Research on security as code approach for cloud-native applications based on Kubernetes clusters

Oleksandr Vakhula[1,†] and Ivan Opirskyy[1,*,†]

[1] Lviv Polytechnic National University, 12 Stepan Bandera str., 79000 Lviv, Ukraine

Abstract

The fast evolution of cloud-native applications and the widespread adoption of Kubernetes clusters have revolutionized how modern software is developed, deployed, and managed. However, this paradigm shift has introduced new security challenges that require innovative solutions. This research explores the "Security as Code" (SaC) approach, which integrates security policies and practices into the development and deployment pipelines of cloud-native applications on Kubernetes clusters. The study begins by outlining the theoretical foundations of the SaC approach, emphasizing the need for automated and consistent security measures across all stages of the software development lifecycle. We then explore the implementation of the policy engine and its gatekeeper component, as core tools for enforcing security policies within Kubernetes environments. The research details the setup process on AWS using a cost-effective configuration, augmented with GitOps tool for continuous deployment and container image vulnerability scanner. Our methodology includes configuring OPA Gatekeeper for admission control, defining and applying constraint templates, and integrating FluxCD to automate policy deployment and enforcement. We provide a step-by-step guide for setting up the environment, ensuring that the approach is practical and reproducible. The findings demonstrate that the SaC approach significantly improves security management in cloud-native environments, offering a scalable and flexible framework for integrating security into DevOps workflows. This research contributes to the broader understanding of how security can be codified and automated, paving the way for more secure and resilient cloud-native applications.

Keywords

security-as-code, cloud-native applications, Kubernetes clusters, open policy agent, gatekeeper, GitOps, continuous deployment, container security, DevOps, policy-as-code, service mesh, shift-left security

1. Introduction

The introduction of cloud-native applications has marked a significant shift in the landscape of software development and deployment. These applications, designed to leverage the advantages of cloud computing, offer unparalleled scalability, flexibility, and resilience. At the heart of this transformation is Kubernetes, an open-source container orchestration platform that has become the de facto standard for deploying, scaling, and managing containerized applications.

While Kubernetes simplifies many aspects of application management, it also introduces new security challenges. Traditional security practices often struggle to keep pace with the dynamic and ephemeral nature of cloud-native environments. This gap has led to the emergence of the "Security as Code" (SaC) paradigm, which aims to embed security directly into the development and operational processes through code.

Security as Code involves defining security policies and controls as code, allowing them to be versioned, reviewed, and deployed alongside application code. This approach ensures that security measures are consistently applied and automatically enforced across all environments, from development to production. By integrating security into the DevOps pipeline, organizations can achieve continuous security and compliance, reducing the risk of vulnerabilities and misconfigurations.

This research focuses on implementing the SaC approach within Kubernetes clusters, leveraging Open Policy Agent (OPA) and its Gatekeeper component. OPA is a general-purpose policy engine that enables the enforcement of fine-grained, context-aware policies. Gatekeeper extends OPA's capabilities by integrating with Kubernetes admission controllers, allowing policies to be enforced at the time of resource creation and modification.

Additionally, this study incorporates FluxCD, a continuous delivery tool for Kubernetes, and Trivy, a comprehensive vulnerability scanner for container images. By combining these tools, we aim to create a robust framework for automating security policy enforcement and continuous monitoring of container vulnerabilities.

The goal of this research is to demonstrate the practical implementation and effectiveness of the Security as Code approach in Kubernetes environments. Specifically, we aim to integrate and automate security policies, ensuring

CSDP-2024: Cyber Security and Data Protection, June 30, 2024, Lviv, Ukraine

*Corresponding author.
†These authors contributed equally.
oleksandr.p.vakhula@lpnu.ua (O. Vakhula); ivan.r.opirskyi@lpnu.ua (I. Opirskyy)
0009-0008-5367-3344 (O. Vakhula); 0000-0002-8461-8996 (I. Opirskyy)

continuous security and compliance throughout the development lifecycle of cloud-native applications.

- Define Security Policies: Create and codify security policies that can be enforced within Kubernetes clusters using OPA Gatekeeper.
- Automate Policy Enforcement: Integrate security policies into the CI/CD pipeline to ensure automated and consistent enforcement.
- Implement Continuous Deployment: Use FluxCD to manage and automate the deployment of Kubernetes manifests, ensuring the desired state of the cluster is maintained.
- Vulnerability Scanning: Incorporate Trivy to scan container images for vulnerabilities, adding a layer of security.
- Cost-Effective Setup: Establish a cost-effective environment for testing and deploying the SaC solutions, particularly in cloud environments like AWS.

The swift adoption of Kubernetes as a container orchestration platform has revolutionized the deployment and management of cloud-native applications. However, this shift has also introduced significant security challenges that traditional security approaches are ill-equipped to address. The need for dynamic, automated, and scalable security measures has become crucial.

The primary problem this research addresses is the implementation of a "Security as Code" (SaC) approach in Kubernetes-based cloud environments. Specifically, the challenges include вynamic and ephemeral nature of containers, containers are inherently ephemeral and dynamic; complexity of Kubernetes, Kubernetes, while powerful, introduces significant complexity with its numerous components (e.g., API server, etcd, scheduler, controllers); multi-tenancy and Isolation; integration with existing security tools; continuous security and compliance; visibility and monitoring.

Through this research, we aim to explore and address these challenges by providing a detailed implementation guide, evaluating the effectiveness of automated security policies, and offering practical insights into integrating security as code into Kubernetes-based workflows. By doing so, we contribute to the broader discourse on enhancing the security of cloud-native applications through innovative code-centric approaches.

2. Related works

The field of container orchestration and security has evolved significantly over the past decade, with numerous contributions from both academia and industry. This section reviews some of the seminal works and current research related to the "Security as Code" approach in Kubernetes environments.

The article "Borg, Omega, and Kubernetes" by Burns et al. from Google Inc. provides an in-depth look at the evolution of container management systems within Google, starting with Borg, moving to Omega, and finally to Kubernetes. This progression highlights the increasing sophistication and scalability of container orchestration, emphasizing key innovations such as Borg, the initial system developed to manage both long-running services and batch jobs. Borg introduced resource sharing between different types of applications, significantly improving resource utilization. Omega, built to improve the software engineering of Borg, introduced a more consistent and principled architecture, using a centralized Paxos-based transaction-oriented store. Kubernetes was designed for a broader developer audience, emphasizing ease of use for deploying and managing distributed systems, leveraging a shared persistent store accessed through a REST API. Kubernetes' architecture is designed to support scalability and flexibility while enforcing consistent security policies. This is achieved through a centralized API server that ensures all state changes are validated, defaulted, and versioned, providing a robust foundation for enforcing policies and maintaining system invariants. Reconciliation controllers improve resiliency by continuously aligning the desired and observed states, a concept shared with Borg and Omega [1].

"Kubernetes Security: Securing Microservices and Applications in the Cloud" by O'Reilly Media is a comprehensive guide that explores various security challenges and best practices for securing Kubernetes clusters. It covers topics such as securing the Kubernetes API server, controlling access with RBAC, network security policies, and monitoring and auditing Kubernetes clusters. The book emphasizes the importance of integrating security throughout the development lifecycle and provides practical examples of implementing security measures [2].

Practical guides and best practices for Kubernetes security, such as those by AquaSec and Red Hat, provide comprehensive overviews of necessary security measures. These include network policies to control traffic between pods, secret management to securely manage sensitive information, and admission controllers to enforce security policies at the point of deployment, ensuring that only compliant configurations and container images are deployed [3–4].

The concept of Policy as Code, particularly with tools like Open Policy Agent (OPA), has gained traction for automating and enforcing security policies within Kubernetes. Works such as those by O'Reilly and industry blogs discuss implementing OPA for dynamic policy enforcement, highlighting its flexibility and power in managing complex security policies programmatically [5].

The article "XI Commandments of Kubernetes Security: A Systematization of Knowledge Related to Kubernetes Security Practices" provides a systematic approach to securing Kubernetes environments by identifying eleven critical security practices. These practices, derived from a comprehensive analysis of internet artifacts, include Role-Based Access Control (RBAC), network policies, and regular security patching. The systematic approach offers a structured framework for practitioners to enhance the security posture of their Kubernetes deployments [6].

The research article "Comprehensive Approach for Developing an Enterprise Cloud Infrastructure" by Khoma et al. emphasizes the need for a multilevel security approach in cloud environments. The article outlines the limitations of existing "Security as Code" practices and proposes a

comprehensive framework to enhance cloud infrastructure security. Key aspects include effective access and privilege management, logical isolation of network resources, continuous monitoring, and automated response to anomalies. This comprehensive approach aligns well with the principles of "Security as Code" by integrating security into every layer of the cloud infrastructure, thus providing a robust foundation for secure cloud-native applications [7].

The paper "Cloud Container Technologies: A State-of-the-Art Review" by Pahl et al. provides a systematic mapping study of container technologies and their orchestration, particularly in cloud environments. The study identifies and classifies 46 selected studies on container technologies, highlighting the key concerns and trends in the field. It reveals that container technologies positively impact both development and deployment aspects, supporting continuous development and deployment pipelines. However, the study also notes the lack of tool support to automate and facilitate container management and orchestration, particularly in clustered cloud architectures. The findings underscore the need for advanced orchestration support and the importance of container-based orchestration techniques in balancing optimized resource utilization and performance in the cloud [8].

The paper "Expanding DevSecOps Practices and Clarifying the Concepts within Kubernetes Ecosystem" by Alawneh and Abbadi discusses the integration of DevSecOps principles within Kubernetes environments. The authors highlight the importance of incorporating security by design within organizational processes, including development, deployment, and operational management. The paper outlines several real-life examples that illustrate the integration of security into each practice, emphasizing how DevSecOps practices can enhance application delivery, resilience, elasticity, availability, and reliability. The paper also addresses the challenges of establishing robust mechanisms for integrating security within existing DevOps practices and provides insights into the roles of DevSecOps practices in securing the Kubernetes ecosystem. This work aligns with the Security as Code approach by demonstrating how security can be seamlessly integrated into the Kubernetes lifecycle, thereby enhancing the overall security posture of cloud-native applications [9].

The literature review highlights the importance of adopting a security-as-code approach in modern cloud-native environments. By automating and codifying security policies, organizations can achieve continuous security, maintain compliance, and build more resilient systems. The articles and studies reviewed provide valuable insights, practical examples, and best practices that can guide practitioners in enhancing the security of their Kubernetes deployments. These resources emphasize the critical role of policy automation, continuous monitoring, and integration of security into DevOps practices in building secure and compliant cloud-native applications.

The research and resources analyzed above underscore the critical importance of integrating security into the development and operational processes of cloud-native applications. By adopting a Security as Code approach and leveraging tools like OPA Gatekeeper, FluxCD, and Trivy, organizations can achieve continuous security and compliance, ensuring that their Kubernetes environments remain secure and resilient. These resources provide valuable insights, practical examples, and best practices that can guide practitioners in enhancing the security of their cloud-native applications.

The field of container orchestration and security has evolved significantly over the past decade, with numerous contributions from both academia and industry. This section reviews some of the seminal works and current research related to the "Security as Code" approach in Kubernetes environments.

3. General overview of container cluster and Kubernetes orchestration

Container clusters and orchestration are fundamental to modern application deployment and management. Containers encapsulate an application and its dependencies, providing a consistent environment across development, testing, and production. Orchestration is crucial for managing these containers at scale, ensuring efficient resource utilization, high availability, and automated workflows. Kubernetes has emerged as the leading orchestration platform, offering robust tools for deploying, scaling, and operating containerized applications across clusters of machines.

Kubernetes clusters are highly versatile and can be utilized to manage and orchestrate a range of innovative technologies. In blockchain [10], Kubernetes supports platforms like Hyperledger Fabric and Ethereum, ensuring scalable and resilient node deployment. For machine learning and AI, tools like TensorFlow Serving and Kubeflow benefit from Kubernetes' scalability and automated management. In big data, Kubernetes efficiently manages Apache Spark and Elasticsearch clusters. Kubernetes is also pivotal in IoT with edge computing solutions like KubeEdge and in CI/CD with Jenkins X and Argo CD. Additionally, it supports microservices and serverless architectures through Istio and Knative, and manages databases such as Cassandra and PostgreSQL, making it an essential tool for modern, cloud-native applications.

Core Concepts

Containers are lightweight, portable, and consistent units of software that include everything needed to run an application. Unlike traditional virtual machines, containers share the host system's kernel but operate in isolated user spaces. A container cluster is a group of interconnected nodes that work together to provide a scalable and resilient environment for running containerized applications. This clustering allows for efficient resource sharing, load balancing, and fault tolerance, making it possible to manage thousands of containers seamlessly.

Kubernetes Architecture (Fig. 1):

Kubernetes architecture consists of a master node and multiple worker nodes. The master node controls the cluster, housing components such as the API server, etcd (a key-value store for cluster data), the scheduler, and the controller manager. The API server serves as the main interface for interaction with the cluster. Etcd stores all

cluster configuration data persistently. The scheduler assigns workloads to nodes based on resource availability, and the controller manager handles routine tasks like replication and state management. Worker nodes run the actual applications in containers, managed within pods, which are the smallest deployable units in Kubernetes [11–12].

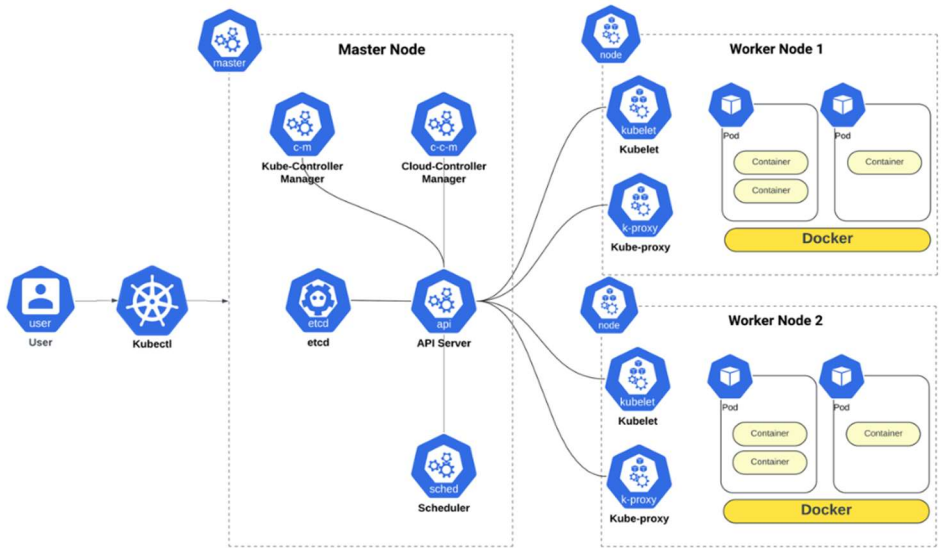

Figure 1: Kubernetes architecture

Operational Features:

Kubernetes provides several critical operational features to enhance the management and resilience of containerized applications. Self-healing capabilities automatically replace failed containers, ensuring continuous availability. Load balancing distributes network traffic evenly across all running containers, optimizing resource usage and performance. Automated rollouts and rollbacks allow for seamless updates and rollbacks of applications without downtime, ensuring that deployments are both reliable and consistent. These features collectively contribute to the robustness and efficiency of Kubernetes as an orchestration platform.

Declarative Configuration:

Kubernetes employs a declarative approach to configuration management, where the desired state of the system is defined in configuration files using YAML or JSON. Users specify what the end state should be, and Kubernetes takes responsibility for achieving and maintaining that state. This approach simplifies management, as Kubernetes continuously monitors the current state and makes necessary adjustments to align it with the desired state. It ensures consistency, and repeatability, and reduces the complexity of managing configurations manually.

In summary, Kubernetes has revolutionized the orchestration of container clusters, providing robust tools for managing containerized applications at scale. By leveraging a sophisticated architecture, operational features, and a declarative configuration approach, Kubernetes ensures efficient, reliable, and consistent application deployment and management. [13] This overview sets the stage for understanding the complexities and benefits of Kubernetes orchestration.

In the next chapter, we will dive into the challenges associated with implementing security in Kubernetes environments, exploring the issues that must be addressed to maintain robust security postures.

4. Problem statement: Challenges in implementing security for containerized services in cloud environments

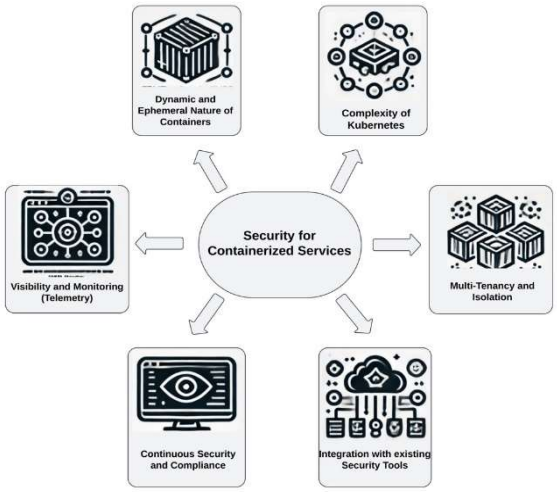

Figure 2: Main problems of ensuring a needed level of security for containerized services

Let's dive deeper into each of them:

1. Dynamic and Ephemeral Nature of Containers:

Containers are inherently ephemeral and dynamic, often created and destroyed within seconds. This transient nature makes it difficult to maintain consistent security policies and apply traditional security measures. Ensuring that security policies are consistently applied to every instance of a container can be challenging, leading to potential security gaps.

2. Complexity of Kubernetes:

Kubernetes, while powerful, introduces significant complexity with its numerous components (e.g., API server, etcd, scheduler, controllers). Securing each component and ensuring secure communication between them requires a deep understanding of the Kubernetes architecture. Misconfigurations and overlooked security settings can lead to vulnerabilities, making the cluster susceptible to attacks.

3. Multi-Tenancy and Isolation:

In a multi-tenant environment, ensuring proper isolation between different tenants' workloads is crucial to prevent unauthorized access and data leakage. Achieving strong multi-tenancy security requires robust network policies, resource quotas, and effective namespace management, which can be complex to implement and manage.

4. Integration with existing Security Tools:

Integrating Kubernetes with existing security tools and processes can be difficult due to differences in how these tools are designed to operate. Organizations may struggle to leverage their existing security investments, leading to potential gaps or redundant efforts in securing Kubernetes environments [14].

5. Continuous Security and Compliance:

Maintaining continuous security and compliance in a fast-paced, CI/CD-driven development environment is challenging. Automated pipelines need to incorporate security checks without hindering development velocity. Ensuring that security checks are seamlessly integrated into the CI/CD pipeline is essential to catch vulnerabilities early and maintain compliance without slowing down development [15].

6. Visibility and Monitoring (Telemetry):

Achieving comprehensive visibility and monitoring of containerized applications across a distributed cloud environment is challenging. Traditional monitoring tools may not provide the granularity needed for container environments. Lack of visibility can hinder the detection and response to security incidents, making it difficult to enforce security policies effectively.

A critical issue highlighting these challenges is the fact that only 0.79% of Kubernetes commits are security-related, suggesting that security-related defects are under-reported and could lead to large-scale security breaches. This statistic underscores the need for a more proactive and integrated approach to security within the Kubernetes ecosystem. By addressing these problems, the implementation of a robust "Security as Code" framework can ensure that Kubernetes environments are secure, compliant, and resilient, protecting them against the evolving threat landscape [16].

Container security is a major concern for companies, with four generalized use cases and solutions relying on software-based and hardware-based solutions. Containers emerged as a lightweight alternative to virtual machines that offer better microservice architecture support. The value of the container market is expected to reach $2.7 billion in 2020 compared to $762 million in 2016. Although they are considered the standardized method for microservices deployment, playing an important role in cloud computing emerging fields such as service meshes, market surveys show that container security is the main concern and adoption barrier for many companies. The literature on container security identifies four generalized use cases that cover security requirements within the host-container threat landscape:

1. Protecting a container from applications inside it.
2. Inter-container protection.
3. Protecting the host from containers.
4. Protecting containers from a malicious or semi-honest host.

The first three use cases utilize software-based solutions that mainly rely on Linux kernel features and Linux security modules, the last use case relies on hardware-based solutions such as trusted platform modules [17].

The swift adoption of Kubernetes as a container orchestration platform has revolutionized the deployment and management of cloud-native applications. However, this shift has also introduced significant security challenges that traditional security approaches are ill-equipped to address. The need for dynamic, automated, and scalable security measures has become crucial.

The primary problem this research addresses is the implementation of a "Security as Code" (SaC) approach in Kubernetes-based cloud environments.

5. Review security as a code concept

The SaC paradigm aims to embed security policies and practices into the development and deployment pipelines, ensuring consistent and automated enforcement across all stages of the application lifecycle. The "Security as Code" approach in cloud environments involves embedding security measures directly into the software development and deployment process. This method enables the automation of various security tasks, enhancing consistency and effectiveness. It is especially crucial in cloud environments, where rapid and flexible responses to changes and emerging security challenges are required. "Security as Code" helps in the early identification of potential vulnerabilities and ensures compliance with regulatory and security standards [18]. Despite its potential, the practical implementation of SaC in Kubernetes environments faces several challenges, which we should take into account:

1. Defining and Enforcing Security Policies:

How can organizations define and enforce security policies in a dynamic and scalable manner that aligns with the ephemeral nature of containers?

2. Automation and Integration:

How can security policies be automated and integrated into existing CI/CD pipelines to ensure continuous security without hindering development velocity?

3. Tooling and Best Practices:

What are the best practices and tools (e.g., OPA Gatekeeper, FluxCD, Trivy) for implementing SaC in

Kubernetes environments, and how can they be effectively configured and managed?

4. Cost-Effective Deployment:
How can organizations set up a cost-effective environment for testing and deploying SaC solutions, particularly in cloud environments like AWS?

This research aims to explore and address these challenges by providing a detailed implementation guide, evaluating the effectiveness of automated security policies, and offering practical insights into integrating security as code into Kubernetes-based workflows. By doing so, it contributes to the broader discourse on enhancing the security of cloud-native applications through innovative, code-centric approaches.

6. Overview of solution based on security as a code in the context of containerized cloud-native application

Open Policy Agent (OPA) is a general-purpose policy engine that enables unified, context-aware policy enforcement across the stack. OPA decouples policy decisions from the application logic, allowing administrators to manage policies centrally [19].

OPA uses a high-level declarative language called Rego to write policies. Rego allows users to define policies based on various data inputs, supporting complex logic and queries to determine policy compliance [20].

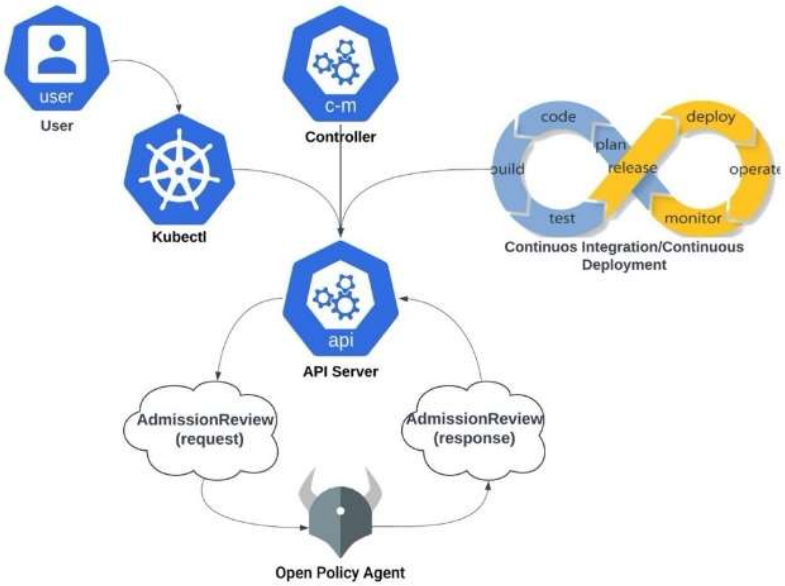

Figure 3: Interaction between Kubernetes components and admission control with the Open Policy Agent (OPA)

OPA can be integrated with a variety of systems, including Kubernetes, CI/CD pipelines, microservices, and more (Fig. 3). In Kubernetes, OPA can enforce policies on resources such as pods, deployments, and services. It can also integrate with CI/CD pipelines to ensure compliance during the build and deployment phases.

Gatekeeper is an admission controller for Kubernetes that uses OPA policies to enforce security and operational rules within the cluster. It provides a framework for policy enforcement and auditing, ensuring that all changes comply with predefined policies before being accepted by the Kubernetes API server.

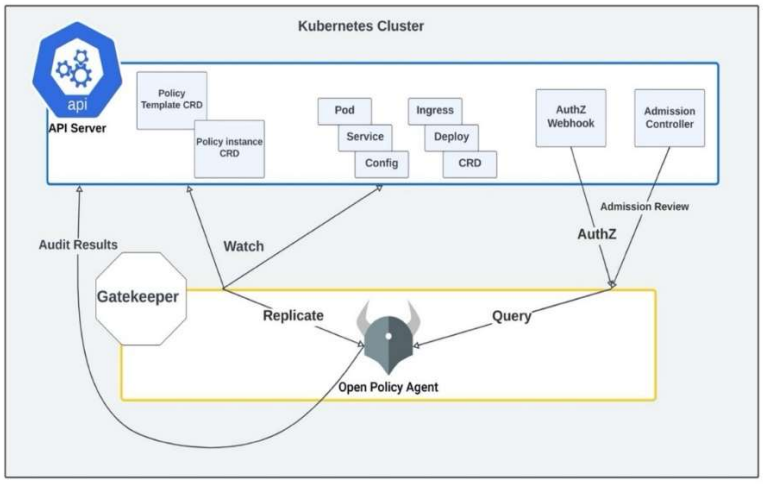

Figure 4: Integration of the Open Policy Agent (OPA) with Kubernetes through Gatekeeper

Fig. 4 shows the integration of the Open Policy Agent (OPA) with Kubernetes through Gatekeeper, detailing how policies are enforced and managed.

1. Policy Templates and Instances:
   – Policy Template Custom Resource Definitions (CRDs) define reusable policy templates.
   – Policy Instance CRDs apply specific policies using these templates.
2. Kubernetes API Server:
   – Handles resources like Pods, Services, and Configurations.
   – Uses Admission Controllers and AuthZ Webhooks for authorization and admission control.
3. OPA and Gatekeeper:
   – Gatekeeper replicates policies to OPA.
   – OPA evaluates AdmissionReview requests against these policies.
   – Results are audited and enforced through the API server, ensuring compliance with defined policies.

Policies in Gatekeeper are written using the Rego language and configured as ConstraintTemplates. These templates define the policy logic and the constraints that must be met. Examples of common policies include restricting certain container images, enforcing namespace-specific policies, and ensuring resource quotas.

Gatekeeper offers auditing and monitoring capabilities, providing visibility into policy violations and historical data for compliance audits. It helps identify non-compliant resources and offers detailed reports on policy enforcement across the cluster.

Adding FluxCD to the setup enables continuous deployment for the Kubernetes environment.

7. Practical implementation

The foundation of the architecture is an Amazon EC2 instance. This instance serves as the host for Minikube, which is used to create a local Kubernetes cluster. Minikube is installed and configured on the EC2 instance. It creates a local Kubernetes cluster within the EC2 environment, enabling Kubernetes functionalities in a contained setup.

The Kubernetes cluster orchestrated by Minikube consists of multiple nodes that manage containerized applications. These nodes handle the deployment, scaling, and operation of application containers. OPA (Open Policy Agent) Gatekeeper is deployed within the Kubernetes cluster. It acts as a policy enforcement tool, ensuring that all resources and configurations within the cluster comply with predefined policies. It intercepts admission requests and validates them against the policies before allowing them into the cluster. [21–22] Trivy is integrated into the Kubernetes environment to scan container images for vulnerabilities. It runs security scans on images either before they are deployed or continuously as part of the CI/CD pipeline. [23] Trivy helps in identifying and mitigating potential security risks in container images. FluxCD is installed in the Kubernetes cluster to manage Kubernetes manifests and automate deployments based on changes in a Git repository. FluxCD continuously monitors the repository for changes and applies them to the cluster, ensuring that the cluster state matches the declared state in the Git repository. This process is known as GitOps. The EC2 instance runs Minikube, which sets up the Kubernetes cluster. Within this cluster, OPA Gatekeeper, Trivy, and FluxCD are deployed as separate services. OPA Gatekeeper enforces security and compliance policies by validating resources during the admission process. Trivy scans the container images used within the cluster for vulnerabilities, ensuring that only secure images are deployed. FluxCD watches the Git repository for changes and updates the Kubernetes cluster configuration accordingly, automating the deployment process and maintaining the desired state.

This guide provides general steps for integrating FluxCD into a Kubernetes environment set up with Minikube on an EC2 instance, along with OPA Gatekeeper and Trivy for security scanning of container images. The steps include:

1. Set Up Minikube on EC2: Install and configure Minikube on an EC2 instance to create a local Kubernetes cluster.
2. Install OPA Gatekeeper: Deploy OPA Gatekeeper to enforce policies within the Kubernetes cluster.
3. Integrate Trivy: Set up Trivy to scan container images for vulnerabilities.
4. Deploy FluxCD: Install FluxCD to manage Kubernetes manifests and automate deployments based on changes in a Git repository.

Step 1: Launch an EC2 Instance

1. Create an EC2 Instance:
   - Open the AWS Management Console and navigate to the EC2 service.
   - Click on “Launch Instance.”
   - Choose an Amazon Machine Image (AMI), such as Amazon Linux 2 AMI (HVM).
   - Select an instance type, such as t3.small, for cost-effectiveness.
   - Configure the instance details, including network settings.
   - Add storage (default settings are typically sufficient).
   - Configure security groups to allow SSH (port 22) access.
   - Review and launch the instance.

2. Connect to the EC2 Instance:
   - Use an SSH client to connect to your EC2 instance.
   ```sh
   ssh -i /path/to/your-key-pair.pem ec2-user@<EC2_Instance_Public_IP>
   ```

Step 2: Install Minikube and Kubernetes Tools

1. Install Docker:
   - Update the package database and install Docker.
   ```sh
   sudo yum update -y
   sudo amazon-linux-extras install docker -y
   sudo service docker start
   sudo usermod -aG docker ec2-user
   ```

2. Install Minikube:
   - Download and install Minikube.
   ```sh
   curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
   chmod +x minikube
   sudo mv minikube /usr/local/bin/
   ```

3. Install kubectl:
   - Download and install kubectl.
   ```sh
   curl -LO "https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl"
   chmod +x kubectl
   sudo mv kubectl /usr/local/bin/
   ```

4. Start Minikube:
   - Start Minikube with a specific driver (e.g., Docker).
   ```sh
   minikube start --driver=docker
   ```

Step 3: Install OPA Gatekeeper

1. Deploy Gatekeeper:
   - Apply the Gatekeeper manifest to deploy OPA Gatekeeper in your Minikube cluster.
   ```sh
   kubectl apply -f https://raw.githubusercontent.com/open-policy-agent/gatekeeper/master/deploy/gatekeeper.yaml
   ```

2. Verify Installation:
   - Check the Gatekeeper pods to ensure they are running.
   ```sh
   kubectl get pods -n gatekeeper-system
   ```

Step 4: Install FluxCD

1. Install Flux CLI:
   - Download and install the Flux CLI.
   ```sh
   curl -s https://fluxcd.io/install.sh | sudo bash
   ```

2. Bootstrap FluxCD with GitHub:
   - Bootstrap FluxCD with your GitHub repository.
   ```sh
   flux bootstrap github \
     --owner=<your-github-username> \
     --repository=<your-repo-name> \
     --branch=main \
     --path=clusters/my-cluster \
     --personal
   ```

Step 5: Configure Admission Control Policies

1. Create a ConstraintTemplate:
   - Define a custom constraint template YAML file. For example, `k8srequiredlabels.yaml`:
   ```yaml
   apiVersion: templates.gatekeeper.sh/v1beta1
   kind: ConstraintTemplate
   metadata:
     name: k8srequiredlabels
   spec:
     crd:
       spec:
         names:
           kind: K8sRequiredLabels
           ...
     targets:
       - target: admission.k8s.gatekeeper.sh
         rego: |
           package k8srequiredlabels

           violation[{"msg": msg}] {
             ...
           }
   ```

2. Apply the ConstraintTemplate:
   ```sh
   kubectl apply -f k8srequiredlabels.yaml
   ```

3. Create a Constraint:
   - Define a constraint to enforce the policy, e.g., `requiredlabels.yaml`:
   ```yaml
   apiVersion: constraints.gatekeeper.sh/v1beta1
   kind: K8sRequiredLabels
   metadata:
     name: ns-must-have-gk
   spec:
     match:
       kinds:
         - apiGroups: [""]
           kinds: ["Namespace"]
     parameters:
       labels: ["gatekeeper"]
   ```

4. Apply the Constraint:
   ```sh
   kubectl apply -f requiredlabels.yaml
   ```

Step 6: Check Container Image Vulnerabilities

1. Install Trivy:
   - Trivy is a popular tool for scanning container images for vulnerabilities. Install it on your EC2 instance.
   ```sh
   wget https://github.com/aquasecurity/trivy/releases/download/v0.28.0/trivy_0.28.0_Linux-64bit.deb
   ```

sudo dpkg -i trivy_0.28.0_Linux-64bit.deb
```

2. Scan an Image:
- Use Trivy to scan an image, e.g., `nginx:latest`.
```sh
trivy image nginx:latest
```

3. Automate Scanning in CI/CD:
- Integrate Trivy into your CI/CD pipeline to automate image scanning. For example, add a step in your GitHub Actions workflow:
```yaml
name: Scan Docker image for vulnerabilities
on: [push]
jobs:
  scan:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v2
      - name: Set up Trivy
        run: |
          sudo apt-get install wget apt-transport-https gnupg lsb-release -y
          wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | sudo apt-key add -
          echo deb https://aquasecurity.github.io/trivy-repo/deb $(lsb_release -sc) main | sudo tee -a /etc/apt/sources.list.d/trivy.list
          sudo apt-get update
          sudo apt-get install trivy -y
      - name: Scan image
        run: trivy image nginx:latest
```

Through our study, we demonstrated the practical implementation of SaC using Open Policy Agent (OPA) and its Gatekeeper component, integrated with FluxCD for continuous deployment and Trivy for container image vulnerability scanning. The research provided a detailed, step-by-step guide for setting up a cost-effective testing environment on AWS using Minikube, making it accessible for practitioners and researchers alike [24–25].

Key findings

1. Automated and Consistent Security Policies:
By defining security policies as code, organizations can ensure consistent enforcement across all stages of the application lifecycle. OPA Gatekeeper enables fine-grained control over Kubernetes resources, preventing misconfigurations and unauthorized changes.

2. Seamless Integration with CI/CD Pipelines:
Integrating security checks into CI/CD pipelines using FluxCD ensures that security is continuously maintained without hindering development velocity. Automated deployments and policy enforcement reduce the risk of human error and accelerate the development process.

3. Effective Vulnerability Management:
Incorporating Trivy for container image vulnerability scanning provides an additional layer of security by identifying and mitigating potential vulnerabilities before they can be exploited. This proactive approach helps maintain a secure application environment.

4. Cost-Effective Setup:
Using Minikube on an EC2 instance as a testing environment offers a cost-effective alternative to managed Kubernetes services like EKS. This setup allows for comprehensive testing and validation of security policies in a controlled, affordable manner.

8. Comparison of approaches for Kubernetes cluster security

This table compares three approaches to security in Kubernetes environments: Security as Code with OPA Gatekeeper, Traditional Security Methods, and Admission Controllers. The comparison is made based on various criteria, highlighting the strengths and weaknesses of each approach based on the aforementioned research.

9. Implications in practice

While this research provides a robust framework for implementing SaC in Kubernetes environments, several areas warrant further investigation:

- Service Mesh Integration: Exploring the integration of service mesh solutions like Istio to enhance security, observability, and traffic management within Kubernetes clusters [26–27].

The topic of my next research will be centered around the benefits of fine-grained access control by integrating Open Policy Agent (OPA) with Istio. This integration allows for a more detailed and context-aware approach to access control within a Service Mesh architecture. Unlike traditional access control mechanisms, fine-grained access control enables policies that consider multiple attributes, including request context, user roles, resource types, and more. This level of granularity is crucial for implementing robust security measures, especially in complex microservices environments [28].

The research underscores the importance of adopting a security-as-code approach in modern cloud-native environments. By embedding security directly into the development and deployment processes, organizations can achieve a higher level of security automation, reduce the attack surface, and improve overall resilience. The findings provide practical insights and best practices that can be leveraged by DevOps teams, security engineers, and IT professionals to enhance the security of their Kubernetes deployments.
```

Table 1
SaC with OPA Gatekeeper vs. Traditional Security Methods vs. Admission Controllers in Kubernetes Clusters

Criteria	Security as Code with OPA Gatekeeper	Traditional Security Methods	Admission Controllers
Policy Management	Centralized and consistent policy management through code	Often decentralized, policies may be managed manually or with scripts	Centralized management, predefined rules
Automation	The high degree of automation in policy enforcement and compliance checks	Varies, and often requires manual intervention or custom automation scripts	Automated enforcement of predefined policies
Integration with CI/CD	Seamless integration with CI/CD pipelines for continuous security	Integration may require additional effort and custom tooling	Limited integration, primarily focused on runtime enforcement
Flexibility	Highly flexible, supports complex and fine-grained policies using Rego language	Limited flexibility, often constrained by predefined rules and configurations	Moderate flexibility, dependent on the admission controller's capabilities
Scalability	Scales well with Kubernetes clusters, suitable for large environments	Scalability varies and may face challenges in large or dynamic environments	Generally scales well, but can add latency
Visibility and Monitoring	Enhanced visibility and auditing capabilities for policy violations	Basic visibility often lacks detailed auditing features	Basic visibility, some support for logging violations
Complexity	Initial setup and policy definition can be complex	Generally simpler setup but may lack advanced features	Moderate complexity, setup and configuration can vary
Vendor Lock-in	Open-source and vendor-neutral	Varies, some methods may involve vendor-specific solutions	Generally open-source or native to Kubernetes
Community and Support	Strong community support, extensive documentation and tutorials	Varies, proprietary solutions may have limited community support	Strong community support for popular admission controllers
Real-time Enforcement	Enforces policies at admission control, ensuring compliance before deployment	Enforcement may be reactive, relying on post-deployment scans and checks	Real-time enforcement at resource creation and modification
Resource Overhead	Low to moderate, depending on the complexity of policies	Varies, can be high depending on the security tools used	Low to moderate, depending on the controller and policies enforced
Adaptability	Easily adaptable to new security requirements and evolving threats	Adaptability depends on the flexibility of the chosen security method	Moderate adaptability, predefined rules may need updates for new threats

Table 2
Pros and Cons of SaC with OPA Gatekeeper

Pros	Cons
Centralized and consistent policy management through code	Initial setup and policy definition can be complex
The high degree of automation in policy enforcement and compliance checks	Requires learning Rego language for policy writing
Seamless integration with CI/CD pipelines for continuous security	Potential performance impact on admission control with complex policies
Highly flexible, supports complex and fine-grained policies using Rego language	May require significant time investment for initial configuration and policy creation
Enhanced visibility and auditing capabilities for policy violations	Debugging policy violations can be challenging
Strong community support, extensive documentation, and tutorials	Need for ongoing maintenance to keep policies up to date

10. Conclusion

The introduction of cloud-native applications has marked a significant shift in the landscape of software development and deployment. These applications, designed to leverage the advantages of cloud computing, offer unparalleled scalability, flexibility, and resilience. At the heart of this transformation is Kubernetes, an open-source container orchestration platform that has become the de facto standard for deploying, scaling, and managing containerized applications.

While Kubernetes simplifies many aspects of application management, it also introduces new security challenges. Traditional security practices often struggle to keep pace with the dynamic and ephemeral nature of cloud-native environments. This gap has led to the emergence of the "Security as Code" (SaC) paradigm, which aims to embed security directly into the development and operational processes through code.

Security as Code involves defining security policies and controls as code, allowing them to be versioned, reviewed, and deployed alongside application code. This approach ensures that security measures are consistently applied and automatically enforced across all environments, from development to production. By integrating security into the DevOps pipeline, organizations can achieve continuous security and compliance, reducing the risk of vulnerabilities and misconfigurations.

This research focuses on implementing the SaC approach within Kubernetes clusters, leveraging Open Policy Agent (OPA) and its Gatekeeper component. OPA is a general-purpose policy engine that enables the enforcement of fine-grained, context-aware policies. Gatekeeper extends OPA's capabilities by integrating with Kubernetes admission controllers, allowing policies to be enforced at the time of resource creation and modification. Additionally, this study incorporates FluxCD, a continuous delivery tool for Kubernetes, and Trivy, a comprehensive vulnerability scanner for container images. By combining these tools, we aim to create a robust framework for automating security policy enforcement and continuous monitoring of container vulnerabilities.

The swift adoption of Kubernetes as a container orchestration platform has revolutionized the deployment and management of cloud-native applications. However, this shift has also introduced significant security challenges that traditional security approaches are ill-equipped to address. The need for dynamic, automated, and scalable security measures has become crucial. The primary problem this research addresses is the implementation of a "Security as Code" (SaC) approach in Kubernetes-based cloud environments.

Through this research, we aim to explore and address these challenges by providing a detailed implementation guide, evaluating the effectiveness of automated security policies, and offering practical insights into integrating security as code into Kubernetes-based workflows. By doing so, we contribute to the broader discourse on enhancing the security of cloud-native applications through innovative code-centric approaches.

The arrival of cloud-native applications and the widespread adoption of Kubernetes have ushered in a new era of software development and deployment. While these technologies offer significant benefits in terms of scalability, flexibility, and resilience, they also present unique security challenges that traditional security practices are often ill-equipped to handle. This research aimed to explore and implement a "Security as Code" (SaC) approach within Kubernetes environments to address these challenges effectively.

By leveraging OPA's powerful policy language, Rego, organizations can define precise access control policies that are dynamically enforced across the Kubernetes ecosystem. This not only enhances security by ensuring that only authorized requests are permitted but also allows for the rapid adaptation of policies in response to emerging threats or changes in compliance requirements. Additionally, the separation of policy logic from application code simplifies the development process, allowing developers to focus on business functionality while security teams manage access policies independently.

Furthermore, the ability to update policies without redeploying services ensures minimal disruption and continuous enforcement of up-to-date security measures. This capability is essential for maintaining a strong security posture in an ever-evolving threat landscape. Through this research, we have demonstrated the practical implementation and effectiveness of the Security as Code approach in Kubernetes environments, offering insights into best practices and potential challenges.

By integrating OPA Gatekeeper, FluxCD, and Trivy, we have established a comprehensive framework for automating security policy enforcement and continuous monitoring of container vulnerabilities. This integrated approach not only addresses the inherent challenges of dynamic and ephemeral container environments but also ensures continuous security and compliance throughout the development lifecycle of cloud-native applications. Our findings contribute to the advancement of secure Kubernetes deployments and provide a robust foundation for future research and practical implementations in the field of cloud-native security.

References

[1] B. Burns, et al., Borg, Omega, and Kubernetes, Queue 14(1) (2016) 70–93. doi: 10.1145/2898442.2898444.

[2] B. Creane, A. Gupta, Kubernetes Security and Observability, O'Reilly Media (2016).

[3] R. Osnat, Kubernetes Security Basics and 10 Essential Best Practices (2020). URL: https://www.aquasec.com/cloud-native-academy/kubernetes-in-production/kubernetes-security-best-practices-10-steps-to-securing-k8s/

[4] M. Isberner, 11 Kubernetes Admission Controller Best Practices for Security (2019). URL: https://www.redhat.com/en/blog/11-kubernetes-admission-controller-best-practices-for-security

[5] J. Ray, Policy as Code (2018). URL: https://www.oreilly.com/library/view/policy-as-code/9781098139179/ch04. html

[6] Md S. Shamim, F. A. Bhuiyan, A. Rahman, XI Commandments of Kubernetes Security: A Systematization of Knowledge Related to Kubernetes Security Practices, IEEE Secure Development (SecDev) (2020) 58–64. doi: 10.1109/SecDev45635.2020.00025.

[7] V. Khoma, et al., Comprehensive Approach for Developing an Enterprise Cloud Infrastructure, in: Cybersecurity Providing in Information and Telecommunication Systems, vol. 3654, (2024) 201–215.

[8] C. Pahl, et al., Cloud Container Technologies: A State-of-the-Art Review, IEEE Transactions on Cloud Computing 7 (2024) 677–692. doi: /10.1109/TCC.2017.2702586.

[9] M. Alawneh, I. Abbadi, Expanding DevSecOps Practices and Clarifying the Concepts within Kubernetes Ecosystem, Ninth International

Conference on Software Defined Systems (SDS) (2022) 1–7. doi: 10.1109/SDS57574.2022.1006 2874.
[10] S. Vasylyshyn, et al., A Model of Decoy System Based on Dynamic Attributes for Cybercrime Investigation, Eastern-European J. Enterp. Technol. 1 (9(121)) (2023) 6–20. doi: 10.15587/1729-4061.2023.273363.
[11] The Linux Foundation, Official Kubernetes documentation (2024). URL: https://kubernetes.io/docs/concepts/overview/components/#control-plane-components
[12] Veritis, An Advanced Approach for Deploying Containerized Applications in a Cloud Environment (2024). URL: https://www.veritis.com/solutions/devops/kubernetes/
[13] The Linux Foundation, Official Kubernetes documentation (2024). URL: https://kubernetes.io/docs/concepts/security/
[14] The Linux Foundation, Cloud Native Security and Kubernetes (2024). URL: https://kubernetes.io/docs/concepts/security/cloud-native-security/
[15] Y. Martseniuk, Automated Conformity Verification Concept for Cloud Security, in: Cybersecurity Providing in Information and Telecommunication Systems, vol. 3654 (2024) 25–37.
[16] D. Bose, A. Rahman, M. Shamim, 'Under-reported' Security Defects in Kubernetes Manifests, IEEE/ACM 2nd International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS) (2021) 9–12. doi: /10.1109/EnCyCriS52570.2021.00009.
[17] Sultan, S., Ahmad, I., & Dimitriou, T. "Container Security: Issues, Challenges, and the Road Ahead", (2019), IEEE Access, 7, 52976-52996. doi: 10.1109/ACCESS.2019.2911732.
[18] O. Vakhula, et al., Security-As-Code Concept for Fulfilling ISO/IEC 27001:2022 Requirements, in: Cybersecurity Providing in Information and Telecommunication Systems, vol. 3654 (2024) 59–72.
[19] O. Vakhula, I. Opirskyy, O. Mykhaylova, Research on Security Challenges in Cloud Environments and Solutions based on the "Security-As-Code" Approach, in: Cybersecurity Providing in Information and Telecommunication Systems, vol. 3550 (2023) 55–69.
[20] Guest Expert on GitGuardian blog, What is Policy-as-Code? An Introduction to Open Policy Agent (2020). URL: https://blog.gitguardian.com/what-is-policy-as-code-an-introduction-to-open-policy-agent/
[21] OPA Official documentation (2024). URL: https://www.openpolicyagent.org/docs/latest/kubernetes-tutorial/
[22] S. Ragonessi, Secure your Kubernetes environment with OPA and Gatekeeper (2023). URL: https://www.cncf.io/blog/ 2023/10/09/secure-your-kubernetes-environment-with-opa-and-gatekeeper/
[23] G. Olaoye, A. Luz, DevSecOps and Integrating Security into the Cloud Development Lifecycle (2024). URL: https://www.researchgate.net/publication/378233643_DevSecOps_and_integrating_security_into_the_cloud_development_lifecycle
[24] G. Sayfan, Policy as Code and the Open Policy Agent (2022). URL: https://blogs.cisco.com/developer/policyascode01
[25] SecureFlag blog, Securing Kubernetes: Using Gatekeeper to Enforce Effective Security Policies (2024). URL: https://blog.secureflag.com/2024/03/13/security-policy-enforcement-in-kubernetes/
[26] Taikun, A Beginner's Guide to Istio: A Service Mesh for Kubernetes (2024). URL: https://taikun.cloud/beginner-guide-to-istio-service-mesh-for-kubernetes/
[27] AquaSecurity, Service Mesh: Architec-ture, Concepts, and Top 4 Frameworks (2021). URL: https://www.aquasec.com/cloud-native-academy/container-security/service-mesh/
[28] R. Chandramouli, Z. Butcher, A Zero Trust Architecture Model for Access Control in Cloud-Native Applications in Multi-Location Environments, NIST (2023). doi: 10.6028/NIST.SP.800-207A.