



A Generative AI Framework for Resilient DevOps in Cloud-Native Fintech

Luca Bianchi

Asst Professor

Abstract

Scalability is a critical non-functional requirement for cloud-native financial platforms. Open banking and fintech ecosystems built on such platforms must also satisfy extensive requirements around risk management, security, and compliance with confidentiality and privacy standards. Meeting these requirements demands a DevOps approach that ensures appropriate governance, monitoring, verification, validation, and testing throughout the system lifecycle — needs that vary not only between development and operations phases but also across architectural layers of the platform.

Generative AI presents significant opportunities to automate DevOps practices in cloud-native environments. Building on Generative AI techniques, the proposed reference architecture enables intelligent automation of key DevOps activities within defined boundaries, supported by continuous learning and improvement loops. Automation scenarios include policy-driven provisioning and configuration aligned with internal standards and external regulations; intelligent incident response and remediation for rapid recovery with minimal human intervention; and AI-assisted testing and verification that use synthetic data to expand coverage in underrepresented areas.

Keywords : Generative AI, DevOps, Automation, Cloud-Native, Financial Services, Governance, Data Privacy, Explainability.

1. Introduction

The emergence of Generative AI presents an opportunity to radically rethink the way complex systems operate. Enabling appropriate levels of automation of routine workloads and minimizing human involvement could lead to better, faster, and cheaper services. Models supporting such automation need to have built-in mechanisms for governance to provide controls and assurance to key stakeholders. Of course, the financial services industry is under pressure to deliver new technology at speed but with minimal risk, while demonstrating that it can operate in a safe, secure and controlled way. The application of DevOps and MLOps has been particularly useful in accelerating deployment velocities. However, financial institutions are frequently under-resourced, with manual processes causing bottlenecks at all stages of the pipeline. The integration of Generative AI into the DevOps pipeline, therefore, allows for greater levels of intelligent automation of operational tasks.

Generative AI-Enabled DevOps maps the DevOps lifecycle for the financial services industry, scoped to consider the areas of Risk, Security and Compliance. Building on this foundation, the thesis demonstrates how the DevOps pipeline can be intelligently automated with appropriate levels of governance and controls. AI and Automation have become synonymous with the promise of better, faster, cheaper services. For DevOps, reducing manual effort across the pipeline—and thereby minimizing the risk of human error—offers the best opportunity to deliver the required outcomes. New and emerging AI models can reduce effort and increase controls across the DevOps process in a variety of areas, but adoption must be done prudently, with risk and compliance policies in mind. Cloud-native architectures provide the technological capabilities that support this transformation.



1.1. Problem Statement

The research problem emerges against the backdrop of increasing DevOps adoption by financial institutions. While granting cloud-native development and deployment speed and flexibility, the approach introduces new risks related to security, compliance, and data protection. As a reaction, the deployment of governance mechanisms, segregation of responsibilities, and control mechanisms requires heightened coordination to protect these cloud-native assets.

Such considerations outline the need for a multi-layered Risk and Compliance DevOps reference architecture capable of reliably delivering financial-grade cloud-native applications and services. Furthermore, the adoption of a well-defined set of Automated Infrastructure and Application Assurance as Code practices guarantees the collation of sufficient evidence along the development and operation phases of the software development lifecycle to demonstrate compliance with the required levels of risk, security, and internal and external controls. Integrating these support and assurance activities into the CI/CD pipelines goes beyond merely generating the required evidence and allows for intelligent testing and verification using Synthetic Data.

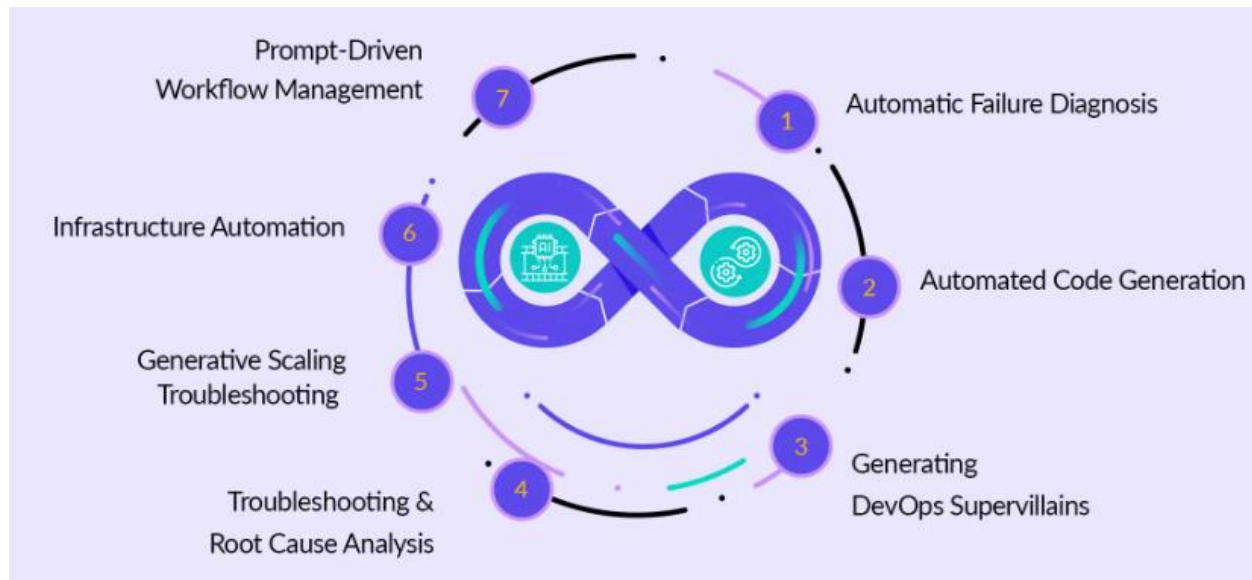


Fig 1: DevOps Automation

1.2. Scope and Definitions

Generative AI-Enabled DevOps automation for scalable cloud-native financial platforms is framed as a research question, followed by motivation and context. The annotated outline serves as a navigational aid. Covering DevOps for financial services, Generative AI, automation, and cloud-native architectures, it focuses on guiding automated deployment, quality assurance, risk management, and incident response of software delivery pipelines across the GitOps model. As Generative AI rapidly evolves, associated risks must be actively surveilled.



Market pressure to deliver applications and services is at an all-time high. Cloud-native architectures with a microservices structure offer scalable solutions deployed and managed like consumer apps. Nevertheless, these systems also Face challenges. Open-source code repositories expose known vulnerabilities, regulators expect reconciliation of changes with risk and security requirements, and incidents like service outages prompt demands for a rethink of the deregulation of systemic financial infrastructure, systems and controlled environments. Generative AI-assisted software development promises to accelerate creation. However, the lack of monitoring together with the growing dependence on unseen black-box systems amplifies risk. Therefore, command-and-control structures are needed to ensure compliance and management of build and run environments commensurate with the capability deployed, thus ensuring user and provider risk remain measured and controlled. Cloud governance empowers the definition of such oversight within DevOps processes across the lifecycle of multiple microservice-based applications in an on-demand shared infrastructure-as-a-service context.

1.3. Relevance to Cloud-Native Financial Platforms

Recent technological advances enable the timely launch of new digital products without sufficient security, compliance, or operational quality. Consequently, organizations risk jeopardizing the trust of their customers, partners, and regulators. Regulatory expectations demand effective governance and controls for speed and business impact associated with digital innovation, yet crafting and enforcing such controls in a non-intrusive manner remains a challenge. Generative AI offers the potential to mitigate these opposing forces by enabling creative capabilities while embedding compliance mechanisms at the same time. Generative AI-Enabled DevOps automation applies to a wide range of digital services that support business innovation and deal-making. Adoption is of particular interest to financial services firms running cloud-native platforms for multiple partners and customers, enabling an open-banking ecosystem with fintech innovation. Furthermore, such platforms deliver sensitive data and services for use in adversarial systems, where partners and customers experiencing service outages may lose customers and revenue. DevOps at speed, scale, and business impact therefore depends on Automation and AI, with increasingly easy access to these fields. Third-party risks arising from an extensive and growing ecosystem are also top-of-mind for boards, with regulators expecting organisations to actively monitor the health of third-party relationships, changes in exposure, associated risks, and the controls put in place with third parties.

Equation 1: Deployment Frequency

Let

- N_d = total number of successful production deployments in an observation window
- T = length of the observation window in days

Then deployment frequency is

$$DF = \frac{N_d}{T}$$

Step-by-step derivation

1. Start from the meaning of “frequency”:



$$\text{frequency} = \frac{\text{number of occurrences}}{\text{time}}$$

2. Here, the occurrence is a production deployment:

$$\text{number of occurrences} = N_d$$

3. The time unit chosen by the paper is days:

$$\text{time} = T$$

4. Substitute:

$$DF = \frac{N_d}{T}$$

2. Theoretical Foundations

DevOps in Financial Services: Financial platforms have a very high rate of change with respect to code. The short release cycles embrace continual development but confront the challenges of proper testing, response to unauthorised access, and maintaining compliance with regulatory requirements at all stages. The DevOps process reflects these requirements across four key stages: code, build, test, and production, with testing as a fully-fledged stage preceding deployment. In every domain of the innovation, risk and policy framework requirements apply and are supported by specialised assurance and security teams responsible for testing processes and tools.

Generative AI, Automation, and Learning Loops: Generative AI in its various formulations and configurations includes Large Language Models (LLMs), Diffusion Models (DMs), and Generative Adversarial Networks (GANs). Systems can be prompted to perform specific tasks or create tailored content either through direct instruction or interaction history. Generative AI is often seen as a technology that enables development by ‘writing code’ or authoring natural language within a certain context and under certain vocabulary constraints. It also has many other applications of direct substance. In principle, all processes can be automated. A task is a candidate for automation when every single execution step, method, and decision is known. Contrast this with forms that must conform to personal demands that differ across consumers. Large-scale production of Cloud-Native Financial Apps: The Cloud-Native concept is best understood as a set of architectural patterns and principles governing the creation of an application that can be deployed on a Cloud service provider (CSP) platform. Cloud-Native applications embrace a microservices architecture, in which the full application consists of a set of independently deployable units that communicate over a service-bus, service-mesh , or similar mechanism.



DevOps stage	Main activities	Financial-platform concerns	GenAI-enabled automation role	Expected outcome
Code / Development	Code authoring, configuration, policy alignment, feature changes	Secure coding, compliance alignment, traceability, segregation of duties	Assist with code generation, policy-aware recommendations, prompt-guided developer support	Faster development with reduced manual error
Build	Compile, package, integrate dependencies, validate artifacts	Dependency risk, build integrity, vulnerability exposure, governance evidence	Generate validation scripts, automate build checks, identify anomalies in pipeline behavior	More reliable and policy-aligned build outputs
Test / Verification	Unit, integration, security, compliance, synthetic-data testing	Coverage sufficiency, non-production data usage, auditability, regulatory evidence	Generate test cases, synthesize safe test data, expand coverage criteria, analyze test results	Better quality assurance and reduced release delay

Table 1: DevOps lifecycle in cloud-native financial platforms

2.1. DevOps in Financial Services

Effective deployment of Banking, Financial Services, and Insurance (BFSI) applications is critical for a successful Digital Transformation of the financial sector. With growing user expectations for instantaneous services and response times, along with stringent regulatory demands and potential threat vectors, the DevOps strategy among organizations has evolved in financial services. Yet, the risk appetite relating to the automation of high-tempo CI/CD stages—development, test, and deploy—remains low. Barriers and concerns are present in areas such as policy compliance, security hygiene, and incident response of environments during operations. A curated Richard Stallman quote reveals such governance challenges for regulators and organizations alike in the race to embrace the new-age cloud-native development.

DevOps is a combination of software development (“Dev”) and IT operations (“Ops”). DevOps aims to help an organization produce software and IT services more rapidly, with frequent iterations and a reliable quality across the entire software development life cycle. In the financial industry, DevOps greatly accelerates the release and deployment, allows early risk detection and mitigation, meets increasing compliance requirements, increases quality, and introduces automation testing and compliance verification. Throughout the DevOps life cycle, risks associated with preventing, detecting, and responding to security and compliance issues are a matter of concern. Generative AI-enabled DevOps automation will create a cycle for risk quantification, detection, and incident response and remediation activities across the stages of build, development, and run of an organization.

2.2. Generative AI and Automation

DevOps practices are being considered in financial services and their supporting platforms – banks, consulting firms, and system integrators. Financial services are exposed to multiple risks, including operational risk. Operations typically deal with production



incidents, which also include failures of applications developed either internally or externally in the fintech ecosystem. Financial services must fulfill compliance and regulatory demands. The entire software engineering and operational life cycles are exposed to risks, including the following: architecture considerations; development (e.g., quality gaps, acceptance test failures); deployment (e.g., throttled releases, downtime); disaster recovery (e.g., data-backup failures); and security (e.g., data breaches, vulnerability to automated bots). Thus, applying DevOps practices while managing risks and compliance gaps is imperative.

Generative AI is an emerging area of research, fueled by different types of data—text, image, video, 3D models, and program code—and availability of multimodal and cross-modal pre-trained-language models. Recent developments in generative AI, such as ChatGPT, have popularized the use of large language models (LLMs) for different tasks, including creative writing and code generation. Generative AI and automation are often conflated due to the evolving nature of AI. Simply put, automation is the designing of systems, devices, or processes that minimize human interventions in production environments. When suitable Guardrails are in place, AI in general—more specifically generative AI—can be seen as another tool that facilitates automation.

Equation 2: Lead Time for Changes

Let for the i -th change:

- $t_{commit}^{(i)}$ = time of commit
- $t_{prod}^{(i)}$ = time that change reaches production

Then the lead time for one change is

$$LT_i = t_{prod}^{(i)} - t_{commit}^{(i)}$$

For n changes, average lead time is

$$\bar{LT} = \frac{1}{n} \sum_{i=1}^n (t_{prod}^{(i)} - t_{commit}^{(i)})$$

Step-by-step derivation

1. Time duration is always:

$$\text{duration} = \text{end time} - \text{start time}$$

2. For a software change:

- start time = commit time
- end time = production release time

3. So for one change:



$$LT_i = t_{prod}^{(i)} - t_{commit}^{(i)}$$

4. For multiple changes, take the arithmetic mean:

$$\bar{LT} = \frac{LT_1 + LT_2 + \dots + LT_n}{n}$$

5. Substituting each LT_i :

$$\bar{LT} = \frac{1}{n} \sum_{i=1}^n (t_{prod}^{(i)} - t_{commit}^{(i)})$$

2.3. Cloud-Native Architectures and Microservices

Cloud-native development shifts responsibility of app performance from the Dev team to the Operations team, with production services tested and monitored at scale. Modern enterprise application architecture uses microservices developed with agile principles. Microservices are small units that can be changed quickly with a very limited scope of impact. Microservices allow organizations to respond directly to business unit requests. A microservice infrastructure is developed using 12-factor principles and is hosted on a cloud platform.

A service mesh establishes service-to-service connections in a consistent way without requiring custom code in application services. The service mesh handles control plane functions, such as service discovery, routing, canary deployment, and enabling strategies for resiliency testing. The cloud provider's native pricing models leverage the multi-tenant nature of large-scale Cloud and PaaS services, paying only for usage rather than ownership.

The containers are deployed in containers orchestrated in rotation like a ferry service. The template-based constructs eliminate drift and promote consistent deployment of services at any location. Users can view the services within a service mesh as interfaces giving services the visibility of APIs. A monitoring and alerting service monitors the core application parameters and SLAs and stores them in a time-series service. The observability service uses resources from an observability platform that spans all teams and environments.

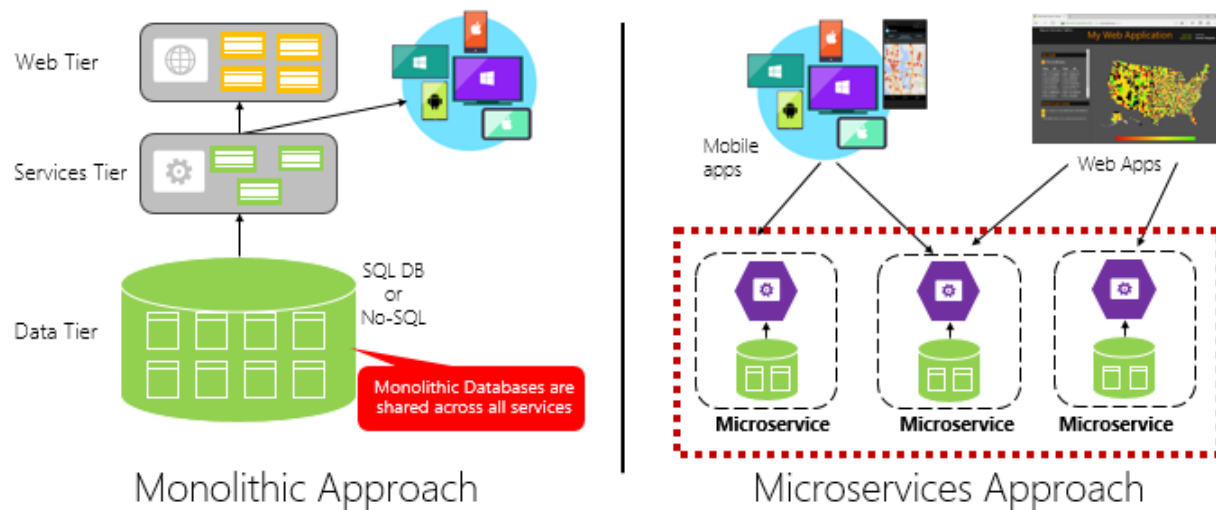


Fig 2: Cloud-Native Architectures

3. Reference Architecture

The reference architecture defines core components, data flows, and interaction patterns across development, operations, and security domains. Three groups of users are identified: application developers, operational engineers, and security and compliance officers. The Infrastructure as Code (IaC) and GitOps approaches for managing infrastructure and application environments are described, covering versioning, storage, and drifts. The Continuous Integration and Continuous Delivery (CI/CD) pipelines for enterprise-grade financial applications are outlined, including automated testing strategies, quality gates, and deployment patterns.

The reference architecture depicts the main constituents of the Estate Provisioning and Management, Continuous Compliance, and Continuous Delivery capabilities that enable the DevOps Automation, Control, and Security patterns. These patterns cluster the operational requirements into three areas of concern: Development (programming, testing, verification), Operations (availability, performance, security), and Security (protection, governance, conformity). Installation, production deployment, and compliance confirmation of a financial application consume services from all three areas. The three groups of users primarily concerned with these operations are application Developers, operational Engineers, and Security+Compliance officers. Communication is two-way between Developers and Engineers and one-way from Security+Compliance to the other two groups. Following GitOps terminology, the entire Git repo, including the IaC and model code/services, is termed the “Estate”.

3.1. Core Components and Data Flows

A cloud-native financial services application comprises a core application component, supported by a cloud-native SSL gateway



that handles TLS termination and security requirements. The application component comprises the application and its dependencies (including infrastructure, runtime, API gateway, etc.). The next layer—DevSecOps—automates security and policy validations throughout the application lifecycle. The outer layer automates the GitOps-based provisioning of the IaaS environment using Infrastructure as Code principles. Automated SQL deployments and infrastructure drift detection mechanisms ensure that production environments remain in a deployed state.

Several data interaction patterns exist among these components. Triggers embedded in the application component invoke tests defined in the DevSecOps layer using synthetic data, with the results forming input for a high-level health check that aggregates coverage results across testing and SLIs. Completed certificate signing requests initiate configurations for the respective Certificates service. When policy violations are detected in the application, these also trigger automated remediation calls to the Infrastructure layer. Cloud logging consolidates logs for these components, facilitating root-cause analysis in case of failure, while data pipelines enable archival data retention and consumption.

Equation 3: Deployment Duration

Let

- $t_{start}^{(i)}$ = deployment start time
- $t_{end}^{(i)}$ = deployment completion time

Then for deployment i :

$$DD_i = t_{end}^{(i)} - t_{start}^{(i)}$$

For average deployment duration over n deployments:

$$\overline{DD} = \frac{1}{n} \sum_{i=1}^n (t_{end}^{(i)} - t_{start}^{(i)})$$

Step-by-step derivation

1. Duration formula:

$$\text{duration} = \text{finish} - \text{start}$$

2. For a deployment:

$$DD_i = t_{end}^{(i)} - t_{start}^{(i)}$$

3. For multiple deployments, compute the mean:



$$\bar{DD} = \frac{DD_1 + DD_2 + \dots + DD_n}{n}$$

4. Substituting:

$$\bar{DD} = \frac{1}{n} \sum_{i=1}^n (t_{end}^{(i)} - t_{start}^{(i)})$$

3.2. Infrastructure as Code and GitOps

Two core components of DevOps automation are Infrastructure as Code (IaC) and GitOps. IaC enables automated resource management on cloud platforms through declarative specifications and controlling deployments in the Operations phase. Cloud resources—networks, virtual machines, databases, Kubernetes clusters—can be defined, provisioned, and configured using code. Code enables the implementation of versioning, testing, validation, roll-back, and drift management practices. It also allows teams to manage sharing and access control across complex environments in a granular way.

GitOps extends the Git-based Source Code Management model by applying it to IaC. The resource specification code is stored in a Git repository, which serves as the single source of truth. Changes are logged, enabling tracking of who changed what, potential root cause analysis, and rollback capabilities. These capabilities support the audit and control requirements inherent to banking and financial services. Changes to IaC code trigger automated pipeline execution, resulting in new resources being provisioned, existing ones reconfigured, or defunct resources removed. A separate automation agent continually monitors the cloud environment and compares real-world resource configurations with those defined in the IaC code. If discrepancies are found, the agent uses the IaC definition to correct them.

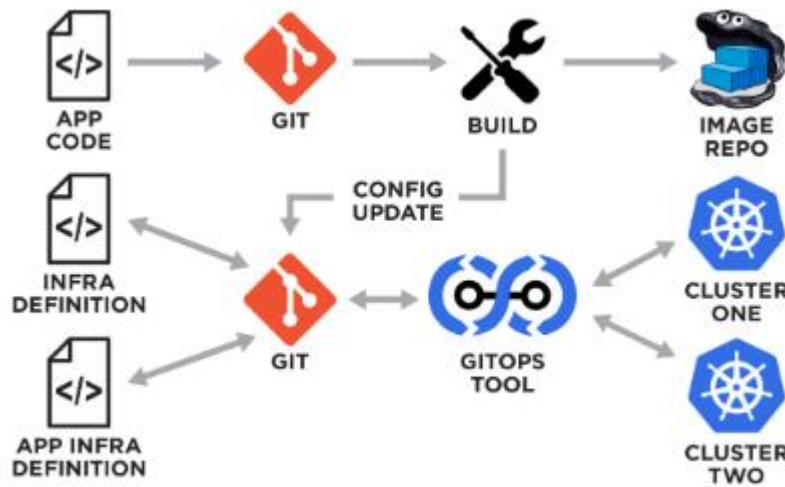




Fig 3: IaC vs GitOps

3.3. Continuous Integration and Delivery Pipelines

In a financial services context, Continuous Integration and Delivery (CI/CD) pipelines automate the building, testing, and deployment of applications. Automated testing of quality-gate criteria can occur at different stages, enabling early fault identification and reducing recovery costs. To align with regulatory requirements, a well-defined deployment strategy addresses security breaches, provides rollback mechanisms, and ensures that any code change has associated unit test coverage. An integrated data-management policy requires that production-use data remain separate from test and development data. This necessitates the generation of valid-test data for different scenarios and datasets or the application of data-anonymization techniques.

As services supporting financial applications are heterogeneous, security-critical services can be integrated without impacting the overall deployment process. Security testing becomes an integral yet independent part of the CI/CD pipeline with policy-as-code definitions that check for known security vulnerabilities, security-risk levels, and compliance-alignment requirements. These pipelines—combined with appropriate Infrastructure as Code (IaC) and GitOps definitions for infrastructure provisioning and testing—enable automated, secure, and reliable deployments. As tools continually provide validated code, the development velocity increases. To reinvent testing within the pipeline while meeting compliance requirements, generative AI can play a significant role in AI-assisted testing and verification.

4. Generative AI-Driven Automation Scenarios

To ensure compliance with security and regulatory policies during the provisioning and operation of cloud infrastructure and application resources, a policy-driven approach can be employed within the concept of Policy-as-Code. Developers and operators adopt guardrails aligned with the organization's policies, thereby minimizing the possibility of non-compliance. When a policy violation is detected (e.g., through cyclical monitoring of the production environment), a solution can be automatically proposed to remediate the issue, and a team can be notified. The actual remediation is performed through the automated execution of policy-recommendation code (the Runbook Code) that contains the script for implementing the proposed solution.

Generative AI can further assist in creating a new set of Guardrails that more accurately describe the Security and Compliance requirements, automatically composing checks that provide a broader range of coverage based on the attributes of the cloud resources under constant development. Such checks can be expressed using Policy-as-Code concepts (SOPHOS, 2023). Policies are stored in a repository under GitOps principles and versioned, which enables an audit trail that complements policy-execution verification. Additionally, these policies can directly trigger the manual provisioning of Services and Kubernetes Handlers via ChatOps integrations. When a new Service is created, a new policy can be generated from predefined Sweet16 ComplianceGuardrails templates.

Generative AI is also used for intelligent incident response. In this scenario, a possible security or compliance incident has been detected by an observability tool, and the initial response is managed through chat technology, which serves as a service interface for presenting both information and recommended actions. Based on the incident's context, Generative AI composes the informational messages and suggests a remediation action aligned with established Runbook Code.



Architecture layer / domain	Core elements mentioned in the paper	Primary responsibilities	Key data / control flows	Main stakeholders
Application layer	Core application, dependencies, API gateway, runtime components	Deliver business functionality and interact with surrounding controls	Emits triggers for tests, policy checks, observability signals	Developers
DevSecOps layer	Security checks, policy validation, testing, verification, compliance controls	Enforce security, compliance, and quality gates throughout the lifecycle	Receives triggers from application layer; sends results to health checks and compliance functions	Developers, Security & Compliance officers
Infrastructure / GitOps layer	IaC templates, cloud resources, Kubernetes clusters, drift detection, provisioning automation	Provision and maintain target-state infrastructure from code	Reconciles live environment with IaC definitions; remediates drift	Operational engineers
Observability & logging layer	Monitoring, alerting, cloud logging, incident data, SLIs/SLOs	Detect failures, support root-cause analysis, provide runtime intelligence	Collects logs/metrics/events; feeds incident response and audit trails	Operational engineers, Security teams

Table 2: Reference architecture layers and responsibilities

4.1. Policy-Driven Provisioning and Configuration

In a cloud-native financial ecosystem, the need for provisioning and configuration across development, operations, and security domains scales exponentially with the growth of the application fleet. Control policies established by architects, cybersecurity officers, compliance officers, and risk officers can be expressed as code and aligned with various regulatory frameworks. Addressing these guardrails as code improves compliance posture and reduces audit complexity. Policy-as-Code solutions enable the encoding of these as a decision tree of Guards, yet policy violation still remains an incidental type of failure.

Provisions and controls are specified and expressed as input parameters. These are then used in Industry-Standard Infrastructure as Code (IAAS) templates and versions of reinforcement learning. The generated IaC templates are made available in version control, and drift checks are deployed as a CI/CD (continuous integration and continuous delivery) pipeline. In deviation from the Generate & Remediate As-Code practice seen in the previous section, this practice is aligned towards the Guard & Govern approach.

This approach emphasizes a **Guard & Govern** model, where provisions and controls are defined upfront as structured input parameters that guide infrastructure deployment and management. These parameters are integrated into industry-standard Infrastructure as Code (IaC) templates, often enhanced through reinforcement learning techniques to improve decision-making and adaptability over time. The generated IaC templates are systematically stored and maintained in version control systems, ensuring traceability, consistency, and auditability. To maintain alignment with defined policies, drift detection mechanisms are



embedded within CI/CD pipelines, continuously monitoring infrastructure for unauthorized or unintended changes. Unlike the Generate & Remediate As-Code model, which focuses on correcting issues after deployment, this approach proactively enforces governance by embedding controls directly into the deployment lifecycle, thereby reducing risk, enhancing compliance, and ensuring that infrastructure remains consistently aligned with organizational standards.

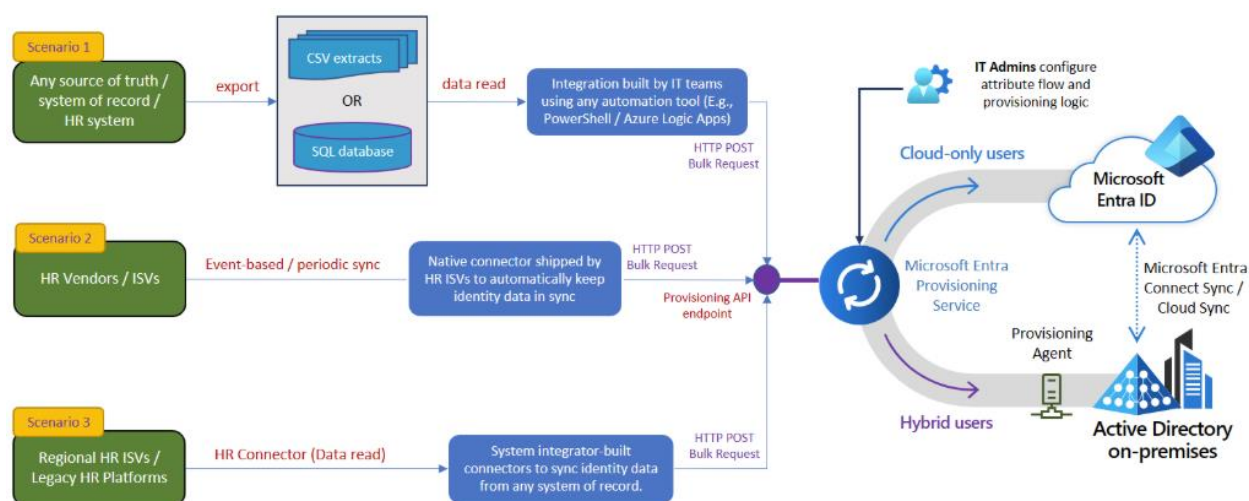


Fig 4: API-driven inbound provisioning

4.2. Intelligent Incident Response and Remediation

Policy-driven provisioning and configuration enable automatic detection and correction of policy drift. Guardrails express configuration policy in natural language, which AI can transform into policy-as-code form for enforcement by policy engines. Deviations from guardrails trigger alerts, while policy violations initiate automatic corrective action. However, interpretation and root-cause analysis are challenging even with the best monitoring and observability capabilities, and human intervention is critical for non-automated steps. Automation ultimately minimizes mean-time-to-recovery (MTTR) but has as little impact on mean-time-to-failure (MTTF).

A sentiment analysis use case illustrates the potential for intelligent incident management and remediation automation. ChatGPT receives data from enterprise chat channels preprocessed by a natural language processing (NLP) pipeline for check, classify, and tag. Detection captures incoming sentiment, with rules set for action on adverse sentiment spikes in any department, channel, or subject area. Root causes are searched for in incident databases focussing on time, area, topic, or department. Most probable causes are forwarded for human validation, and recovery is automated for incidents flagged in advance as repetitive enough for a playbook. Improvised alerts are issued if emergency action is clearly needed but no prior playbook exists. Centralized incident reports help departmental leaders propose improvement actions for such recurring incidents.



4.3. AI-Assisted Testing, Verification, and Compliance

Generative AI plays a crucial role in supporting the design, creation, and execution of test cases and verification artifacts within AI-assisted verification pipelines or as part of the broader CI/CD process. Especially for financial applications that process sensitive live data, adhering to comprehensive testing and verification coverage requirements and regulatory compliance is essential. These requirements can often be difficult to fulfill, leading to testing and verification-driven release delays and slowing down the development process. Generative AI can assist in all phases, from identifying coverage criteria, such as testing paths and conditions, to generating test data and test cases, including mock interfaces to the underlying microservices, APIs, or data stores.

To minimize risks, the generated test cases need to be subsequently executed, ideally in the native CI/CD pipeline of the application. The results should be automatically analyzed, and any deviations from the expected behaviors communicated to the development team. Also, regulatory compliance verification is crucial for cloud-native financial applications. However, compliance requirements for production systems can differ significantly from such for test and development environments. Generative AI can automatically devise policy-precompliance-testing requirements and create synthetic data to fulfill them. Also, compliance-conforming data access can be assured during production data usage by deploying data access demarcation during the runtime of the application allowing policy-conform testing.

Equation 4: Change Failure Rate

Let

- N_f = number of failed changes
- N_c = total number of deployed changes

Then

$$CFR = \frac{N_f}{N_c} \times 100\%$$

Step-by-step derivation

1. A rate is generally:

$$\text{rate} = \frac{\text{undesired events}}{\text{total events}}$$

2. Here:

- undesired events = failed changes = N_f
- total events = deployed changes = N_c

3. So the raw fraction is:



$$\frac{N_f}{N_c}$$

4. Because the paper uses percentage language, multiply by 100:

$$CFR = \frac{N_f}{N_c} \times 100\%$$

5. Data Management and Privacy Implications

Data minimization constitutes a fundamental principle of data governance, especially in the context of financial services handling sensitive personal data. To enforce this principle effectively, all synthetic data generation tasks must fulfil the following criteria. The data required for training or other AI-related purposes must not contain any real production data. The synthetic data generation process must automatically determine the minimum set of production data required. Additionally, all synthetic data created for training or testing must not be used in other processes, thus limiting exposure.

Synthetic data generation serves as an enabling mechanism for policy-driven provisioning and intelligent incident response. Machine-learning-based anomaly detection techniques rely on normal, non-anomalous training data, which is often hard to come by, especially during the development phase. The problem can be addressed by creating normal-status synthetic data on demand. Such data can be used to train detection models for specific microservice instances, with coverage criteria determining when to generate synthetic data.

Generative AI models sometimes produce toxic or offensive outputs or provide inaccurate content. Guardrails—software limitations on the generative model—aim to prevent such occurrences. Access controls, containing blacklists and whitelists of topics, serve as initial means of safe prompting. A safe prompting framework helps build several safe prompting techniques for a variety of scenarios. Misuse of Generative AI models can cause reputational damage and financial loss. Creating misuse-monitoring models helps mitigate unsafe outputs within the AI-DevOps interaction pathways. Auditability, provenance, and explainability of the AI models' decisions within DevOps workflows are critical. Models should generate audit trails with information about prompting and the data generated or used, which enable their auditability.

Equation 5: Mean Time to Recover (MTTR)

Let for incident i :

- $t_{incident}^{(i)}$ = time incident starts or is detected
- $t_{restore}^{(i)}$ = time service is restored

Recovery time for incident i :

$$RT_i = t_{restore}^{(i)} - t_{incident}^{(i)}$$



If there are m incidents, then

$$MTTR = \frac{1}{m} \sum_{i=1}^m (t_{restore}^{(i)} - t_{incident}^{(i)})$$

Step-by-step derivation

1. Recovery duration for one incident:

$$RT_i = \text{restore time} - \text{incident time}$$

2. Write it symbolically:

$$RT_i = t_{restore}^{(i)} - t_{incident}^{(i)}$$

3. Mean means average over all incidents:

$$MTTR = \frac{RT_1 + RT_2 + \dots + RT_m}{m}$$

4. Substitute each RT_i :

$$MTTR = \frac{1}{m} \sum_{i=1}^m (t_{restore}^{(i)} - t_{incident}^{(i)})$$

5.1. Data Minimization and Synthesis for Training

Data minimization is a key principle in the design and operation of cloud-native financial platforms, and extends to the use of Generative AI within associated DevOps workflows. Sensitive data should not be retained for longer than necessary, exposure minimized during operations using industry-standard data-encryption measures, and never recorded when no longer required to facilitate a transaction or other function. Within the context of data synthesis, automatic engine and service interactions send minimal metadata traces describing communication intent. However, when building and testing AI models for development, service reliability, compliance, and risk mitigation in cloud-native ecosystems and the increasingly connected FinTech domain, an associated issue arises. The data required to fulfil the training criteria for support pipelines is often fragmented, scarce, and even if available, difficult to provide, with major concerns surrounding the retention, transcription, labeling, and consequent unauthorized use of sensitive data. Generative AI techniques can also minimize the gathering of information from customers for training purposes by synthesizing training data corresponding to the expected quality, quantity, and coverage ratios. The process follows the principles set forth by Braune et al. in support of back-end FinTech testing where data labels also require customer input.



Techniques belonging to the data-minimization domain reduce exposure while generating training data. For example, Privacy-Preserving Deep Learning (PPDL) techniques build Deep Learning models without transferring the local training data to the third-party servers. The american fuzzy lop (AFL) approach synthesizes semantic preserving data and reinforces DLP frameworks by checking the existence of that data in the dataset, while the Data-Shuffling-based Neural Networks technique adds noise to the data and redistributes it on a homomorphic basis. The AI-GTV approach preserves how the sensitivity of each feature is regarded in previous works towards testing the data but enhances the remaining solutions by also achieving good utility. Hidden Markov Models are used for assisting in the synthesis of missing testers across multiple stages while also improving the test data generation process via a space buffer.

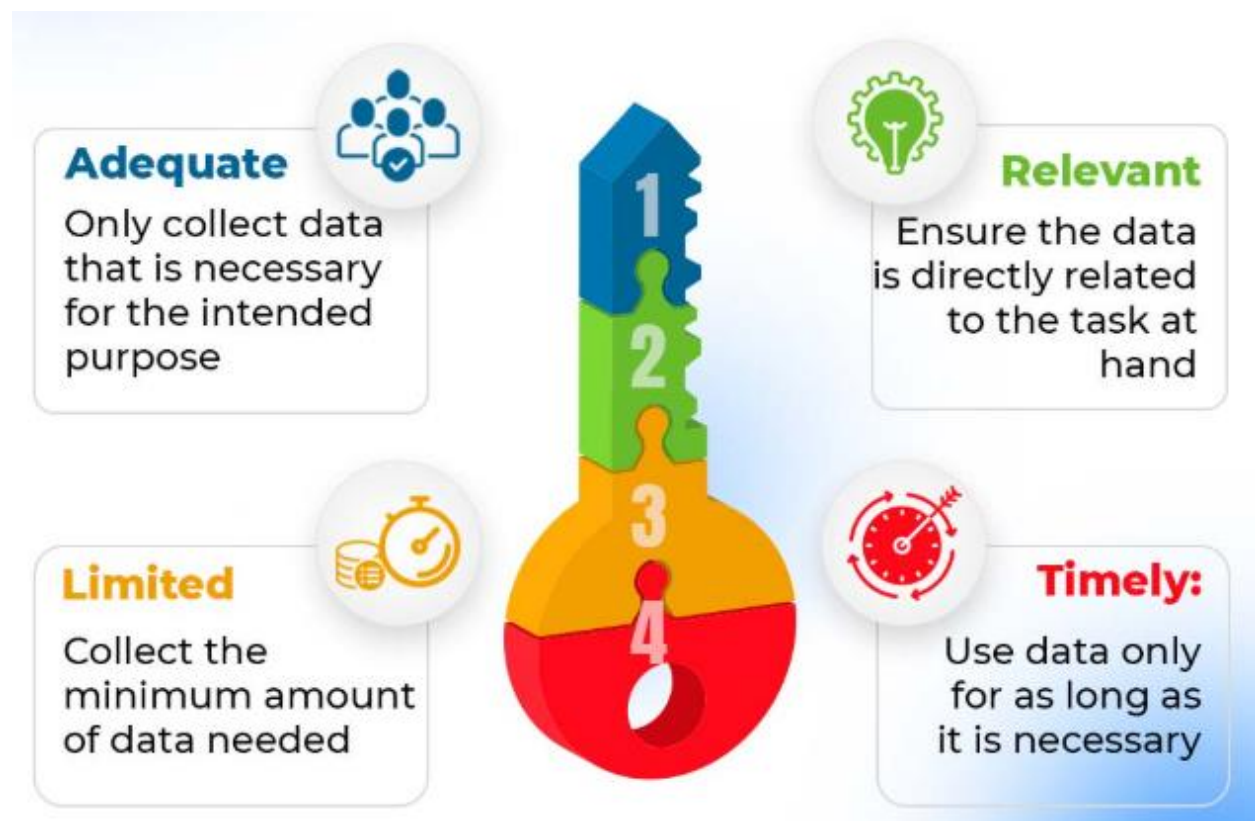


Fig 5: Data Minimization and Synthesis for Training

5.2. Safe Prompting and Guardrails

Misuse scenarios for generative AI in DevOps environments cover prompting, including prompt design, user (mis)use, and controlling the prompt space, and how these scenarios factor into the provision or service of models and applications that use them. Concerns over prompting fall into various groups: lateral prompts (the prompt lives on one host) elevate the notion of



prompt engineering to a privileged activity; a special category of prompt engineering is known as safe prompting; antagonist prompts attempt to unlock toxic content via prompting. Policy-as-Code can be used to mitigate risk and as a strategy for exploitation detection. Policy-as-Code guardrails narrow the prompt space given to users and record those input prompts for future alerting or investigation.

Models as Services provide prompt management, including logging. Another wrapping strategy is to expose the model as a service and restrict the prompt space for people assigned to lower roles. Efforts have also gone into reducing the impact of dangerous or toxic generations. When the goal is model provision rather than use, supervision and monitoring of input can ensure that prompt design has not made privileged prompts unsafe. The prompt space for those using the model may also be restricted, either limiting its utterances or providing the utterances via APIs without access to changer.

5.3. Auditability, Provenance, and Explainability

Integrating Generative AI into DevOps enables intelligent automation of core operational tasks. Such automation invariably raises questions related to the traceability of decisions leading to production system actions, operational incident resolution, model quality, and auditability in sensitive domains like financial services. Minimizing the volume of sensitive production data in DMLC pipelines and during operation is crucial for risk and compliance, and supporting AI model data-growing diversity by generating and utilizing synthetic data mitigates the impact. Intelligent augmentations of these DevOps and progressive automation stages are often used to establish such controls. AI-assisted testing and verification require multiple forms of coverage criteria, including security testing, certification, and IT audit trail in distributed systems.

Prompt-based models are susceptible to unbounded hallucinations. Containing error classes exhibits a Boolean behavior for certain errors; that is, prompting alone can control errors by being a potentiation of their occurrence. Properly safeguarding such prompting scenarios by implementing guardrails and sufficient role-based access guarantees remediates this vulnerability.

Scenario	Problem addressed	GenAI capability used	Inputs	Automated actions	Benefits	Risks / controls needed
Policy-driven provisioning and configuration	Manual provisioning does not scale and can violate policy	Natural-language-to-policy generation, template generation, policy recommendation	Guardrails, regulatory requirements, infrastructure parameters, cloud resource attributes	Generate IaC/policy artifacts, validate configurations, trigger remediation workflows	Faster provisioning, improved consistency, lower audit burden	Guardrail quality, policy accuracy, approval controls, versioning
Intelligent incident response and remediation	Incident analysis and recovery are slow and complex	NLP, contextual summarization, remediation recommendation, runbook mapping	Logs, alerts, chat activity, incident history, observability context	Classify incidents, suggest root causes, initiate repetitive runbooks, notify teams	Reduced MTTR, quicker coordination, better reuse of institutional knowledge	False diagnosis, unsafe autonomous remediation, need for human



Scenario	Problem addressed	GenAI capability used	Inputs	Automated actions	Benefits	Risks / controls needed
AI-assisted testing, verification, and compliance	Testing and compliance evidence are costly and incomplete	Test-case generation, synthetic-data generation, result analysis, requirement expansion	Coverage criteria, policy requirements, service interfaces, expected behaviors	Generate test cases/data, execute in CI/CD, analyze deviations, surface compliance gaps	Increased coverage, less manual effort, quicker release readiness	validation on critical events Synthetic data fidelity, hallucinated test logic, audit trail requirements

Table 3: Generative AI-driven automation scenarios

6. Evaluation Methodologies

Metrics for assessing velocity, reliability, security, and cost of deployments, along with measurement methods, are specified. Risk assessment strategies and evaluation of control effectiveness in AI-enabled pipelines are also introduced. To validate the proposed concepts, case studies and benchmarking plans are presented.

Metrics

The velocity of deployment relates to the frequency of deployments to production and the lead time required to deploy a fix or feature. To assess the safety associated with the frequency of deployments, the rate of failures is noted along with the mean time to recovery. For DevOps-driven security testing, security-related incidents detected after deployment are measured. The cost associated with running the security and quality gates can be monitored as a measure of cost control.

Risk assessments and evaluation of control effectiveness are considered. With respect to financial services, concentration risk is regarded as the risk of failure of a single customer, supplier, or application, while model risk is the risk of reliance on a model that is mis-specified or misused. For AI-enabled pipelines, the plausible risk scenarios identified are plain prompts leading to plain misuse of the AI service without an intermediate human review and plain prompting leading to successively amplified errors in autonomous release procedures.

To verify these effects and for accuracy, case studies involving existing financial-grade cloud-native financial platforms and companies are pursued. As per regulation, such institutions will soon evolve—if not already—into risk-taking conglomerates in a true ecosystem.



6.1. Metrics for Deployment Velocity and Reliability

DevOps embraces a set of practices that enables rapid development and operation of application software. Successful DevOps adoption enhances an organization's ability to deliver applications and services at high velocity: evolving and improving products faster than the organization's competitors. With AI-enabled DevOps in place, the time from development to deployment can be decreased, meaning the deployment velocity increases. Conversely, downtime and unplanned service outages can be reduced, which increases the service reliability.

Deployment frequency indicates how often a service goes into production. Normally, organizations choose to measure this metric in deployments per day. A higher deployment frequency can be indicative of higher productivity and efficiency. The lead time for changes describes the time it takes for a commit to get into production. Shortening this lead time is often an important goal of DevOps initiatives. Deployment duration indicates how long it takes for a production deployment to be completed. Having rapid deployment durations allows for an increase in deployment frequency without increasing the burden on teams to deploy. Change failure rate measures the percentage of changes that fail. Reducing this rate is a crucial goal of many DevOps initiatives, usually because recovering from these failures takes considerable time and cost. Mean time to recover measures how long it takes to restore service when a service incident occurs. Decreasing mean time to restore improves the reliability of the service from an end-users perspective.

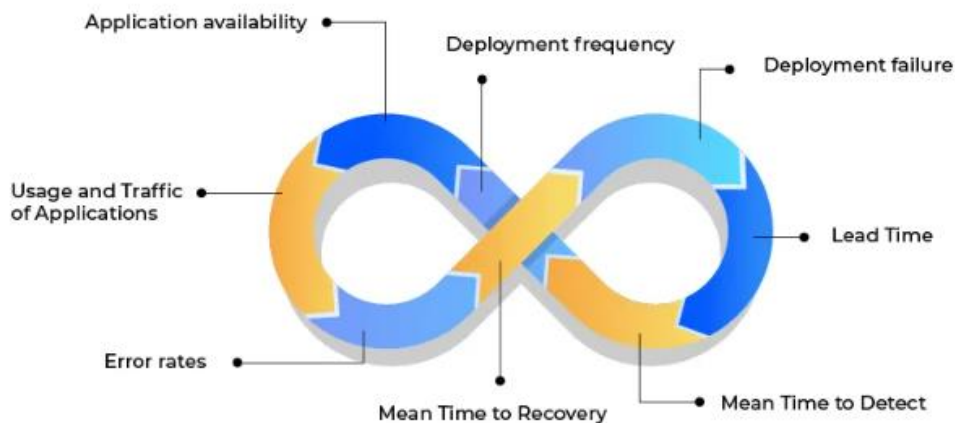


Fig 6: Key Metrics of DevOps

6.2. Risk Assessment and Control Effectiveness

Automated testing and verification enhance deployment velocity and quality, and introduce additional risk and compliance requirements. Synthetic data generation supports testing of solutions with sensitive workloads such as financial services, healthcare and other regulated environments. The bespoke nature of Generative AI workloads makes it difficult to establish



coverage criteria, and similarly testing for detected bias can be scarce or even impossible. During generative testing scenarios, audit trails are of utmost importance for investigation purposes.

Policy-driven provisioning and configuration provide a solid foundation for infrastructure and platform configuration. Expanding into Secure-by-Design or Secure-by-Default territory enables Policy as Code support for applications and services, validating that functional Requirements and Security Controls are met. The decentralized nature of testing in financial grade cloud-native systems aligned to Open Coalition patterns focus on methods to ensure tests are properly supported avoiding scenarios of over-commitment / under-delivery, devices that should not be deployed.

6.3. Case Studies and Benchmarking

Cloud-native financial platforms leverage DevOps practices, including continuous integration and continuous delivery (CI/CD), to accelerate deployment speed. Generative AI offers the potential for DevOps to provide compliance, security, and risk assurance without restricting agility. GenAI capabilities such as natural language processing and code generation, combined with domain-shared data, and expanded training, can be harnessed to boost policy-driven automation and make aligned intervention timely, reliable, and efficient. However, it is essential to incorporate risk assessment techniques and evaluate the effectiveness of controls within the generative pipelines.

Case studies and quantitative validation based on historical data from existing financial platforms can help assess the relevance and efficiency of the generative automation points outlined. Metrics associated with the deployment velocity, reliability, security, and cost of cloud-native financial applications can inform evaluation. GenAI provides an extensive domain and reduces supervision needs, offering avenues to advance testing levels and present evidence of the broad coverage of compliance and risk controls. Automating Guardrails-as-Code tooling that emphasizes policies, procedures, and practices as code in a bank can further enhance their alignment with the financial industry.

7. Implementation Roadmap

Phased adoption with milestones and criteria confirms feasibility, clarifies dependencies, and governs organizational change. Tooling requirements address DevOps enablement, Generative AI integration, risk management, and assurance through standardization, interoperability, and embedding in cloud-native financial platforms.

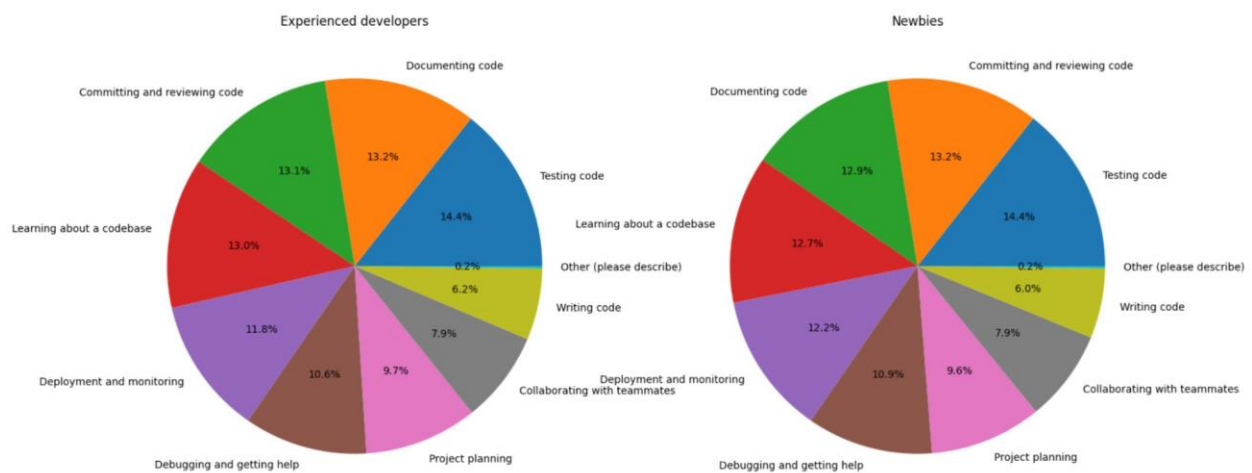
The proposed Generative AI-enabled DevOps automation represents an ambitious conceptual reference architecture. Deployment velocity, reliability, security, and operational cost constitute key dimensions of any financial service. Metrics justify investment by establishing process baseline, benchmarking effort and risk, calculating cost of changes, measuring requisite control effectiveness, and assessing deployment across security zones. Addressing the intersection of Generative AI and DevOps adoption within FinTech clouds, the confluence of governance, detection of anomalous behaviour, and closed-loop compliance engenders trust. Timely and intelligent detection of Service Level Objective infringement, security, and compliance violations facilitates fast, risk-aware remediation.

Adoption comprises initiation and onboarding of Generative AI technology within Financial-DevOps. Generative Ai deployment emerges as a true dual-use technology: leveraging closed-source models for core functions and fine-tuning safe open-source



variants across other Deployment Zones. Nitbacked integrating Infrastructure as Code tooling augments governance and control. Policy Risk and Cyber Risk Assessment complement. Co-designing policy guardrails with Policy as Code tools and integrating Generative Ai into testing and audit trail mechanisms complete the Foundation Phase. Customer interaction scripting and policy-driven security-camera provisioning exemplify initial Scalable AI use cases.

Provide incident recovery for financial applications** with integrated bank policy awareness. The ability to detect, respond, and recover using the financial application-provided basic observability requirements, potentially supported by an initial frictionless simulation concept.



7.1. Phased Adoption Strategy

Bridging the gap between a traditional DevOps setup and the envisioned AI-enabled automation framework requires a planned approach supported by a roadmap. One possible strategy for incrementally embarking on the necessary journey is illustrated in the figure below. The organization needs to first be aligned on an agreed-upon baseline governance model that matches the overall cloud-native enabling strategy and roadmap. Subsequent focuses for each increment and their associated outcomes are as follows:

1. ****Adopt the first-generation Infrastructure as Code blueprint**** that facilitates Cloud Service Provider provisioning and resource updating fast through standard operating procedures and patterns. Standard templates should be established as GitOps modules. The policy gating will be added in the next increment.
2. ****Introduce Infrastructure as Code policy and GitOps drift management**** that action the desired operating model with managed policy violations through GitOps architecture, integrated with the incident management flows, and automated remediation. Subsequent violations can therefore be highlighted in near-real time.



7.2. Organizational and Governance Considerations

For automation to provide the anticipated advantages, Gen-AI-enabled DevOps processes require suitable organizational changes. Both internal and external controls must evolve to enable quicker decisions. Stakeholders could shift from being mere spectators, relegated to post-hoc checks, to active participants throughout a product's lifecycle. Failure modes may change from traditional security breaches to undesirable data leaks, toxicity, and bias. New possibilities may also arise. For instance, cloud-native architectures facilitate the declaration of quality attributes via Machine Learning Model Operations (MLMLOps) policy-as-code specifications that accompany application deployment.

Recent events motivate the establishment of a Gen-AI policy. Indeed, organizations are already establishing common sense policy specification and checking frameworks to enable safe usage of Gen-AI tools. The outcomes of policy-constrained prompting must also achieve auditability, provenance, and explainability (PEA). The compliance requirements for privacy and security must also take into account the risk of Gen-AI models being trained on confidential information or generating they-image content.

7.3. Tooling Stack and Standardization

Standardization is key for effective tooling adoption—especially in an interconnected environment like financial services—where misaligned policies can lead to information security breaches. The tooling stack, policy specifications, and assessment templates for governing Generative AI across the DevOps ecosystem should thus be standardized. Domain-centric templates for policy as code, guardrails, AI safety, testing, compliance verification, incident detection, and recovery should be developed.

Collaboration with industry forums can ensure interoperability across the open banking ecosystem. A tailored generative AI system for guiding prompt and feedback construction within the organization can be integrated with the DevOps pipeline to ensure central control without hindering creativity and exploration. Adopting open-source tooling and fulfilling compliance obligations will form the basis for establishing a sustainable cloud-native financial platform.

8. Conclusion

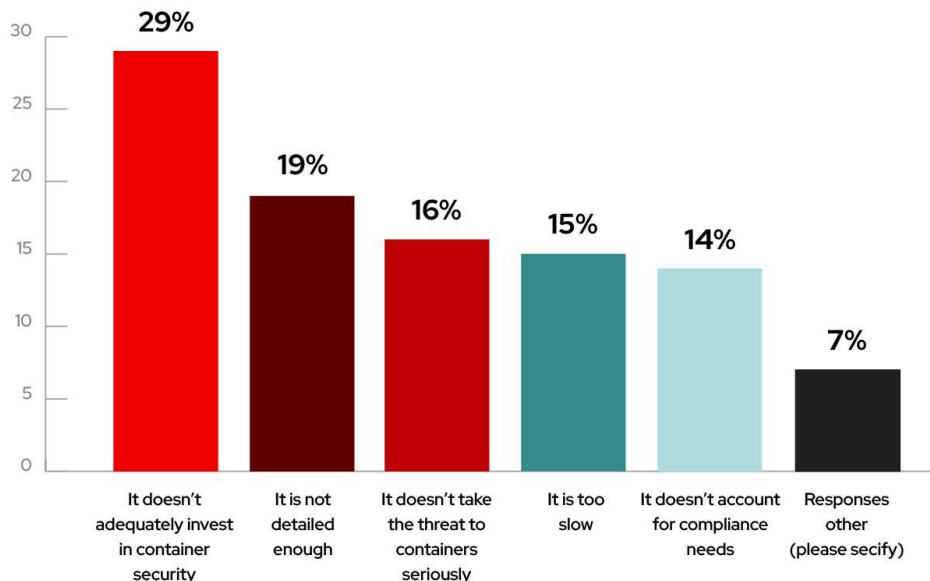
Generative AI-Enabled DevOps Automation for Scalable Cloud-Native Financial Platforms

Findings, limitations, and implications for future research are synthesized. The proposed concepts advance state-of-the-art automation and governance of DevOps workflows and AI models, facilitating faster, safer deployment of financial applications in public and hybrid clouds.

Emerging digital-native banks, fintech ecosystems, and the open-banking paradigm are fundamentally transforming the financial services industry. To meet increasing competition and regulatory pressure for rapid innovation with reduced costs while maintaining government-grade reliability, banks are evolving from traditional operation models to scalable cloud-native architectures based on large-scale use of outsourcing and third-party service provisioning. New applications must be delivered at high speed while mitigating deployment-related risks. Generative AI is poised to transform many service operations across different domains and functions. Applying Generative AI in DevOps Automation is one of the exciting possibilities attracting a lot of attention. Development Operations—commonly referred to as DevOps—is a software development and delivery



methodology that combines the fields of software development (Dev) and information-technology operations (Ops) that aims to help organizations produce software and IT services more rapidly and with greater quality.



8.1. Emerging Trends

The multi-faceted capabilities of Generative AI are currently receiving extensive attention and experimentation. In addition to the potential for quality, cost, and speed improvements in automated or assisted content creation, innovative applications are emerging in the areas of intelligence augmentation, automation, and programming assistance. The use of Generative AI to enhance DevOps automation and governance offers great promise for cloud-native financial applications—reducing complexity and risks while accelerating deployment velocity and increasing system reliability. However, these technologies are not without challenges. The potential for exploitation of vulnerable models, as well as the risks of automating poor-quality decision-making, underline the need for careful prompt design, safe prompting, and governance of AI-generated code.

Organizations employing Generative AI for DevOps automation and governance must prioritize data minimization, embedding privacy-preserving mechanisms, protecting against misuse, and ensuring the auditability and explainability of AI-assisted decisions within the DevOps workflow. The overall effectiveness of an AI-enabled DevOps pipeline can be assessed in terms of deployment velocity, security, and reliability, and appropriate evaluation methods incorporated into the implementation roadmap.



9. References

1. Bogner, J., Fritzsche, J., Wagner, S., & Zimmermann, A. (2021). Industry practices and challenges for the evolvability assurance of microservices. *Empirical Software Engineering*, 26, Article 104.
2. Ghantous, G. B., & Gill, A. Q. (2021). Evaluating the DevOps reference architecture for multi-cloud IoT-applications. *SN Computer Science*, 2, Article 123.
3. Kumara, I., Garriga, M., Romeu, A. U., Nucci, D. D., Tamburri, D. A., van den Heuvel, W.-J., & Palomba, F. (2021). The do's and don'ts of infrastructure code: A systematic gray literature review. *Information and Software Technology*, 137, Article 106593.
4. Loganathan, R. (2022). Converging Security Architecture and Compliance Management in Enterprise Data Center Ecosystems: A Unified Control Framework. *International Journal of Scientific Research and Modern Technology*, 1(12), 295-312.
5. Mäkinen, S. S., Skogström, H., Laaksonen, E., & Mikkonen, T. (2021). Who needs MLOps: What data scientists seek to accomplish and how can MLOps help? In *2021 IEEE/ACM 1st Workshop on AI Engineering—Software Engineering for AI (WAIN)*. IEEE.
6. Granlund, T., Kopponen, A., Stirbu, V. A., Myllyaho, L., & Mikkonen, T. (2021). MLOps challenges in multi-organization setup: Experiences from two real-world cases. In *2021 IEEE/ACM 1st Workshop on AI Engineering—Software Engineering for AI (WAIN)*. IEEE.
7. Renggli, C., Rimanic, L., Gürel, N. M., Karlas, B., Wu, W., & Zhang, C. (2021). A data quality-driven view of MLOps. *IEEE Data Engineering Bulletin*, 44(1), 11–23.
8. Ruf, P., Madan, M., Reich, C., & Ould-Abdeslam, D. (2021). Demystifying MLOps and presenting a recipe for the selection of open-source tools. *Applied Sciences*, 11(19), 8861.
9. Shahin, M. (2021). A qualitative study of architectural design issues in DevOps. *Journal of Software: Evolution and Process*, 35(5), e2379.
10. Chen, M., Tworek, J., Jun, H., Yuan, Q., Ponde de Oliveira Pinto, H., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., et al. (2021). Evaluating large language models trained on code. *arXiv*.
11. Mattaparthi, R. (2022). Engineering Predictive Industrial Systems Through IoT-Driven Asset Monitoring and Machine Learning Prognostics. *International Journal of Future Innovative Science and Technology (IJFIST)*, 5(1), 7790.
12. Akbar, M. A., Smolander, K., Mahmood, S., & Alsanad, A. (2022). Toward successful DevSecOps in software development organizations: A decision-making framework. *Information and Software Technology*, 147, Article 106894.
13. Alnafessah, A., et al. (2022). A survey on DevOps practices and challenges in software development organizations. *Journal of Systems and Software*.
14. Camilli, M., & Russo, B. (2022). Modeling performance of microservices systems with growth theory. *Empirical Software Engineering*, 27, Article 39.
15. Hewage, N., & Meedeniya, D. (2022). Machine learning operations: A survey on MLOps tool support. *arXiv*.
16. Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Dal Lago, A., Hubert, T., Choy, P., d'Autume, C. de M., Babuschkin, I., Chen, X., Huang, P.-S., Welbl, J., Goyal, S., Cherepanov, A., Molloy, J., Mankowitz, D. J., Sutherland Robson, E., Kohli, P., de Freitas, N., Kavukcuoglu, K., & Vinyals, O. (2022). Competition-level code generation with AlphaCode. *Science*, 378(6624), 1092–1097.



17. Nguyen, D. K. (2022). Big data, artificial intelligence and machine learning: A transformative symbiosis in favour of financial technology. *European Financial Management*, 29(2), 517–548.
18. Nijkamp, E., Pang, B., Hayashi, H., Tu, L., Wang, H., Zhou, Y., Savarese, S., & Xiong, C. (2022). CodeGen: An open large language model for code with multi-turn program synthesis. *arXiv*.
19. Xia, H., Wang, Y., Gauthier, J., & Zhang, J. Z. (2022). Knowledge graph of mobile payment platforms based on deep learning: Risk analysis and policy implications. *Expert Systems with Applications*, 208, 118143.
20. Singh, A., et al. (2022). Machine learning for cloud-native applications: A systematic review. *Journal of Cloud Computing*.
21. Chen, Z., et al. (2022). A systematic review of machine learning applications in financial technology. *Electronic Commerce Research and Applications*.
22. Ayas, H. M., Leitner, P., & Hebig, R. (2023). An empirical study of the systemic and technical migration towards microservices. *Empirical Software Engineering*, 28, Article 85.
23. Erdei, R., & Toka, L. (2023). Minimizing resource allocation for cloud-native microservices. *Journal of Network and Systems Management*, 31, Article 35.
24. Reddy, V. A. R. (2022). Data-Driven Healthcare Operations: Architecting Unified Member, Provider, and Claims Intelligence Platforms. *International Journal of Science, Research and Technology*, 5(5), 8511-8521.
25. Fan, A., Gokkaya, B., Harman, M., Lyubarsky, M., Sengupta, S., Yoo, S., & Zhang, J. M. (2023). Large language models for software engineering: Survey and open problems. In *Proceedings of the 2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)* (pp. 31–53). IEEE.
26. Kreuzberger, D., Kühl, N., & Hirschl, S. (2023). Machine learning operations (MLOps): Overview, definition, and architecture. *IEEE Access*, 11, 31866–31879.
27. Ozkaya, I. (2023). Application of large language models to software engineering tasks: Opportunities, risks, and implications. *IEEE Software*, 40(3), 4–8.
28. Zheng, Z., Ning, K., Zhong, Q., Chen, J., Chen, W., Guo, L., Wang, W., & Wang, Y. (2023). Towards an understanding of large language models in software engineering tasks. *arXiv*.