National College of Ireland

Designing and Scaling OPA for PCI-DSS and HIPAA Compliance in AWS

MSc Research Project
Cloud Computing

Dhanusha Siva Priya Chintakayala
Student ID: 23192887

School of Computing
National College of Ireland

Supervisor: Dr Giovani Estrada

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Dhanusha Siva Priya Chintakayala
Student ID:	23192887
Programme:	Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Dr Giovani Estrada
Submission Due Date:	24/04/2025
Project Title:	Designing and Scaling OPA for PCI-DSS and HIPAA Compliance in AWS
Word Count:	10077
Page Count:	26

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Dhanusha
Date:	26th May 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Designing and Scaling OPA for PCI-DSS and HIPAA Compliance in AWS

Dhanusha Siva Priya Chintakayala
23192887

Abstract

As the cloud native infrastructure gets more dynamic and complex, the level of difficulty maintaining its compliance with regulatory standards, such as PCI-DSS and HIPAA, pose challenges to DevOps teams. The traditional manual compliance verification methods are known to be time consuming, error prone and are in most cases taking to configuration drifts. This research proposes an automated solution using Terraform for infrastructure provisioning and Open Policy Agent (OPA) for policy enforcement within an AWS CodePipeline-based CI/CD workflow.

The declarative Rego policies stored in version controlled S3 buckets are continuously validated before the infrastructure code is applied to the cloud environment. The performance of the system is evaluated experimentally across different dimensions including formation time, accuracy of compliance, execution timing stage-wise, and scalability (Policies of Observation Planning) with the growing number of OPA policies.

Results show that the automated compliance pipeline improves configuration accuracy by more than 30% and minimally reduces formation time (up to 86% improvement) over manual methods. For instance, the validation of 35 policies takes less than 0.3 seconds. This research describes in detail how policy enforcement can be made operational with Terraform and OPA to ensure compliance as well as deployment agility in cloud infrastructure management.

1 Introduction

Cloud computing adoption has experienced extraordinary growth during recent years because businesses need elastic and cost-efficient computing solutions. Organizations now choose cloud-based platforms to boost their operational performance, while at the same time decreasing the need to manage their infrastructure. The rise of Infrastructure as a Service (IaaS) among different cloud service models brings significant value because it enables system administrators together with developers to manage virtualized computing resources through infrastructure-as-code (IaC) tools including Terraform. Market research shows that Amazon Web Services (AWS) leads the cloud industry with the most substantial market share, yet both Microsoft Azure and Google Cloud Platform also play an important role in the global cloud market space [1].

The software development company HashiCorp created Terraform which has gained widespread popularity as an IaC tool since it uses a declarative configuration write and

[1] `https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-serv`

supports multi-cloud resource management. The platform uses cloud-agnostic design principles for provisioning infrastructure across AWS and Azure and Google Cloud Platform and other platforms. Infrastructure automation capabilities were enhanced, while consistent deployment standards became possible using Terraform [2].

Following the use of IaC, the management of infrastructure has become easier, but it does not mean that new challenges in compliance and regulatory standards are less stringent. As applications and services are rolled out to the cloud, enterprises are increasingly required to comply with data protection regulations such as Payment Card Industry Data Security Standard (PCI DSS), the General Data Protection Regulation (GDPR), as well as Health Insurance Portability and Accountability Act (HIPPA). Data breaches, legal penalties and reputational damage are some of things that follow not enforcing of compliance policies.

Since cloud resources are dynamic and ephemeral by nature, maintaining compliance in cloud environments is very hard. In most of these cases traditional security controls are insufficient to protect the new environments (containerized and also serverless). In addition, cloud resources can introduce vulnerabilities that lead to negatively impacting the compliance requirements by breaching the divergence between the desired state and the actual state of the cloud resources, known as configuration drift.

In order to address concerns in the above, Policy as Code (PaC) has brought into the picture an approach of expressing compliance and security policies in machine readable formats that can be statically enforced during provisioning of the infrastructure. Perhaps the most powerful tool of that, the Open Policy Agent (OPA), is an open source general policy engine, that allows fine grain access control and compliance checking in the entire cloud native ecosystem [3].

Rego, a declarative query language for writing policies[4] is supported as well as a large number of platforms such as Kubernetes, Terraform, and CI/CD pipelines. And it is now a key piece of cloud native security framework like the Cloud Native Computing foundation's (CNCF) Gatekeeper and Conftest. IaC configurations and Kubernetes manifests are validated with these tools by applying custom or standard compliance rules and are used to validate them before deployment [5].

1.1 Research Gap

It is a fundamental problem to design scalable, efficient, and cloud agnostic OPA policies. For most enterprise use cases, OPA is needed to run across many services in a distributed microservices architecture, which may give rise to the various performance tradeoffs of latency, resource overhead, and policy evaluation times in a system.

The research presented in this work seeks to bridge the field between the automation of infrastructure in a cloud-agnostic manner and the dynamic policy enforcement using OPA as a policy decision point across heterogeneous clouds. It is more focused on automating PCI-DSS compliance in the AWS environment using Terraform managed environment infrastructure.

Although OPA is commonly used to monitor policies in cloud-native environments, most studies have considered single clouds or Kubernetes runtime situations. Few studies

[2] https://learn.microsoft.com/en-us/training/modules/terraform-introduction-to-infrastructure-as-c

[3] https://www.openpolicyagent.org/docs/latest/

[4] https://www.openpolicyagent.org/docs/latest/policy-language/

[5] https://www.env0.com/blog/how-policy-as-code-enhances-infrastructure-governance-with-open-policy

have looked at how OPA can be carefully merged with tools such as Terraform to include compliance standards like PCI-DSS and HIPAA in CI/CD pipelines before deployment. There is no full framework available today that assesses how OPA policies scale and work in various cloud environments. To address this gap, this paper describes how to set up an automated CI/CD pipeline with Terraform which uses OPA to enforce compliance.

1.2 Research Question

Novelty is about showing how Terraform and OPA could fit both in a CI/CD pipeline for compliance automation on the deployment stage. It brings together both HIPAA and PCI-DSS compliance checks. The research looks at how to bring together various standards and automate them:

- How can Open Policy Agent (OPA) policies be developed and integrated in a Continuous Integration and Deployment pipeline of a Terraform-managed AWS infrastructure as a code (IaC), in such a way that it orchestrates various cloud services to maintain PCI-DSS and HIPAA compliance across the account in a cloud-agnostic approach?

The experiments collect important metrics, such as compliance accuracy, efficiency, and scalability, to assess the feasibility of the proposed pipeline.

1.3 Research Objectives

The goals of the key research objectives in this research are:

- Design and implement a CI/CD pipeline orchestrated with Terraform to integrate a codepipeline that creates an AWS infrastructure using Terraform with OPA for automatic compliance enforcement.
- Develop and encode Rego policies that represent PCI-DSS and HIPAA standards.
- Validate a policy within a pipeline using JSON formatted Terraform plans.
- Evaluate metrics between manual and automated compliance setup in accuracy and time to setup.
- Analyze the variation in compliance evaluation time based on the number of policies assessed

These are the core milestones that are needed to answer the research question and validate the proposed approach.

1.4 Outline

The document is structured as follows: Section 2 presents the theoretical background and related work on Infrastructure as Code and Policy Enforcement with OPA. The research methodology and strategy used are discussed in Section 3. Section 4 details the system architecture and design specifications of the CI/CD pipeline. In Section 5, the implementation of the proposed solution is presented, as well as its integration into Terraform and policy validation. Evaluation methodology, experiments, results, and discussion are presented in Section 6. In the final Section 7, important findings from this research, research implications, and work suggestions for the future are provided.

2 Related Work

Focus on compliance has been increasing as the loads of sensitive data are deployed on the on-prem, private, and public cloud. The integration of policy enforcers into infrastructure-as-code (IaC) stands as a main research subject for cloud computing security. As per (Agarwal et al.; 2022) states that the need for proper compliance is surging as many organizations are adopting cloud infrastructure. Some authors, such as (Kavas; 2023), have examined how Terraform, an IaC tool, could potentially be used to explain cloud resource management with reference to AWS.

A recent work (Paul et al.; 2024) has served as the foundation for the present research, which emphasises more on infrastructure than standards such as HIPPA and PCI-DSS in the fintech and insurance organizations. This is addressed in the proposed research by including these policy standards and others as part of Compliance as Code (CaS) across the AWS cloud environment. The goal is to put this study into practice in a way that will provide these policy enforcement results with the least amount of human configuration and effort, and without causing errors.

2.1 Infrastructure as Code (IaC) and Terraform

The introduction of Infrastructure as Code has changed the way of infrastructure provisioning and management, and Terraform has become a leading choice in this. As per the guidance distributed by IEEE Computer Society [6], Terraform modules are very similar to the programming functions and they encapsulate resource groups that can be reused across projects and environments.

According to (Dalvi; 2022), Terraform plays a vital role in Infrastructure as a Service (IaaS) automation because it enables infrastructure resources definition through code that decreases errors in manual provisioning while improving reproducibility. This study identifies two main challenges regarding complex dependency management and security compliance maintenance within Terraform configurations, which can be addressed in the given work.

Also (Yongsiriwit et al.; 2016) has outlined a resource management system for clouds that implements Topology and Orchestration Specification for Cloud Applications (TOSCA), Open Cloud Computing Interface (OCCI), and Cloud Infrastructure Management Interface (CIMI) standards. The research center addresses interoperability requirements between diverse cloud services since it supports cloud-agnostic systems deployments. The research field lacks exploration of how compliance checks should be integrated into these models.

Furthermore, (Banse et al.; 2021) introduced Cloud Property Graph (CloudPG), which merges between static code evaluation results and runtime security results to generate a vendor-independent visualization of cloud service security levels. The ability of the framework to detect misconfigurations remains strong, but it does not integrate with Infrastructure as Code systems such as Terraform to perform automated compliance checks. Also, (Teppan et al.; 2022) gives special attention to creating security templates that IaC can utilise to enforce security policies and procedures across various implementations. They also suggest the usage of Terraform and highlight shortcomings in current architectures that have problems with multi-cloud interoperability. The outcomes of this

[6] `https://www.computer.org/publications/tech-news/trends/terraform-guide`

work pay attention to the different orchestration techniques and broaden the scope of the significance of unification.

Similarly, (Battula; 2024) investigates software service compliance challenges in hybrid cloud structures and provides solutions for automated IaC template rule-checking. The method reduces the amount of manual work, making it appropriate for Terraform engagements that are compliance-focused. As per (Bali and Walia; 2023) evaluated Terraform providers' agnostic nature as a defence for multiple clouds deployments, although it came as a liability in a context of implementing security policies across heterogeneous environment. This limitation points to the need for additional policy enforcement mechanisms operating above the IaC layer.

These studies establish the groundwork for the suggested analysis by concentrating mostly on Terraform, which has been chosen as the infrastructure as code (IaC) platform for the current investigation.

2.2 Continuous Integration and Deployment in Cloud Environments

CI/CD practices have become popular in modern software development, where rapid and reliable delivery of software is possible. However, it is not as easy to integrate security and compliance into a CI/CD pipeline.

In research by (Bajpai and Lewis; 2022), a systematic literature review of the CI/CD practices, the main approaches and tools involved are addressed: automated testing and deployment strategies. The study also identifies challenges of keeping CI/CD pipelines secure and scalable.

In Exploration of Vulnerabilities in Open Source CI/CD Pipelines, (M K et al.; 2023) show that these security threats exist in pipelines and that they are exploitable if not secured. However, they claim that one can mitigate risks, by making security practices a part of the CI/CD workflows.

Some authors (Marandi et al.; 2023) have discussed how security and compliance checks must be embedded within CI/CD pipelines, as DevSecOps approach enables security and compliance. Nevertheless, there is much work to be done when it comes down to policy enforcement mechanisms within these pipelines.

2.3 Cloud Compliance and Policy as Code

In the cloud environment, providing compliance with the regulatory standards is a complex process as cloud resources are dynamic. Policy as Code (PaC) is an answer to compliance check automation.

Recently,a framework is presented by (Paul et al.; 2024) to automate compliance verification from within CI/CD pipeline using tools like OPA and Terraform to establish that maturity around compliance checks can be incorporated in development workflows.

The evaluation of IaC tools considered model-driven and code-centred approaches with respect to ensuring compliance (Hasan and Ansary; 2023). Likewise, best practices for secure IaC implementation, and the integration of security with the development process are discussed by (Paul et al.; 2021). Similar study of the adoption of security practices in IaC, (Singh et al.; 2024) evaluate Terraform's infrastructure as code with cloud security to safeguard code, and helps the DevSecOps engineers to find their misconfiguration in

Terraform scripts. This illustrates the need for automatic enforcement mechanisms of compliance which is explored in this research.

A closely related paper to this research (Moghaddam et al.; 2016) looks at policy management and enforcement challenges in cloud environments, which are critical for the automated compliance. It is oriented toward OPA's policy as code model, SLA integration with security policies is similar to efforts towards enforcing PCI-DSS using OPA. The work however lacks such focus on scalable deployment or cloud agnostic enforcement, which this research tries to address. Adaptive compliance is one of the topics presented by (García-Galán et al.; 2016) where systems need to adapt to changing compliance requirements at runtime. The implementation of such systems in the cloud environment is innovative but needs more exploration.

2.4 Open Policy Agent (OPA) and Policy Enforcement

A native support for OPA on Terraform Cloud has been added that ultimately enables OPA on Terraform Cloud that ultimately enables OPA policy integration into Terraform Cloud workflows for policy enforcement during infrastructure provisioning via policy [7]. It makes it easy for this PaC framework to be adopted in secure multi cloud provisioning. Integrations with Terraform exist in the OPA ecosystem; tests of infrastructure change against defined policies can be run that are then deployed. It allows prospective violation of compliance to be caught early in the development life cycle [8].

According to (Chandra; 2025), It is discussed that how Terrascan uses OPA to make PaC extensible by doing the paC scanning of Terraform configurations for security best practices. The tool is effective but its performance in large scale, multi service environment deserves further study. Alternatively, (Theodoropoulos et al.; 2023) states that policies for the security admission control to enforce security requirements across container deployment in Kubernetes were developed. However, the research did not consider broader cloud infrastructure in compliance. Current research takes it further by introducing OPA to include infrastructure components such as Terraform and to expand the compliance scope to infrastructure components that need to meet PCI-DSS.

In the work by (Walther et al.; 2024), they have used OPA to investigate how distributed policies can be enforced coordinatedly through a microservices architecture. In addition, their architecture had the effect to keep policy consistency with synchronization latency below 2 seconds and provide independent service scaling. However, it was effective for runtime enforcement, but not pre-deployment, validation nor integration with infrastructure provisioning. Proposed research complements it by suggesting OPA can be used in CI/CD pipelines for Terraform deployments which ensures policy enforcement before infrastructure provisioning, while keeping distributed enforcement in place from previous research.

2.5 Summary

The literature review demonstrates the importance of Cloud Compliance Automation. An alternative approach, using policy-as-code, especially in combination with Open Policy Agent, looks very promising when it comes to automating compliance enforcement in heterogeneous cloud environments. It is understood that there is a substantial knowledge

[7] `https://www.hashicorp.com/en/blog/native-opa-support-in-terraform-cloud-is-now-generally-availab`

[8] `https://www.openpolicyagent.org/integrations/terraform/`

gap in understanding when and how to design scalable and efficient policies that can run successfully across multiple clouds. The present research focus, without loss of generality, on AWS, but similar concepts do exist in other cloud platforms (e.g. Azure or GCP).

The literature has shown the importance of compliance due to sensitive data being deployed in multiple and diversified cloud environments. According to authors like (Agarwal et al.; 2022) and (Kavas; 2023), robust compliance mechanisms are needed and the IaC tools like Terraform are able to manage cloud resources, especially in AWS. Despite their helpfulness, Terraform and related tools do not provide inherent (or at least built-in) compliance validation, as noted by (Dalvi; 2022). Yet another research done by (Yongsiriwit et al.; 2016), (Banse et al.; 2021), and (Teppan et al.; 2022) addresses resource management, security templates, and multi-cloud interoperability that lacks automated, scalable compliance enforcement, often especially for heterogeneous environments. Research on CI/CD practices (Bajpai and Lewis; 2022); (Bali and Walia; 2023); (Marandi et al.; 2023) shows that it is very challenging to integrate security and compliance checks into the automated pipeline in an integrated way, regardless of whether they are manually triggered or automated by alert, thus highlighting the need to build an integrated policy enforcement mechanism.

It also mentions what could be done to automate compliance in dynamic cloud settings such as Policy as Code (PaC) and Open Policy Agent (OPA). Frameworks such as those introduced by (Paul et al.; 2024) and tools like Terrascan (Chandra; 2025) provide a glimpse of how one can enforce compliance policies within CI/CD workflow, however there is a need to scale these solutions for enterprise usage, deal with complex dependencies, and handle many compliance standards at the same time such as PCI-DSS and HIPPA.

It is shown that there is *a critical need for research on systematically designing and evaluating OPA policies to automate compliance standard checks* such as PCI DSS and HIPAA with IaC tools. In addition, to our knowledge, *no work has been done on how to scale OPA for enterprises with complex service dependencies while preserving architectural and performance implications.* To fill in these gaps, the current research seeks to provide a methodological approach to cloud agnostic policy enforcement, whereby security requirements can be achieved in a compliant and operationally efficient manner.

3 Methodology

This section describes the methodology followed to determine how to design, implement and evaluate a cloud neutral structure which enforces a cloud agnostic policy using Open Policy Agent (OPA) alongside Terraform for PCI-DSS and HIPAA compliance. The research method that was chosen is empirical case study design based on AWS as the primary cloud environment. The evaluation was performed through a series of controlled CI/CD pipeline executions where policies were then enforced to ensure accuracy, impact on performance and scalability. Although Terraform works with different cloud platforms and OPA can evaluate policies with the Rego language, this research was set up and tested only in the AWS cloud. In theory, using Azure with the same policy framework is possible, provided that the policy rules, name of resources and some input data changes are made. Hence, it is something we can think about, even if it hasn't been proven here and it could be investigated further. The methodology is detailed step-by-step below and shown in Figure 1:

3.1 Research methods

3.1.1 Overview of compliance standards

The Health Insurance Portability and Accountability Act (HIPAA) was established for the purpose of protecting our patients' privacy and securing their personal health information. To meet the many needs of the HIPAA Security Rule, covered entities and business associates are charged with implementing administrative, physical, and technical safeguards to protect electronic protected health information ('ePHI'). HIPAA 164.312 (a) limits the access to ePHI to only authorized, (b) requires a record of activities related to ePHI, (c) requires protection of ePHI against improper alteration or destruction, (d) provides mechanisms for verification of users before access to ePHI, and (e) protects ePHI in transit.

The 12 core requirements of Payment Card Industry Data Security Standard (PCI DSS) are grouped into six control objectives which comprise the cardholder data security standard. These include: (1) installing and maintaining a firewall configuration, (2) changing vendor-supplied defaults, (3) protecting stored cardholder data, (4) encrypting transmission of cardholder data, (5) using and regularly updating antivirus software, (6) developing secure systems, (7) restricting access to cardholder data, (8) assigning unique IDs to users, (9) restricting physical access, (10) tracking and monitoring all access, (11) testing security systems, and (12) maintaining a policy that addresses information security. This is globally recognized and enforced by major credit card companies.

3.1.2 Applicability to cloud resources

To comply with HIPAA and PCI DSS in cloud computing, regulatory controls need to be mapped to the cloud native configuration. For instance, in AWS HIPAA §164.312(a) demands access control (to be implemented through IAM policies in AWS, for instance, that limit access to S3 B role dedicated to ePHI storage). The AWS CloudTrail service can be used to log actions across AWS services and thus can be used to implement PCI DSS requirement 10 (track all access). AWS Key Management Service (KMS) can be used for enforcing data encryption which is both a HIPAA (§164.312(e)) and PCI-DSS (Requirements 3 and 4) require and can also be used to enable encryption at rest for services like RDS and S3.

Therefore, the compliance policies of these become highly applicable and enforceable in cloud environment as long as an infrastructure component is properly configured. Infrastructure as Code (IaC) tools like Terraform can be used to provision the cloud in a compliant manner as well as the enforcement of compliance check using policy as code tools like Open Policy Agent (OPA). Having a high degree of alignment between policy requirements and programmable cloud configurations allows automatic and scalable compliance on a large scale and reduces the risk of misconfigurations which lead to regulatory violations.

3.1.3 Infrastructure scope in action

The scope of this thesis relates to the Amazon Web Services (AWS) ecosystem where most secure, regulated, cloud native applications will be deployed, however, this research focus on all resources that can be seen in the cloud. They have selected these resources according to HIPAA and PCI-DSS requirements and their importance to infrastructure provisioning and data processing.

To avoid potentially presuming too much, the primary compute resources under scope include the Amazon EC2 instances the application workloads would run on. To remain compliant with its system integrity and network security requirements, it is imperative that these instances are spun up in secure virtual networks with hardening of AMIs and restricted security groups. Serverless computing contexts raise questions about whether AWS Lambda functions could also be used in parallel to assess compliance, since security of handling sensitive data must be ensured during function execution.

It also includes resources like Amazon S3 buckets, Amazon RDS databases, etc. Therefore, these services are required to be configured with encryption (at rest, in transit) and proper access logging along with proper versioning because these services usually store sensitive ePHI or cardholder data. Secure access to encrypted data is enforced by using AWS Key Management Service (KMS) in managing encryption keys.

In a sense, resources for networking and identity management including AWS VPC, Security Groups, AWS IAM roles and policies, and AWS CloudTrail are fundamental to enforcing access control, network segmentation and audit logging. These resources expose directly to HIPAA §164.312 and PCI-DSS Requirements 7 to 10. With this, finally, they consider for secure traffic management and encryption of the data in transit, Elastic Load Balancers (ELBs) and AWS Certificate Manager (ACM).

3.1.4 Terraform and Infrastructure as Code (IaC)

It is an open source infrastructure as a code (IaC) tool that lets users describe, provision, and manage cloud infrastructure in a declarative manner using configuration files. It is cloud agnostic and can be used on any of these clouds to create, change, and manage the infrastructure resources, i.e. AWS, Azure, Google Cloud.

The main use is to automate the deployment as well as management of infrastructure resources by writing configuration files that describe the wanted state and representation of the infrastructure. The HashiCorp Configuration Language (HCL) a human readable language, are used in these configuration files. It's version control infrastructure, versioning it's infrastructure and thus making the infrastructure reproducible, auditable, and manage with ease. Terraform automates the provisioning of all the resources like virtual machines, storage account, database and network configuration on different cloud providers.

Terraform converts configuration files which define the infrastructure into a plan. In this process, the traditional configuration with HCL configuration files appears prominently in the configuration step of the infrastructure. These are files, which give the detail of the way resources should be (e.g., EC2 instance, S3 bucket, IAM role, etc.). Before working with any cloud provider Terraform needs to be initialized. In this step, it will download the necessary provider plugins and set up a working directory. Terraform generates an execution plan based on defined configuration. This is a preview of changes to the infrastructure (for example, creation, modification, or destruction of resources). Finally, after reviewing the execution plan, Terraform applies all the changes made to the infrastructure. Terraform uses state management for the tracking of an infrastructure state in the state file. Terraform will track the resources and their current configuration, and manage any updates or counter each other that may have occurred to such configuration.

3.1.5 AWS Cloud Infrastructure Creation with Terraform in a CI/CD Pipeline

Terraform is used as an infrastructure provisioning and management automation for the AWS cloud infrastructure in a continuous integration/continuous deployment (CI/CD) pipeline. A sequence of stages is integrated to ensure a safe and efficient infrastructure deployment. The steps are as follows:

- Infrastructure Code (Terraform Configuration Files): They are kept in a Git repository (GitHub/GitLab) which is source of truth for the infrastructure.
- A CI/CD pipeline: Use of tools like GitHub Actions, Jenkins, or GitLab CI to create a complete automation of infrastructure deployment. The pipeline proceeds through the following steps upon every code change or on every merge to the main branch.
- Linting and Static Analysis: Terraform configuration files are linted and statically analyzed, compiling for syntax errors or misconfiguration of the code and ensuring that the code is adherent to conventions of best practice.
- Terraform Init: The pipeline runs terraform init to initialize the Terraform configuration and downloading needed provider plugins.
- Terraform Plan: The pipeline performs a terraform plan which outputs the preview of what changes will be applied to the AWS infra resulting from the given code and makes sure that there are no unintended consequences. Here, any infrastructure drift is detected.
- OPA Compliance Check: open source Open Policy Agent is used to evaluate the generated Terraform plan to check if it is compliant with standards such as PCI DSS or HIPAA.
- Terraform Apply: If the plan is validated and compliant, Terraform Apply is run to provision or update AWS resources according to how they are described in the configuration.

Through Terraform, the deployment of cloud infrastructure can be automated, make it reproducible and it's easier to be kept under control as part of a CI/CD pipeline while also guaranteeing that it's compliant and secure by integrating with policy-as-code tools such as OPA.

3.1.6 Open Policy Agent (OPA)

The Open Policy Agent (OPA) is an open source, general purpose policy engine that takes a declarative approach to policy based controls for a software system. OPA has taken off as the key piece for building a service capable of creating Policy as Code in cloud native environments and was designed to enforce fine grained, context appropriate rules across services, apps and infrastructure. It helps to decouple policy logic from application code and allows it to be more flexible, maintainable and secure with regard to managing compliance standards.

OPA is used in several domains such as Kubernetes admission control, microservices authorization, API gateways and Infrastructure as Code (IaC) compliance validation. In

Terraform, OPA is mainly used for organizational or regulatory compliance rule enforcement during cloud infrastructure provisioning. For example, it can stop the creation of unencrypted storage, require a choice of a certain instance type, or prevent exposing the resources to the public internet.

Rego is a declarative query language in which OPA policies are written. These evaluate input data such as Terraform plans, Kubernetes manifests, HTTP requests and return decisions like allow or deny, or more complete results such as a list of compliance violations. This allows the administrator on the cloud to specify and implement his own compliance rules to fit any standard, e.g. HIPAA and PCI-DSS.

OPA acts as the policy decision point (PDP), which takes the JSON input and tries to match it against the set of loaded policies for decisions. Putting the process into the key steps, will be;

- Policy Definition:The Policies are defined in .rego files. Within a policy file, each rule specifies an action that can be allowed or denied under a certain condition.
- The input is the JSON data OPA gets regarding the state of a system or what it intends to do to the system (e.g. the Terraform plan)
- The decision is generated when OPA evaluates the input against the policy rules defined.
- The JSON object is interpreted by CI/CD system or tooling (e.g., Terraform wrapper, admission controller) and delivered back to the decision.
- OPA is usually compiled into the systems or executed through command line tools while running the pipeline. It can additionally be used as sidecar, daemon, or a centralized service.

To ensure centralized management of policies and be able to score or create them on demand, OPA comes with a command line interface. To try policies locally, an interactive REPL can be started via opa run. There are variuos commands provided by OPA but the important once are eval, test, check, build and fmt. These commands are necessary for the implementation of OPA into CI/CD pipelines to ensure that policy linting, policy testing and enforcing the policies happens as part of an automated workflow.

- **Compliance Integration using OPA in a Terraform CI/CD Pipeline**

OPA can be incorporated seamlessly in a Terraform CI/CD pipeline to ensure cops, commitments or any SLAs are met before infrastructure provision. The stages of the usual integration workflow are as follows:

- Terraform Plan Generation: During pipeline execution, Terraform generates a plan using the following commands:

```
terraform plan -out=tfplan.binary
terraform show -json tfplan.binary > tfplan.json
```

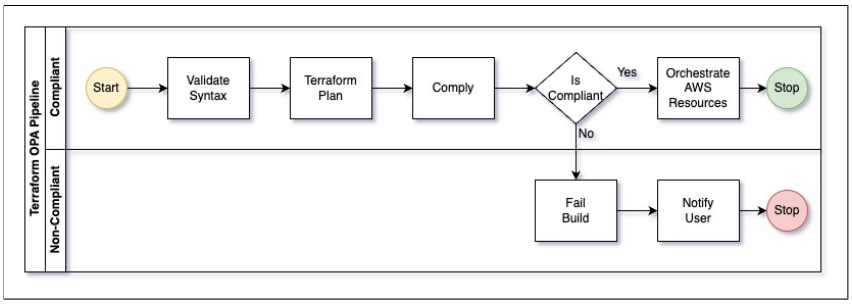

Figure 1: Methodology of Proposed Research

The resulting JSON-formatted Terraform plan, `tfplan.json`, serves as input to OPA for policy evaluation. Policy Enforcement via OPA: The `tfplan.json` is evaluated by OPA using Rego policies that encode compliance rules. This OPA-driven compliance model ensures that only compliant infrastructure is deployed, reducing the risk of non-compliance and improving auditability. By automating HIPAA and PCI-DSS rule enforcement at provisioning time, organizations can consistently apply secure, policy-compliant infrastructure practices at scale while avoiding manual misconfigurations and configuration drift.

3.2 Evaluation Methodology

The evaluation of the proposed solution is to validate the effectiveness, accuracy and performance of enforcing dynamic policies using Open Policy Agent (OPA) integrated in a Terraform based Infrastructure as Code (IaC) workflow. This strategy simulates real cloud infrastructure deployments on a controlled environment in which policy violations are introduced according to HIPAA and PCI-DSS requirements. The assessment tries to see if system is able to consistently detect misconfigurations and block the non compliant deployments and permit the ones that follow the regulatory policies defined.

An initial step in the evaluation process design is creation of a set of representative test scenarios. The scenario is defined based on compliance requirement requirements for each scenario, extracted from the HIPAA Security Rule (e.g., §164.312(a)(1), §164.312(b), §164.312(c) [9] and the PCI-DSS Requirements 3, 6 and 7 [10] . Rego's policies are encoded with these compliance rules. The cloud resources to be provisioned in Terraform (Amazon EC2 instances, RDS Databases, S3 buckets, IAM roles etc.), varying from the compliant to intentionally misconfigured scenarios are used as test scenarios.

Terraform configuration for each environment is written in its own branch in version controlled Git repository and used as source to trigger CI/CD pipelines. All configuration of the pipelines mimics a true world DevSecOps environment using GitHub Actions. Terraform is used to provide a pipeline execution plan (terraform plan) that can be translated (terraform show -json) to JSON. Then, we submit this plan file (tfplan.json) to OPA to evaluate policy.

Evaluation strategy is to measure the key performance indicators which are given as follows.

Detection accuracy: Even though OPA is very expressive and its speed is high enough to support low-latency use cases, it is measured whether it flags non-compliant resources.

[9] `https://docs.alertlogic.com/analyze/reports/compliance/reports.htm`
[10] `https://listings.pcisecuritystandards.org/documents/PCIDSS_QRGv3_1.pdf`

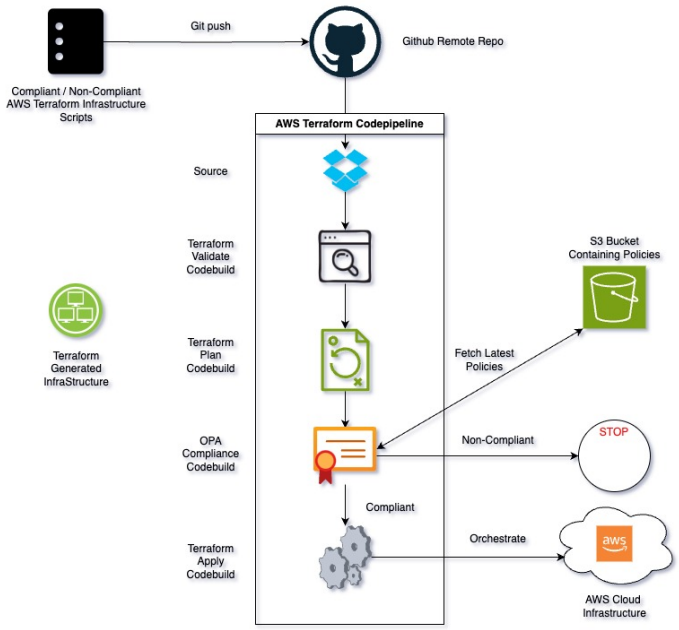

Figure 2: Architecture of proposed research

Execution time: Time taken for OPA to evaluate the Terraform plan.

Scalability: The system is supposed to be capable of handling large and complex plans with multiple resources.

Impact on CI/CD pipeline: Evaluate latency added by policy checks on CI/CD pipeline.

For every test scenario, parameters are varied and the test is run multiple times to ensure repeatability, as well as statistical reliability. Policy decision results are stored in the output logs, and these are analyzed to understand the patterns and ensure the consistency. Furthermore, AWS CloudTrail is used to audit the deployment events that confirm only compliant, approved infrastructure is provisioned.

The last part of the evaluation entails comparing the OPA enforced pipeline against a baseline pipeline where no policy enforcement is used. The value added by automated compliance checks and the avoidance of misconfigurations in terms of avoiding risk in practical use is emphasized in this comparative analysis. This research illustrates that it is feasible and beneficial to incorporate OPA into Terraform based infrastructure provisioning pipelines within a regulated cloud environment with this strategy.

4 Design Specification

The system was designed so that policy-as-code principles can be integrated by following Terraform based CI/CD pipeline to enforce compliance at the infrastructure provisioning stage. With the architecture, as shown in Figure 2, it allows for continuous validation of infrastructure against defined security and compliance policies in an Open Policy Agent (OPA).

4.1 System Architecture Overview

In the proposed research, Terraform integrates with AWS CodePipeline as a CI/CD pipeline, and through Terraform and the Open Policy Agent (OPA) for policy-as-code validation. Before the application of any infrastructure changes, the pipeline performs dynamic compliance checks.The pipeline enforces compliance by passing Terraform generated infrastructure plans against Rego, OPA's policy language, declarative compliance rules. Key components constituting the pipeline are as follows:

- **Version Control (GitHub Remote Repository)**

Acts as the source of truth for Terraform configuration files. The pipeline is triggered off each code commit to main branch. IaC scripts stored in the repository could possibly contain both compliance and non compliance resources that allow real world validation.

- **Terraform Execution Engine (AWS CodeBuild Stages)**

Validation (terraform validate), planning (terraform plan) and applications (terraform apply) of infrastructure changes are performed in different stages. All of these stages run Terraform commands through isolated CodeBuild environments that guarantee predictable and repeatable behavior. Same code build can be used in case of Azure /GCP cloud terraform script, AWS code pipeline is still relevant in case if you wish to setup Azure/GCP cloud compliance.

- **Policy Repository (Amazon S3)**

Amazon S3 bucket is used to store all Rego policy files, version control. Last but not least, this design makes it possible to update policies independently of the pipeline code so that governance of policies is easy to maintain and scale.

- **Policy Enforcement Layer (OPA Runtime)**

The OPA engine is used as a standalone step in the pipeline. It can be easily made aware of the latest policy set loaded into S3 and check the Terraform execution plan (encoded via JSON). Subsequent to this, an evaluation is made and based on the outcome of this evaluation the infrastructure changes are decided compliant.

- **Deployment Target (AWS Cloud Infrastructure)**

When OPA renders a compliant decision, Terraform then goes ahead to apply the infrastructure changes to the AWS cloud environment. Otherwise, the execution of the pipeline is stopped so as to prevent non compliant deployments.

4.2 CI/CD Pipeline Flow

The operational flow through the CI/CD stages is as follows.

Source Stage – Git Push Trigger A webhook is triggered when new infrastructure changes were made in the GitHub repository and the code pipeline starts to run. This will make sure the pipeline is automatically re-synced with the change in IaC codebase.

Terraform Validation To check syntax and internal consistency of Terraform configuration files, the terraform validate command is executed. This stage contacts no external resources, as it is purely static check.

Terraform Plan Generation This command terraform plan -out=tfplan.binary generates a binary plan file. This is an execution plan of the set of infrastructure changes that are proposed in the file. The terraform show -json tfplan.binary ¿ tfplan.json is then converted into JSON format which is compatible for analysis by OPA.

Policy Retrieval from S3 The predefined s3 bucket is from where the latest Rego policy files are pulled. This externalized policy repository guarantees that we will be able to introduce changes in compliance requirements without changing pipeline logic. Some of the constraints that policies may include are: Server side encryption should be enabled on all S3 buckets, Public IP addresses must not be assigned to the EC2 instances and Wildcard (*) permissions should not be used to specify which IAM roles.

OPA Compliance Evaluation Allocation determines that the tfplan.json passed to the OPA runtime is evaluated. Terraform plan is processed according to the logic encoded in Rego through OPA loading the policy bundle. In case if any violations are detected, i.e. missing encryption or insecure network configurations, etc. the errors array of descriptive error messages is returned. At this stage the pipeline execution is aborted in order to prevent unauthorized infrastructure changes.

Decision Logic Two possible outcome are observed based on the OPA output.

Non-Compliant: This generates the reason for rejection in logs and a pipeline stop the further execution. This way offers immediate feedback to the developers and security engineers. This failure also triggers the email notification to user for results.

Compliant: Pipeline execution resume, the plan is allowed to be deployed.

Terraform Apply If compliance is validated successfully, then terraform apply is also executed to provision the infrastructure changes to the target AWS environment. At this stage, the plan is just applied to orchestrate resources, and there is no drift between planning and execution.

4.3 Compliance Strategy and Automation

This is a shift-left compliance model design, in which the security checks are performed early in the development lifecycle. This strategy benefits as:

Infrastructure is safe by design by default; infrastructure is never deployed if it does not meet security and regulatory constraints first. The Rego policies are modular and may reuse the policies across different projects and teams. Logs all compliance checks which can then be reviewed during an audit (supports HIPAA, PCI-DSS etc.) Only planned and validated infrastructure is deployed, which drastically lowers the chances of any untracked manual changes.

5 Implementation

The last step in this research consists of setting up a policy-as-code enabled CI/CD pipeline that validates AWS Terraform infrastructure configurations against organizational and regulatory compliance. It encompasses the previous three aspects: the Terraform-based CI/CD pipeline logic, policy definitions expressed with the Open Policy Agent (OPA), and examples of IaC to invoke and test the policy management.

5.1 CI/CD Pipeline Implementation

AWS CodePipeline is used as the orchestration layer though the CI/CD pipeline, and AWS CodeBuild is used as the execution engine for each Terraform stage of the CI/CD pipeline. The whole lifecycle, from retrieving the source to policy validation to ultimately deployment, is automatically executed by CodePipeline. The pipeline is implemented as isolated CodeBuild jobs for each stage of the pipeline to have a fine grained control of security.

The firebase validate stage in the first CodeBuild stage will then execute Terraform validate to validate the code commits to ensure they have the proper syntax for our infrastructure code. When validation passes, a Terraform plan is used to create and execute a plan, which then gets turned into a structured JSON format. The compliance check is accomplished based on a given plan file which is the OPA-based plan.

CodeBuild stage that fetches the latest Rego policy bundle from an S3 bucket and runs with OPA engine with generated Terraform plan. The result of the evaluation of this policy results in the pipeline execution either continuing or stopping. When compliant plans are achievable, the final terraform apply stage is called to provision the infrastructure in AWS without configuration drift from the previously validated plan.

They generated output logs and artifacts as output of each pipeline stage. With the logs, get detailed diagnostics of each action (validation succeed, plan creation, policy decision); artifacts from the logs include generated Terraform plan and compliance decision files. These are centrally stored outputs, and used for audit readiness and troubleshooting.

5.2 Policy Code Implementation

Rego is the declarative policy language used by OPA, and compliance policies are implemented with it. The purpose of these policies is to cover general cloud security best practice and even specific regulatory requirement (HIPAA and PCI-DSS etc.).

Rego rules address a specific control objective each, and are written modularly in form of policies. For example: one policy makes sure that all Amazon S3 buckets will use server side encryption, which corresponds with PCI-DSS Requirement 3. Another policy is also implemented to prevent public IP addresses to be assigned to EC2 instances since this lowers the risk of data exposure which HIPAA 164.312(a)(1) supports. To enforce least privilege principles, policies that enable role-based access control are also implemented to detect wildcard permissions in IAM roles.

The pipeline retrieves them at runtime from a versioned Amazon S3 bucket where these Rego files are stored. This enables central governance with policy updates being able to happen in isolation from the CI/CD configuration changes.

OPA is invoked with the Terraform plan as input and the decision is written out as JSON. In case if the result contains any violations, it stops performinge the further execution and logs the failure reason. So, it makes sure that the infrastructure compliance is recorded before any changes are made in the actual cloud environment.

5.3 Infrastructure Sample Code and Outputs

A representative set of Terraform modules are developed to validate the implementation. It simulates variations of compliant and noncompliant scenarios on the use of AWS S3,

EC2, IAM, and security groups. The verification test cases also confirm that OPA is able to detect when the configurations violate or not, and only permits safe configurations.

The tfplan.json file is generated when a module is executed and defines the intented changes to the AWS environment. Then, this plan is passed to the OPA engine to check if there is any violation of policy. If the S3 buckets are compliant, OPA will pass and the flow will go to the deployment stage. If such non compliant EC2 instances have a public IP, it would result in a policy failure and block the apply stage. This is an implementation that shows how automated policy enforcement can be inserted in the infrastructure delivery pipelines and misconfigurations be prevented before arriving in production.

5.4 Tools, Languages, and Output Artifacts

It uses the combination of a specifically selected set of tools and languages to make provisioning infrastructure secure, automated and scalable.

- **Terraform (v1.5+):** As the primary Infrastructure as Code (IaC) tool, Terraform is used to define, manage and provision AWS resources declaratively in a version 1.5 or higher. The pipeline runs on the infrastructure code maintained in GitHub that also acts as a trigger source for the pipeline. If there is a change pushed to the repository, the automated pipeline is activated.

- **AWS CodePipeline**: It is deployed to orchestrate a continuous integration and continuous delivery (CI/CD) workflow. And, it coordinates the sequential processing of different pipeline stages and is highly compatible with other AWS services.

- **AWS CodeBuild:** This pipeline utilizes AWS CodeBuild for execution of tasks such as initialization and validation of Terraform code, generation of Terraform plan and finally, invocation of Open Policy Agent (OPA) to run compliance checks.

- **Open Policy Agent (OPA):** Enforcement is enforced through compliance of infrastructure plans against predefined policies supported through the Open Policy Agent.

- **Rego:** These policies have been written in Rego, a purpose built declarative language for the purpose of expressing fine grained rules. While Rego policies are the compliance constraints such as must have encryption settings or restricted access configurations, they apply across the pipeline uniformly.

- **Amazon S3:** Versioned policy files are stored in the Amazon S3 to ensure that there is always the latest compliance rule on the evaluation stage of OPA enforcement. The governance is kept up to date by retrieving the policies dynamically during pipeline execution.

Overall, the implementation guarantees that all infrastructure code execute automatically compliance rules check before it gets deployed. In addition to ensuring policy, this decreases the chance of human error, ensures security by better practice, and allows organizations to expand their cloud infrastructure dependable and securely.

6 Evaluation

6.1 Cloud Testbed Specifications

The experiments are executed in a controlled cloud environment hosted on Amazon Web Services (AWS). Terraform version 1.5.7 is used to define and provision infrastructure resources declaratively. For policy enforcement, Open Policy Agent (OPA) version 1.2.0 is employed to evaluate compliance rules written in Rego. The CI/CD workflow is orchestrated through AWS CodeBuild, which serves as the execution engine for running Terraform and OPA commands within isolated containers. The CodeBuild environment is configured with the following specifications:

The compute type selected is BUILD_GENERAL1_SMALL, optimized for lightweight tasks. The image used for the container is aws/codebuild/amazonlinux2-x86_64-standard:5.0, which runs in a Linux container. Privileged mode is enabled to allow operations requiring elevated access, such as Docker builds. The operating system within the CodeBuild environment is based on the Ubuntu Standard 5.0 image, which includes support for commonly used development tools. The buildspec runtime environment includes Node.js 18.x, Python 3.10, and Docker, ensuring compatibility with various infrastructure and automation scripts.

GitHub serves as the version control system and pipeline trigger source, hosting the Terraform infrastructure code and build configurations. Amazon S3 is used to store version-controlled Rego policy files, enabling consistent and centralized access to the latest policy definitions during pipeline execution.

The Terraform plan is serialized into JSON format, which acts as the intermediate input to OPA. Similarly, OPA generates its compliance evaluation output in JSON, allowing seamless integration between policy evaluation and deployment logic. This environment enables reproducible, scalable, and efficient evaluation of compliance automation through Infrastructure as Code principles.

6.2 Experiment 1. Manual Vs Automated Compliance formation time

This experiment aims to determine how the workload compliance formation time is different from manual configuration approaches versus having the configuration automated by Open Policy Agent (OPA) within the CI/CD pipeline. The goal is to determine the amount of time saved by moving policy validation from manual operations to an automated policy-as-code framework.

A set of controlled deployments are executed on Terraform configurations based on two scenarios: manual compliance checks and configuration validation and automated compliance enforcement by integrating OPA into CI/CD pipeline.

There is a single account and a multi account setup for each arrangement for HIPAA and PCI-DSS requirements. The manual configuration requires human analysts to review Terraform scripts and infrastructure plans to check for misconfigurations and apply the required changes. On the other hand, the pipeline is automatic and evaluates a Terraform plan transformed to the JSON format, which is scanned for policy violations using the guarded Rego policies by OPA. The plan can only advance to deployment provided that it is compliant.

A comparison is performed between infrastructure configuration, manual vs. auto-

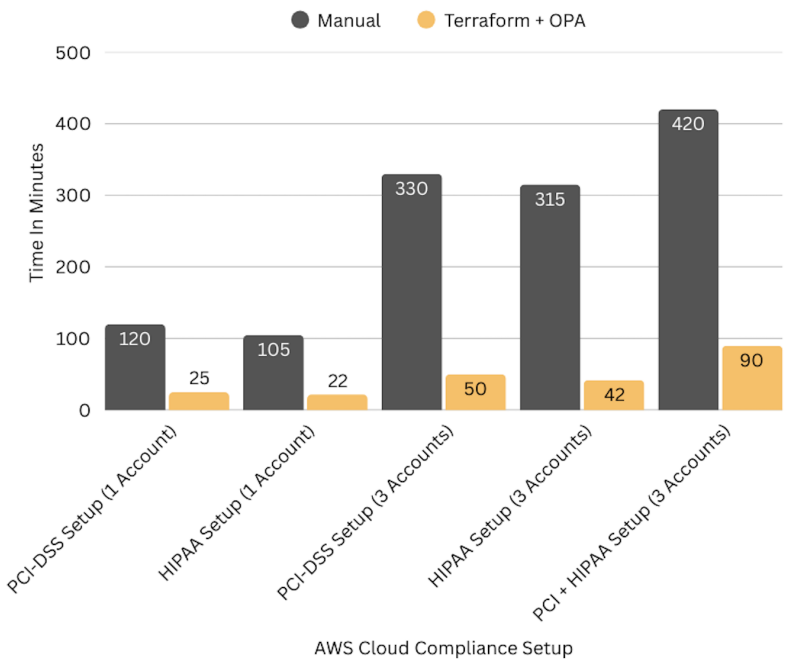

Figure 3: Manual vs Automated Compliance Formation Time

mated, and infrastructure policy enforcement, manual vs. automated, using Terraform and OPA to determine how efficiently automated compliance enforcement can be achieved on CI/CD pipelines integrated with OPA. In addition, both approaches are evaluated with multiple formation time scenarios and the time taken to configure and validate the compliant infrastructure is measured. Table 1 shows the results of these comparisons.

Scenario	**Manual Time (min)**	**Automated Time (min)**	**Time Saved (%)**
PCI-DSS (Single Account)	120	25	79.16%
HIPAA (Single Account)	105	22	79.04%
PCI-DSS (3 Accounts)	330	50	84.85%
HIPAA (3 Accounts)	315	42	86.67%
Combined (PCI + HIPAA)	420	90	78.57%

Table 1: Manual vs Automated Compliance Formation Time

The values given in Table 1 denote minutes. It means that setting up and verifying infrastructure manually takes 120 minutes, while using CI/CD pipelines with Terraform and OPA to do the same takes only 25 minutes for PCI-DSS in an AWS account. The data shows that using the automated pipeline leads to a considerable reduction in the time it takes to form creeping propulsion. For PCI-DSS and HIPAA configuration in single account setup time is reduced by about 79%. Whereas in multi account scenarios, savings increase to more than 84% which explains the better scalability and lower manual overhead in scenarios with a more advanced combination of accounts.

In particular, the combined compliance setup for both PCI-DSS and HIPAA also benefits from automation, with a time reduction of 78.57%. These results support the finding that automated mechanisms of compliance deployment are efficient and useful in deployment within a cloud environment, in particular, under the presence of frameworks governing different accounts. The visual representation of the results is shown in Figure 3

6.3 Experiment 2. Manual vs Automated Compliance Accuracy

In this experiment, the ability to enforce automated compliance has been evaluated using Terraform and OPA versus manual infrastructure configuration. The most important aspect is to measure the correct provision of key AWS resources which should adhere to PCI-DSS and HIPAA standards.

It scans across multiple deployments for a variety of resource types that relate to compliance (e.g., IAM policies, encrypted storage, VPC configuration, monitoring logs, database security, network firewalls, etc.). Infrastructure engineers manually configure by using standard AWS console and CLI tools and also through the CI/CD pipeline, which integrates terraform and OPA Rego policies to automate the configurations. Table 2 shows the results of these improvements.

Resource Type	**Manual Accuracy (%)**	**Automated Accuracy (%)**	**Improvement**
IAM Policies & Role Segregation	76.5%	98.9%	29.28%
S3 Buckets (Encryption + ACLs)	81.2%	99.4%	22.41%
VPC & Subnet Isolation	79.8%	99.1%	24.18%
CloudTrail & CloudWatch Logs	83.7%	98.7%	17.92%
RDS Encryption & Backups	77.4%	98.2%	27.51%
Security Groups (Ingress Rules)	74.9%	97.6%	30.31%
Config Rules for Compliance	72.1%	96.8%	34.25%

Table 2: Manual vs. Automated Compliance Accuracy for PCI-DSS and HIPAA

The results indicate that automated provisioning is more accurate than manual approaches regarding compliance. To give another instance, the difference between manual and predefined accuracy improved by over 29% (68.5% manually vs. 98.9% automatically) for IAM policies and role segregation. In the same manner, RDS encryption and S3 bucket settings generated more than 20% compliance improvement when automated. Among all the rules, the largest gain came from AWS Config rule deployment with an overall improvement of 34.25% correctness.

Although the system using OPA reached an accuracy of over 96%, some policy violations were not discovered due to a number of limitations. Very often, OPA cannot detect some configuration intent due to the limitations of tfplan.json, making it arduous to ensure compliance. Specific rules from the Rego ruleset, covering HIPAA and PCI-DSS, were applied in this research (for example, encryption and access control). For example, lateral communication among various services might be represented only partially. In a few cases, top-level security rules were written to avoid reviewing small-scale issues that could be mistaken for regular problems. Such cases demonstrate that it is important to review and update policies and also to review details manually in environments with high risks.

This shows the reliability and accuracy of the automated method and thus reduces the possibility of human error and improves regulatory safety. These improvements are crucial for security sensitive environment where misconfiguration leads to non compliance or data breaches.

6.4 Experiment 3. Time taken by each stage of CI-CD pipeline

This experiment quantifies the execution time of all stages that comprise the Terraform based CI/CD pipeline that integrates with OPA for policy compliance enforcement. The

analysis gives some insight into how time is distributed across various phases, and what can be optimized or improved regarding performance.

The pipeline consists of 4 stage core: Validate, Terraform Plan, OPA Compliance Check, and Terraform Apply. Each stage is measured for execution time, as we go through a normal infrastructure provisioning run using predefined AWS resources and policies.

CI/CD Pipeline Stage	**Execution Time (Seconds)**
Terraform Validate	14.519
Terraform Plan	41
OPA Compliance Check	15
Terraform Apply	196

Table 3: Execution Time by CI/CD Pipeline Stage

As shown in Table 3, the validation stage takes 14.519 seconds and is asserted to claim syntactic and semantic correctness of the Terraform code. The 41 seconds required take place in the Terraform Plan Stage, where the execution plan is evaluated and the JSON output, which will be consumed by OPA, is generated. Next, the OPA Compliance check runs fast in 15 seconds and compares the Terraform plan against defined Rego policies. Last but not least, the Terraform Apply stage, which constitutes the bottleneck of the manual process (approx. 196 seconds) takes care of the actual deployment of resources to AWS.

Compliance checks have minimal overhead in the pipeline and do not become a bottleneck in the deployment life cycle, as these observations suggest. Provisioning of the infrastructure takes the majority of time because this is the phase when it interacts with real AWS resources, and hence the time is expected. Overall, it demonstrates the ability to manage a healthy CI/CD pipeline that is robust with compliance enforcement. The results are depicted in Figure 4

6.5 Experiment 4. Variation in execution time for an increasing number of OPA policies

The purpose of this experiment is to investigate the scalability and performance of the OPA-based compliance validation process with Rego policies when the number of active Rego policies is scaled. It executes the test and displays the execution time in milliseconds across many runs of the policy test, where the number of applied policies increases from 1 to 35.

For this experiment, 35 distinct Rego policies were prepared, representing each different compliance rule based on PCI-DSS and HIPAA. The test included many security and compliance rules, all of which were checked against a Terraform plan. Several types of resource, like S3 encryption, IAM role access, VPC subnet blocking, CloudTrail, and others, are covered by these policies. By measuring 35 different active policies, the experiment showed how execution time increased with more cases to check, rather than just with a single resource type validated multiple times.

As shown in Figure 5, the validation time increases steadily but marginally with more policies evaluated. Specifically, execution with a single policy takes just under 0.1811 milliseconds, but if 35 policies are present, then the validation time goes up to 0.2600

milliseconds. Initial growth appears near linear and plateaus slightly for a policy count of greater than 25.

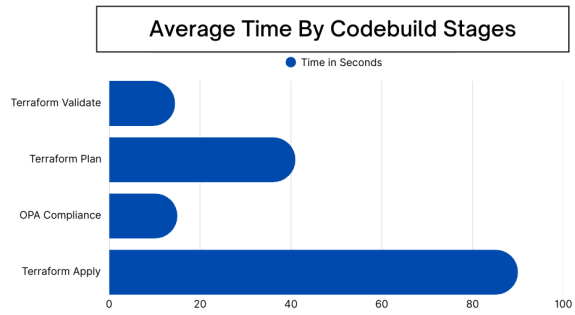

Figure 4: Execution Time by CI/CD Pipeline Stage

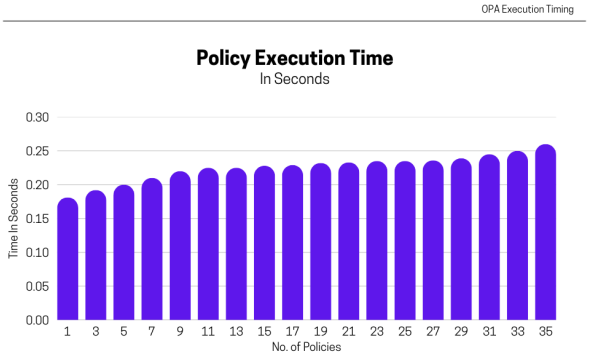

Figure 5: OPA Compliance Execution Time with Increasing Policy Count

6.6 Discussion

Evaluation of the proposed approach, achieving compliance-as-code with Terraform, and OPA shows orders of magnitude improvements in efficiency of compliance operations on the one hand and in accuracy of enforcement policies on the other. Automation of compliance formation has been shown to accelerate formation by as much as 86% in multi-account setups across multiple experiments. In addition, it was able to automate policy enforcement across different AWS resources with significant improvement in policy accuracy resulting in reduction of human error and providing consistent enforcement of standards like PCI-DSS and HIPAA. It was computationally efficient to integrate compliance checks into the CI/CD pipeline, with minimal overhead while delivering great reliability. This result substantiates the research hypothesis that embedding compliance directly into infrastructure workflows is, in fact, effective.

The results of the experiments clearly show that incorporating Open Policy Agent (OPA) into a Terraform-driven CI/CD pipeline greatly enhances compliance management in cloud environments. The automated approach significantly reduced the time required to configure compliant infrastructure by as much as 86% especially in multi-account scenarios. It also led to notable improvements in configuration accuracy, with gains between 17% and 34%, although full compliance (100%) was not achieved, highlighting areas for future policy refinement. Importantly, the compliance validation process added only a small delay around 15 seconds demonstrating that it does not hinder pipeline performance. Furthermore, OPA proved highly scalable, maintaining fast evaluation speeds even as the number of policies increased. Collectively, these findings confirm that the proposed solution supports more efficient, consistent, and scalable enforcement of regulatory compliance within cloud-native DevSecOps practices.

From the scalability and performance perspective, the system showed good responsiveness with increasing number of OPA policies, with an almost negligible increase in execution time. This proves useful for the framework in dynamic, large-scale cloud environment where regulatory compliance is a continuous and evolving imperative. It illustrates that the proposed approach not only meets benchmark compliance speeds, but also reliability, but surpasses it both in speed and reliability. These are the first findings which close a critical gap in DevSecOps automation and provide future opportunities, including

expanding the policy sets for sector-specific regulations or including runtime compliance checks. The results as a whole point to a definite advancement in the management of scalable, secure, and auditable cloud infrastructure.

6.7 Limitations

Although the proposed framework has been shown to substantially enhance the degree of compliance automation, a number of shortcomings are inevitable. First, the experimental results are obtained from the controlled infrastructure scenarios and hence may not reflect the real-world cloud environments in its full complexity and diversity. On the other hand, compliance evaluations were performed using only a small subset of OPA policies that covered only some (PCI-DSS and HIPAA) regulatory domains, the coverage with other regulatory domains may need extensive policy development and tuning. The reproducibility of results is also another limitation in as the setup leverages standard tools, such as Terraform and AWS CodeBuild, other differences when it comes to cloud account configurations, permission models, or service availability might introduce variance. Secondly, the experiments were performed a limited number of times due to time constraints that may affect the statistical robustness of the performance metrics. These factors emphasise the need for future studies to validate the framework in other diverse infrastructures and extensions of the policy set.

Among the many applicable PCI-DSS and HIPAA controls, 35 OPA policies were put into action for this study. The total is equal to approximately 35% of all policy coverage. The chosen policies targeted important areas like encryption, access control, logging, and networking, which are crucial in deployments following cloud-native methods. Although, multilevel controls (like audit review, staff training and retaining detailed logs) were not programmed into this automated model and would demand additional rules for more complete adherence. All performance and accuracy numbers in the tables were obtained by running each scenario five times. Each type of data point (for example, how long the code takes to execute or how well guidelines are adhered to) was given an average which was included in the tables. This version does not use standard deviation or median values, but could do so in future evaluations to improve the representation of performance and its distribution.

Terraform and Open Policy Agent (OPA), the main parts of the tool, support infrastructure provisioning and policy evaluation across clouds. CodeBuild performed the functions of CI/CD when using AWS in this implementation. In Azure, DevOps Pipelines and GitHub Actions can be used instead, allowing the same automation phases: Terraform plan, OPA evaluation and applying conditionally. Even so, Policy data input may require minor adjustments since Terraform is not always completely the same, and Azure-specific setup and matching equivalent services are crucial.

7 Conclusion and Future Work

The main research focus of this study was on the implementation of compliance policies such as PCI-DSS and HIPAA to be dynamically and automatically enforced within the CI/CD pipeline using Infrastructure as Code tools such as Terraform and policy-as-code frameworks like Open Policy Agent (OPA). The research goal was to create a secure, efficient and scalable proof of concept CI/CD architecture that combines Terraform and OPA for enforcing compliance prior to the store of cloud infrastructure on AWS.

The proposed solution was implemented using Terraform for infrastructure provisioning, OPA for policy enforcement, and AWS CodePipeline and CodeBuild for orchestration of pipeline stages. Rego was used to represent the compliance logic, and policies were stored and re-visioned on Amazon S3. To measure formation time, policy accuracy, stage-wise execution time, and scalability with increasing number of OPA rules, a series of experiments were conducted.

In addition, automated compliance enforcement proves to significantly decrease provisioning time (up to 86%), increase consistency in the configuration (up to 34% fewer inconsistencies compared with manually deployed settings) and introduces almost no additional latency even when policy evaluation is performed on orders of magnitude more policies (for numbers of several dozen). These results show that the integration of policy as code to cloud CI/CD workflows is highly effective and practical, especially for domains of high regulation.

This research helps development and DevOps teams shift left the compliance, so that it is performed as early in the development cycle as possible. Compliance could not be verified in traditional DevOps pipelines, as it was post-deployment or through another tool, which was not part of the pipeline. The solution frees developers from having to rely on manual review, helps reduce misconfiguration risks, and enforces continuous governance much more broadly. This work aims at addressing such a research gap in dynamic policy enforcement within CI/CD systems to establish secure, scalable infrastructure automation.

There are several meaningful directions in which the approach could be extended for future work. Second, enforcing compliance on a multi-cloud architecture would verify if the framework is the same for different service providers such as Azure and Google Cloud. Second, real-time threat intelligence or compliance monitoring could be used for integrative compliance in order to perform adaptive compliance. Finally, integrating machine learning to recognise misconfigurations and offer policy refinements would make the system go from a static validator to a predictive advisor. Finally, commercialisation of the solution could be packaged as a plug-and-play DevSecOps toolkit for enterprises trying to simplify secure deployment without deep policy expertise.

Finally, the research successfully models an efficient and scalable compliance as code automation model. It caters from security governance to continuous deployment, adding up to the modern DevSecOps practices and establishing a healthy assumption for secure infrastructure provisioning at scale.

References

Agarwal, V., Butler, C., Degenaro, L., Kumar, A., Sailer, A. and Steinder, G. (2022). Compliance-as-code for cybersecurity automation in hybrid cloud, *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*, pp. 427–437.
URL: *https://doi.org/10.1109/CLOUD55607.2022.00066*

Bajpai, P. and Lewis, A. (2022). Secure development workflows in ci/cd pipelines, *2022 IEEE Secure Development Conference (SecDev)*, pp. 65–66.
URL: *https://doi.org/10.1109/SecDev53368.2022.00024*

Bali, M. K. and Walia, R. (2023). Enhancing efficiency through infrastructure automation: An in-depth analysis of infrastructure as code (iac) tools, *2023 International Con-*

ference on Computing, Communication, and Intelligent Systems (ICCCIS), pp. 857–863.
URL: *https://doi.org/10.1109/ICCCIS60361.2023.10425162*

Banse, C., Kunz, I., Schneider, A. and Weiss, K. (2021). Cloud property graph: Connecting cloud security assessments with static code analysis, *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, pp. 13–19.
URL: *https://doi.org/10.1109/CLOUD53861.2021.00014*

Battula, M. (2024). A systematic review on a multi-tenant database management system in cloud computing, *2024 International Conference on Cognitive Robotics and Intelligent Systems (ICC - ROBINS)*, pp. 890–897.
URL: *https://doi.org/10.1109/ICC-ROBINS60238.2024.10533959*

Chandra, T. R. (2025). (PDF) Cloud-Native DevSecOps: Integrating Security Automation into CI/CD Pipelines, *IJIRCT* **10**(6): 1–19.
URL: *https://www.ijirct.org/viewPaper.php?paperId=2503017*

Dalvi, A. (2022). Cloud infrastructure self service delivery system using infrastructure as code, *2022 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, pp. 1–6.
URL: *https://doi.org/10.1109/ICCCIS56430.2022.10037603*

García-Galán, J., Pasquale, L., Grispos, G. and Nuseibeh, B. (2016). Towards adaptive compliance, *2016 IEEE/ACM 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pp. 108–114.
URL: *https://doi.org/10.1145/2897053.2897070*

Hasan, M. R. and Ansary, M. S. (2023). Cloud Infrastructure Automation Through IaC (Infrastructure as Code), *International Journal of Computer (IJC)* **46**(1): 34–40.
URL: *https://www.ijcjournal.org/index.php/InternationalJournalOfComputer/article/view/2043*

Kavas, E. (2023). *Architecting AWS with Terraform: Design resilient and secure Cloud Infrastructures with Terraform on Amazon Web Services*, Packt Publishing Ltd. Google-Books-ID: YajoEAAAQBAJ.

M K, N., B S, M., Khandelwal, N., Pai, N. and L, S. (2023). Ci/cd pipeline with vulnerability mitigation, *2023 International Conference on Recent Advances in Science and Engineering Technology (ICRASET)*, pp. 1–6.
URL: *https://doi.org/10.1109/ICRASET59632.2023.10419921*

Marandi, M., Bertia, A. and Silas, S. (2023). Implementing and automating security scanning to a devsecops ci/cd pipeline, *2023 World Conference on Communication & Computing (WCONF)*, pp. 1–6.
URL: *https://doi.org/10.1109/WCONF58270.2023.10235015*

Moghaddam, F. F., Wieder, P. and Yahyapour, R. (2016). Policy engine as a service (peaas): An approach to a reliable policy management framework in cloud computing environments, *2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*, pp. 137–144.
URL: *https://doi.org/10.1109/FiCloud.2016.27*

Paul, A., Manoj, R. and S, U. (2024). Amazon web services cloud compliance automation with open policy agent, *2024 International Conference on Expert Clouds and Applications (ICOECA)*, pp. 313–317.
URL: *https://doi.org/10.1109/ICOECA62351.2024.00063*

Paul, D., Soundarapandiyan, R. and Krishnamoorthy, G. (2021). Security-First Approaches to CI/CD in Cloud-Computing Platforms: Enhancing DevSecOps Practices, *Australian Journal of Machine Learning Research & Applications* **1**(1): 184–225. Number: 1.
URL: *https://sydneyacademics.com/index.php/ajmlra/article/view/131*

Singh, R., Yeboah-Ofori, A., Kumar, S. and Ganiyu, A. (2024). Fortifying cloud devsecops security using terraform infrastructure as code analysis tools, *2024 International Conference on Electrical and Computer Engineering Researches (ICECER)*, pp. 1–6.
URL: *https://doi.org/10.1109/ICECER62944.2024.10920371*

Teppan, H., Flå, L. H. and Jaatun, M. G. (2022). A survey on infrastructure-as-code solutions for cloud development, *2022 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 60–65.
URL: *https://doi.org/10.1109/CloudCom55334.2022.00019*

Theodoropoulos, T., Rosa, L., Benzaid, C., Gray, P., Marin, E., Makris, A., Cordeiro, L., Diego, F., Sorokin, P., Girolamo, M. D., Barone, P., Taleb, T. and Tserpes, K. (2023). Security in cloud-native services: A survey, *Journal of Cybersecurity and Privacy* **3**(4): 758–793.
URL: *https://doi.org/10.3390/jcp3040034*

Walther, R., Weinhold, C., Amthor, P. and Roitzsch, M. (2024). Multi-stakeholder policy enforcement for distributed systems, *Proceedings of the 10th International Workshop on Container Technologies and Container Clouds*, WoC '24, Association for Computing Machinery, New York, NY, USA, p. 7–12.
URL: *https://doi.org/10.1145/3702637.3702958*

Yongsiriwit, K., Sellami, M. and Gaaloul, W. (2016). A semantic framework supporting cloud resource descriptions interoperability, *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pp. 585–592.
URL: *https://doi.org/10.1109/CLOUD.2016.0083*