Optimizing Kubernetes Security through automated Policy Enforcement in Multi-Cloud Environment

MSc Research Project
MSc Cloud Computing

Soham Yadav
Student ID: X23173394

School of Computing
National College of Ireland

Supervisor: Prof. Rashid Mijumbi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Soham Yadav
Student ID:	X23173394
Programme:	MSc Cloud Computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Prof. Rashid Mijumbi
Submission Due Date:	12/12/2024
Project Title:	Optimizing Kubernetes Security through automated Policy Enforcement in Multi-Cloud Environment
Word Count:	2686
Page Count:	26

Signature:	Soham Yadav
Date:	28th January 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.	☐

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

Optimizing Kubernetes Security through automated Policy Enforcement in Multi-Cloud Environment

Soham Yadav
X23173394

Abstract

The use of Kubernetes has become crucial for the management of containerized applications in cloud environments. With rising advantages, there is a rise in complexity due to its dynamic nature, this brings in significant security challenges. Traditional security tools namely AnchoreCLI generate a higher rate of false positives, lack the ability to customize, and have a restricted ability to adapt to multi-cloud environments, exposing the Kubernetes environments vulnerable to evolving threats.

The study in this research proposes a novel approach by creating an automated security framework for Kubernetes to address the limitations. The framework will consist of a custom-built security scanning agent, which will perform active vulnerability detection by pulling data from the National Vulnerability Data (NVD).

In addition to the custom-built scanning agent, a dynamic and adaptive policy enforcement engine is also created. The policy engine automatically updates the policies based on the scanning results by the scanning agent. The policy enforcement engine applies Kubernetes configurations in real time. The crucial part of the framework is its seamless automation with the continuous integration and continuous deployment (CI/CD) pipeline. This provides the facility for automation of initialization of the security scanning agent and policy enforcement engine at appropriate stages during deployment to every environment. This pipeline ensures a secure deployment and minimizes manual work leading to the low rate of human errors. This creates a proactive solution to the evolving security threats.

The combination of real-time security scanning, adaptive policy enforcement, and seamless CI/CD automation of the process, the proposed framework provides a dynamic solution to the Kubernetes security complications. The study is a demonstration of the automation, customization, and adaptive approach for enhancing the security of Kubernetes clusters in multi-cloud environments.

Key Words - *Custom Security Scanning Agent, Adaptive Policy Enforcement, Kubernetes, Continuous Integration and Continuous Deployment (CI/CD), Azure, Google Cloud, National Vulnerability Database (NVD)*

1 Introduction

In this modern world, cloud-native technologies have changed the process of application, development, merging, and deployment. Kubernetes has emerged as a promising container orchestration platform, and its abilities, such as flexibility to merge, deploy, and scale the developed application over a multi-cloud environment, have catalyzed this transformation. Kubernetes seamlessly executes tasks like load balancing, scaling, and self-repair of workload, because of its dynamic nature of processing. This dynamic nature comes with its advantages but also increases the complexity, which introduces various security challenges, Rahman et al. (2023). As the load or tasks over a container increases, the possibility of exposure to vulnerabilities increases. For e.g. a tech giant like Tesla faced a Kubernetes consol attack by hackers and sensitive data was exposed Rahman et al. (2023), and another company like Shopify Bose et al. (2021) also faced a similar attack where restrictions were imposed to access the data, such incidents highlight the importance of Kubernetes security. This research aims to address such security challenges under the research question: "How can the security of Kubernetes clusters be enhanced through automated solutions?"

Tools that are already available such as AnchoreCLI, provide functionality for basic vulnerability scanning and policy enforcement, Wende (2024). However, there are certain limitations experienced with these tools, generation of high rates of false positives, and lack of customization ability in the tool, especially when it is required during working with a multi-cloud environment, Kamieniarz and Mazurczyk (2024). Another identified limitation is its restriction towards real-time changes in deployment policies, this can open the possibility of exposure to vulnerabilities causing a serious security issue. These voids highlight the requirement for a more dynamic, adaptive, and automated approach towards Kubernetes security.

The objectives of the research is an attempt to propose a novel framework for the enhancement of Kubernetes security by using automated solutions that consist of custom-built active security scanning and adaptive policy enforcement with the integration of the CI/CD pipeline.

- The first component of this research is the custom-built security scanning agent constructed to address the existing limitations. This tool extracts the real-time vulnerability data from the National Vulnerability Database (NVD) and performs scanning of container images. It focuses on accuracy and adaptability in order to reduce the rate of false positives and to incorporate customizations for environment-specific scans in a multi-cloud environment.

- Following the custom-built security scanning agent, an adaptive policy enforcement engine is also proposed. This policy enforcement engine acts as a bridge between the steps such as scanning and mitigating vulnerabilities. Based on the scanned results by the security scanning agent, and identified vulnerabilities, the policy engine dynamically generates the configurations YAML that make required changes in the policy to mitigate the issues. These policies include restrictions on incoming and outgoing traffic exposing vulnerable components or blocking entire access. This dynamic approach of policy enforcement focuses on Kubernetes cluster security against frequently evolving threats. The need for manual intervention in policy changes is minimized and the risk of human error is also lowered.

- The integration of the framework with CI/CD pipelines is one of the important aspects of this research. The custom-built security scanner and adaptive policy enforcement in this pipeline make sure that the security scans and policy updates get triggered automatically with each deployment. This automation process significantly reduces manual work and minimizes the possibility of human errors and this increases the rate of mitigation of security threats. This constructs a proactive security model that continuously scans and adapts to the dynamic nature of the Kubernetes environment, Raghunathan (n.d.).

The novel part proposed in this research consists of three components, real-time vulnerability detection, adaptive policy enforcement, and CI/CD pipeline automation into a scalable framework. This proposed solution addresses the limitations of the pre-existing tools responsible for carrying out similar tasks. The framework works parallel to the requirements of modern cloud architecture. Constructing a self-regulating security framework, this research aims to improve the overall Kubernetes security over a multi-cloud environment. It provides a robust solution for security and dynamic security challenges in the modern world's containerized application management.

This research is a contribution to Kubernetes in the field of security, Kamieniarz and Mazurczyk (2024), by providing a display of the potential of automated solutions for the healthy security of containerized workloads. It focuses attention on the integration of a custom-built real-time security scanning agent and adaptive policy enforcement engine with a CI/CD pipeline to construct a scalable security framework. The proposed framework addresses the challenges and complexity of security, immediately, as well as opens the door for future research in the automated security framework in cloud environments.

The research paper is segregated into the following sections, Section 2 provides a brief summary of the literature review or the related work done in the past, Section 3 describes the methodology for this research, Section 4 gives a detailed understanding of the implementation of the tools and technologies for this research, Section 5 discusses and evaluated the results and findings of the research along with the challenges faced and lastly Section 6 concludes the research along with the possible future scope of the research.

2 Related Work

Studying the existing related work, this research is an introduction to a automated framework for enhancing the security of the Kubernetes cluster. Unlike the existing work, the research addresses the gap between security scanning and policy enforcement based on scanned results. This focuses on end-to-end automation of the process and minimization of manual errors.

This section is bifurcated into 3 types based on the relevance of the research papers: subsection 2.1, subsection 2.2, and subsection 2.3.

2.1 Vulnerability Detection and Management

(Bose et al.) Bose et al. (2021) In his paper, the under-reported security defect in Kubernetes manifest, by analyzing 5,193 commits over a range of 38 open-source repositories. The study generated an output, that, just 0.79% of all the considered commits, were related to security which suggested that several security-related issues were unidentified. The author explains that such under-detected security issues can cause significant security risks, leading to exposure of vulnerabilities. The proposal is to create a tool, a detection mechanism for the detailed identification of security defects in the configurations and focuses on deep research to identify the cause behind these issues.

The research paper “Automated Vulnerability Scanning of Kubernetes During the CI/CD Process” Wende (2024) by Florian Wende focuses on the integration of vulnerability scanning tools within Kubernetes CI/CD pipelines. The vulnerabilities in Kubernetes are bifurcated into different categories such as Iac misconfigurations, third-party component flaws, and leaked secrets, highlighting the complexities of managing vulnerabilities in the automated environment. The research displays the upper hand of combining open-source scanning tools to enhance the vulnerability detection rate.

Rahman et al. Rahman et al. (2023) in the research have identified 11 different categories of vulnerabilities by analyzing the security misconfigurations in the Kubernetes manifests, to name a few, absent security context, resource limits, and misuse of privileged configurations. The author proposes a tool, Security Linter for Kubernetes Manifests (SLI-KUBE), this tool improves vulnerability detection using static analysis and minimizes false positives with the help of def-use chain analysis. The research focuses on the critical role of rigorous static analysis and emphasizes best practices.

Kamieniarz and Mazurczk Kamieniarz and Mazurczyk (2024)studied and analyzed Kubernetes security in four deployment models. These deployment models are divided into two categories, firstly 0n-premise clusters namely K3S and Rancher Kubernetes Engine 2, and secondly managed services namely Amazon Elastic Kubernetes Service and Google Kubernetes Engine. The author proposed and developed a tool “kube-security-scanner” which can aggregate outputs of open-source tools like Kubescape, trivia, kube-score, and kube-bench which are already available in the market to provide the functionality of automatic security scanning and provide valuable insights related to misconfigurations and container vulnerabilities. The result of the analysis depicts that the managed services have better security whereas the on-premises clusters possess more misconfiguration issues.

The author Oluyede et al.Oluyede et al. (2024) in the research studies the security challenges in container-based systems in the cloud environments. The focus is on identifying the vulnerabilities related to container images, misconfigurations, and weak runtime environments. It highlights enhancements in the CI CD pipeline such as Kernel isolation mechanisms and access control strategies. The article identifies four important use cases for container security. It also suggests combining software solutions with hardware technologies for significant protection.

The article “Testing the Security of a Kubernetes Cluster in a Production Environment”, Giangiulio and Malmberg (2022), by Francesco Giangiulio and Sebastien Malmberg focuses on the security vulnerability in Kubernetes which contains custom-managed applications. The research paper analyses Precio Fishbone’s Omnia application deployed on Azure Kubernetes Service and experiments with cross-tenant isolation within the Kubernetes namespaces. The crucial vulnerabilities found during the experiment were a lack of network policies, default access to the root container, and misconfiguration in admin privileges. Penetration testing is used to recreate the attack scenarios and demonstrate the exploitation of the gaps. The author of this research paper proposes mitigating processes such as implementing Role-Based Access Control, creating network policies, and restricting account permissions for security enhancement.

2.2 Multi-Cloud and Network Policy Integration

The research paper "Network Policies in Kubernetes: Performance Evaluation and Security Analysis",Budigiri et al. (2021) studies the Network policy of Kubernetes, focusing on the performance and security effects on the 5G edge computing environments. The research focuses on the implementation of Container Network Interface plugins such as Calico and Cilium that enforce the network policies without affecting much of the performance, providing an advantage for low-latency communication. The author emphasizes the need for automation of policy verification for robust security.

Osmani et al.Osmani et al. (2021) in the research paper point out the limitations of Kubernetes with regard to multi-cloud and multi-cluster environments within 5G networks. The novel proposal of integrating Federated Kubernetes (KubeFed) with Network Service mesh (NSM) to provide multi-cloud connectivity and management workload dynamically. The solution proposed consists of secondary networking, service chaining, and connectivity without network address translation (NAT). The proposed framework incorporating NSM proxyless data plane and KubeFed API, results in low latency and high throughput. The research work completely focuses on innovating multi-cloud Kubernetes networking.

The research paper "Enabling Service Mesh in Multi-Cloud Environment" Giandonato (2021) studies the service mesh technologies such as Linkerd and Istio within Liqo multi-cloud architecture. The author describes the identified challenges in a multi-cloud Kubernetes environment. The challenges such as namespace reflection, mutual TLS (mTLS) certificate management, and cross-cluster service communication. Aligning with Liqo's capabilities, this research paper proposes a prototype to enhance cloud security and communication. In all, it focuses on extended service mesh solutions.

The author Muhammad Waseem et al.Waseem et al. (2024) studies containerization in multi-cloud environments. The research paper has divided the complications or challenges into four different frameworks, namely, security, automation, deployment, and monitoring. There are 74 implementation strategies and frameworks, which also focus on limitations in automated container security scanning and adaptive policy enforcement. It provides a wide range of exposure to the architectural patterns, strategies, and attributes of containerization.

The author Ganne Ganne (2022) analyzes the comparison between the security mechanism of Kubernetes and Docker Swarm in managing containerized workloads for cloud environments. The author displays the advantages of Kubernetes over Docker specifically with respect to scalability, fault tolerance, and features such as network policies and role-based access control. Kubernetes proves to have the upper hand in secure workload isolation and dynamic scaling, essential aspects for data protection in the distributed cloud environment. The article also inspects the encryption techniques for secure data transmission and storage.

In the research paper "Kunerva: Automated Network Policy Discovery Framework for Containers", Lee and Nam (2023) the author focuses on a framework for discovering network policies in a dynamic containerization environment such as Kubernetes. The network logs are used to automate the generation of the network policies that are used for container isolation. The framework integrated with the existing policy enforcement tool Gatekeeper to validate and enforce policies to reduce the risk of misconfiguration. In addition to this, the real-life implementation of the framework demonstrated its adaptability to factors like network changes with minimal overhead, and better scalability, as it proves to be a useful asset for managing security in complex and dynamic environments.

2.3 DevSecOps and Automation in Kubernetes

The author Savitha Raghunathan Raghunathan (n.d.) addresses several challenges such as rapid development cycles, dynamic deployments, and evolving security threats by exploring the strategies for embedding DevSecOps practices in Kubernetes environments. The research focuses on threat modeling, static and dynamic code analysis, and container scanning during the CI/CD pipeline workflow. The container vulnerability is identified using tools like Clair and Trivy and the network policy in Kubernetes is monitored for secure pod communication. In addition to this, anomaly detection and trust verification are carried out using Falco and in-toto tools.

Blomqvist et al.Blomqvist et al. (2021) in the research describe the utilization of Hashicorp Vault for automating secrets management in multi-cloud Kubernetes environments. The author focuses on the vault's capabilities in terms of secure storage, secrets rotation, and centralized lifecycle management and its integration with the CI/CD pipeline. It highlights the use of tools like Terraform for Kubernetes security injection. The article also highlights the challenges related to secrets management in multi-cloud settings such as secret sprawl and ensuring secure success control.

The authors Amir Boroufar and Pasquale Boroufar (2020) Lepera contribute to the study of challenges that occur during the deployment of microservices in a multi-cloud environment. This is aligned with a focus on Kubernetes and Istio to handle traffic management and system flexibility. The proposal architecture in this research integrates OpenVPN for secure networking, Istio for traffic control and service mesh, and CI/CD pipelines for automated software delivery. The research also identifies some gaps in securing the CI/CD pipelines and container management in multi-cloud scenarios, but the study successfully displays the advantages of using multi-layered DevOps for distributed microservices.

Mustyala and Tatineni Mustyala and Tatineni (2021)in the research paper "Advanced Security Mechanisms in Kubernetes Isolation and Access Control Strategies" explore the security of Kubernetes and it mainly focuses on isolation techniques, sandboxing tools, and access control strategies. The crucial points discussed in this study are namespaces, network policies, and pod security policies for isolation methodology with an aim to minimize the attack surface and mitigate risks of privilege escalation and lateral movements. The tools used for the purpose of sandboxing are gVisor and kata containers, which enhance workload isolation due to the lightweight virtual machine architecture. The author in this research touches on several aspects such as access control mechanisms, specifically Roal-Based Access Control and Attribute-Based Access Control with the integration of an Open Policy Agent for policy enforcement and external secret management.

The authors Voievodin and Rozlomii Voievodin and Rozlomii (2024) explore the possible security complications in container orchestration systems such as Kubernetes, Docker Swarn, and Apache Mesos. The research discusses the possibility of unauthorized access or service disruptions due to insecure configurations, supply chain vulnerabilities, and misconfigurations. The focus is on the strategic scheduling decisions to improve security using distributed solutions. The critical components are distributed among the nodes to reduce the impact of attacks. The proposal by the authors is to use node-weighted and ranking algorithms to balance load properly.

Kermabon-Bobinnec et al., Kermabon-Bobinnec et al. (2022), in the research paper "ProSPEC: Proactive Security Policy Enforcement for Containers" propose an approach for security policy enforcement in Kubernetes. The paper points towards the limitations of reactive security tools like OPA/Gatekeeper which have limitations such as introducing delays and exposing vulnerabilities due to their state-replication mechanism. The ProSPEC uses predictive modeling using Bayesian networks to anticipate security events and pre-verify policies. This ensures quick policy enforcement in about 10 ms for around 800 pods. It focuses on the seamless integration of Kubernetes with policy tools and suggests future improvements in the form of support for other orchestrators.

3 Methodology

The methodology for this research is an attempt to provide a detailed and systematic approach to address and mitigate the Kubernetes security challenges in a multi-cloud environment. As Introduced in the beginning, Kubernetes is a commonly used container orchestration platform but often faces vulnerabilities with increased complexities during dynamic deployments, policy enforcement, and real-time security scanning. The research proposes a novel framework to integrate the custom-built components into the CI/CD pipeline to create seamless automation for mitigating challenges. This section focuses on the key elements of this research.

This section is divided into specific sections and subsections describing the crucial elements of this research in detail:

Architecture Design: This subsection provides a high-level description of the framework's architecture design. It provides an understanding of the integration of key components and emphasizes multi-cloud adaption for scalability.

Key Components: This sub-subsection delves into the crucial components of this research. The components such as "Custom Security Scanning Agent", and "Adaptive Policy Enforcement" are explained in detail.

The proposed methodology is built upon the relative existing research with an introduction to innovative features. The existing solutions such as "Calico", "Cilium" address Kubernetes security like network policies and vulnerability scanning but lack real-time adaptability. This research proposal fills the gaps by connecting Security Scanning Agents and Policy Enforcement and automating them through the CI/CD pipeline.

3.1 Architecture Design

The architecture design of the proposed framework for Kubernetes security in a multi-cloud environment is planned for a proper connection between the key components including "Custom Security Scanning Agent", "Adaptive Policy Enforcement", CI/CD pipeline, and multi-cloud environment.

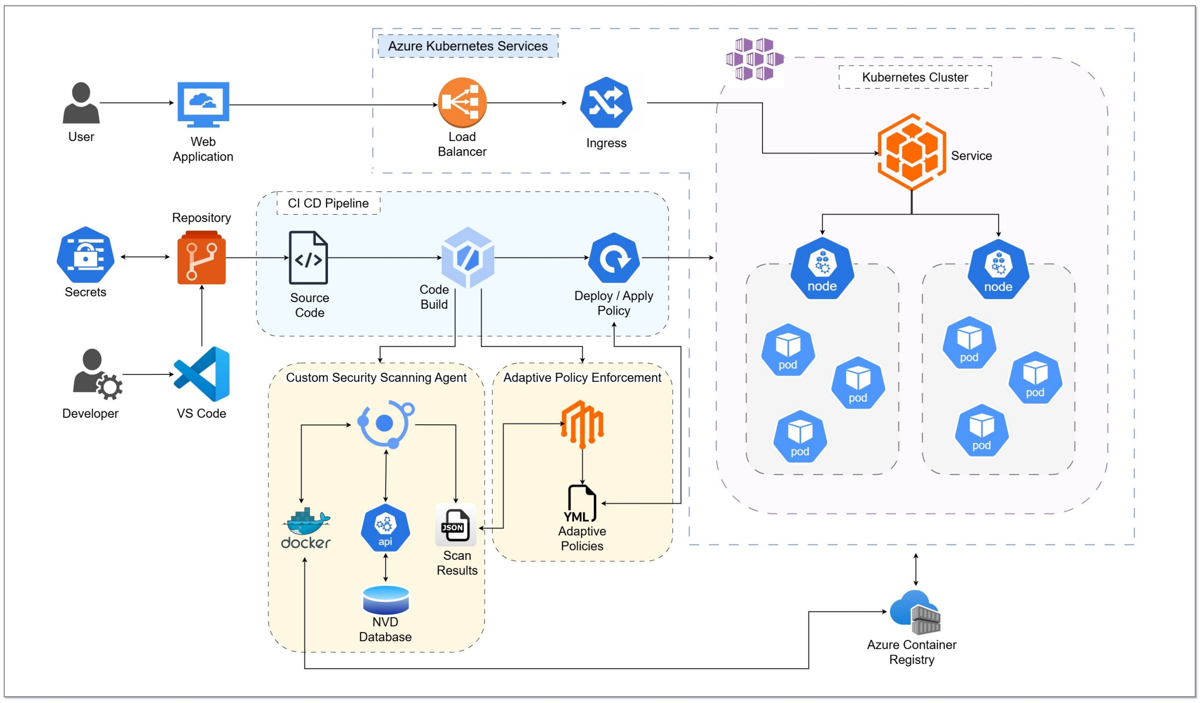


Figure 1: Architecture Diagram

The architecture diagram, refer to the image 1 demonstrated a detailed connection of all the components used in this research as well as provides an understanding of the sequential workflow between all the components.

A developer develops a code for any application in any of the desired IDE, In this case "VS Code" is used. Once the development is completed, along with all the required YAML files for container and Kubernetes configuration, the code is pushed to the version manager, in this case, it is "GIT". "GIT" has crucial keys stored in its secrets, which can be used whenever required. Once the code is pushed to "Git", the CI/CD pipeline "Git Jobs" gets triggered. These jobs are the steps or actions that are linked to each other to create seamless automation. At the code build stage, the "Custom Security Scanning Agent" job is executed, in which the metadata of "Docker" images created in the earlier action is extracted from the container registry of "Azure" or "Google Cloud" and that metadata is compared or matched with the "National Vulnerability Data NVD", then if the vulnerabilities are identified, they are stored in "Scan Results.json" YAML file. Further, these scanned results are then provided to "Dynamic Policy Enforcement" where the policies are created to mitigate the scanned vulnerabilities, and these policies are stored in another YAML file "Adaptive Policies.json". This YAML file with stored policies is applied based on the coverage required such as pod level, node level, and cluster level. Then the secured application is deployed and available for the user to access. This process is a complete automation of identifying the vulnerabilities in Kubernetes and mitigating them by policy enforcement and applying required restrictions for security enhancement. When the user tries to access the application, it will get access to only those areas that are allowed by the policies, securing unwanted privileges.

3.1.1 Architecture Overview

The design can be understood through different sections or layers:

1. Development and Containerization layer:
   - A small application is developed using the "Java Spring Boot" framework and the version control is provided by "GIT".
   - Docker technology is used for containerization and the created images can be scanned, stored, and deployed.
   - The configuration files in the form of YAML are stored at the application code level to define Kubernetes configurations in terms of deployments, services, and policies.
2. CI/CD Automation Layer:
   - All the processes such as application building, containerizing, scanning vulnerabilities, and adaptive policy enforcement are automated using the CI / CD pipeline.
   - The pipeline stages can be summarized as Build, Scan, Enforce policy, Deploy, and Monitor.
3. Deployment Layer:
   - The use of Kubernetes is deployed across clouds such as Azure ensuring scalability, fault tolerance, and distributed workload management.

3.1.2 Key Components:

1. Custom-built Security Scanning Agent:

   One of the crucial components of this research is the custom-built security scanning agent. The aim is to counter the limitations of the existing security scanning tools by providing an adaptive and automated solution for the Kubernetes environment in multi-cloud scenarios. This agent is designed to dynamically analyze the containerized application for vulnerabilities during the CI/CD processes and at the built phase. It makes use of the National Vulnerability Database to validate with real-time data. The approach increases accuracy in vulnerability detection and minimizes the rate of false positives.

   Purpose and Scope of the Custom-built Security Scanning Agent:

- Customizable scanning logic: Customization ability to specific deployment environments and applications needs.
- Integration with multiple cloud environments: Seamless operations across Kubernetes cluster in "Azure Cloud Services", "GCP".
- Real-time validations: Active synchronization with "NVD" to ensure up-to-date vulnerability assessments.
- Actionable Results: Focus on critical and high-severity vulnerabilities, providing concise and relevant insights to ensure Kubernetes security.

Use of National Vulnerability Database (NVD):

NVD is a public database that provides details about the vulnerabilities, including the level of severity (CVSS scores) as well as the version of the software that was affected during the process. It is a key component of the Custom-built Security Scanning Agent.

- Reliability: "NVD" is widely known as the best for vulnerability information in security research and practices.
- Real-time data access: The agent uses the "NVD CVE API 1.0" to fetch the latest vulnerabilities.
- Standard Severity Scoring: It provides a trustable and uniform severity assessment of the data stored in "NVD".

Mechanism of Vulnerability Detection:

(a) Container Metadata Extraction:
   - The metadata such as the package name and version are extracted from the docker images.
   - The metadata should be compatible with the dynamically generated Images during the CI/CD process.

(b) Querying the NVD:
   - An HTTP GET request is sent to NVD API with the required keywords such as package names.
   - Filters must be able to match the results within the package versions present in the container.

(c) Severity-Based filtering:
   - Based on the CVSS scores, the filtering can be done based on the levels of severity, and relative results can be obtained.
   - This helps in decreasing noise and priorities that are actual threats for immediate remediation.

Code Workflow and Functionality, refer to img 2:

(a) Image analysis:
   - The metadata of the docker image is extracted.

   (b) Real-time vulnerability matching:
      - Querying the NVD to match vulnerabilities based on the extracted metadata.
      - The CVE entries to the package version to identify vulnerabilities relevant to the current application specifications.
   (c) Severity Assessment:
      - Categorizes vulnerabilities based on the level of severity which is defined based on CVSS score, such as critical, high, medium, and low.
   (d) Report Generation:
      - A JSON report is structured, that includes details about the detected vulnerabilities, that include package name and version, CVE ID, severity score, and description of the vulnerabilities.

   Integration with the Framework:

   The Custom-Built Security Scanning Agent integrates seamlessly into the research framework.

   - Triggering during CI/CD: Whenever the new images are pushed or built, scanning gets initiated automatically.
   - Providing real-time insights: The scan generated ascendable reports that act as input to the adaptive policy enforcement.
   - Reducing Manual Intervention: It completely automates vulnerability scanning and reporting, streamlining security processes.

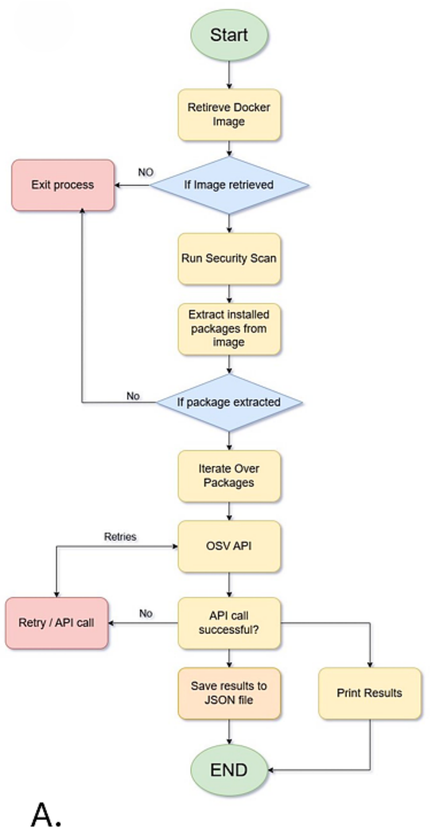
2. Adaptive Policy Enforcement:

   Adaptive Policy Enforcement is a fundamental part of this research, that addresses the limitations and acts as a bridge to overcome the limitations of the static policies in the Kubernetes environment. The proposed mechanism automates policy enforcement in Kubernetes based on the scanned results of the vulnerabilities by the Custom-built Security Scanner Agent. This dynamic functionality ensures increased Kubernetes security and its robustness towards evolving security, as well as reduces manual intervention minimizing the scope of human errors.

   Purpose and Scope of the Adaptive Policy Enforcement:

   The traditional Kubernetes policy configurations are defined manually and are dynamic in nature. This limits its ability to modify the policies dynamically or automatically for the detection of vulnerabilities. Existing tools such as Calico and Cilium provide a significant policy framework but lack dynamic adaptability based on the Custom-built Security Scanning Agent's results. To overcome this gap in the security process, Adaptive Policy Enforcement is proposed in integration with the CI/CD pipeline that will seamlessly mitigate the identified vulnerabilities.

   Mechanism of policy generation and application:



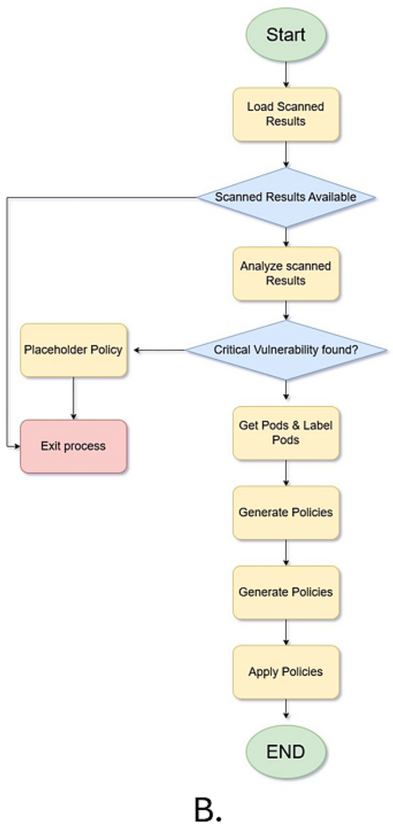


Figure 2: Algorithm flow-chart of A. Custom Security Scanning Agent and B. Adaptive Policy Enforcement

The key steps taken by Adaptive Policy Enforcement for policy generation are as follows:

(a) Input from Custom-built Security Scanner Agent:

- The Custom-built Security Scanner Agent generated results of vulnerabilities with details such as affected packages, severity levels, and potential impacts. A critical vulnerability detected in the application may trigger an immediate policy update.

(b) Adaptive Policy Enforcement:

- Based on the scanned results, the system dynamically generates YAML-based Kubernetes policies.
- If a severity level is high for the detailed vulnerabilities, policies can block all incoming and outgoing traffic to the affected container.
- For lower levels of security, traffic may be restricted to a specific IP range.
- These policy updates are done programmatically using Python.

(c) Automated policy application:

- The generated policies are applied automatically to the Kubernetes cluster. The step gets triggered automatically by the CI/CD pipeline and minimizes the latency between vulnerability detection and policy enforcement.

Explanation of the Adaptive Policy Enforcement Code Flow, refer to Img 2:

The policy enforcement is established using a Python script that automates the entire workflow, from reading scan results to applying updates.

(a) Triggering Security Scans:
  - The script periodically involves a custom security scanning Agent to fetch the latest vulnerability data via an API endpoint.

(b) Policy Decision Logic:
  - Based on the scanning results, the policy is modified to mitigate the identified vulnerabilities.
  - Example 1: Block incoming and outgoing traffic if a critical vulnerability is detected.
  - Example 2: Restrict incoming and outgoing traffic to a certain IP range only.

(c) Yaml Policy Generation:
  - This Adaptive Policy Enforcement script generates Kubernetes network policy YAML files based on the new policy changes.

(d) Applying Policies:
  - The script applies the generated policies to the Kubernetes cluster, to reduce latency in threat mitigation.

3. Integration into CI/CD pipeline:

   The CI/CD pipeline, refer to Img 3 and 1, is one of the crucial parts of this research. The key components of this research, which are “Custom-built Security Scanning Agent” and “Adaptive Policy Enforcement” are integrated into the CI/CD pipeline for seamless automation of the process. This subsection will provide an understanding of the connection between the key components and Kubernetes over a multi-cloud Kubernetes management.

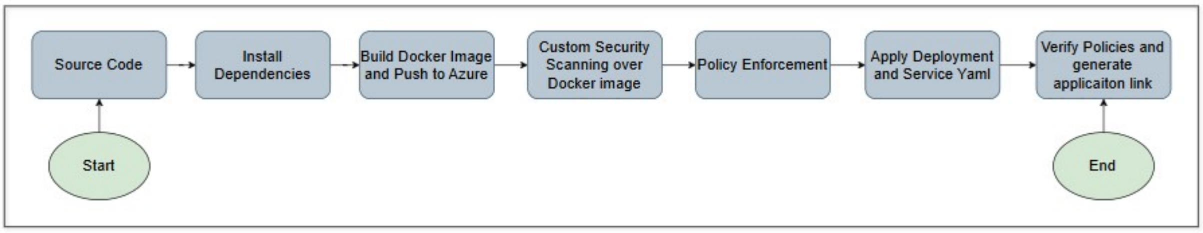

Start
Source Code
Install Dependencies
Build Docker Image and Push to Azure
Custom Security Scanning over Docker image
Policy Enforcement
Apply Deployment and Service Yaml
Verify Policies and generate applicaiton link
End

Figure 3: CI CD Pipeline Build Phase

Purpose and Scope of CI/CD pipeline:

The CI/CD pipeline aims to achieve complete automation for deploying secure Kubernetes.

(a) Automating vulnerabilities Detection:

- The pipeline triggers the custom security scanning agent for the identification of vulnerabilities during the build phase. This provides results of open vulnerabilities in the docker image.

(b) Adaptive Policy Enforcement:

- The automatic enforcement of the policies is triggered based on the scanned results of the Custom-built Security Scanning Agent.

(c) Continuous Monitoring:

- Active monitoring throughout the application and its health is achieved without any manual intervention.

Workflow of the Stages, refer to Img 3 and 1:

The pipeline can be divided into several stages for understanding purposes:

(a) Code Commit: The code developed is pushed to a version control technology, which triggers the CI/CD pipeline.

(b) Build Stage: The pipeline builds Docker images from the application code by utilizing pre-defined configurations.

- Custom-built Security Scanning Agent: The custom-built Security Scanning Agent analyses the docker image for the image details and verifies it with NVD data for vulnerability identification.
- Policy Updating Stage: The updated policies are enforced and applied to Kubernetes as per the changes required in scanned results provided by the Custom-built Security Scanning Agent.

(c) Deployment Stage: Once all security checks are passed, the pipeline deploys the application to the Kubernetes cluster in the cloud.

4 Implementation

The implementation section of this research report deals with the implementation of an automated security framework, which is the proposed solution for challenges in security faced by Kubernetes in a multi-cloud environment. This section provides an overview of the steps for configuring the required environments and technologies for the research. The workflow of the framework is seamlessly automated using Git Workflows which is the integration of two critical components, "Custom Security Scanning Agent" and "Adaptive Policy Enforcement" enhancing the software delivery lifecycle. The components are designed to adapt customization and can be scaled based on the requirements enabling secured deployments of Kubernetes clusters in Azure and Google Cloud environments.

The primary objective is to provide details about the steps taken into consideration while developing the automated framework tailored to securing Kubernetes clusters. Primarily the tools and technologies considered for the implementation of this research idea are summarized as follows:

1. Programming Languages:

- Java Spring boot:
  It is used to develop a small Java web-based application for testing or deployment purposes. It is a robust framework for creating applications and provides the ability to vide a range of integrations of tools and other technologies.
- Python:
  It is used for developing the “Custom Security Scanning Agent” and “Adaptive Policy Enforcement” scripts as Python has a huge set of libraries that are easy to use and are easy to integrate with API and YAML files.
- Bash:
  It is used for shell scripting for Kubernetes and Docker integrations and management.

2. Infrastructures:

   - Kubernetes:
     Works as the container orchestration technology for deployment and management of applications and security policies.
   - Docker:
     It is used to containerize the application and its dependencies for enhancing application maintenance over cloud environments.

3. CI CD Pipeline:

   - Git Workflows:
     It is used for creating the automation CI CD pipeline using Git Actions, which includes application image building, Scanning, Policy Enforcement, and deployment of code or images including push or pull events of images.

4. Cloud Environments:

   - Azure Kubernetes Services and Google Kubernetes Engine:
     Deployment over a multi-cloud infrastructure to test the scalability and effectiveness of the proposed framework.

5. Data Source:

   - National Vulnerability Data:
     The source of vulnerability data used by the “Custom Security Scanning Agent” to validate the vulnerabilities in the image is accessed via NVD API.

4.1 Development Environment Setup

The development environment setup is the foundational layer of the implementation that deals with the base setup of the tools, infrastructure, and other essential requirement of the research. Each component used in this process enhances the automation ability of the proposed framework.

1. Tools and Technologies Setup:

- Integrated Development Environment (IDE):
  Visual Studio Code is used as the IDE to develop all the codes and configuration files required for this implementation. A wide range of extensions were installed into VSCode to support seamless integration. Extensions such as Azure, Spring Boot, Docker, Kubernetes, and other relative plugins.
- Version Control:
  Git and GitHub are used for managing the versioning of application code, Docker files, Kubernetes manifests, and almost every file in the project folder. In addition to this, CI CD is utilized in the form of Git actions to create and automate the pipeline of seamless end-to-end automation of triggering the research components, analyzing the images, and applying policies and deployments.
- Kubernetes CLI:
  The Kubernetes command line tool is set up for the management of Kubernetes clusters, deploying applications, and applying changes in policies. These Kubectl commands are integrated within the CI CD workflow. In addition to this, Helm libraries are also installed to support the integration of tools like Prometheus and Grafana for monitoring purposes.

2. Cloud Environment Setup:
   - Azure Kubernetes Services (AKS):
     For this research, a free version of Azure cloud is purchased and the setup is done using Azure CLI in integration with Azure Container Registry for storing container images. It is directly connected to the CI-CD pipeline for the deployment and applying policies at the end of the CI-CD process.
   - Google Kubernetes Engine (GKE):
     A free version of Google cloud is purchased and it is setup using an API configuration for communication between the CI CD pipeline and Kubernetes Cluster.
3. Application Development and Setup:
   - Project Structure:
     - Maven Project:
       The Maven ecosystem was used for managing dependencies and jar files of the test application.
     - Application:
       The Application is created under the Maven ecosystem and the code, dependencies, and configuration files are stored in Git version control.
   - Containerization:
     - Docker technology is used for containerizing the application. For this process, a Docker desktop is downloaded and a Dockerfile is created at the application level to install the required packages and set the Docker configuration to expose the application on port 8081.
     - The images created are dynamically pulled during the Continuous Integration and Continuous Deployment process for ease of analysis and deployment.

- Configurations:
    - The Kubernetes configuration files, which include deployment and service YAML files are developed at the project folder level within the VSCode. These files define the amount of replicas required, resource limits, and initial security of pods.

4.2 Kubernetes Security Framework Setup

The framework is a combination of two crucial components developed for this research. These components facilitate automated vulnerability detection and adaptive policy enforcement and these components are integrated into a CI CD workflow to secure Kubernetes deployments.

4.2.1 Custom Security Scanning Agent

The Custom Security Scanning Agent is a Python script developed at the root level in the project folder. The idea behind developing this script is to detect vulnerabilities within docker images by extracting its metadata and querying the National Vulnerability Data, database using an API. The script process is as follows:

- Image Meta Data Extraction:
  The script extracts the package metadata from the docker images by communicating with ACR and using "dpkg-query". This metadata provides information such as package names and versions installed in the Kubernetes. This provides compatibility with Devian images which are used for the Spring Boot application.

- NVD Database:
  Each extracted image metadata is queried upon by using the data extracted from the NVD database for vulnerabilities. The vulnerabilities are prioritized based on the Common Vulnerability Scoring System (CVSS) scores which define the criticality level of the vulnerability.

- Results:
  The processed results are stored in a JSON file by filtering the duplicate entries and these results will be further utilized for Adaptive Policy Enforcement.

4.2.2 Adaptive Policy Enforcement

The Adaptive Policy Enforcement is another Python script developed at the root level of the project folder. It provides a set of actionable Kubernetes policies by considering the vulnerabilities listed in the scan results JSON by the Custom Security Scanning Agent. The script process is as follows:

- Policy Generation:
  The Python script extracts the data listed in the scan results JSON file and generates a YAML configuration file with the list of policies that are required to be applied. Just like, traffic to a pod can be blocked which has a high severity of vulnerabilities, and securing the IP range.

- Policy Application :
  Policies listed in the YAML file are then applied to Kubernetes using “Kubectl” commands, minimizing the latency between the detection and mitigation of vulnerabilities. The strict pod security admission policy is achieved by updating the Namespace-level security labels such as “pod-security.kubernetes.io/enforce”.

- Adaptability:
  The policies applied are in the process of being applied to target specific pods and namespaces based on the analysis and findings.

4.3 CI CD Pipeline

The GitHub Actions is used in this implementation for developing the CI CD pipeline defined in ci-cd-pipeline YAML which is stored in the project folder, for integrating all the components and providing complete automation of the workflow of the framework, triggered by code commits. The pipeline is aligned with the DevSecOps principle by integrating security validation within the pipeline workflow. The Pipeline is developed accordingly:

- Build Stage:
  The pipeline builds a docker image of the Spring Boot application by using Docker file which has configuration details and is created at the root level in the project folder. The build created is tagged with a unique identifier for ease of traceability.

- Security Scanning Stage:
  After the image is created, it triggers the Custom Security Scanning Agent to analyze the image for any available vulnerabilities and if found any, storing it in scan results JSON file.

- Adaptive Policy Enforcement:
  This stage is triggered after the successful implementation of the Security Scanning Stage. The scan results are analyzed by the Adaptive Policy and generated adaptable Kubernetes policies which are applied to the Kubernetes cluster using “Kubectl” commands.

- Deployment Stage:
  The pipeline then deploys the application to Kubernetes clusters which are hosted on Azure and Google Cloud. The stage makes use of deployment, service, and adaptable Kubernetes policies YAML files which are also developed and stored at the root level in the project folder.

- Verifications:
  Post deployment the policies applied are verified using “kubectl” commands and the external IP for accessing the application is provided in the form of logs to access the application.

4.4 Multi-Cloud Deployment

The implementation of a multi-cloud environment provides consistent security. This subsection deals with two cloud services used in this research.

1. Azure AKS:

    - The Azure Kubernetes clusters are set up using Azure CLI and Azure Active Directory. These clusters are integrated with Docker using Azure Container Registry, ensuring continuous push and pull of images across clusters.
    - The Last phase of the CI CD pipeline deploys the application and applies the adaptive policies to the Kubernetes cluster within the AKS.

2. Google GKS:

    - The GKS clusters are set up using the Workload Identity in Google Cloud Console for communication between GCP services and Kubernetes.
    - The last phase of the CI CD pipeline communicates with the GKE Authentication to ensure that the images deployed are scanned and analyzed.

4.5 Monitoring and Validation

To subsection deals with integrating the ability to monitor the application performance and overall cluster health from a security point of view. Also, this will validate the framework mechanism. To achieve this, the following tools are considered in this research:

1. Prometheus:

    - To configure Prometheus, Helm needs to be installed first, after that the Prometheus repository is added to the Helm and it is installed in the next step in the monitoring namespace.
    - The installed Prometheus is accessed locally by port forwarding the Prometheus UI to a specific port, in this research, it is 9090, and the address to access the Prometheus dashboard is localhost:9090.

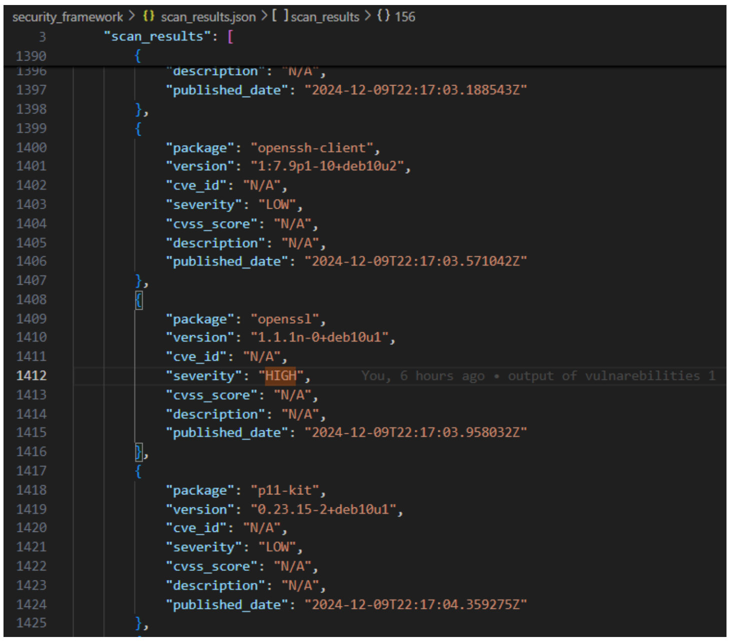
2. Grafana:

    - If the Helm is already installed, directly install Grafana using the command in the CMD or PowerShell.
    - The visualization of the matrix of Prometheus is available in Grafana using the predefined dashboards. The different panels on the dashboard display different information.
    - Grafana dashboard is port forwarded to a specific port, in this research, it is 3000, and the address to access the Prometheus dashboard is localhost:3000.

5 Evaluation

This section evaluates and discusses the successful implementation of the research on the Kubernetes security framework and provides a comprehensive understanding of the framework and its components with the help of results and findings acquired during the process.

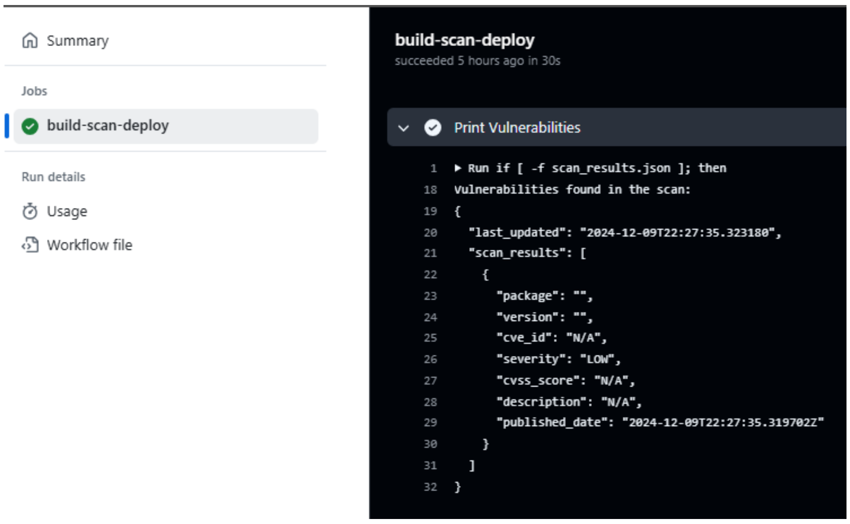
5.1 Case Study 1 - Custom Security Scanning Agent



```
security_framework > {} scan_results.json > [ ] scan_results > {} 156
    "scan_results": [
        {
            "description": "N/A",
            "published_date": "2024-12-09T22:17:03.188543Z"
        },
        {
            "package": "openssh-client",
            "version": "1:7.9p1-10+deb10u2",
            "cve_id": "N/A",
            "severity": "LOW",
            "cvss_score": "N/A",
            "description": "N/A",
            "published_date": "2024-12-09T22:17:03.571042Z"
        },
        {
            "package": "openssl",
            "version": "1.1.1n-0+deb10u1",
            "cve_id": "N/A",
            "severity": "HIGH",        You, 6 hours ago • output of vulnerabilities 1
            "cvss_score": "N/A",
            "description": "N/A",
            "published_date": "2024-12-09T22:17:03.958032Z"
        },
        {
            "package": "p11-kit",
            "version": "0.23.15-2+deb10u1",
            "cve_id": "N/A",
            "severity": "LOW",
            "cvss_score": "N/A",
            "description": "N/A",
            "published_date": "2024-12-09T22:17:04.359275Z"
        },
```

Figure 4: CI CD Pipeline Build Phase

One of the crucial components of this study is the Custom Security Scanning Agent, it is integrated with the CI CD Pipeline to analyze the docker images and quarry the metadata to find vulnerabilities. The scanned results provide output stating the name of the vulnerability (e.g. openssl and p11-kit) along with a few more details like severity (e.g. HIGH and LOW) based on its security score by CVSS. This output depends upon the response received by the “NVD API”, where if available, it also provides a description of the recorded vulnerabilities. The scanning process took approximately 30 seconds which demonstrates efficient functionality. The approximation of policy percentage with HIGH severity can be around 5 percent, refer to img 4 and 5.



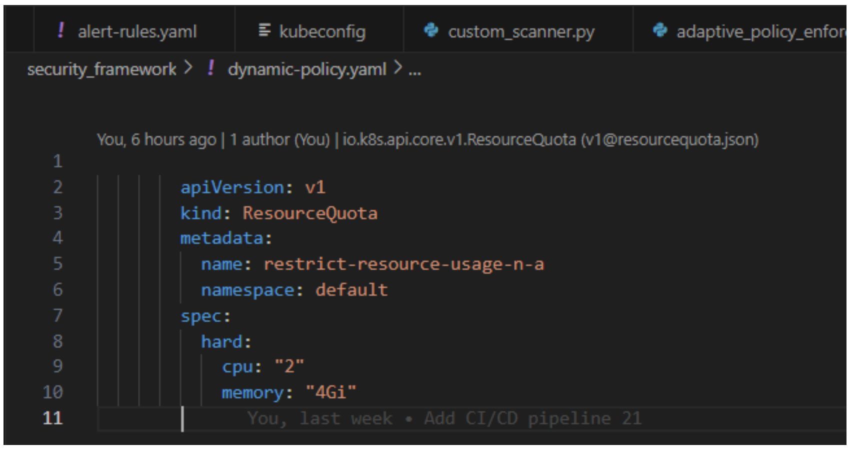
build-scan-deploy
succeeded 5 hours ago in 30s

Print Vulnerabilities

```
▶ Run if [ -f scan_results.json ]; then
Vulnerabilities found in the scan:
{
  "last_updated": "2024-12-09T22:27:35.323180",
  "scan_results": [
    {
      "package": "",
      "version": "",
      "cve_id": "N/A",
      "severity": "LOW",
      "cvss_score": "N/A",
      "description": "N/A",
      "published_date": "2024-12-09T22:27:35.319702Z"
    }
  ]
}
```

Figure 5: CI CD Pipeline Build Phase

5.2 Case Study 2 - Adaptive Policy Enforcement

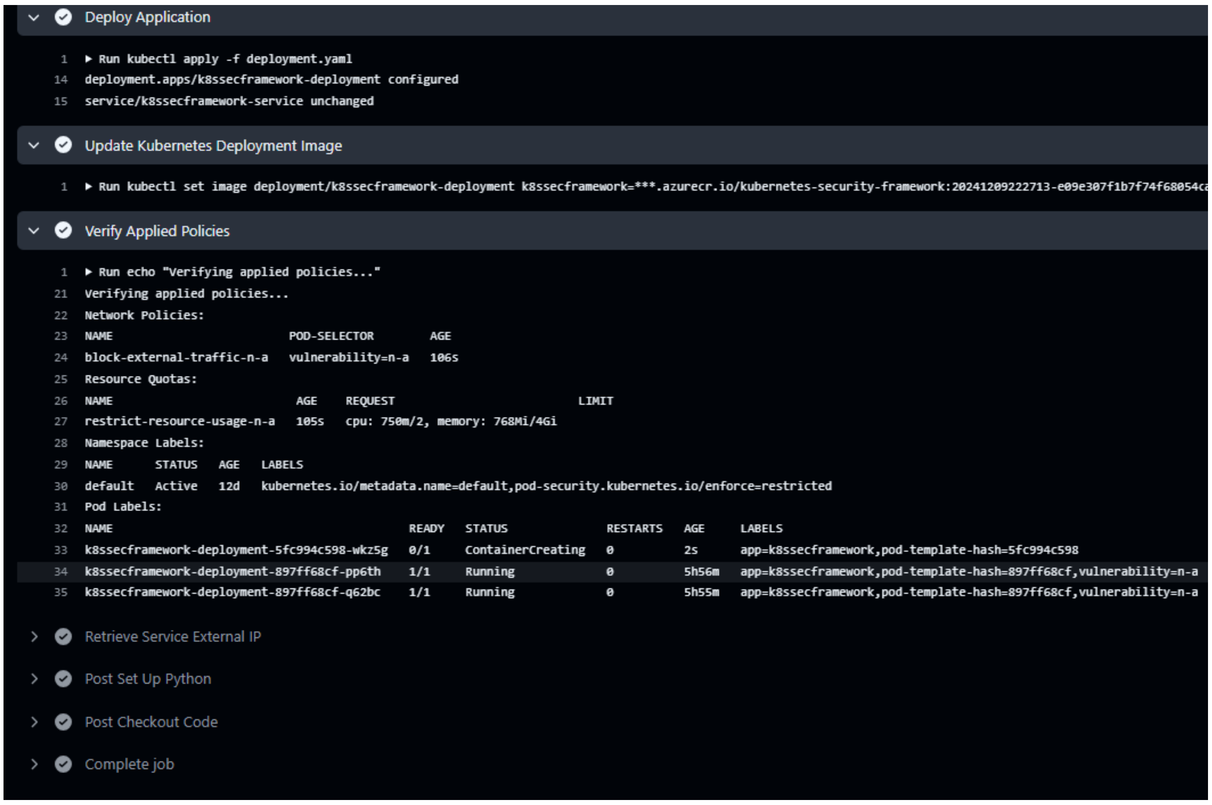

! alert-rules.yaml
kubeconfig
custom_scanner.py
adaptive_policy_enfor
security_framework > ! dynamic-policy.yaml > ...
You, 6 hours ago | 1 author (You) | io.k8s.api.core.v1.ResourceQuota (v1@resourcequota.json)

```yaml

apiVersion: v1
kind: ResourceQuota
metadata:
  name: restrict-resource-usage-n-a
  namespace: default
spec:
  hard:
    cpu: "2"
    memory: "4Gi"
      You, last week • Add CI/CD pipeline 21
```

Figure 6: CI CD Pipeline Build Phase

The second crucial component of the research is Adaptive Policy Enforcement, it is also integrated in the CI CD pipeline and takes the scanned results stored in JSON file as its input. Based on that policies like Network policy (e.g. blocking external traffic for pods with certain vulnerabilities)and Resource Quota policy (e.g. restricting CPU utilization to 2 cores or memory utilization to 4 Gb)have been applied. These policies effectively isolate the sensitive pods resulting in enhanced security, refer to img 6 and 7.



```
▸ Run kubectl apply -f deployment.yaml
deployment.apps/k8ssecframework-deployment configured
service/k8ssecframework-service unchanged
```

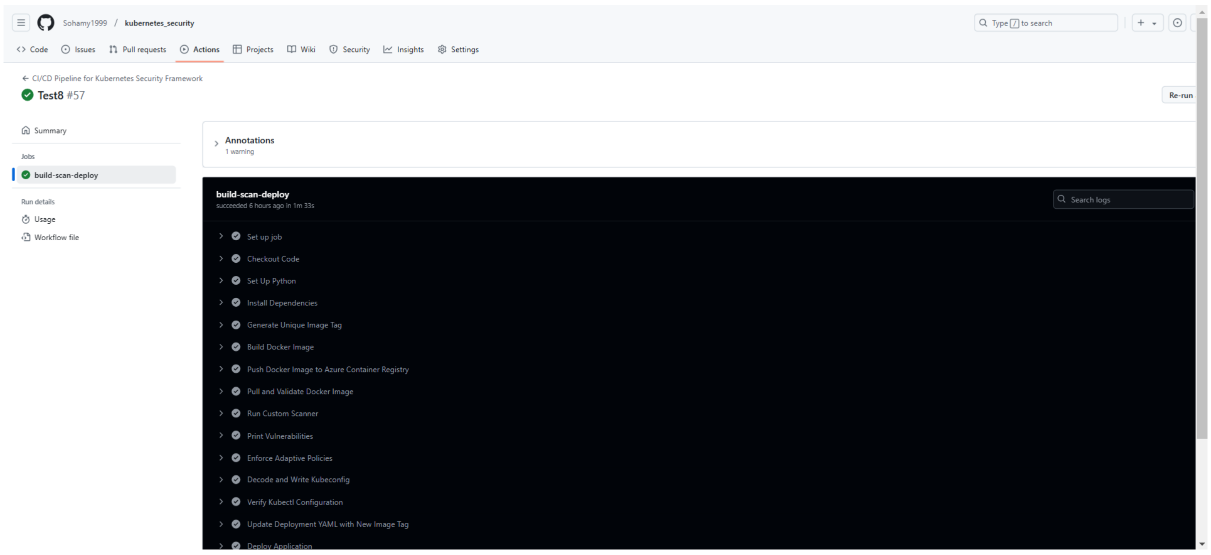
```
▸ Run kubectl set image deployment/k8ssecframework-deployment k8ssecframework=***.azurecr.io/kubernetes-security-framework:20241209222713-e09e307f1b7f74f68054ca
```

```
▸ Run echo "Verifying applied policies..."
Verifying applied policies...
Network Policies:
NAME                        POD-SELECTOR        AGE
block-external-traffic-n-a  vulnerability=n-a   106s
Resource Quotas:
NAME                         AGE    REQUEST                          LIMIT
restrict-resource-usage-n-a  105s   cpu: 750m/2, memory: 768Mi/4Gi
Namespace Labels:
NAME      STATUS   AGE   LABELS
default   Active   12d   kubernetes.io/metadata.name=default,pod-security.kubernetes.io/enforce=restricted
Pod Labels:
NAME                                        READY   STATUS              RESTARTS   AGE     LABELS
k8ssecframework-deployment-5fc994c598-wkz5g 0/1     ContainerCreating   0          2s      app=k8ssecframework,pod-template-hash=5fc994c598
k8ssecframework-deployment-897ff68cf-pp6th  1/1     Running             0          5h56m   app=k8ssecframework,pod-template-hash=897ff68cf,vulnerability=n-a
k8ssecframework-deployment-897ff68cf-q62bc  1/1     Running             0          5h55m   app=k8ssecframework,pod-template-hash=897ff68cf,vulnerability=n-a
```

Figure 7: CI CD Pipeline Build Phase

5.3 Case Study 3 - CI CD Pipeline

The CI CD Pipeline completely automates the CI/CD pipeline along with the steps or stages or jobs executed during the pipeline process. The key stages such as “Docker image creation” where the pipeline builds and tags docker images. “Run Security Scan”, is where the “Custom Security Scanning Agent” gets executed. “Enforce Adaptive Policies”, is where the “Adaptive Policy Enforcement” gets executed and then there is the deployment stage that deploys the application to the Kubernetes cluster. The time taken for the execution of the complete pipeline is approximately 2 minutes and 30 seconds. Azure is completely integrated as part of automation where as Google Cloud services are partial part of this research. The logs provide a detailed understanding of the CI CD stages and their successful implementation, refer to Img 8.



← CI/CD Pipeline for Kubernetes Security Framework
Test8 #57

build-scan-deploy

Annotations
1 warning

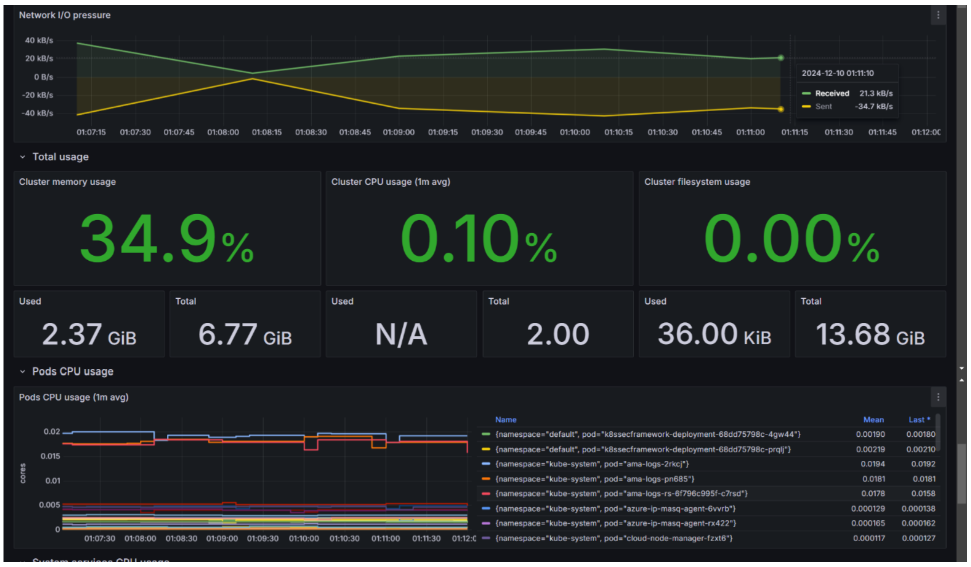
build-scan-deploy
succeeded 6 hours ago in 1m 33s

```
Set up job
Checkout Code
Set Up Python
Install Dependencies
Generate Unique Image Tag
Build Docker Image
Push Docker Image to Azure Container Registry
Pull and Validate Docker Image
Run Custom Scanner
Print Vulnerabilities
Enforce Adaptive Policies
Decode and Write Kubeconfig
Verify Kubectl Configuration
Update Deployment YAML with New Image Tag
Deploy Application
```

Figure 8: CI CD Pipeline

5.4 Case Study 4 - Monitoring

Prometheus and Grafana monitoring tools are integrated with Kubernetes to monitor the health of the overall cluster. A few important observations provide insights such as Cluster Memory Utilization which is marked 34.9 percent of total memory (i.e. 2.37 GB out of 6.77 GB), in addition to this the Cluster CPU usage is 0.10 percent of the 2 available cores. And, Pods Network I/O traffic range is monitored to be 0 to 50 KBs, refer to Img 9 and refer to Table 1.



Name	Mean	Last *
{namespace="default", pod="k8ssecframework-deployment-68dd75798c-4gw44"}	0.00190	0.00180
{namespace="default", pod="k8ssecframework-deployment-68dd75798c-prqlj"}	0.00219	0.00210
{namespace="kube-system", pod="ama-logs-2rkcj"}	0.0194	0.0192
{namespace="kube-system", pod="ama-logs-pn685"}	0.0181	0.0181
{namespace="kube-system", pod="ama-logs-rs-6f796c995f-c7rsd"}	0.0178	0.0158
{namespace="kube-system", pod="azure-ip-masq-agent-6vvrb"}	0.000129	0.000138
{namespace="kube-system", pod="azure-ip-masq-agent-rx422"}	0.000165	0.000162
{namespace="kube-system", pod="cloud-node-manager-fzxt6"}	0.000117	0.000127

Figure 9: Monitoring using Grafana and Prometheus

Metric	Description	Values
Cluster Memory Usage	Percentage of memory used by the cluster	34.9% (2.37 GiB out of 6.77 GiB)
Cluster CPU Usage	CPU usage in cores.	0.10% (2 cores)
Cluster Filesystem Usage	Percentage of filesystem space used	0.00% (36 KiB out of 13.68 GiB)
Pods CPU Usage	CPU utilization per pod	max ∼0.02 cores
System Services CPU Usage	CPU Utilization for key system services	Max ∼0.015 cores
Pods Memory Usage	Memory usage by individual pods in the cluster.	Max ∼256 MiB per pod
System Services Memory Usage	Memory usage for system services operating in the cluster.	Max ∼236 MiB
Pods Network I/O	Input/Output network traffic	Range: ∼0–50 KiB/s
Containers CPU Usage	CPU utilization per container	Max ∼0.015 cores
Containers Network I/O	Input/Output network traffic per container	Max ∼40 KiB/s
All Processes CPU Usage	CPU utilization across all processes	Max ∼0.02 cores
All Processes Memory Usage	Memory consumption across all processes	Max ∼1 GiB
All Processes Network I/O	Combined network traffic for all processes	Max ∼60 KiB/s

Table 1: Important Monitoring Matrix

5.5 Discussion

The research has aligned with its objective of creating a sustainable, effective, and completely automated Kubernetes security framework. However, there are some key points that were identified during this research.

- Strengths: The automation process through the CI CD pipeline has positively affected the execution time of security scanning and policy enforcement. The monitoring provides every detail related to the cluster, and pods which can be beneficial in scaling the framework and managing more complex workflows. In addition to this, the scaling can also be done from an adaptability point of view where a wide range of policies can be enforced with required customization.

- Challenges: Integrating multi-cloud in single automated work turned out to be a difficult task, considering the cloud Kubernetes registry will have to be toggled dynamically followed by logging into that specific cloud service while deployment turned out to be a tricky part. In addition to this, the dependency on the external API for vulnerability details might turn out a bit tedious when the response received from the API is not perfect, it provides more accurate respons at night when traffic over the APIs is low.

- Future improvements: The incorporation of multi-cloud functionality with the automation workflow will provide a complete customization ability as per the requirement. Another study would be to find an alternative to the external APIs or enhancing the APIs to reduce any kind of dependency for the framework.

6 Conclusion and Future Work

This research is an attempt to address the security challenges and complications in the Kubernetes environment by developing a completely automated framework that is hosted by the CI CD pipeline and provides functionalities such as real-time vulnerability scanning and adaptive policy enforcement. The framework is developed with the intention of overcoming the limitations of existing tools, such as high false-positive rates and limitations in customization for better adaptability. The two crucial components were created in this research "Custom security scanning agent" that integrates with the National Vulnerability Data (NVD) database and "Adaptive Policy Enforcement" for a multi-cloud Kubernetes structure.

The results obtained have demonstrated the abilities of the proposed framework such as real-time vulnerability analysis, automated end-to-end flow, and the ability to customize based on requirements. It effectively answered the research question "How Kubernetes clusters can be secured through automated solutions?" by providing a completely automated framework with a dynamic nature. However, in addition to these merits, there are a few limitations for this framework, those are dependency on external (NVD) API for real-time vulnerability data and implementing policies in large-scale deployments.

Future scope for this research could aim to extend the capabilities of this framework to create an advanced threat detection framework. The possible areas to explore could be the integration of behavioral anomaly analysis, and can also work towards commercialization of the framework as a security platform for Kubernetes that provides end-to-end automated security to Kubernetes. This research could be considered a process of creating an ecosystem of self-healing Kubernetes clusters.

References

Blomqvist, M., Koivunen, L. and Mäkilä, T. (2021). Secrets management in a multi-cloud kubernetes environment.

Boroufar, A. (2020). *Software Delivery in Multi-Cloud Architecture*, PhD thesis, Politecnico di Torino.

Bose, D. B., Rahman, A. and Shamim, S. I. (2021). 'under-reported'security defects in kubernetes manifests, *2021 IEEE/ACM 2nd International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS)*, IEEE, pp. 9–12.

Budigiri, G., Baumann, C., Mühlberg, J. T., Truyen, E. and Joosen, W. (2021). Network policies in kubernetes: Performance evaluation and security analysis, *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, IEEE, pp. 407–412.

Ganne, A. (2022). Cloud data security methods: Kubernetes vs docker swarm, *International Research Journal of Modernization in Engineering Technology* **4**(11): 1–6.

Giandonato, F. (2021). *Enabling Service Mesh in a Multi-Cloud Environment*, PhD thesis, Master's Thesis. Politecnico di Torino.

Giangiulio, F. and Malmberg, S. (2022). Testing the security of a kubernetes cluster in a production environment.

Kamieniarz, K. and Mazurczyk, W. (2024). A comparative study on the security of kubernetes deployments, *2024 International Wireless Communications and Mobile Computing (IWCMC)*, IEEE, pp. 0718–0723.

Kermabon-Bobinnec, H., Gholipourchoubeh, M., Bagheri, S., Majumdar, S., Jarraya, Y., Pourzandi, M. and Wang, L. (2022). Prospec: Proactive security policy enforcement for containers, *Proceedings of the Twelfth ACM Conference on Data and Application Security and Privacy*, pp. 155–166.

Lee, S. and Nam, J. (2023). Kunerva: Automated network policy discovery framework for containers, *IEEE Access* .

Mustyala, A. and Tatineni, S. (2021). Advanced security mechanisms in kubernetes: Isolation and access control strategies, *ESP Journal of Engineering & Technology Advancements (ESP JETA)* **1**(2): 57–68.

Oluyede, M. S., Mart, J., Olusola, A. and Olatuja, G. (2024). Container security in cloud environments, *ScienceOpen Preprints* .

Osmani, L., Kauppinen, T., Komu, M. and Tarkoma, S. (2021). Multi-cloud connectivity for kubernetes in 5g networks, *IEEE Communications Magazine* **59**(10): 42–47.

Raghunathan, S. (n.d.). Strengthening kubernetes: Strategies and tools for enhanced devsecops integration.

Rahman, A., Shamim, S. I., Bose, D. B. and Pandita, R. (2023). Security misconfigurations in open source kubernetes manifests: An empirical study, *ACM Transactions on Software Engineering and Methodology* **32**(4): 1–36.

Voievodin, Y. and Rozlomii, I. (2024). Application security optimization in container orchestration systems through strategic scheduler decisions.

Waseem, M., Ahmad, A., Liang, P., Akbar, M. A., Khan, A. A., Ahmad, I., Setälä, M. and Mikkonen, T. (2024). Containerization in multi-cloud environment: Roles, strategies, challenges, and solutions for effective implementation, *arXiv preprint arXiv:2403.12980* .

Wende, F. (2024). *Automated vulnerability scanning of Kubernetes during the CI/CD Process*, PhD thesis, University of Applied Sciences Technikum Wien.