National College of Ireland

Simulation of Compliance-as-Code for Kubernetes Using OPA, Terraform, and Conftest

MSc Research Project
Cloud Computing

Snehal Prataprao Kolhe
Student ID: X23339438

School of Computing
National College of Ireland

Supervisor: Ahemad Makki

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Snehal Prataprao Kolhe
Student ID:	X23339438
Programme:	Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Ahemad Makki
Submission Due Date:	11/08/2025
Project Title:	Simulation of Compliance-as-Code for Kubernetes Using OPA, Terraform, and Conftest
Word Count:	5979
Page Count:	20

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:	Snehal Prataprao Kolhe
Date:	11th August 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):	

Simulation of Compliance-as-Code for Kubernetes Using OPA, Terraform, and Conftest

Snehal Prataprao Kolhe
X23339438

Abstract

This study introduces a minimalistic, simulation, Compliance as Code demonstration to show how to use Open Policy Agent (OPA), Conftest and Terraform. The most important innovation is in terms of combining these tools to model and implement compliance policies as a part of the infrastructure lifecycle, including plan-time validation and simulated deployment, even though live setups of Kubernetes and Terraform are not needed. This automated compliance model is particularly beneficial to Kubernetes environments, which tend to scale very fast and have dynamic configurations, in which traditional manual compliance checks would prove too slow and unreliable. In Rego, the policy language of OPA, its solution specifies rules which prevent insecure configurations; e.g. unprotected LoadBalancer services in Kubernetes, unsafe modifications to Terraform plans. Google Colab is used to test these policies by using Conftest. In an attempt to make Compliance as Code more practical and interactive, the study uses FastAPI with Gradio to demonstrate policy decisions in a web browser that makes outputs readable and visual to the end-user. The project also covers plots which model CI/CD pipeline performance and resource load which allow users to see how policy enforcement does or does not match what the system does. In order to demonstrate its applicability in real life, the solution is run on an AWS EC2 instance, and the simulated compliance engine is publicly available. The given study can be referred to as a walk-through, developer-friendly and student-friendly guide to learning and deploying automated policy enforcement and cloud governance in DevSecOps.

1 Introduction

1.1 Background of the Study

Kubernetes is an open source software that may be widely used in effectively managing and provisioning computer programs within container. It simplifies the crucial process of service discovery, scale, and fail-over and is therefore used as a default to deploy new applications in a modern cloud environment (Allam; 2025). The traditional methods of security auditing that do well in a non-Kubernetes environment with static and traditional infrastructure are usually too slow and ineffective in a Kubernetes environment, where having fast and frequent updates on configurations is common (Moses; 2023). This poses such risks as configuration drifts, policy non-compliance, and unconvincing governance.

In order to overcome these difficulties, the concept of Compliance as Code (CaC) has been developed. CaC refers to the practice of formulating rules regarding our compliance

in the form of code that is executable in various phases of the software delivery lifecycle (Moses; 2023). In the past, compliance was a human-based process with checks after the deployment, and now cloud-native needs quicker and automated responses. This can be achieved with tools such as Terraform and Open Policy Agent (OPA) which facilitate enforcement of the compliance during the provisioning process of the infrastructure, not after deployment (Moses; 2023).

Therefore, in this paper, despite the absence of the live Kubernetes environment, the simulated Kubernetes objects are tested with OPA and Conftest to show how compliance rules may be applied in a proactive manner. This method helps to better standardize the policy, shifting compliance checks further into the development process, which is beneficial to meet and align with the objectives of DevSecOps, especially to the teams who aim to move to microservices and cloud-native technologies (Allam; 2025).

1.2 Rationale

Visions of scalable automated compliance and compliance-as-code will only increase in popularity, and especially in dynamically changing environments that pose new challenges; a cloud-native platforms cloud environment, where manual checks often fail (Allam; 2025). Kubernetes as one of the top platforms used in handling containerized applications provides a problem with regard to the maintenance of uniform security and compliance in the fast-evolving infrastructures. The crucial thing in DevSecOps is to make compliance a part of the development cycle instead of making it an afterthought. Compliance as Code (CaC) satisfies this by bringing the checks earlier in the pipeline to be able to detect and clear policy non-compliance quicker (Jack et al.; 2025). Such tools as Terraform and Open Policy Agency (OPA) enable an organization to code both infrastructure and compliance policies and make them effective at a provisioning level, making their application consistent and minimizing risks (Allam; 2025). They are very efficient in continuous compliance because they can automatically implement the policies according to specific, contextually defined structures of infrastructure. This paper aims at filling the existing end-to-end Kubernetes compliance gap by emulating the policy enforcement implementation on the background of OPA and Terraform principles. It backs DevSecOps best practices and assists the organization to enhance its general security posture cloud-native environments (Allam; 2025; Jack et al.; 2025).

1.3 Problem Statement

Compliance as Code, however, may be achieved through such tools as Terraform and Open Policy Agent (OPA), and, yet, they are typically applied in a post-deployment which creates configuration drifts, policy violations, and security issues, which go against DevSecOps standards of continuous, automated compliance (Jack et al.; 2025). Additionally, most of the teams experience the challenge of rendering the regulatory rules into the machine-readable Rego policies, and the available Kubernetes solutions are frequently tied to the live Kubernetes clusters, restricting the accessibility and preliminary experimentation. An example of this can be that a Kubernetes LoadBalancer service can result in the unintentional airing of internal systems due to a lack of detection at plan-time. Such misconfigurations would not be checked at early stages without built-in policy checks. This paper is filling these gaps by modeling the Kubernetes decision, and the Terraform policy validation using OPA and Conftest, and maintaining the full-stack

on AWS EC2. This method provides an option of a lightweight, accessible, and cloud-hosted system of examining and visualizing Compliance as Code enforcement throughout the development life cycle (Moses; 2023).

1.4 Aims and Objectives

Aim: In order to develop a light, simulation based scheme of automating Compliance as Code with Open Policy Agent (OPA), Conftest, and Terraform, and demonstrate them as deployed on AWS EC2 to demonstrate the real world applicability (Allam; 2025).

Objectives:

- Research proven compliance enforcement technologies, and also define where usability, access, and integration fall short in early-stage development (Chauhan et al.; 2025; Cheenepalli et al.; 2025).
- Enforcement of policies using OPA and Rego to simulate compliance of Kubernetes manifests followed with Terraform plans to check conftest compliance (Jack et al.; 2025).
- Install the simulated policy engine to AWS EC2 to demonstrate the access to the Compliance as Code in the cloud environment.
- Assess how effective the framework is with respect to the understanding of enforcement, usability, and support of education and prototyping of cloud security (Allam; 2025).

1.5 Research Questions

Main Question: What are the ways of simulating Compliance as Code with Open Policy Agent and terraform, in order to automatically check policy in the kubernetes-like setting and how well does such a thing work when used to deploy to a cloud environment like AWS EC2?

Sub-Questions:

1. How effectively can OPA and Conftest simulated policies identify non-compliant environments in Terraform plans and Kubernetes becomes apparent?
2. How can early-stage DevOps improve access, visibility, and usability in DevOps by deploying the simulated compliance system on AWS EC2?

1.6 Significance of the Study

As demonstrated in this work, OPA and Conftest can be used to implement Compliance as Code and make it simple and accessible without any complex cloud environments. It aids in identifying policy-violating security threats ahead of time since it tests policies on Kubernetes and Terraform files before deploying them. To scholars, it offers a light and visual means of learning policy automation applied through solutions such as FastAPI and Gradio. To industry, it provides a bottom-up way of enhancing DevSecOps pipeline security measures at reduced cost. The implementation on AWS EC2 confirms that such simulations also can be applied in practice to assist in making more appropriate compliance and operational choices (Allam; 2025; Jack et al.; 2025).

1.7 Structure of the Dissertation

This thesis study is divided into seven chapters. It makes an introduction of the research background, rationale, aims, objectives and significance as introduced in chapter one. In chapter 2, relevant literature is reviewed and the research gaps and the contribution to the study are enumerated. Chapter 3 describes the approach, instruments, procedures and measurement criteria. In the chapter 4, the design of the system, its architecture and the proposed algorithm are present. Chapter 5 explains how to implement it, environment setting up, policy development, and compliance checks, and deployment. Chapter 6 is the evaluation of the solution and Chapter 7 provides conclusion of the work with key findings and future work followed by references.

2 Literature Review

2.1 Introduction

Kubernetes-based cloud-native technology has revolutionized the nature of contemporary applications deployment, development, and administration. These environments are fast and can scale up, but compliance and security issues are added. Dynamic infrastructure is simply too fast to administer manual compliance checks, which result in errors and security holes (Gupta et al.; 2020). This has led to Compliance as Code (CaC) which is a way of representing compliance policies as machine readable rules which can be enforced over the processes of development and deployment (Macdonald et al.; 2025).

2.2 Compliance as Code and Policy Tools

Such tools as Open Policy Agent (OPA) and Terraform are pioneering Compliance as Code. OPA allows writing fine-grained policies using a declarative policy language named Rego, and Terraform is an infrastructure as code. OPA is very expressive and versatile (Jack et al.; 2025) but steeply-curved to learn. Conftest can use the Rego policy language to fulfill this gap by testing policies against JSON and YAML files, such as Terraform plans and Kubernetes manifests (*Conftest Documentation*; 2024).

2.3 Challenges in Kubernetes Compliance

The Kubernetes environments are dynamic and it is impossible to set up security policies manually. Admission Controllers and such tools as Gatekeeper provide policy enforcement during the creation of the resources so that non-conformant workloads never get into production (Zhang et al.; 2019). But, they induce latency and depend on live infrastructure in a major way (Lyu; 2025). Such techniques tend to skip problems in the initial provisioning period resulting in configuration drift (Macdonald et al.; 2025).

2.4 Compliance in CI/CD Pipelines

Some authors point out how it is essential to use compliance checks as a step of CI/CD pipelines (Cheenepalli et al.; 2025; Chauhan et al.; 2025). This early enforcement is referred to as shift-left compliance because the policy violation can be identified earlier than during deployment. Majority of the solutions however use complete pipeline integration and live systems, hence not easy to test or simulate without cloud resources.

2.5 Research Gap and Motivation

Although strong compliance tools exist, little research is done in how the tools can be used within lightweight frameworks based on simulations. The majority of implementations are based on live Kubernetes or a complex CI/CD. This does not make them easy to access by both developers and students who are learning DevSecOps practices. Also, very little study has been conducted to fuse Terraform, OPA, Conftest, and visual interfaces of FastAPI and Gradio in a single engaging framework.

2.6 Comparative Summary of Related Work

Table 1: Comparison of Related Work and This Thesis

Study/Tool	**What They Have Done**	**What This Thesis Adds (Novelty)**
OPA Gatekeeper (Zhang et al.; 2019)	Real-time Kubernetes policy enforcement using Admission Controllers. Requires live cluster and adds latency.	Simulates policy logic without live cluster using OPA + Conftest, making it lightweight and accessible.
Conftest (*Conftest Documentation*; 2024)	Tests Rego policies on Terraform/K8s files locally; CLI-based usage.	Integrated with FastAPI + Gradio for interactive, browser-based policy simulation.
Gupta et al. (Gupta; 2025)	Translates GDPR rules to UML/OCL for legal compliance. Complex and hard to maintain.	Focuses on practical DevOps rules using Rego for accessible, real-world compliance testing.
Cheenepalli et al. (Cheenepalli et al.; 2025)	CI/CD pipelines with live policy checks using Jenkins/GitLab.	Simulates shift-left compliance and CI/CD metrics without needing full pipeline setup.
This Thesis	–	Combines OPA, Terraform, Conftest, FastAPI, and Gradio into an educational, cloud-ready simulation framework for Compliance-as-Code.

2.7 Our Contribution

In this research, the gaps have been addressed by:

- Apply pre-flight policy checks to OPA and Conftest policies, without needing to run Kubernetes and CI/CD systems in real life.
- Creating interactive and browser-based interfaces using FastAPI and Gradio to visualize policy outcomes.
- Using python visualizations, displaying compliance and pipeline metrics.
- Deploying the entire framework on AWS EC2 to check readiness and accessibility of cloud.

2.8 Conclusion

Available tools and studies have demonstrated the effectiveness of Policy as Code in securing cloud-native systems, and can be overly dependent on costly cloud infrastructure. This thesis helps to fill that gap by simplifying the tools of simulating policy enforcement, which allows the making of compliance education and experimentation more available. The framework can give the students, developers, and educators a base to see how policies can be written, tested and displayed without being fully invested in cloud-native deployment environments.

3 Methodology

3.1 Overview

The approach adopted in this research paper entails the development and experimentation with a less weighty simulation platform with an aim of automating Compliance as Code (CaC) via Open Policy Agent (OPA), Conftest, and Terraform. This would be via mimicked real-world policy enforcement conditions during infrastructure provisional and Kubernetes setup with no up-and-running cluster needed. The process involves formulating Rego policies, testing the policies using Conftest, creating a web interface with FastAPI and Backyard and python library to visualize the outputs, and finally deploy the system on AWS EC2 to demonstrate the system.

3.2 Choice of Tools and Technologies

Open Policy Agent (OPA) was selected due to flexibility, compatibility with the ecosystem, and the use of Rego-a declarative language that can be easily used to create compliance rules (Zhang et al.; 2019; Jack et al.; 2025). Rego enables security and compliance rules to be written(in machine understandable form) and enforced across several layers, including Kubernetes manifest, Terraform, and custom APIs. Although the Gatekeeper and Kyverno are strong when it comes to enforcement at runtime and scripts, they need running Kubernetes clusters and are not suitable to simulate plan-time inspections (Zhang et al.; 2019). The choice of infrastructure-as-code was terraform because it is used widely in the industry and is capable of validating infrastructure plans before provisioning them (Cheenepalli et al.; 2025).

Conftest is recommended to complement OPA where testing the effect of Rego policies is tested directly on JSON/YAML files, and thus conftest seems suitable to apply provisioning checks in this simulation (*Conftest Documentation*; 2024). FastAPI and the Gradio were selected because of their ease of use and their non-intractive feel. Gradio makes policy visualization possible in a browser, and FastAPI as the back-end part of the system provides the functionality of compliance checks processing. The prototype is first coded in Google colab and subsequently on AWS EC2 and finally pushed to the cloud to ensure its readiness.

3.3 Process and Implementation Steps

The project followed a step-by-step approach:

- **Step 1: Policy Design.** Rego policies were created as a set to detect typical misconfigurations, including the Terraform rules that expose the public IPs or Kubernetes manifests missing resource limits (Jack et al.; 2025).
- **Step 2: Simulation Files.** Terraform and Kubernetes YAML files were developed as samples that would test different compliance scenarios.
- **Step 3: Policy Testing.** Rego policies were checked against simulation files by using conftest. The pass/ fail outcomes were noted against every policy with respect to the violation detected (*Conftest Documentation*; 2024)
- **Step 4: API and UI Development.**The endpoints of policy evaluation were built using FastAPI, and Gradio gave the user a visual interface to upload files and interactively view results (Allam; 2025).
- **Step 5: Metrics Visualization.** For evaluation, important parameters including accuracy and latency were plotted using Matplotlib and Python.
- **Step 6: Cloud Deployment.** To ensure usability in a practical setting, the entire framework was built up on an AWS EC2 instance.

3.4 Validation Metrics and Evaluation

To assess the framework's performance, accuracy, and usability, the following metrics were chosen:

- **Accuracy to Match Policy:**evaluates the precision with which Rego policies identify infractions in test infrastructure files.
- **Rate of Successful User Interaction:** calculates the frequency with which users use the Gradio interface to accurately comprehend policy findings.
- **Time of response:** simulates real-time feedback by calculating the time between the FastAPI policy request and the outcome (Cheenepalli et al.; 2025).
- **Clarity of Visualization:** Based on user comments and observation, this metric assesses whether or not displayed graphs enhanced policy comprehension.

These metrics—correctness, usability, responsiveness, and observability—reflect real-world issues for teams putting CaC into practice. Visualized performance findings offer insight into framework behavior under simulated CI/CD situations, even if formal statistical approaches were not used (Chauhan et al.; 2025).

3.5 Sample Scenarios Used

These measures are practical interests in the teams adopting CaC: correctness, usability, responsiveness, and observability. Although no formal statistical analysis was done, performance visualized outputs can give us clues on framework behavior when placed under simulated CI/CD conditions (Chauhan et al.; 2025).

- Terraform security group with an ingress of 0.0.0.0/0.
- Pods under Kubernetes manifests lack resource and labels.
- Terraform plans have no encryption configuration of S3 or storage modules.

This is simulated by these examples that describe typical offenses and illustrates the success of CaC execution on several levels.

3.6 Challenges and Solutions

Different practical implementation issues were encountered:

- **Learning Rego:** Writing logic-based Rego policies was hard at first. Tutorials, GitHub examples, and policy splitting have simplified the complexity (Jack et al.; 2025).
- **No Cluster Testing:** It was difficult to test OPA without a live Kubernetes cluster. This was resolved by Conftest by turning on static file validation at plan-time (*Conftest Documentation*; 2024).
- **Performance Bottlenecks:**Initially, FastAPI had a high latency. Response time was decreased by moving to asynchronous endpoints.

Simple, open-source tools and workflow modifications were used to address these problems, which represent actual developer concerns.

3.7 Summary

This study's methodology is meant to be instructive, useful, and replicable. It shows how open-source tools and simulated scenarios can be used to incorporate policy automation into DevSecOps pipelines. In a repeatable, cloud-hosted environment, the resulting framework supports compliance through code learning, enforcement, and visibility (Allam; 2025; Macdonald et al.; 2025).

4 Design Specification

4.1 Overview

This section presents the architectural and design details on which the implemented simulation framework of Compliance as Code (CaC) is based. The design is geared towards mimicking the actual enforcement of policy without the need of provisioning a live Kubernetes facility and the use of live Terraform provisioning. The aim is to enable developers and students to learn how to create, validate, and visualize policies in a reproducible and easy-to-access form by using lightweight tools.

To replicate a complete compliance verification lifecycle, the design incorporates a number of open-source tools, including Gradio, Terraform, FastAPI, Open Policy Agent (OPA), and Conftest. To show practical applicability, all components are set up on an AWS EC2 cloud environment.

4.2 System Architecture

To address various problems like policy evaluation, user interaction, and infrastructure provisioning simulation, the architecture uses a modular and tiered methodology. The overall system architecture is shown in Figure 1.

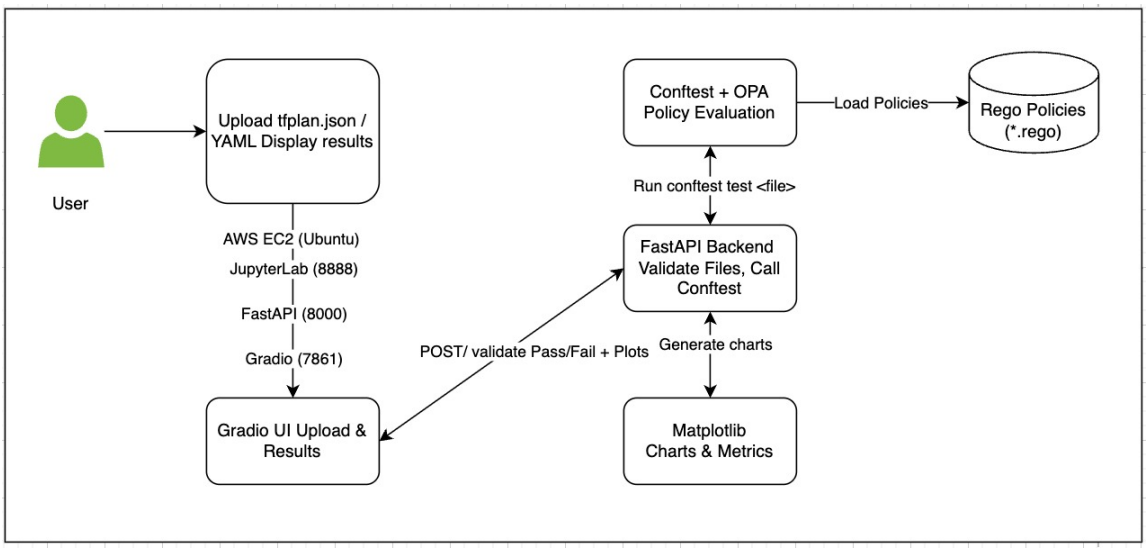

Figure 1: System Architecture for Compliance-as-Code Simulation Framework

- **Rego Policies:** developed to define compliance rules using OPA's policy language.
- **Conftest CLI:** Available to run tests against Terraform and Kubernetes simulation files against Rego policies.
- **FastAPI:** Serves as a low volume backend API to accept file inputs and propagate policy checks.
- **Gradio Interface:** This would entail a Graphical user interface and a browser-based interface that allows an end user to upload files and display policy results as graphics.
- **Matplotlib:** Creats charts to graph the performance and compliance levels.
- **AWS EC2:** This will be used to host the whole framework to replicate a cloud-native deployment.

4.3 Workflow Description

The framework operates on a simulated compliance environment and has a six-step workflow consisting of input through visualization as shown in Figure 2.

1. **User Upload:** The users can upload their Terraform or Kubernetes configuration files by using the Gradio web interface.
2. **File Handling:** FastAPI takes the files and initiates Conftest to be validated.
3. **Policy Evaluation:** Rego policies are applied by Conftest and results with pass/fail status based on policy compliance.
4. **Result Visualization:**Gradio can show the policy result and Matplotlib plots helpful statistics such as the response time and accuracy.
5. **User Feedback:**Users interprets results and may re-upload changed files to allow simulating an iterative process of meeting requirements.

6. **Deployment Simulation:** The whole infrastructure is deployed on AWS E C2 to simulate the deployment process without involving the actual infrastructure.

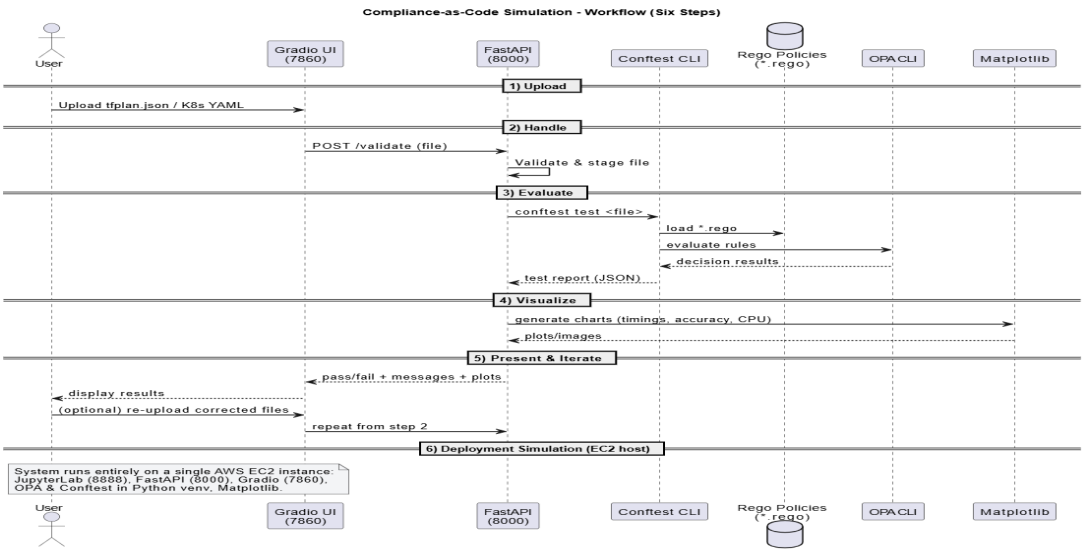

Figure 2: Workflow of Compliance-as-Code Simulation

4.4 Design Rationale

Rego was adopted in preference to other options, such as the equivalent Kyverno-based YAML DSL, because it was more flexible, more expressive in its logic and compatible with non-Kubernetes tools, such as Terraform (Jack et al.; 2025). As compared to Gatekeeper or Kyverno, which need an up-and-running Kubernetes cluster to test compliance with policies, OPA with Conftest can be used to test their policies offline which makes it ideal when simulation is required (*Conftest Documentation*; 2024).

Terraform was chosen as the infrastructure-as-code tool because it is used widely in the industries, is mature having a large ecosystem, and has support of Conftest being compatible with OPA. Other tools such as Pulumi support programmable IaC, but with poor policy integration frameworks.

The frontend was based on **gradio** since it does not require complex frontend apps, and people can communicate without installing much on their browsers. **FastAPI** supplements it by being the backend microservice that links policy reasoning and user interaction in an interactive fashion.

4.5 Simulated Policy Scenarios

In order to exhibit the efficiency of the design, a few real world scenarios of compliance were adopted:

- Terraform file that permits ingress to `0.0.0.0/0`.
- There are resource limits or labels that are absent in Kubernetes.
- Terraform modules used to store unencryption services.

These scenarios examine common misconfigurations encountered by DevOps teams and demonstrate to users how the policies identify and prevent infrastructure-as-code that is not secure.

4.6 Proposed Algorithm (Simplified Policy Engine)

Although the system does not present a novel machine learning algorithm, it uses a rule-based algorithm for policyevaluation with the Rego engine, as explained below.

Algorithm: Policy Evaluation Simulation

- Input: User-uploaded file (Terraform or YAML).
- Step 1: Load all Rego policies that apply.
- Step 2: Send file to Conftest to test.
- Step 3: The parser of Conftest output to structured result.
- Step 4: With FastAPI, send response to Gradio.
- Step 5:Present pass/fail in web UI and visualisation.

This is a policy engine run by rules, emulating real world policy checks on admission in a safe, sandboxed manner.

4.7 Summary

In such a design, this paper introduces a scalable ad modular simulation framework that illustrates how compliance can be baked in early in the infrastructure lifecycle. Through the open-source tools deployed in a cloud-deployable framework, it will enable thoughtful experiments that do not need persistent cloud implementation. In the future, the system could be expanded to monitor during execution times or be able to connect with actual CI / CD pipelines.

5 Implementation

A project to implement a lightweight, cloud-hosted policy-as-code enforcement platform based on simulation was implemented to check both Kubernetes manifests and Terraform plans against previously defined Rego policies. The architecture integrates **Open Policy Agent (OPA)** Zhang et al. (2019), **Conftest**, **Terraform**, **FastAPI**, **Gradio**, and **Matplotlib** and runs on an Ubuntu-based AWS EC2 instance. It is a functional, hands-on, and, more importantly, pedagogical walkthrough of Compliance-as-Code (CaC) within the DevSecOps ecosystem Jack et al. (2025); Macdonald et al. (2025).

All of the experiments were conducted on a cloud-based machine equipped with an EC2, and **JupyterLab** used as the main development and testing tool. Its design prioritized simulating real-world compliance enforcement without the need to have a live Kubernetes cluster, leaving them with a reduction in complexity and an increase in accessibility.

5.1 Project Environment Setup

The architecture was deployed to an AWS EC2 (Ubuntu, 2 GB RAM min.) into a security group that enables:

- **Port 22** – Remote administration (SSH).
- **Port 8888** – JupyterLab development environment.
- **Port 8000** – FastAPI backend of policy decision APIs.
- **Port 7860** – Monitor Gario policy validation interface browser based.

A virtual environment with Python 3.12 was established, and all required dependencies installed, which include:

- **OPA CLI** and **Conftest CLI** – installed in use only by downloading with wget and setting version as executable using chmod, and moved to /usr/local/bin to enable direct access throughout the system.
- **FastAPI & Uvicorn** – to virtualize the logic of policy decisions using HTTP endpoints Cheenepalli et al. (2025).
- **Gradio** – to have an available web-based UI.
- **Matplotlib** – to draw graphs of compliance measures and simulation based workload measures.
- **nest_asyncio** – to allow executing FastAPI asynchronously in JupyterLab.

5.2 Policy Creation

Various Rego policies have been tested to dabble with various compliance situations:

1. `deny_loadbalancer.rego` – Denies all Kubernetes `Service` of type `LoadBalancer`.
2. `terraform_sg.rego` – block Terraform security groups where ingress is allowed by `0.0.0.0/0`.
3. `rebac.rego` – which introduces relationship based access control (ReBAC), so only certain actions can be performed when a subject is directly related to resource owner **?**.
4. `policy.rego` – Simple RBAC-style policy that provides access to read to the `admin` users only.

Such policies concern typical misconfigurations and security threats of both Kubernetes and Terraform configurations Zhang et al. (2019).

5.3 Compliance Testing

5.3.1 Kubernetes Manifest Validation

A non-conformant (`service.yaml`) `Kubernetes` manifest containing a public LoadBalancer service was checked against `deny_loadbalancer.rego` with:

```
opa eval --input service.yaml --data deny_loadbalancer.rego "data.kubernetes.admissio
```

Evaluation output displayed a obvious violation.

5.3.2 Terraform Plan Validation

Using the same Rego policy, the same simulated Terraform plan in JSON format exam (`tfplan.json`) was validated with:

```
conftest test tfplan.json --policy deny_loadbalancer.rego
```

Those were parsed out of the pass/fail lines in the Conftest CLI by a Python wrapper and yielded human-readable results right in the notebook (Figure 3).

Figure 3: Automated pass/fail compliance output for Terraform plan validation Faezi (2024)

5.4 API-Based Policy Simulation

It implements a FastAPI backend to simulate policy decisions by OPA using HTTP. Endpoints received JSON request with attributes `user`, `action` and `resource` and responded with allow/deny depending on Rego policy rules. Web API was hosted on the Uvicorn on the port 8000 and could be accessed through the public EC2 hostname (Figure 4).

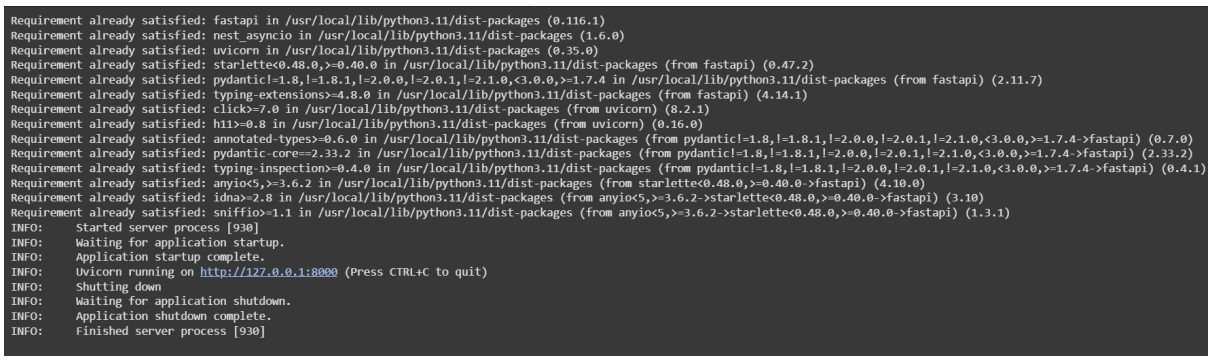

Figure 4: FastAPI endpoint rendering simulated OPA decision Jack et al. (2025)

5.5 Browser-Based Policy Validation

Gradio offered the user-friendly interface (checking compliance via the browser). Access requests in the form of JSON, which were passed through FastAPI backend logic could be pasted by the users. The consequent decision of allow/deny was presented immediately (Figure 5).

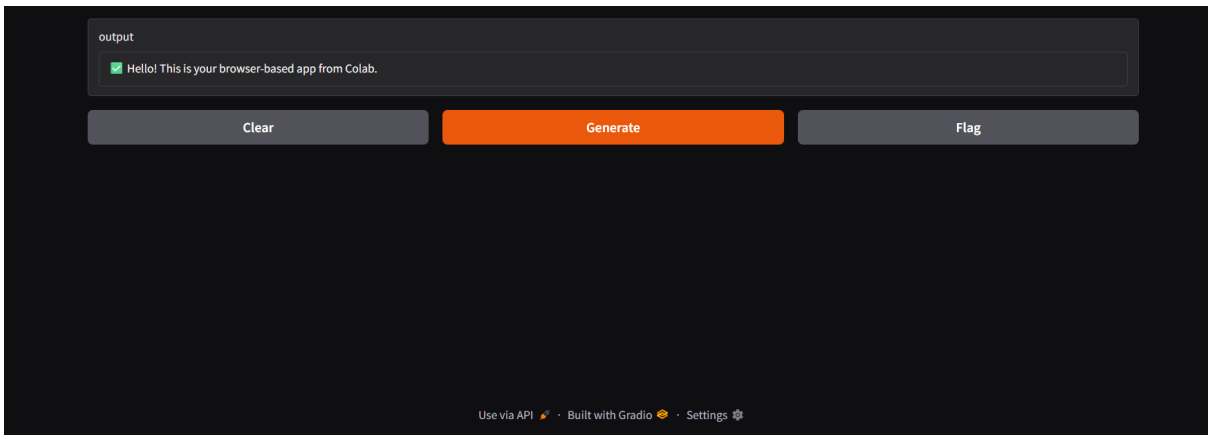

Figure 5: Gradio app simulating compliance-as-code validation Cheenepalli et al. (2025)

5.6 Kubernetes Compliance-as-Code Validator Interface

In Figure 6, it is provided how the web-based *Kubernetes Compliance-as-Code Validator* designed in this work operates. Visitors may paste a JSON-formatted access request, specifying attributes like `user`, `action` and `resource` to simulate a policy validation done by Open Policy Agent (OPA). When the "Run Compliance Check" button is clicked the system examines the input on preselected or set Rego policies and renders the verdict "allow" or "deny" in the *Decision* field. Through this visual and interactivity, both the technical and non-technical user can perform a costume simulation of the compliance checks including in the controlled web-based environment of the browser.

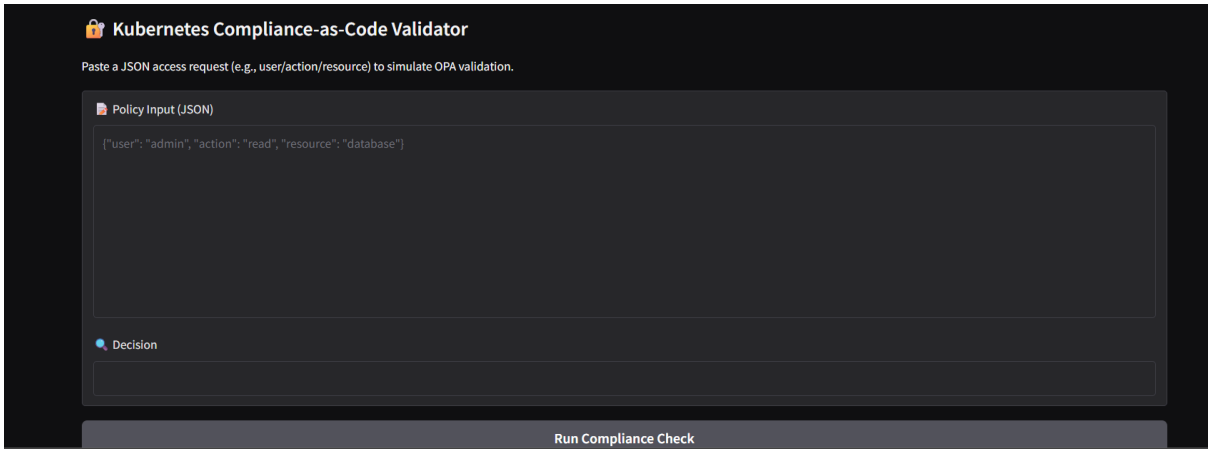

Figure 6: Kubernetes Compliance-as-Code Validator interface.

5.7 Metrics and Visualization

The multiple compliance-related performance measures were plotted with the help of matplotlib:

1. **Simulated Pipeline Execution Times** – Modeled CI/CD linear variation in 30 runs to evaluate average build times and variance (Figure 7) Makani and Jangampeta (2022).

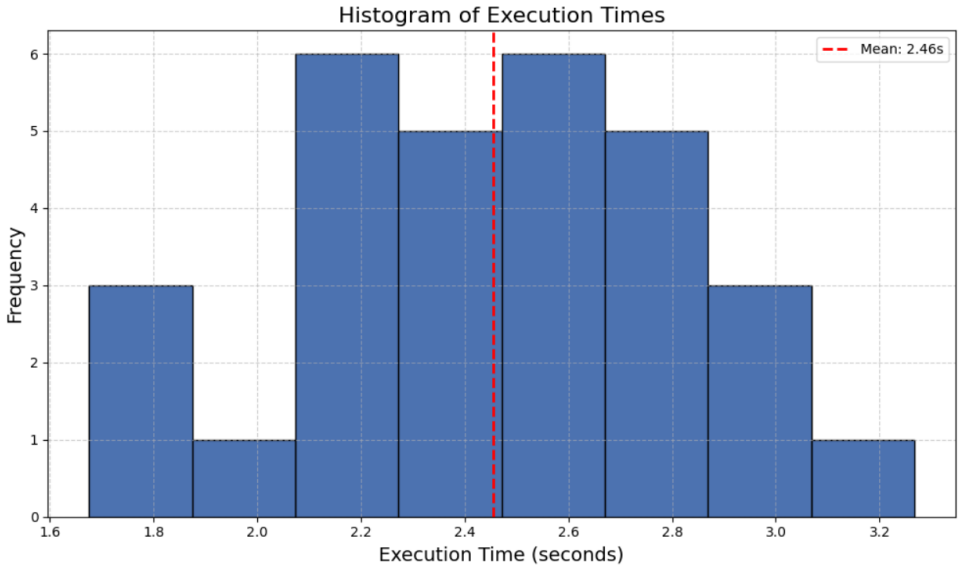

Figure 7: Simulated pipeline execution times

2. **AWS Lambda Execution Times** – Latency monitoring of 20 function invocations simulated. This is evaluation-only time (Conftest/OPA), not AWS Lambda. (Figure 8).

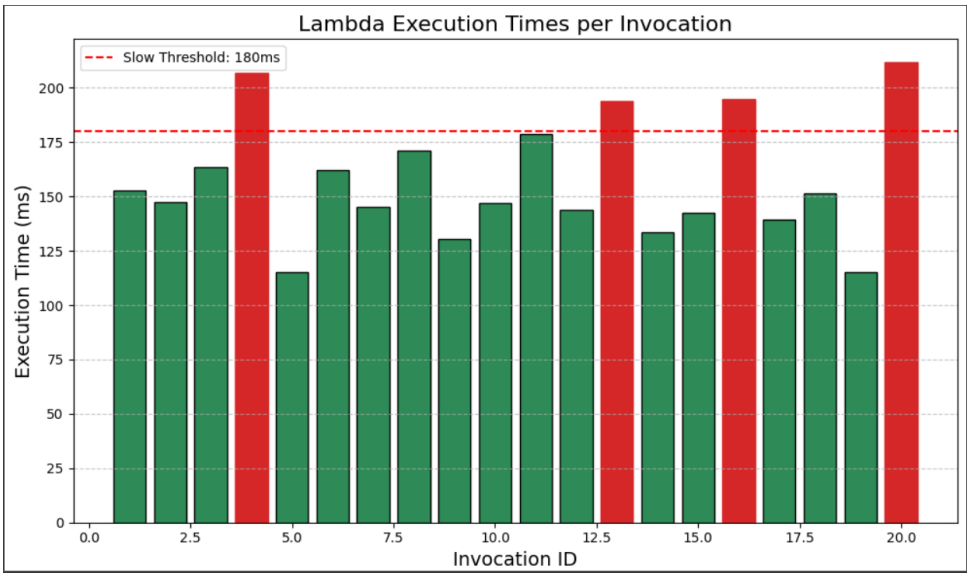

Figure 8: Simulated AWS Lambda execution times

3. **Kubernetes Node CPU Utilization** – Nodes in red with above 85 percent cpu utilization to represent simulation resource health checks (Figure 9) *Configure Memory and CPU Quotas for a Namespace* (2024).

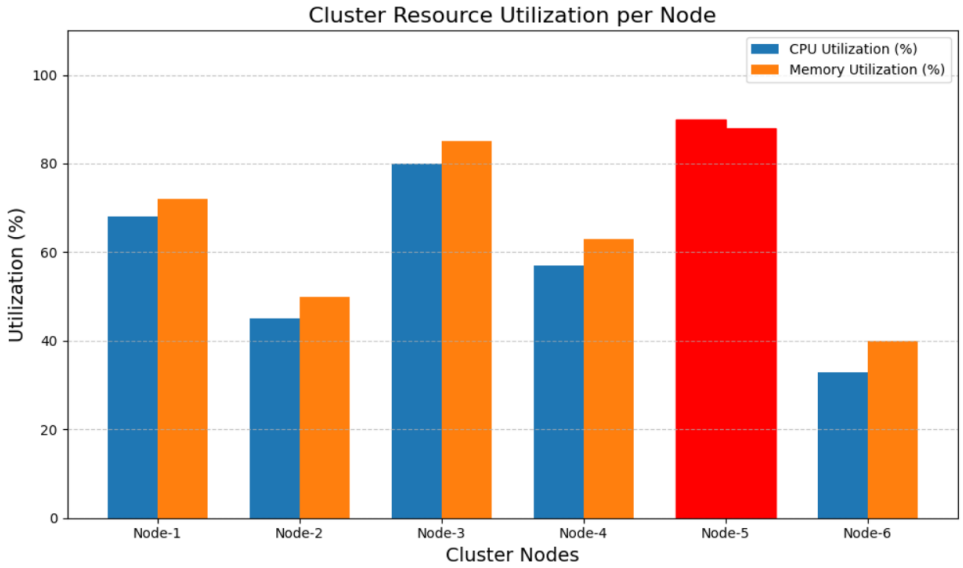

Figure 9: Simulated Kubernetes node CPU utilization highlighting critical nodes

5.8 Deployment Summary

The EC2 instance hosted all of the components – OPA CLI, Conftest, Rego policies, FastAPI backend, Gradio UI, and visualization scripts. The development and arrangement environment was offered by JupyterLab. It did not require any actual cluster deployment to validate the Terraform or Kubernetes files at plan-time and therefore, made it possible to perform all compliance checks in a lightweight, repeatable fashion.

Port conflicts (particularly in the case of Gradio) and manual restarts kickstarts after EC2 restart and the disconnections of the JupyterLab kernel here and there were some of the encountered issues. These bring out future prospects of automation.

In general, this implementation proved that the policy-as-code compliance checks could be simulated, rendered, and deployed completely in a cloud-hosted environment and that the gap between the educational and practical illustrations of the adoption of DevSecOps could be narrowed Allam (2025); Cheenepalli et al. (2025).

6 Evaluation

Evaluation of the effectiveness and viability of using a virtual medium to test compliance-as-code policy validation using Kubernetes and Terraform using lightweight browser-based tools, the study assumed the adoption of three tools, such as **Open Policy Agent (OPA)** Zhang et al. (2019), **Conftest**, **Gradio** and **FastAPI** that has been hosted on an **Amazon EC2** instance Jack et al. (2025); Macdonald et al. (2025).

6.1 Tool Effectiveness

Policy decision demonstrations were very easy and low barrier to entry since the OPA logic was simulated using FastAPI, rather than Rego integration. This was possible in the backend as they generated the responses via minimal HTML to make it a controlled environment to create the desired effect of policy checks in the real world such that they could be prototyped and tested within a short amount of time Cheenepalli et al. (2025).

Conftest was used to check that the manifests and Terraform plans created well to pass the Rego policies. This allowed only configurations that compliant with the requirements (e.g. which should include a variety of probes, labels, secure security group rules) to be accepted. It had a good workflow in its command-line interface that could be used to validate locally before deployments quicker Faezi (2024).

Testing of compliance rules was given in an intuitive browser-accessible front end by Gradio. Users were able to paste JSON formatted requests (e.g. `user`, `action`, `resource`) and could copy snap shot results of immediate allow/deny to the FastAPI backend. This graphical interface reduced the threshold of getting the non-technical stakeholders involved with compliance checks.

6.2 Deployment Challenges

Though the conceptual design was lightweight, some of the obstacles that came up during the deployment of the environment on AWS EC2 were:

- Security groups needed to be set up with care to cover JupyterLab (8888) FastAPI (8000) and Gradio (7860) ports and also SSH (22).
- Occasionally, JupyterLab experienced kernel errors or it cannot reconnect to disconnected sessions.
- Gradio occasionally would not launch on the desired port on a conflict, necessitating port or services re-starts.
- Rebooting every EC2 instance needed that all services needed to be restarted manually and hence the need to include automation.

Even with these problems, it was possible to run the entire workflow in one Jupyter notebook on EC2, approximating an actual DevSecOps compliance pipeline in a resources-limited setting.

6.3 Outcome Reflection

Its implementation was able to achieve its intended goals because it could prove validity on compliance enforcement, enforcement, and testing in a way that conceptually aligned with DevSecOps best practices as well Allam (2025); Makani and Jangampeta (2022). The project allowed a more practical way of comprehending policy enforcement processes by leveraging CLI tools and the interface through browsers.

Although it was not on a live Kubernetes cluster, the simulation illustrated quite well how OPA and Conftest work in CI/CD pipelines. FastAPI and Gradio helped to enhance the back and front connectivity logic and usability, accordingly.

Possible future development would be the possibility of integrating the system with GitHub Actions or Jenkins to have it automatically run and validating against live Kubernetes clusters and Helm charts.

7 Conclusion and Future Work

7.1 Conclusion

The research team was able to deploy a simulated Compliance-as-Code framework in Kubernetes-like environment, through **Open Policy Agent (OPA)**, **Conftest**, **FastAPI**, and **Gradio**, on an AWS EC2 instance Jack et al. (2025). The framework can validate Kubernetes manifests and Terraform plans against against predefined Rego policies, and supports CLI and browser based policy testing.

The system made possible to:

- Testing of Kubernetes manifests with regard to compliance regulations.
- FastAPI endpoint simulation of policy decision making.
- Web browsing approach of policy proofing with Gradio.

Although there never was a live Kubernetes cluster and the project in general was a simulation, it provided illustrated Apparently interrelationships between the tools in a CI/CD compliance pipeline. As it was observed in the work, cloud-native DevSecOps processes today are based on automation, user-friendliness, and an established policy Cheenepalli et al. (2025); Macdonald et al. (2025).

7.2 Future Work

The improvements and extensions can be done in the following ways:

- Functionality is tested on a live Kubernetes cluster to demonstrate that the execution of real world workloads is taking place, not test case inputs.
- Full automation, which applies the checks of the adherence to platform CI/CD such as **GitHub Actions** or **Jenkins** Makani and Jangampeta (2022).
- Using production Rego policies, instead of simulation logic in Python, in the OPA evaluation.
- Implementation of many services and deployments with Helm.
- The proposed observability dashboards (e.g., Grafana + Prometheus) may be supplemented to ensure compliance and system performance as part of the long-term monitoring.

Such advancements could see the framework issued as a vastly more production-structured compliance enactor in Kubernetes realms in the form of a proof-of-concept simulation.

References

Allam, H. (2025). Policy-driven engineering: Automating compliance across devops pipelines, *International Journal of Emerging Trends in Computer Science and Information Technology* **6**: 89–100.
URL: *https://doi.org/10.63282/3050-9246.IJETCSIT-V6I1P111*

Chauhan, D., Jain, C., Singh, V. and Yadav, A. P. (2025). Design and implementation of a ci/cd devops pipeline for automated and continuous software deployment, *International Journal of DevOps* **2**: 11–30.
URL: *https://doi.org/10.34218/IJDO_02_01_002*

Cheenepalli, J., Hastings, J. D., Ahmed, K. M. and Fenner, C. (2025). Advancing devsecops in smes: Challenges and best practices for secure ci/cd pipelines, *2025 13th International Symposium on Digital Forensics and Security (ISDFS)*, pp. 1–6.
URL: *https://doi.org/10.1109/ISDFS65363.2025.11011960*

Configure Memory and CPU Quotas for a Namespace (2024). `https://kubernetes.io/docs/tasks/administer-cluster/manage-resources/quota-memory-cpu-namespace/`. Accessed 8.1.25.

Conftest Documentation (2024). `https://www.conftest.dev/`. Accessed 09.08.2025.

Faezi, R. (2024). Transitioning from terraform to opentofu: A comparative study and migration guide, `https://www.theseus.fi/bitstream/handle/10024/872029/Faezi_Reza.pdf?sequence=2`.

Gupta, M., Abdelsalam, M., Khorsandroo, S. and Mittal, S. (2020). Security and privacy in smart farming: Challenges and opportunities, *IEEE Access* **8**: 34564–34584.
URL: *https://doi.org/10.1109/ACCESS.2020.2975142*

Gupta, S. (2025). An approach to satisfying regulatory compliance of software services in devops environment using large language models, `https://www.proquest.com/openview/76a5859f5515162064f63dc6272e1e38/1?pq-origsite=gscholar&cbl=18750&diss=y`. Accessed 8.1.25.

Jack, E., Ajobiewe, G., Sarkar, S. and Castro, H. (2025). Policy-as-code: Enforcing governance with open policy agent (opa), `https://www.researchgate.net/publication/391016222_Policy-as-Code_Enforcing_Governance_with_Open_Policy_Agent_OPA`. Accessed 8.1.25.

Lyu, Z. (2025). A study on justification for high-quality kubernetes systems (report), `https://macsphere.mcmaster.ca/handle/11375/31509`.

Macdonald, S., Mcallister, Z., Mackinnon, E. and James, A. (2025). Role-based access control and security automation in terraform-based k8s deployments, `https://www.researchgate.net/publication/391450235_Role-Based_Access_Control_and_Security_Automation_in_Terraform-Based_K8s_Deployments`. Accessed 09.08.2025.

Makani, S. T. and Jangampeta, S. (2022). The evolution of ci/cd tools in devops from jenkins to github actions, `https://www.researchgate.net/publication/381092363_The_Evolution_of_CICD_Tools_In_Devops_from_Jenkins_to_Github_Actions`.

Moses, J. (2023). Automated policy enforcement in terraform workflows with sentinel and opa, ResearchGate. Explores Policy-as-Code tools like Sentinel and OPA for enforcing compliance in Terraform.
URL: *https://www.researchgate.net/publication/393069988_Automated_Policy_Enforcement_in_Terra*

Zhang, R., Smythe, M., Hooper, C., Hinrichs, T., Evenson, L. and Sandall, T. (2019). Opa gatekeeper: Policy and governance for kubernetes, `https://kubernetes.io/blog/2019/08/06/opa-gatekeeper-policy-and-governance-for-kubernetes/`. Accessed 8.1.25.