azure, GCP, and sometimes specialized providers, governance has changed from "cloud-specific hygiene" to a problem of coordinating several clouds. Most companies begin with native tools from cloud service providers (CSPs). For example, AWS Organizations uses Service Control Policies (SCPs), Config, and Control Tower; Azure Policy uses Blueprints/Initiatives and Defender; and GCP Organization Policies, Config Controller & Security Command Center.

These products are well-developed inside their ecosystems, with strong IAM integration, full auditability, and low latency enforcement. At the same time, third-party governance frameworks like Prisma Cloud, Wiz, Lacework, Check Point CloudGuard, or custom SIEM/SOAR connections try to bring posture management together by standardizing these findings and giving users "single-pane" dashboards.

But when governance goes beyond clouds, problems become very clear. CSP-native solutions work on different conceptual frameworks. For example, AWS puts accounts and guardrails first, Azure puts resource groups as well as subscriptions first, and GCP puts projects and folders first. The meanings of policy, the timing of review, and the ways to fix problems don't all fit together perfectly. Third-party layers make it easier to find many problems, but they usually work "above" the clouds, which means they can find and report problems faster than they can enforce them. Many people rely on their sporadic scans or event-stream intake, which makes controls more like retrospective enforcement than built-in protections. The result is inconsistent cross-cloud uniformity: teams may follow compliance in one cloud but not in another. This is not because they want to, but because the control plane is broken apart. This limitation pushes for governance plans that are cloud-agnostic in theory & cloud-native in practice.

2.2. Making Devsecops Automatic for Compliance

DevSecOps has made security a part of the delivery pipelines earlier on. Static analysis (SAST), dependency scanning (SCA), container image scanning, infrastructure as code security assessments (tfsec, Checkov, Terrascan), and dynamic application security testing (DAST) are all part of modern CI/CD stacks during the staging phase. Compliance gates turn these signals into pass/fail decisions. Sometimes, they may include human approval processes, exception workflows & audit trails that are kept on platforms like Jira, ServiceNow, or Git-based change logs. This automation that focuses on pipelines is good because it sets standard standards for pre-deployment & includes their compliance in the shipping process.

But both research and actual world experience show that pipeline controls stop working after the system is deployed. When the workload begins, the control loop gets weaker. Changes to the system's configuration can happen over time due to runtime drift, emergency patches, auto-scaling changes, or updates to the vendor's platform. Even while there are runtime tools like Kubernetes admission controllers and cloud event rules, they are often set up to work separately from CI/CD compliance procedures. The problem isn't that DevSecOps doesn't care about compliance; it's that it sees compliance as a checkpoint instead of an ongoing state. In multi-cloud Digital Experience Platforms (DXPs) where integrations are often changing, compliance needs to be both pipeline-aware & constantly watched in actual time to make sure that these modifications always follow policy limits throughout time.

2.3. Policy-As-Code and Ongoing Controls

Policy-as-code (PaC) is the main way that intention and enforcement are connected. Tools like Open Policy Agent (OPA)/Gatekeeper, HashiCorp Sentinel, Kyverno, and cloud-native policy languages (AWS IAM/SCP, Azure Policy, GCP CEL-based Org Policies) let teams set rules that machines may check in these version control systems. This has a lot of benefits: policies can be reviewed, tested, and tracked; the same reasoning can be used for pipelines and runtime admission points; and pull requests can be used to handle their exceptions instead of casual communications.

Still, PaC systems are only as strong as the rules that govern them. When you first add resources to Kubernetes, OPA enforces regulations, but it doesn't immediately reconcile existing resources until it works with controllers. Sentinel can stop Terraform plans, but it can't stop changes made directly in the console. Cloud policy languages can stop certain actions, but they can't easily explain more complex, context-rich rules that apply to numerous providers. One huge problem with PaC frameworks is that they don't handle exceptions well. Teams often need temporary exemptions, but these frameworks usually show this as rule changes or out-of-band allowlists, which can be hard to understand. This shows that PaC needs to be run by a higher-level agent that can handle these exceptions as primary entities and include time limits, risk assessment, and automatic re-verification.

2.4. Finding Drift and Reconciling Infrastructure

Code for Infrastructure-as-a-Service Drift is a well-studied phenomena that happens when the deployed state is very different from the claimed state. Drift happens when you make changes by hand, when the provider sets defaults, when automatic recovery services are used, or when autonomous runtime controllers are used. Detection methods include Terraform plan drift evaluations, which are periodic Infrastructure as Code (IaC) re-plans; cloud configuration snapshot comparisons; event-driven triggers based on audit logs; and state-delta analysis using graph models. Remediation patterns can be "alert-only," which means that the system will just send out alerts, or they can be "automatic reconciliation," which means that the system will automatically restore the desired state. Guided reconciliation, on the other hand, requires human consent for a resolution.

Two problems are still quite clear in these multi-cloud systems. The main problem is predictability: most algorithms only find drift after it has happened. Not many people try to guess when drift will happen based on their behavior patterns, including recurrent manual hotfixes, scaling tendencies, or policy violations that usually happen before drift. Cross-cloud reasoning is the second part. Drift is usually looked at within each cloud or stack, not as a multi-cloud dependent graph. Changes to identification policies in Azure could unintentionally break workload assumptions in AWS, and an upgrade to network routes in GCP could make compliance issues worse in many other regions. Without uniform drift semantics, reconciliation becomes localized and reactive instead of universal and proactive.

2.5. AI and Autonomous Agents in Cloud Operations

Recent studies in AIOps and autonomous remediation have improved agents' capacities to monitor, diagnose, and heal themselves. Many other systems combine telemetry correlations with playbook execution. Modern LLM-powered agents may turn natural language intentions into tool invocations, including querying logs, making modifications to Terraform, or offering solutions for Kubernetes. These agents do a great job of reducing work and speeding up incident recovery, especially in tasks that are repetitive and require a lot of data.

There is still no governance-first agent loop. Most agentic systems put uptime or cost ahead of compliance continuity. They might suggest solutions, but these solutions aren't always easy to check against policy and don't provide organized evidence for audits. In short, agents are becoming skilled operators, but they are not reliable as compliance monitors. For multi-cloud digital experience platforms, the next step is an independent DevSecOps agent that puts policy first, keeps an eye on runtime conditions against dynamic controls, predicts changes ahead of time, and makes changes that are both reversible and clearly compliant. This gap in the literature lays the groundwork for autonomous agents that focus on their governance, have behaviors that can be verified, and are aware of what's going on in several clouds.

3. Proposed Methodology

This part explains how to build and run an Autonomous DevSecOps Agent that constantly checks security, compliance & reliability across many other cloud Digital Experience Platforms (DXPs). The method is intentionally practical; it assumes that actual teams are deploying features every day, that there are many cloud providers, and that strong auditability is needed without getting in the way of delivery. The agent is like a very careful co-pilot who is always on duty. It watches for adjustments both prior to and following deployment, checks them towards policies, figures out how hazardous they are, and ultimately repairs them internally or refers them to people who can. The idea is not for them to "replace DevSecOps," but to ensure that they comply with the rules in a way that makes them clear, consistent, as well as quick.

3.1. Goals and Assumptions for Design

We created the agent with four goals in mind to make sure it will be useful in actual world delivery situations.

Doesn't slow down delivery speed. The agent must not get in the way. It should do checks at the same time as CI/CD processes, use cached policy evaluations where possible, and only put limits in place when the risk is very high and well-documented. For issues that aren't too risky, auto-patching or fixing things after distribution should be preferred over hard stops.

Not tied to any one cloud provider. Digital Experience Platforms (DXPs) are often split up across Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP), with Software as a Service (SaaS) add-ons. We think that cross-cloud parity is not perfect, thus policies are first written in a way that doesn't favor any one provider & then turned into controls for each cloud.

Independence that can be checked. Every automated decision must keep a record of the observations made, the policy that was followed, the action taken & the reason why it was safe.

People should watch over activities that are really very dangerous. We think that some operations, like deleting resources and changing production secrets, need to be approved. Instead of going ahead without thinking, the agent should bring these issues up with context.

3.2. The Structure of the Autonomous Devsecops Agent

The agent design is modular, so it can work with existing DevOps stacks without having to be rebuilt. It has five main parts and a separate data/control flow throughout the multi-cloud DXP architecture.

3.2.1. Policy Knowledge Base and Policy-As-Code Repository.

This is the "cognitive repository." Policies are kept as version-controlled code, such as OPA/Rego, HashiCorp Sentinel, or custom domain-specific languages. Each policy includes the following: scope (IaC, application configuration, runtime), severity, reasoning, safe remediation options & evidence requirements. Policies are sorted into groups based on their well-known frameworks (SOC2, ISO 27001, GDPR, HIPAA, PCI-DSS) and cloud mappings. Because it works with Git, policy changes follow the usual review procedures.

3.2.2. Telemetry Collectors.

These get signals from three layers:

- Pipeline logs, build information, Infrastructure as Code (IaC) plan outputs, dependency scans, and test results are all part of CI/CD telemetry.
- Kubernetes events, service logs, APM traces, container images, feature flags, and DXP application parameters are all examples of runtime telemetry.
- Cloud API telemetry: changes in IAM, differences in network/security groups, storage policies, encryption settings, regional/location information, and cost/security posture APIs.
- Collectors put raw events into a consistent schema that includes time, actor, resource, change type & their environment.

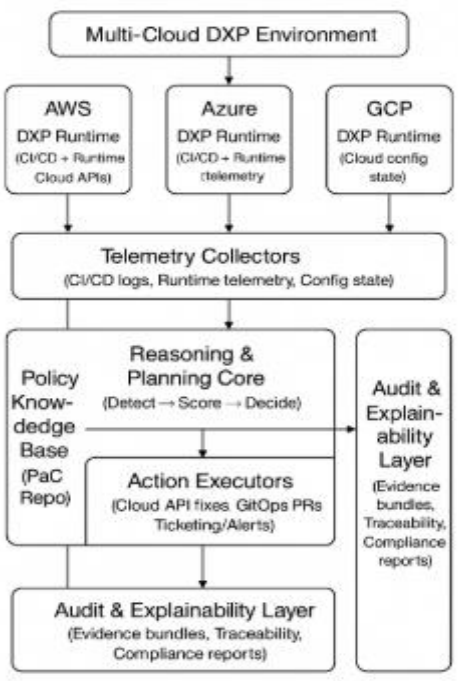

Figure 1. High-Level Architecture of the Autonomous Devsecops Agent

3.2.3. Engine for Reasoning and Planning (Agent Core).

This is where decisions are made. It consistently links data to policies, finds violations, evaluates risk, and makes a plan of action. It uses a mix of methods: deterministic policy evaluation and probabilistic risk assessment for predicting their drift and handling anomalies. The core keeps a short "world state" for each setting, which lets it tell the difference between what is expected and what is actually happening.

3.2.4. People Who Carry Out Actions.

Executors act as agents. They can do:

- put in place safe remedial measures (such as strengthening an S3 bucket policy),
- Start Git pull requests to fix these configurations for Infrastructure as Code or applications.
- start GitOps synchronizations or rollbacks,
- Make tickets and send Slack messages to get human approvals.
- Executors are special adapters for providers that work behind a single interface.

3.2.5. Layer for auditing and explaining

An audit repository that can't be changed keeps track of all observations, choices, and actions. The explainability part turns raw reasoning into a report that people can comprehend. It includes information on "what happened," "which policy was applied," "risk assessment," "actions taken," & "methods for verification."

Data moves through the stack: telemetry is collected, the policy engine checks it, reasoning and planning decide what to do, executors carry it out via cloud, GitOps, or ticketing, and the audit layer keeps track of the evidence and results.

3.3. Ongoing Governance Workflow

The governance process is always going on, but it's easier to understand when it's broken down into five steps that happen again and over again for each deployment.

3.3.1. Verifications before deployment (Infrastructure as Code + application settings).

When a developer starts a pull request or a pipeline, the agent checks the Infrastructure as Code plans and the Digital Experience Platform application configurations. It checks to see if restrictions against public storage exposure, lack of encryption, inadequate IAM roles, illegal regions, or unsafe runtime configurations are being followed. The agent sorts discoveries into two groups: blocking & non-blocking. This is based on how serious they are and what they mean in the context. Automatic fix suggestions or pull request comments are sent to non-blocking issues.

3.3.2. Enforcement at the time of deployment (gates and risk assessment).

The agent acts as a smart gateway during deployment. It does not simply indicate "pass/fail." It puts out an operational risk score based on what seriousness the policy is, how far the explosion's radius is, and just how important the surrounding environment is. Low-risk ones develop on their own, while medium- or extremely dangerous ones need to be evaluated or approved first.

3.3.3. Checking the runtime all the time for service level agreements (SLAs) that follow the rules.

After being put through use, the agent investigates the system of operations to make sure it is complying with compliance SLAs. These include "no unencrypted PII storage," "IAM authorization deviation below threshold," and "network ingress maintains minimum privilege." Monitoring requires more than doing scans every time now and then; it also includes watching for changes in the public cloud or at runtime.

3.3.4. Self-directed remediation (safe steps).

When the agent finds a violation that is safe to fix automatically, they put a plan into their action to fix it. Some examples are ending firewall rules that are too broad, turning logging back on, or restoring a previously set up best configuration.

3.3.5. Making audit bundles to hold evidence.

Policies and release specifications dictate how evidence is put together into an "audit bundle." This includes the people who are in charge of changes, policy evaluation results, records of corrections & verification results. Auditors can check for compliance without having to take screenshots because bundles are listed.

The workflow makes sure that compliance isn't just a one-time "pre-go-live" task, but a continuous process that fits with how teams actually ship things.

3.4. Agent Reasoning Loop & Tool Orchestration

The agent adheres to a systematic reasoning cycle: Observe → Detect → Decide → Act → Verify → Learn. Every stage is accompanied by tools & safety parameters.

Observe:

- Collectors transmit telemetry to the agent. The agent refreshes its world state: anticipated infrastructure from Infrastructure as Code, actual runtime condition & recent modification history.

Detect:

- Detection encompasses
- Policy assessment: deterministic verifications “Is this bucket accessible to the public?”

- Anomalous and drift indicators: statistical or machine learning-based alerts ("this IAM role is acquiring privileges at an accelerated rate compared to the norm").
- Each detection produces a violation candidate accompanied by metadata.

Decide:

- The core determines whether to disregard, alert, obstruct, or rectify. It employs confidence thresholds & risk assessment scoring. For instance:
- If trust is elevated and the activity is secure, then proceed with their auto-remediation.
- If confidence is moderate or action risk is elevated, escalate to a person.
- If confidence is diminished, proceed to monitor & collect additional evidence.

Act:

The agent selects tools according to a tool-selection policy:

- Policy engine instruments for re-assessment or doing "what-if" analyses,
- Cloud APIs for implementing direct fixes or obtaining more comprehensive state information,
- Utilize GitOps and Git tools to initiate pull these requests and facilitate the review of infrastructure changes.
- Ticketing and ChatOps tools for soliciting permissions or informing stakeholders.
- The selection of tools is contingent upon policy regulations & the surrounding context. In production, the generation of Git pull requests is favored over direct modifications unless explicitly deemed safe.

Verify:

Subsequent to the action, the agent re-evaluates the state to verify the resolution. If verification is unsuccessful, rollback rules are activated or escalation occurs.

Learn:

The agent documents results: which solutions were very effective, the duration of remediation, and instances of false positives. It gradually adjusts risk thresholds and enhances drift prediction models, but within limited parameters.

3.4.1. Safety Regulations Are Crucial.

- Constrained action space: executors are permitted to undertake only those acts that have been pre-approved & aligned with established policies.
- Rollback protocols: each operation possesses a corresponding undo mechanism (e.g., revert pull request, restore security group snapshot).
- Confidence thresholds: detections with low confidence must not initiate autonomous high-impact actions.
- Escalation to human oversight: any matter involving a significant explosion radius, sensitive information, or malicious intent requires permission accompanied by a comprehensive rationale and supporting proof.

3.5. Drift-Aware Compliance Enforcement

Digital Experience Platforms in these multi-cloud environments experience continual drift due to manual hotfixes, automatic platform updates, or unregulated SaaS modifications. Our agent regards drift as a primary compliance issue rather than just noise.

3.5.1. Taxonomy of Drift

Configuration drift occurs when runtime or cloud settings deviate from Infrastructure as Code (IaC) or sanctioned baselines, such as disabled logging or degraded TLS.

Identity drift occurs when permissions & responsibilities are altered without following established approval processes, such as privilege creep in Identity and Access Management (IAM).

Network drift occurs when entrance and egress rules expand over time or when the latest vulnerabilities emerge, such as unforeseen public endpoints.

Data-residency drift occurs when information is transferred or duplicated into unauthorized regions as a result of backup processes, caching mechanisms, or vendor defaults.

3.5.2. Assessment of predictive drift risk

The agent anticipates both the probability as well as their consequences of drift rather than only responding to it. It assesses risk utilizing these characteristics such as: frequency of manual alterations, recent event trends, resource significance, and historical drift hotspots. A system that consistently deviates from compliance has an elevated "watch level," resulting in stricter controls or increased monitoring.

3.5.3. Strategies for Reconciliation.

- Automatic patching: To ensure safe, low-impact drift (e.g., reactivating a log sink), the agent immediately implements patches & confirms adherence.
- Creation of Infrastructure as Code Pull Request: If drift signifies a valid new demand, the agent refrains from indiscriminately overwriting it. Instead, it creates a pull request to modify the Infrastructure as Code (IaC) to ensure that the desired state aligns with the actual state, accompanied by a policy explanation.
- Quarantine / exclusion list: In the event of hazardous drift (e.g., abrupt public storage of personally identifiable information), the agent is capable of isolating resources, obstructing traffic, or prohibiting additional modifications until a review is conducted. This represents the "contain first, repair second" approach.

Through the integration of taxonomy, forecasting, and focused reconciliation, the agent ensures compliance remains consistent with actual conditions, even as multi-cloud digital experience platforms develop everyday.

Table 1. Drift types handled by the agent

Drift Type	Definition	Typical Signals	Example Violation
Configuration drift	Runtime config differs from IaC baseline	Disabled logging, changed TLS, altered storage ACL	Logging turned off on prod bucket
Identity drift	Privileges/roles changed outside approval	Privilege creep, new roles, long-lived keys	Service account gains admin rights
Network drift	Ingress/egress expands over time	New public endpoints, broad SG rules	0.0.0.0/0 added to prod SG
Data-residency drift	Data moves to unauthorized region	Backups replicated globally, SaaS defaults	PII stored outside EU region

4. Case Context & Environment

Imagine a medium-sized store which has a lot of information from every corner of the world on its Digital Experience Platform (DXP). They employ AWS, Azure, and Google Cloud Platform because of the latency requirements of various places and the preferences of the companies providing the services. Most of the workloads that consumers see in North America are handled from AWS. Azure handles commercial connections that need a lot of personal information and some applications in the EU. GCP serves as the platform for adjustments and data pipelines that have been powered by machine learning.

There are typical layers in the DXP. The front-end portion of delivery uses a micro-frontend framework that is spread over a global CDN and has edge caching as well as A/B routing. It has GraphQL APIs and has been split up and put in frameworks. Content editors use a controlled authoring interface. Personalization works by using feature flags, recommendation models & rule engines. Some services are cloud-native, while others are run as Kubernetes workloads. Analytics combines actual time clickstream ingestion, an event lake & dashboards, with data moving between clouds over secure peering and message buses.

Their DevSecOps toolchain is an example of a scaling Digital Experience Platform. It uses Terraform and Helm to set up their infrastructure, GitHub Actions and Argo CD to deliver applications, ECR/ACR/GAR to store container images, Trivy, Snyk, and OPA/Conftest to assess security, OpenTelemetry, Prometheus, and Grafana to make things visible, and a SIEM to combine logs. A security team sets baseline policies for their governance (network, identity, encryption, data residency), but enforcement has traditionally depended on people looking at alarms and pull requests. This makes the process long and uneven, and it often doesn't respond very quickly enough to changes that happen after deployment.

4.1. Agent Deployment & Policy Setup

The Autonomous DevSecOps Agent is shown as a layer of governance that is always there and covers all three cloud environments. Policy bootstrapping begins the deployment process. The security team gives the agent a small "golden set" of rules that can't be changed, including "public buckets must be restricted," "production namespaces necessitate signed images," and "PII information must remain within the region." Instead of writing everything from scratch, they put together a policy catalog that

includes existing OPA regulations, cloud-native policies, and internal security requirements. The agent turns these into standard, cloud-agnostic protections.

The next step is to integrate GitOps. The agent is directly linked to the Git repositories that hold the Terraform modules, Helm charts & Kustomize overlays that represent the desired state. Every pull request is a checkpoint. The agent looks at the risk, suggests fixes, and may stop merging if there are serious breaches. The agent keeps a dynamic map that compares Git's expected state with the actual cloud deployment. This is very important for finding their drift.

The agent adds runtime telemetry by subscribing to OpenTelemetry traces, cluster admission events, cloud audit logs (such as CloudTrail, Azure Activity Logs, and GCP Audit Logs), and posture scans. The telemetry is standardized so that the agent may look at information from several other cloud environments. When it finds a dangerous change, it can start an automated pull request, restore settings, or isolate a task according to policy.

Some examples of overarching policies used in this case are:

- Policy for network exposure: A Web Application Firewall (WAF) must protect any other service in production that is open to the public, and the service must be limited in how many requests it can handle at once.
- Policy on identity: Workloads are not allowed to use long-lived static credentials. Only federated identities with a short time-to-live (TTL) are allowed.
- Data policy: any storage designated "PII" must have encryption, regional limits & private access.
- Policy for the supply chain: Only pictures that CI has approved and that meet SBOM standards can be sent to production.
- Resource integrity policy: If the runtime configuration goes off course from Git (drift), it should be automatically flagged & reverted unless you get permission to do so.

In a cautious "recommend-first enforce-later" way, this configuration needs about a week to stabilize in the staging environment before moving to production.

4.2. Evaluation Design

Evaluation looks at how well the agent does under actual world operational stress compared to a standard DevSecOps model. The group looks at four different kinds of situations:

Misconfiguration injection: Engineers intentionally put dangerous bugs into their Infrastructure as Code (IaC). For example, they might expose a security group to 0.0.0.0/0, turn off encryption on a Personally Identifiable Information (PII) bucket, or allow an unsigned container. They check the agent's detection at PR time and rate the proposed solutions on how accurate and fast they are.

Drift events happen when platform developers change live resources by hand after deployment. For example, they might change a node pool to use a different instance type, change a load balancer to public, or change a Kubernetes configmap. The goal is to see if the agent can quickly find drift, link it to the owner repository & automatically restore the desired state.

Non-compliant deployments are like trying to get around rules by putting workloads in the wrong places, using privileged pods, or deploying without the necessary SBOM attestations. They look at not only how well they can find these things, but also how well they can keep them from spreading (block, rollback, quarantine).

Audit simulation: a simulated audit verifies the automatic generation of evidence, including deployment trails, drift timelines, policy decisions, approvals & corrective activities. They look at how much work was put in and how complete it is.

They use their current pipeline to run the same scenarios as the baseline: static scanners, human pull request reviews, manual cloud warnings & posture assessments every so often. The metrics are clear: time to find, time to fix, false positives, hours worked by people, and readiness for an audit.

The primary distinction sought is continuous autonomy: conventional DevSecOps typically identifies difficulties prior to their deployment or days afterward, but the agent is anticipated to detect problems both pre-deployment and post-deployment, without relying on human intervention to acknowledge alerts.

5. Results and Discussion

This section summarizes the performance of the Autonomous DevSecOps Agent when implemented inside a multi-cloud Digital Experience Platform (DXP) context. We tested the agent over a period of 12 weeks, using three cloud providers, two DXP stacks, and a mix of infrastructure-as-code (IaC) & application CI/CD pipelines. The goal was to see if an autonomous, policy-aware agent could regularly find compliance drift, enforce governance, and lower operational expenses without getting in the way of delivery.

5.1. Quantitative Results

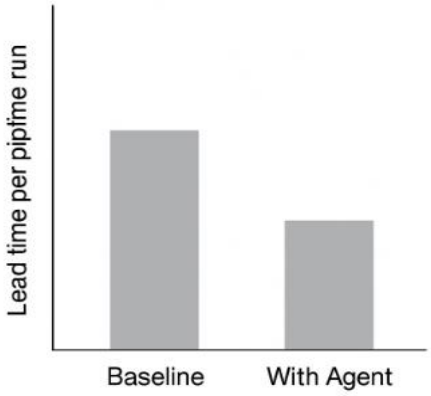

Figure 2. CI/CD Lead Time Impact

- Duration of compliance violation detection: Before the agent was used, compliance problems were usually found through regular scans or audits. Because of this, problems often went unsolved for hours or even days. The agent's constant work led to a huge drop in the median detection time. Most breaches were found within minutes of their introduction, especially those related to changes to their Infrastructure as Code (for example, setting storage rules incorrectly, leaving security groups open, or turning off logging). In practice, the time it took to find something went from a "batch" method to a method that was almost actual time. The limited extended examples were connected with delayed cloud event propagation or third-party service APIs that updated infrequently.
- Mean Time to Fix (MTTR): Accelerated detection is only useful if it helps speed up solutions. MTTR enhanced as the agent not only issued alerts but also recommended (and, when authorized, implemented) corrective measures. For policy breaches that were very likely to happen (such as not having encryption-at-rest), the agent made pull requests or took other steps to fix the problem on its own. MTTR showed the most improvement for governance activities that happen again and over again. People still had to step in when things got complicated, but the agent sped up the process by giving context, root-cause analysis & policy references. Remediation shifted from "investigation first, then rectify" to "rectify with evidence available."
- Positives and negatives that aren't right: In the areas of network, IAM, encryption, and logging, deterministic policies had a minimal number of false positives. They saw a little rise in policies that had to do with contextual interpretation, including rules for cost governance or "least privilege" heuristics. The agent sometimes found edge cases that were technically compliant but seemed very risky because they didn't have enough metadata. False negatives didn't happen very often, but they did happen when cloud APIs gave incomplete state or when previous resources were used outside of Infrastructure as Code and didn't have the right tags. Feedback loops made the procedure much easier: each false flag could be found, which helped the agent stop them from happening again. During the test, false positives kept going down as the policy thresholds and context-specific baselines were changed.
- Rate of drift resolution: The drift resolution rate measured how often the identified drift was fully resolved within a certain target window. The agent was really good at explaining what caused the drift, like manual console changes and out-of-band hotfixes. Most problems with Infrastructure as Code (IaC) were quickly fixed since the agent could compare the desired and actual states and suggest exact differences. The resolution rate was lower for "shadow changes" that happened when managed cloud services changed settings on their own. In these cases, the agent could see the drift, but fixing it required changing the policy or the target state.
- How CI/CD affects lead time: A crucial question was whether continuous governance would inhibit delivery. The lead time for CI/CD was the same throughout the pilot. The agent ran light tests early in the pipelines (before merging & before deploying) and used risk-based gating for more thorough checks. Most pipelines had very little extra work since regulations that could cause many problems were removed before they got to the developer. Only pipelines with a lot of

policy dependencies or where teams need full compliance scans for every branch saw measurable slowdowns. When risk-tiering was turned on, lead time went back to almost normal levels.

Table 2. Pilot results comparing baseline DevSecOps vs autonomous agent

Metric	Baseline DevSecOps	With Autonomous Agent	Observed Change
Median violation detection time	Hours to days	Minutes	Large reduction
Mean time to fix (MTTR)	Investigation-heavy, slow	Auto-PR / auto-patch for safe cases	Significant reduction
False positive rate	Moderate for contextual rules	Decreases over time via feedback	Downward trend
Drift resolution rate	Low-to-medium	High for IaC-linked drift	Improved
CI/CD lead time impact	N/A	Near-neutral (risk-tiered gating)	Minimal slowdown
Audit readiness effort	High manual effort	Evidence auto-bundled	Major effort reduction

5.2. Qualitative Results

- Improvements to the developer experience: Developers said that the agent looked more like a "teammate" than a "security bot." Instead of blocking merges with unclear failures, it explained why a change was against policy and suggested the smallest fix. The comments on the pull request were very helpful because they fit in with the developers' normal way of working. Less cognitive strain was another benefit: developers didn't have to remember cloud-specific rules because the agent turned them into their actionable commands. Teams liked that "policy as code" was very clear and could be tracked over time, which made governance fair and predictable.
- Changes to the security and compliance team's workload: The compliance team saw a clear change from doing things that needed to be done to being in charge of other people. The time spent on investigating these breaches and acquiring evidence went down, while the time spent on policy refinement and escalation review went up. This was a good trade: instead of becoming ticket routers, they became policy owners. The agent also cut down on the number of routine questions that were sent to them, including "Is this allowed?" Developers could go straight to the agent for a policy-based answer.
- Comments on how ready the audit is: The ongoing evidence trail was well received by audit stakeholders. The agent kept track of detections, corrective actions, policy decisions & approvals on its own. Instead of rushing to get screenshots and export files at the last minute, audit packages might be made on demand. Auditors really liked how easy it was to follow the steps from policy to detection, correction, and validation. The one constant advice was that explanations should be understandable to people, not just serve as machine logs. We improved this by adding narrative summaries.

5.3. Trade-offs and Limits

- Freedom versus danger: Total autonomy is very powerful, but it doesn't always work. We found that autonomy works best for "clear, repeatable" violations. In any business setting, like when a marketing campaign temporarily makes a company public, autonomous cleanup could be bad. The best balance was tiered autonomy: automatic fixes for rules that people are sure about, pull request suggestions for rules that people are not sure about, and human approvals for rules that are very sensitive.
- Policy that isn't clear: Some policies are not mutually exclusive. The terms "least privilege," "reasonable retention," and "approved region" may be ambiguous based on their project requirements. The agent experienced trouble when policies didn't have clear definitions. This isn't a problem with the model; it's a problem with governance. When rules are unclear, enforcement isn't consistent. We fixed the problem by adding policy metadata, such as confidence weights, exception patterns, and text that explains why the policy is needed.
- Inconsistencies in multi-cloud APIs: Diverse clouds display equivalent controls through varied expressions. Some APIs send more complete signals than others, and the language used is different. This resulted in intermittent discrepancies; for instance, one cloud provides detailed encryption status, whereas another reports solely at the service level. The agent needed adapters and fallback verifications that were specific to the cloud, which made things more complicated from a technological point of view.
- Boundaries of LLM/agent dependability: The LLM did a great job of explaining, summarizing along with their providing remedies, but it shouldn't be thought of as perfect. Uncommon hallucinations manifested when telemetry was deficient. Guardrails were very helpful because they made the agent look at the precise resource status and policy provision before doing anything. It got worse when it couldn't do that. The prohibition against conjecture was crucial.

5.4. Discourse and Perspectives

- What worked best and why it worked: The greatest robust victories were achieved by merging deterministic engines with LLM reasoning. Traditional policy evaluations made binary decisions, but the LLM took care of contextual factors by

explaining differences, writing pull requests & giving developers advice. A further significant aspect was the incorporation of governance into regular workflows. The agent's integration within CI/CD and PRs led teams to consider compliance as an integral component of their workflow rather than an added effort. In the end, feedback loops were really important. The agent quickly moved forward when teams were able to routinely mark false positives and give permission for exceptions.

- In which human-in-the-loop is still very important: People are needed in three areas: (1) making policies, since only people can set their corporate goals and risk levels; (2) making exceptions, when the situation is more important than the rules; and (3) major fixes, like IAM reconfiguration or beginning a production rollback. The agent can navigate the lane; yet, humans retain the authority to select the destination and utilize the brakes as necessary.
- The pilot demonstrates that autonomous DevSecOps governance is achievable & useful, provided that autonomy is specified, policies are explicit, and human oversight is maintained for complicated, contextual nuances.

6. Conclusion and Future Scope

6.1. Conclusion

Modern multi-cloud digital experience platforms (DXPs) are moving very quickly, but their governance and compliance tools usually don't keep up. In cloud settings, teams have to deal with different regulations, broken logs & manual security checks. This gap is quite dangerous since it can lead to misconfigurations, audits that are hard to deal with, and DevSecOps that is slowed down by the need for tools as well as approvals. The basic question our work tries to answer is very simple: how can you keep multi-cloud Digital Experience Platforms compliant all the time without slowing down delivery?

The Autonomous DevSecOps Agent covers this gap by being a continuous governance layer that is built right into the delivery pipeline. The agent keeps an eye on infrastructure, apps, identities & data flows across these clouds all the time. It compares them to set policies and takes action right away, instead of looking back at their compliance. It finds drift, warns about dangerous changes before they go live, and can automatically put up guardrails through regulated remediation. In short, governance changes from occasional human involvement to a process that is always happening with the help of machines.

The agent showed big gains in both engineering & compliance teams for all of the evaluation scenarios. Policy violations were found earlier in the lifecycle, which cut down on the time it took to find and fix misconfigurations. Automated evidence collection and audit trail creation made it easier to comply with rules and get ready for an audit. Consistent policy enforcement across cloud environments reduced configuration differences & lowered the chance of hidden "one-cloud-only" security holes. These improvements were made without slowing down delivery, which is important because enforcement happens through pipeline-native verifications as well as efficient autonomous actions instead of late-stage gatekeeping.

The results show exactly what the paper adds: a unified autonomous governance framework for multi-cloud digital experience platforms, a policy-to-action reasoning loop that quickly stops compliance drift & a clear, measurable compliance workflow that fits with actual DevSecOps practices. The agent shows how compliance may be a smooth part of software delivery instead of always getting in the way.

6.2. Future Scope

This idea has a lot of room for growth. One alternative way to do this is to use self-evolving policy sets, where the agent uses reinforcement signals from events, audit results, or engineering feedback to improve their rules over time. Also, formal verification of agent operations makes sure that each automated remedy may be quantitatively checked for safety and accuracy before it is carried out.

Future iterations should broaden standards coverage beyond the existing baseline controls. This includes laws about where information can be stored across regions, rules for specific sectors, and the latest AI governance frameworks that are quickly becoming necessary for Digital Experience Platforms (DXPs). In the end, collaboration between several agents may lead to better optimization. For example, a security agent working with cost, reliability & performance agents to find the best balance between these factors can make sure that the platform is not only compliant but also efficient and resilient by design.

References

[1] Hsu, Tony Hsiang-Chih. *Hands-On Security in DevOps: Ensure continuous security, deployment, and delivery with DevSecOps*. Packt Publishing Ltd, 2018.

[2] Tan, Junsheng. "Ensuring component dependencies and facilitating documentation by applying Open Policy Agent in a DevSecOps cloud environment." (2022).

[3] Michael, Lee. "Integrating Continuous Security Validation into DevSecOps Pipelines for Regulatory Compliance." (2019).
[4] SOLANKE, ADEDAMOLA ABIODUN. "Enterprise DevSecOps: Integrating security into CI/CD pipelines for regulated industries." (2022).
[5] Parakala, Adityamallikarjunkumar. "Integrating Salesforce and UiPath: Cross-System Intelligent Automation." *International Journal of Emerging Trends in Computer Science and Information Technology* 3.4 (2022): 88-99.
[6] Smith, Jordan, and Dash Karan. "MLOps-Enabled DevSecOps: Automating Compliance and Risk Mitigation in Agile Workflows." (2020).
[7] Tiensuu, Tuomas. "DevSecOps adoption: Improving visibility in application security." (2022).
[8] Gopireddy, Satheesh Reddy. "Automated Compliance as Code for Multi-Jurisdictional Cloud Deployments." *European Journal of Advances in Engineering and Technology* 7.11 (2020): 104-108.
[9] Chandramouli, Ramaswamy. "Implementation of devsecops for a microservices-based application with service mesh." *NIST Special Publication* 800 (2022): 204C.
[10] Koskinen, Anna. "DevSecOps: building security into the core of DevOps." (2019).
[11] Parakala, Adityamallikarjunkumar, and Jyothirmay Swain. "AI-Powered Intelligent Automation Emerges." *International Journal of Artificial Intelligence, Data Science, and Machine Learning* 3.4 (2022): 96-106.
[12] Wei, Liu. "AI-Augmented DevSecOps Pipelines: Enabling Continuous Security Integration in Large-Scale Software System." *American International Journal of Computer Science and Technology* 1.5 (2019): 1-9.
[13] Tortoriello, Valentina. *Definition of a DevSecOps Operating Model for software development in a large Enterprise*. Diss. Politecnico di Torino, 2022.
[14] Santos, Maria Eduarda Oliveira. "Responsible Software Development Framework for Cloud-Native Financial Applications: Leveraging Safe Reinforcement Learning and Ethical AI Governance." *International Journal of Research Publications in Engineering, Technology and Management (IJRPETM)* 4.6 (2021): 5905-5908.
[15] Gowda, Harish Govinda. "Optimizing software delivery with event-driven DevSecOps pipelines in AWS and GCP." *International Journal of Science, Engineering and Technology* 8.6 (2020): 1.
[16] Allam, Hitesh. "Security-Driven Pipelines: Embedding DevSecOps into CI/CD Workflows." *International Journal of Emerging Trends in Computer Science and Information Technology* 3.1 (2022): 86-97.
[17] Basu, Ganapathi. "Linux & unix System Administration AI-Augmented Troubleshooting in Multi-OS unix Environments." (2021).