Generative DevOps AI System with In-Account Micro-Agents and Governance-Aware Automation

Field of the Invention

This invention relates to cloud computing automation and DevOps systems. In particular, it concerns a **generative AI-based DevOps system** that operates entirely within a user's cloud account while enforcing governance, security, and compliance guardrails in real time.

Background of the Invention

Modern organizations are increasingly adopting **generative artificial intelligence (AI)** to automate DevOps tasks such as infrastructure provisioning, code generation, and configuration management. AI coding assistants and autonomous agents can dramatically accelerate development and operations, but they also introduce new challenges in **security, compliance, and cost management**. Research shows nearly *half of AI-generated code is insecure* due to models reproducing flawed patterns[1]. For example, AI-generated Infrastructure-as-Code (IaC) often passes syntax checks yet fails security audits – one study found only *9% of AI-generated cloud infrastructure code met compliance standards*, with common issues including missing mandatory tags, unencrypted data storage, and overly permissive IAM policies[2][3]. In one case, dozens of Terraform modules generated by AI contained an average of 47 security findings each – e.g. S3 buckets without encryption, Lambda functions with wildcard (*"*") *permissions, and zero enforcement of organizational tagging conventions[3]. This highlights that* speed without guardrails* can create compounding technical debt[2].

Another trend is the use of **reinforcement learning (RL)** to optimize cloud operations for cost and performance. RL can dynamically learn how to scale resources or tune configurations to achieve *cost-effective, energy-efficient cloud deployments while maximizing performance*[4]. However, a *purely performance-driven RL agent may ignore regulatory or policy constraints*, inadvertently exposing organizations to compliance risks[5]. For instance, an RL system might choose an action that reduces cost or latency but violates a security policy or regulatory requirement (e.g. deploying an unapproved resource type or in an unallowed region)[6]. Traditional cloud automation tools do not simultaneously account for **cost, performance, and compliance risk** in their decision-making logic.

Moreover, most generative DevOps AI solutions today rely on external SaaS or public cloud services (e.g. calling a public API of a large language model), which can raise **data governance and security concerns**. Many enterprises in regulated industries require that sensitive data and AI operations remain within a controlled environment[7]. There is strong demand for mechanisms to *establish a data perimeter* so that generative AI can operate without sending proprietary data over the public internet[8]. For example, Amazon's Bedrock service now supports private VPC endpoints using AWS PrivateLink, allowing foundation model API calls to

remain entirely within the customer's VPC and not traverse public networks[9]. This reduces the attack surface and helps satisfy compliance requirements that data stay in-network[10]. However, simply hosting an AI model privately is not enough; the AI **agents executing actions** must also be governed. If an AI agent is granted excessive cloud permissions or acts without oversight, it could unintentionally modify critical infrastructure or access sensitive data. Best practices for AI in operations therefore dictate **least-privilege access**, rigorous monitoring, and real-time policy enforcement for any autonomous agent[11][12].

Finally, as multiple teams or accounts deploy such AI DevOps systems, there is an opportunity for **federated learning** to continuously improve the AI models. Each cloud account's AI agent could learn from its own environment (e.g. which optimizations worked, which actions were policy-compliant) and share insights with others *without revealing sensitive data*. Federated machine learning techniques (e.g. sharing sanitized model gradients instead of raw data) have been shown to enable collaborative training under privacy constraints[13]. For instance, agents can exchange **differentially private parameter updates** that allow collective improvement of policies while preserving data confidentiality[14][13]. Such approaches can maintain high performance and convergence even when each participant's data (and compliance context) stays local[15]. To date, however, no integrated solution exists that combines *in-account generative DevOps agents, multi-objective RL optimization (cost–performance–compliance), real-time policy guardrails, and federated continual learning*. There is a need for a comprehensive system that addresses all these aspects to enable safe, efficient, and compliant AI-driven automation in cloud operations.

Summary of the Invention

Overview: The present invention provides a **governance-aware generative DevOps AI system** that operates within a user's cloud account and introduces multiple novel components to ensure safe and optimal automation. In one aspect, the system comprises: *(1)* **Generative AI micro-agents** deployed entirely inside the user's cloud environment (e.g. within an AWS account) and running under least-privilege IAM roles; *(2)* a **Tri-Vector Reinforcement Learning (RL) optimizer** that evaluates potential actions along three dimensions – cost, performance, and compliance risk – to choose actions that best balance these sometimes competing objectives; *(3)* a dynamic **"Policy Weaver" guardrail layer** that intercepts and analyzes proposed infrastructure-as-code (IaC) changes or other actions **before** deployment, enforcing organizational policies (IAM policies, service control policies, tagging requirements, budget limits, config rules, etc.) in real time; and *(4)* a **federated continual learning framework** that allows multiple instances of the system (e.g. in different accounts or regions) to improve their models collaboratively via encrypted parameter sharing and privacy-preserving aggregation.

Some key features and advantages of the invention include:

- **In-Account Generative Agents with Least Privilege:** Each AI *micro-agent* (a lightweight specialized agent) runs within the user's own cloud account or VPC, rather than as an external service. This ensures all actions (and data) stay inside the trusted boundary[9]. The agents assume dedicated IAM roles granting *only the minimum permissions required* for their tasks, drastically limiting the blast radius of any unintended actions[11]. For example, an agent that manages S3 backups might only have

access to specific S3 buckets and nothing more. By keeping agents **resident in-account** and **sandboxed by least privilege**, the system aligns with zero-trust security principles for AI operations.

- **Tri-Vector RL Optimization (Cost, Performance, Compliance):** The system's decision engine uses a novel RL-based approach that optimizes a *multi-component reward function*. Unlike conventional cloud autoscalers or cost optimizers, this *Tri-Vector* RL considers three simultaneous goals: minimizing cost, maximizing performance, and minimizing compliance risk. Each potential action (e.g. deploying a server, modifying a security group, refactoring code) is evaluated for its predicted impact on cost (e.g. $$ spend, resource usage), performance (e.g. latency, throughput improvements), and a **compliance risk score**. The RL policy is trained (initially via simulations and continuously in production) to favor actions that improve performance and efficiency *without* incurring undue compliance risk. Compliance risk may be quantified, for instance, by penalty signals if an action violates or even approaches a guardrail (like deploying an untagged resource or increasing attack surface). This tri-objective optimization leads to more **balanced decisions** – e.g. the agent might choose a slightly less cost-efficient solution if it significantly reduces security risk. By incorporating compliance into the reward function, the system *proactively avoids* high-risk actions (a departure from typical single-metric RL)[5].

- **Real-Time Policy Weaver Guardrails:** A core component, referred to as the *Policy Weaver*, acts as a real-time compliance and governance filter interwoven with the agents' workflow. Whenever a micro-agent generates proposed changes (for example, a Terraform script to create infrastructure, or an API call to modify a configuration), the Policy Weaver intercepts it **prior to execution**. It parses and analyzes the change against a set of **machine-readable guardrail policies** (which can be provided as code, rules, or AI models themselves). These guardrails cover organizational and regulatory requirements: **Identity & Access Management (IAM) policies**, **Service Control Policies (SCPs)** limiting actions, required resource **tagging conventions**, **budget thresholds**, **security baseline configurations** (AWS Config rules, etc.), and more. If the proposed change violates any rule – e.g. tries to create a resource without required tags, or grant an IAM role wildcard "*" permissions – *the Policy Weaver will block or adjust the change in real time. In some cases, it can auto-correct the proposal (for instance, injecting missing tags or tightening a policy) and feed it back to the agent. This dynamic "weaving" of policies into the agent's output ensures that AI-driven changes are compliant by construction. It shifts governance all the way to the point of code/action creation, rather than after-the-fact audits[16][17]. Essentially, the Policy Weaver serves a similar role to known* safe RL shields* – it guarantees that unsafe actions are filtered or projected to the nearest safe alternative before proceeding[18]. By providing immediate feedback and enforcement, it prevents the AI from ever deploying something that would create a security or compliance incident.

- **Federated and Continual Learning:** The system improves over time through built-in learning mechanisms. Each micro-agent and the RL optimizer can continuously train on data from its local environment (e.g. outcomes of past actions, feedback from the Policy Weaver, observed cost/performance metrics, etc.). Furthermore, when the system is

deployed in **multiple accounts or regions**, it uses a *federated learning* approach to share knowledge. Periodically, each instance will **encrypt its learned model updates** (such as gradient vectors or updated neural network weights) and send them to a central aggregator or peer-to-peer network. Using secure aggregation protocols[13], the updates are combined into a global model improvement that is then sent back to all agents. Crucially, the sharing is done **without exposing any raw sensitive data** from individual accounts[14]. For example, an agent might share a differentially private gradient that captures the essence of what it learned about a new efficient configuration, but an outside observer cannot decipher the specific infrastructure or data that produced that gradient[13]. Over time, this **federated continual learning** means that if one account's AI discovers a better way to, say, optimize storage costs under a compliance constraint, all accounts' AIs can benefit from that discovery. The models thus become more effective with each deployment, while *respecting data silo boundaries*. This collaborative learning also helps in scenarios where one account alone has limited training data; collectively the agents approach the performance of a centralized training, yet remain *privacy-preserving and compliant*[15].

In sum, the disclosed system provides a **comprehensive generative DevOps AI platform** that maximizes the benefits of AI automation (speed, efficiency, intelligent decision-making) *without sacrificing organizational control*. By keeping AI agents in-account with least privilege, using multi-objective RL to factor in compliance, weaving in policy guardrails in real time, and leveraging federated learning, the invention addresses the key pain points identified in current approaches. The system can autonomously manage cloud infrastructure and DevOps workflows – such as provisioning resources, deploying code, configuring services, optimizing CI/CD pipelines, or remediating incidents – while continuously **balancing cost, performance, and risk**. This yields an autonomous cloud operations agent that is **safe, cost-effective, high-performing, and auditably compliant**. The patent coverage is broad, spanning the core system as well as numerous embodiments, including plug-in module architectures, model retraining processes, and hybrid deployment options, as described below.

Brief Description of the Drawings

FIG. 1 illustrates an exemplary architecture of the generative DevOps AI system (100). The system's major components – including in-account micro-agents, RL optimizer, Policy Weaver, and federated learning coordinator – are depicted in a cloud account environment.

FIG. 2 is a schematic diagram showing the Tri-Vector Reinforcement Learning optimizer (200) in operation. It highlights how the RL agent evaluates actions against three metrics (cost 210, performance 220, compliance risk 230) and receives aggregated reward signals.

FIG. 3A depicts a sample guardrail enforcement scenario via the Policy Weaver (300). In this illustration, an AI-generated Infrastructure-as-Code change (310) is intercepted and checked against multiple policy rules before deployment. **FIG. 3B** zooms into a specific policy check (320) – for example, verifying that required resource tags are present and IAM permissions are not overly broad – and shows the action being allowed or blocked accordingly.

FIG. 4.1 shows an embodiment of federated learning across multiple cloud accounts. Instances of the system in Account A (401) and Account B (402) train locally and periodically send

encrypted model updates (411, 412) to a central aggregator (420). **FIG. 4.2** illustrates the privacy-preserving aggregation process (430) and distribution of the updated global model back to the account-specific agents.

FIG. 5 illustrates a plug-in architecture variant of the system (500). The core orchestration module (510) loads various micro-agent plug-ins (521, 522, 523) and guardrail plug-ins (531, 532) that extend the system's capabilities (for example, adding support for new cloud services or custom compliance checks).

(Note: These figures are provided for illustrative purposes in describing embodiments and may not be drawn to scale. Like-numbered elements refer to the same or similar components in different figures.)

Detailed Description of Embodiments

System Overview (FIG. 1)

Referring to **FIG. 1**, the generative DevOps AI system **100** is deployed within a cloud computing environment **105** (for example, an Amazon Web Services account, Microsoft Azure subscription, Google Cloud project, or a private cloud). The system comprises several cooperating components: one or more **micro-agents 110** that perform domain-specific DevOps tasks, a central **orchestrator/optimizer 120** that coordinates agent decisions using reinforcement learning, a **Policy Weaver module 130** that enforces guardrails, and a **federated learning coordinator 140** for cross-account model training. These components execute on computing instances or serverless functions (e.g., AWS Lambda) inside the user's environment. All interactions with cloud provider APIs **150** (such as AWS SDK calls to manage resources **151** or fetch metrics **152**) occur via endpoints within the account/VPC, optionally through a **private AI model endpoint 160** for any foundation model inference.

Generative AI Micro-Agents 110: Each micro-agent is a specialized AI routine designed to handle a particular set of tasks – for example, a "Database Tuner" agent to optimize database configurations, an "Infra Deployer" agent to generate Terraform or CloudFormation scripts from high-level intents, or a "Compliance Remediator" agent to auto-fix security group settings. These agents are powered by generative AI models (such as large language models or code generation models) which may be hosted on **private endpoint 160**. In one embodiment, the system utilizes an AWS Bedrock foundation model deployed via a VPC interface endpoint (i.e., AWS PrivateLink), ensuring that all prompts and completions between agents and the model stay within the account's private network[9]. This guarantees sensitive input (like infrastructure state or config data) and output (like recommended code) are not exposed to the public internet. Each micro-agent runs under a distinct IAM role **112** configured with **least-privilege permissions**[12]. For instance, the Database Tuner agent might only be allowed to describe and modify the specific databases it manages, and nothing else. This granular privilege segregation means even if an agent behaves unexpectedly, it cannot affect resources outside its narrow scope. The system may automate the generation of such IAM policies – e.g. using templates or an AI "Privilege Guard" tool – to ensure minimal access for each agent[19][11]. The micro-agents are typically event-driven or goal-driven. They receive high-level goals or triggers (for example, a request to "provision a new QA environment" or a CloudWatch alarm indicating an application slowdown), and then they autonomously figure out the steps to achieve the goal, consulting the generative

model for planning or code generation as needed. Importantly, they do not execute any cloud-altering actions directly; instead, they pass proposed actions or changes to orchestrator 120 for evaluation.

Tri-Vector RL Orchestrator 120: At the heart of the system is an orchestrator or decision engine that uses **reinforcement learning** to evaluate and approve agent actions. This component can be thought of as a "governance-aware brain" overseeing the micro-agents. It maintains an internal state representation **122** of the environment (e.g., current infrastructure metrics, budget utilization, known compliance risk levels, etc.) and uses a policy network (or other RL model) **124** to choose the next action that an agent should perform. The novel aspect is the **multi-objective reward mechanism 125**. Rather than optimizing a single metric, the orchestrator considers **three reward vectors**: **cost efficiency 125a**, **performance impact 125b**, and **compliance/safety 125c**. Each time an action is taken and results observed, the system computes feedback in these three dimensions. For example, if the Infra Deployer agent proposes deploying two new servers, the orchestrator might simulate or predict: the cost impact (e.g. +\$100/month), the performance improvement (e.g. 20% faster response under load), and the compliance risk (e.g. a low risk if using approved AMIs in allowed regions, or a high risk if any policy violation is involved). These feedback signals are combined (e.g. as a weighted sum or via a multi-objective RL algorithm) to update the policy. The orchestrator's goal is to maximize an overall reward that captures *high performance and low cost, without compliance violations*. One way to implement this is via a **constrained Markov decision process (CMDP)** or a custom reward shaping: e.g., +10 reward for each percent of performance gain, –5 for each \$ of cost, and a large –1000 penalty for any compliance rule violation, ensuring such violating actions are strongly discouraged. Over time, the RL agent **124** learns policies that reflect efficient yet compliant operations. Notably, the compliance aspect may involve predicting the *likelihood or severity of a violation* even before it happens. The orchestrator may utilize a **risk predictor** model (trained on historical incidents or rule breaches) to evaluate an action's compliance risk as part of the reward calculation. This integrated approach is a significant improvement over prior cloud RL optimizers that ignore compliance[5]. If no viable action can improve one objective without harming another beyond acceptable trade-off, the orchestrator may decide to take no action (or request human review), thereby erring on the side of safety. The system can be configured with organization-specific preferences, for example to bias the RL more strongly against compliance risk in highly regulated environments, effectively treating compliance as a hard constraint (unsafe actions simply *never* chosen, similar to how a **Shield module** in safe RL will override actions outside allowed bounds[18]).

Policy Weaver Module 130: This component acts as the execution gatekeeper, deeply integrated ("woven") into the agent workflow. As shown in FIG. 3A, whenever orchestrator 120 approves an action plan from an agent, that plan is fed into the Policy Weaver **130** before actual deployment. The Policy Weaver comprises a set of **policy rules and models 132** and a real-time evaluation engine **134**. The policies 132 can be written in a *policy-as-code* language (e.g., Open Policy Agent Rego rules, AWS Config Rules, or Terraform Sentinel policies) and/or defined via machine-learning classifiers for more complex conditions. They cover multiple domains of governance:

- **Identity & Access Policies:** e.g., "No IAM role can have * admin privileges," "All new IAM roles must include MFA enforcement," or "Deny any policy that grants S3:GetObject on buckets with PII unless condition X is present." These prevent the AI

from creating overly permissive access configurations. (See FIG. 3B, element 320, illustrating detection of a wildcard Action:* in a generated IAM policy – a guardrail forces least privilege instead[20][21].)

- **Resource Tagging and Configuration:** e.g., "All EC2 instances must have Environment and Owner tags,"[3] "S3 buckets must have encryption enabled,"[22] "No public IP addresses on databases," etc. These ensure that even if the generative agent wasn't aware of internal conventions, any output is retrofitted with required organizational context and security settings before launch. (For instance, if an agent creates a new S3 bucket and forgets to specify encryption, the Policy Weaver will catch this and either auto-enable encryption or block the deployment[22].)

- **Budget and Quota Checks:** e.g., "Prevent launching of resources if monthly budget for project X is exceeded," or "Do not allow more than N of instance type Y in region Z." This ties into the cost vector: even if RL suggests adding resources for performance, a hard budget guardrail can veto the action, prompting the agent to find alternatives (like optimizing existing resources instead).

- **Service Control and Compliance Rules:** e.g., "Disallow deployment of services not on the approved list for this account," (perhaps an SCP at the AWS Organizations level), or "All data storage must be in us-east-1 or us-west-2 regions" (for data residency compliance). The Policy Weaver has access to such organizational policies and can simulate the effect of the proposed changes against them. If a change would be blocked at a higher level (say by an SCP), the Policy Weaver flags it immediately, saving time and preventing policy breaches.

- **Security Baselines and Best Practices:** e.g., checking generated code against static analysis or known security linting tools. The Policy Weaver might incorporate scanners (like HashiCorp Terraform checkov, Docker image scans, etc.) as part of its pipeline. It can thus catch issues such as using outdated images with CVEs, missing vulnerability patches in scripts, or known anti-patterns.

The Policy Weaver engine **134** can operate in multiple modes. In a strict mode, any violation causes the proposed action to be rejected; the system may log the incident or ask for human approval. In a permissive mode, the Weaver could automatically remediate certain issues (e.g. append missing tags or adjust IAM policies to be narrower) and then allow the action to proceed. In all cases, it provides immediate feedback to the generative agent and the orchestrator. This feedback loop means the RL optimizer also learns from the Policy Weaver: if many proposed actions in a certain direction get blocked, the RL will receive poor reward for those, reinforcing the policy-compliant behavior. Over time, the micro-agents and the RL policy effectively **internalize the guardrails**, generating fewer violating proposals. This approach implements in real-time what historically was done through manual code reviews or CI/CD pipeline checks. By shifting governance "further left" to the moment of creation, the system greatly reduces the latency and friction of compliance checks[16][17]. It becomes feasible to trust AI automation in production because every step is under watch. As an example outcome: using such guardrails, a company MyCoCo was able to reduce security findings from 47 to just 3 per Terraform module, while still retaining ~70% of the velocity gains from AI[23]. This underscores that automated policy enforcement can effectively bridge the gap between what AI generates and what the

organization requires[20] – the Policy Weaver is designed to do exactly that, serving as the "immune system" of the AI.

Federated Learning and Cross-Account Collaboration (FIG. 4.1 & 4.2)

The invention further provides mechanisms for **federated learning** across different deployments to continuously improve the underlying AI models (both the generative models powering micro-agents and the RL policy models). Consider an enterprise with multiple cloud accounts (dev, test, prod, or per business unit), or a SaaS provider deploying the system for multiple client organizations. Each instance of the system operates with its own data and within its own policies. Directly sharing raw data (like configuration states or usage metrics) between accounts may be disallowed due to privacy and regulatory reasons. Instead, as shown in FIG. 4.1, each account's orchestrator **120** and micro-agents **110** locally log their experiences and periodically train **local model updates 411** (e.g., fine-tuning the foundation model on domain-specific terminology, or improving the RL value function on recent cost-performance outcomes). These updates are then sent to a **federated aggregator 420**, which could be a central server or a decentralized coordination service. To preserve privacy, the system applies **encryption and secure aggregation** techniques: for example, each account could encrypt its model gradient with a key such that only the aggregator can sum the gradients but not read individual ones, or use differential privacy to add noise that masks individual contributions[13]. As indicated in FIG. 4.2, the aggregator combines the contributions (step 430) – e.g., averaging the weight updates from all accounts – resulting in an improved global model. This aggregated model (now incorporating learnings from all participating environments) is then sent back to each instance as **updated model parameters 412**. Because the exchange was of parameters and not raw data, no sensitive configuration or proprietary code left any account in intelligible form[14][24]. Each local system then continues operation with a smarter model.

This federated loop can occur on a fixed schedule (say nightly or weekly) or be triggered by certain events (e.g., when a significant improvement is achieved by one node). The system may use techniques from Federated Averaging, Federated RL, or split learning as appropriate. In some embodiments, the system employs a **privacy budget** – ensuring that over time, the amount of information that could potentially leak via parameter sharing remains below a threshold (for compliance with things like GDPR, HIPAA if applicable). The result is a network of AI DevOps agents that **learn collectively**: for instance, if one account's agent learns a better way to tune a Kubernetes cluster for cost savings under policy constraints, all other accounts' agents will soon adopt that method, *without any direct exchange of the cluster's data*. Prior research has validated that such federated approaches can maintain performance while enforcing regional compliance and data locality[25][15], which this invention leverages in the DevOps context.

Additionally, the federated learning coordinator **140** can facilitate **multi-party governance**: different organizations might agree on certain shared model aspects (like general knowledge of cloud best practices), and yet keep specific rules private. The aggregator can use secure multi-party computation such that each participant only learns the final combined model and not any other's contributions. Optionally, a *federated model registry* can track versions and allow each participant to vet updates (e.g., via a hash or signature) before accepting them, providing assurance against poisoning attacks in the federated process. This approach transforms the system into a **network-effect platform** – the more it is used across environments, the smarter and more compliant it becomes for all.

Plug-In Modular Architecture (FIG. 5)

In one set of embodiments, the architecture is designed to be highly modular and extensible via a **plug-in framework** (see FIG. 5). The core system (510) can dynamically load or integrate additional **micro-agent modules 521–523** and **policy modules 531–533** as plug-ins. For example, an organization might develop a custom agent for a niche task (521) – say, managing a legacy on-prem resource via a hybrid cloud connector – and plug it into the system without altering the core. Each micro-agent plug-in implements a defined interface so the orchestrator 120 can interact with it (e.g., send it tasks, get action proposals). Similarly, the Policy Weaver 130 can accept **guardrail plug-ins** that add new rules or compliance checks. An example might be a region-specific data sovereignty rule module (532) that a European deployment can add to enforce GDPR-related constraints. The plug-in architecture also allows integration of **external tools**: for instance, a security scanning plug-in might call out to an API of a static analysis tool, or a cost forecasting plug-in might embed a third-party cost simulation library. The system ensures that each plug-in (especially micro-agents) still runs with *least privilege* – e.g., assigning separate roles to each new agent plugin.

This plug-in approach future-proofs the platform by **covering extension pathways**. As cloud providers introduce new services or as new compliance requirements emerge, additional modules can be added to handle them, without redesigning the whole system. The claims cover such modular embodiments to secure exclusivity over the concept of a governance-aware AI DevOps platform that can be **extended via plugins** for tasks and policies.

Model Retraining and Continuous Improvement Flows

The invention encompasses various **model retraining flows** beyond the federated scenario already described. In one embodiment, the system supports **on-demand fine-tuning** of the generative AI model inside the account. For example, after the system has been in use for some time, an administrator might provide feedback or additional training data (such as exemplary infrastructure code that was approved or instances where the AI's suggestion was overridden by a human). The system can retrain or fine-tune the foundation model (or a smaller adapter model) on this data within the secure environment, thus making the model more aligned with the organization's needs (akin to Reinforcement Learning with Human Feedback, RLHF). Because this fine-tuning happens in-account on the organization's data, it remains private and compliant. Periodic retraining can also apply to the RL optimizer – e.g., resetting and retraining it with updated reward weightings if the business shifts priorities (say, performance becomes more critical than cost in a busy season, or new compliance rules are added which effectively alter the risk calculations).

Another retraining flow involves **simulation-based training**. The system can spin up a simulated cloud environment or use historical logs to train the RL agent faster than real-time. This is done in a controlled sandbox, possibly using generative models to simulate various what-if scenarios (like a surge in traffic, or a security event) and training the AI on how to respond optimally under the tri-objective criteria. Once validated, these learned strategies are deployed to the production agent. The patent covers such training pipelines, ensuring that any method of improving the AI's decision-making within the scope of cost-performance-compliance optimization is protected.

Hybrid Deployment Scenarios

While the primary embodiment runs entirely in a single cloud account, the architecture can also be deployed in **hybrid scenarios**. For instance, an organization might keep the **Policy Weaver** and sensitive guardrail checks on-premises (or in a highly secure account), while allowing micro-agents to operate in various cloud accounts. In this setup, agent proposals are sent to a central Policy Weaver service via secure channels; approvals or rejections are then returned. This might be useful if an enterprise wants a unified control plane for compliance that oversees multiple cloud accounts or even multi-cloud deployments (AWS, Azure, etc.) from one place. Another hybrid variation is where the generative model endpoint (160) resides in one environment (say, a specialized AI-centric cloud or an on-prem GPU server cluster), and the micro-agents operate in another environment, communicating over an encrypted link. The invention's claims cover such distributed deployments where components of the system span different network domains yet collectively function as described.

One notable hybrid case is **multi-cloud governance**: e.g., an organization uses AWS for some workloads and Azure for others, and deploys our AI system in each. A central orchestrator could be designated to coordinate or ensure policies are consistent across them. Federated learning in this case would occur across cloud platforms – our design anticipates abstracting the cloud-specific details so that knowledge can transfer (for example, an optimization learned in AWS about storage performance vs cost could inform an Azure agent's policy by analogy).

Security and Logging

Given the critical nature of infrastructure changes, the system implements extensive **logging and oversight** capabilities. Every action taken by a micro-agent, every decision by the RL optimizer, and every intervention by the Policy Weaver is logged in a tamper-evident audit log **170** (FIG. 1). These logs can be reviewed by human operators or connected to a SIEM (Security Information and Event Management) system. The logs include the rationale for decisions (e.g., "Action X blocked by Policy Weaver due to missing tag 'Owner'"), which improves transparency of the AI's operations – often a requirement for AI governance. Optionally, the system can provide **explanations** for its suggestions (leveraging the generative model to justify why a particular change is proposed, including references to compliance or past successes). This helps build trust with DevOps engineers who may supervise the AI.

Security-hardening features are also built in. For example, **ephemeral credentials** can be used for agents (rotating IAM credentials frequently to reduce risk of compromise)[19]. The system can integrate with secret managers to securely fetch any secrets an agent needs (never exposing them in logs or to the model). Additionally, if any agent attempts an action outside its scope or if the RL agent were to produce an output sequence that tries to circumvent the guardrails, detection triggers can halt the agent and raise an alert. In essence, the system treats AI agents as **privileged identities** that must be subject to the same controls as human admins[26][11] – continuous monitoring, least privilege, and rigorous access control.

Example Use Case Workflow

To illustrate an end-to-end flow, consider a scenario: A developer requests via a chat interface, "Deploy a testing environment for Project X." The request is received by a micro-agent (say, an

Orchestration Agent) inside the account. The agent breaks down the task and uses a generative model to draft Infrastructure-as-Code for the environment (perhaps a Terraform module with an EC2 cluster, a database, etc.). Before doing anything, it submits this draft plan to the orchestrator **120**. The RL optimizer, using its policy, evaluates the plan: it predicts the environment will cost \$500/month (cost vector), improve team productivity by providing needed test capacity (performance vector), and poses low compliance risk since it uses standard templates (compliance vector). The RL approves moving forward, giving a positive reward internally. The plan is then handed to the Policy Weaver **130**, which runs its checks. The Policy Weaver finds that the generative agent omitted the corporate "CostCenter" tag on the resources – a violation of tagging policy. It automatically inserts the required tag (since this is a straightforward fix) and also notices the database storage is not encrypted, which violates a security rule. It modifies the configuration to enable encryption at rest. It then passes the corrected, compliant template back to the agent orchestrator. The orchestrator triggers the actual deployment (for instance, by pushing the Terraform plan to the infrastructure pipeline or calling cloud APIs to create the resources). The new test environment is created in compliance with all policies. The micro-agent logs a summary: "Deployed Project X test env with 3 EC2 instances, 1 RDS database (encrypted), tags applied." The RL optimizer gets feedback: the performance goal is met, cost is within budget, and compliance was achieved after minor adjustments (some small negative reward for initially missing tags, teaching the agent to include them next time). Meanwhile, this experience – deploying a tagged, encrypted environment – is captured as a training instance. That night, the federated aggregator collects gradients from this and other accounts; our agent's experience helps update the global model to more strongly emphasize encryption and tagging going forward.

This workflow shows how the components interplay to result in an outcome that satisfies the user's request *and* the organization's governance needs, with minimal human intervention.

Advantages Over Prior Approaches

Compared to naive AI DevOps implementations, this invention's in-account, governance-aware design offers multiple advantages. It eliminates exposure of sensitive cloud credentials and data to third-party AI services by keeping the AI **within the customer's environment**[9]. It actively prevents unsafe changes by integrating guardrails at creation-time rather than relying on after-the-fact detection[23][20]. It manages the often conflicting goals of cloud operations (cost vs performance vs security) using a principled learning approach, rather than static rules or simple cost optimizers[5]. And it continuously learns from collective usage, leading to an AI that gets smarter and more adapted to compliance requirements over time (addressing the common problem that AI assistants lack organizational context by infusing that context through federated learning and policy feedback). The architecture is flexible enough to adapt to various cloud platforms and evolving policies, thanks to its plugin and modular structure.

In various embodiments, the system can be offered as a managed service (with the control plane possibly hosted by a provider but with data staying in the user's account), or as on-premises software for maximum control. The claims that follow capture the breadth of the invention, including system and method aspects, as well as tangible computer-readable media embodying the software, and various extensions such as plugin modules, retraining workflows, and hybrid deployment options.

[1] [16] [17] From Gatekeeper to Guardrail: Embracing the Role of Governance for the AI Era | Snyk

https://snyk.io/de/articles/from-gatekeeper-to-guardrail-embracing-the-role-of-governance-for-the-ai-era/

[2] [3] [20] [21] [22] [23] [27] Guardrails for AI-Generated IaC: How MyCoCo Made Speed Sustainable | MYCLOUDCONDO

https://www.mycloudcondo.com/ai-iac-guardrails.html

[4] Leveraging Reinforcement Learning to Optimize Cloud Resource Costs - Techstrong IT

https://techstrong.it/cloud/leveraging-reinforcement-learning-to-optimize-cloud-resource-costs/

[5] [6] [18] Safe and Compliant Cross-Market Trade Execution via Constrained RL and Zero-Knowledge Audits

https://arxiv.org/html/2510.04952v2

[7] [10] Secure AWS Bedrock Agents with PrivateLink: Implementation Patterns and Best Practices

https://www.axrail.ai/post/secure-aws-bedrock-agents-with-privatelink-implementation-patterns-and-best-practices

[8] [9] Use AWS PrivateLink to set up private access to Amazon Bedrock | Artificial Intelligence

https://aws.amazon.com/blogs/machine-learning/use-aws-privatelink-to-set-up-private-access-to-amazon-bedrock/

[11] [12] [26] When AI agents become admins: Rethinking privileged access in the age of AI

https://www.cyberark.com/resources/blog/when-ai-agents-become-admins-rethinking-privileged-access-in-the-age-of-ai

[13] [14] [15] [24] [25] journal.esrgroups.org

https://journal.esrgroups.org/jes/article/download/8781/5880/15962

[19] AI Agent Security Best Practices | Wiz

https://www.wiz.io/academy/ai-agent-security