

Enterprise Kubernetes Method Modification at S&P 500 Companies^{*}

Oleh Faizulin^{1,†} and Mariia Nazarkevych^{1,*,†}

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

Abstract

Kubernetes has become the backbone of modern enterprise application architectures, offering unmatched scalability and flexibility. However, operating Kubernetes securely at the scale of an S&P 500 company introduces unique challenges. This paper explores typical challenges, including, but not limited to, multi-cluster management, CI/CD pipeline security, and insider threats, while presenting best practices and tools to mitigate risks. It also discusses experimental insights, lessons learned from real-world S&P 500 deployments, and emerging trends like zero-trust architecture and AI-driven threat detection. By combining technical expertise and practical strategies, enterprises can confidently scale Kubernetes while maintaining robust security postures.

Keywords

Kubernetes security, enterprise Kubernetes, S&P 500, multi-cluster management, CI/CD security, container vulnerabilities, zero-trust architecture, AI-driven threat detection, compliance, DevOps security

1. Introduction

Kubernetes has transformed the way enterprises design, deploy, and manage applications, becoming a critical enabler for digital transformation in S&P 500 companies. Its ability to orchestrate containerized workloads with agility and scalability has driven adoption across industries, from finance to healthcare to e-commerce. However, this ubiquity comes with significant challenges, particularly in securing Kubernetes environments at scale.

Enterprises in the S&P 500 face unique pressures, including protecting sensitive data, ensuring compliance with stringent regulations, and maintaining business continuity against a backdrop of increasing cyber threats. Misconfigurations, supply chain vulnerabilities, and insider threats can expose clusters to significant risks, making robust security strategies indispensable.

This paper delves into the complexities of securing Kubernetes in enterprise settings, offering insights into challenges, best practices, experimental findings, and future trends. By addressing these issues proactively, organizations can harness the full potential of Kubernetes without compromising on security.

2. Kubernetes security challenges at scale

Operating Kubernetes on the enterprise scale, particularly within S&P 500 companies, presents a host of security challenges that require robust and innovative strategies to address. Below, we explore some of the most pressing issues and provide insights into their potential impact.

2.1. Multi-cluster management

Enterprises often operate multiple Kubernetes clusters across environments such as production, staging, and development. Managing these clusters at scale involves ensuring consistent security policies and configurations across diverse environments. Misconfigurations in one cluster can

^{*}CPITS 2025: Workshop on Cybersecurity Providing in Information and Telecommunication Systems, February 28, 2025, Kyiv, Ukraine

^{*}Corresponding author.

[†]These authors contributed equally.

✉ oleh.r.faizulin@lpnu.ua (O. Faizulin); mariia.a.nazarkevych@lpnu.ua (M. Nazarkevych)

ORCID 0000-0001-5781-0600 (O. Faizulin); 0000-0002-6528-9867 (M. Nazarkevych)



© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

potentially expose sensitive workloads or create gaps that attackers may exploit. Studies such as Haque et al. [1], highlight the risks of misconfigured orchestrator settings in multi-cluster setups.

2.2. CI/CD pipeline security

Kubernetes thrives in environments with automated CI/CD pipelines, but these pipelines can also be a significant attack vector. Compromised CI/CD workflows can lead to the injection of malicious code into container images or clusters. Tools like Trivy and Clair are crucial for image vulnerability scanning, but their effectiveness depends on enterprise-wide adoption and integration [2].

2.3. Container image vulnerabilities

Containers provide portability and efficiency, but they also introduce risks related to outdated or unverified images. Public container registries can inadvertently harbor images with known vulnerabilities. Enterprises need to implement strict policies for image scanning and verification, as described in Red Hat's [3] findings on container vulnerability management.

2.4. Misconfigurations

One of the most common challenges in Kubernetes security is misconfigurations, such as overly permissive Role-Based Access Control (RBAC) policies, lack of resource quotas, and unbounded service accounts. Studies like Islam Shamim et al. [4] identify misconfigurations as the leading cause of Kubernetes security breaches. Automated tools like Kyverno and OPA/Gatekeeper help enforce security policies to mitigate these risks.

2.5. Insider threats and supply chain attacks

Insider threats, whether intentional or accidental, pose a unique challenge. Employees with excessive privileges can inadvertently expose critical data or systems. Similarly, supply chain attacks, such as the injection of malicious dependencies, have become increasingly prevalent. Reports from SentinelOne [5] emphasize the importance of detecting anomalies and unauthorized access patterns as part of an overall defense strategy.

2.6. Compliance with regulatory standards

S&P 500 companies operate under strict regulatory environments, including SOC 2, GDPR, and HIPAA. Ensuring compliance across distributed Kubernetes environments requires continuous monitoring, audit trails, and adherence to security benchmarks provided by cloud providers like Google Cloud [6] and Microsoft Azure [7].

2.7. Evolving threat landscape

The evolving nature of cybersecurity threats requires proactive and creative security approaches. AI/ML-driven anomaly detection and predictive analytics, as suggested by Malul et al. [2], are becoming integral to identifying and mitigating zero-day vulnerabilities and advanced persistent threats in Kubernetes clusters.

3. Kubernetes security method

3.1. Network security

Securing Kubernetes network communications is foundational for protecting enterprise clusters. Implementing service meshes like Istio or Linkerd provides fine-grained control over traffic flows, including encryption, authentication, and policy enforcement. Service meshes also enable detailed observability, which is critical for detecting and mitigating attacks. Kubernetes network policies allow administrators to define rules restricting traffic between pods or namespaces, reducing the

risk of lateral movement during attacks. According to recent studies, such as Song et al. [8], integrating service meshes with Kubernetes-native policies significantly improves overall network security in multi-tenant environments. Additionally, integrating firewalls and external security tools helps secure traffic entering and leaving the cluster. Recent studies, such as Liu et al. [9], emphasize the importance of enhancing network observability and minimizing trust boundaries using advanced network security policies.

3.2. Identity and access management

Effective Identity and access management (IAM) ensures that users, applications, and services have access only to the resources necessary for their functions. Role-Based Access Control (RBAC) is central to Kubernetes IAM, allowing administrators to grant specific permissions to users and service accounts. Best practices include:

- Using the principle of least privilege to restrict access.
- Segmenting clusters for multi-tenancy to isolate workloads.
- Enforcing multi-factor authentication (MFA) for user logins.

Recent studies, such as those by Liu et al. [9], highlight that fine-grained RBAC policies combined with MFA implementation significantly mitigate unauthorized access risks in Kubernetes environments. Similarly, underscore the importance of auditing IAM configurations regularly to ensure compliance and security. According to Ali et al. [10], fine-grained IAM controls reduce the attack surface and provide actionable insights for compliance and auditing.

3.3. Data security

Kubernetes stores sensitive data such as configuration secrets, API keys, and certificates. Encrypting data at rest and in transit is vital to prevent unauthorized access. Secrets management tools, like HashiCorp Vault or Kubernetes-native solutions, enable secure storage and rotation of credentials. Implementing volume encryption for persistent storage ensures compliance with data protection standards like GDPR and HIPAA. Haque et al. [1] suggest that integrating data encryption with identity-aware proxies further strengthens Kubernetes data security frameworks.

3.4. Continuous monitoring and runtime security

Real-time monitoring of Kubernetes environments allows for early detection of security incidents. Tools like Falco and Sysdig monitor runtime activities, flagging suspicious behavior, such as unauthorized container access or unexpected privilege escalations. Additionally, integrating these tools with centralized logging systems (e.g., ELK stack or Splunk) enhances threat investigation and response capabilities. Recent advancements, such as runtime analysis features in tools like Tracee, provide deeper insights into container and workload behavior, enabling enterprises to detect zero-day vulnerabilities and other sophisticated threats more effectively. Malul et al. [2] argue that runtime monitoring combined with predictive analytics can proactively prevent container-based vulnerabilities.

3.5. Compliance and auditing

Meeting regulatory requirements is crucial for enterprises. Kubernetes security frameworks, like CIS Benchmarks, provide detailed guidelines for configuration and compliance checks. Audit logs should capture all cluster activities, including API requests, user authentication, and resource changes. Enterprises can leverage tools such as Kubescape and Aqua Security to automate compliance checks and generate audit reports, simplifying adherence to standards like SOC 2 and PCI DSS. References like Check Point [11] highlight the increasing adoption of automated compliance management solutions in large-scale deployments.

4. Building an enterprise-grade secure Kubernetes infrastructure

Operating Kubernetes securely at the enterprise level requires a multifaceted approach that addresses infrastructure, tools, and workflows. This chapter outlines strategies for building a robust, scalable, and secure Kubernetes infrastructure suitable for S&P 500 companies.

4.1. Managed vs. on-premises Kubernetes

Enterprises face a critical decision between adopting managed Kubernetes services, like AWS EKS, Google Kubernetes Engine (GKE), or Azure Kubernetes Service (AKS), and deploying on-premises Kubernetes clusters. Managed services simplify operations with automated updates, scaling, and integrated cloud-native tools. However, they can limit control over security configurations and customization.

Conversely, on-premises Kubernetes offers unparalleled control, enabling organizations to implement tailored security measures, particularly for regulated industries. However, this requires a higher level of expertise, operational overhead, and resource commitment. Hybrid models, which combine managed and on-premises clusters, are increasingly popular for balancing flexibility with operational efficiency, as highlighted in Haque et al. [1].

4.2. Centralized identity and access management

As enterprises scale their Kubernetes deployments, centralized identity and access management (IAM) becomes vital. Disparate IAM systems across clusters can lead to inconsistent access controls and increased attack surfaces. Key practices include:

- **Unified Authentication:** Leveraging single sign-on (SSO) systems, such as Okta or Azure AD, ensures consistent authentication mechanisms across all clusters. Integrating these with Kubernetes-native tools like kube-oidc-proxy enables seamless authentication workflows.
- **Granular RBAC Policies:** Centralizing the management of RBAC policies across clusters prevents privilege escalation and ensures adherence to the principle of least privilege.
- **Auditing and Logging:** Regular audits of IAM configurations and continuous monitoring of authentication logs help detect anomalies, such as unauthorized access attempts.

Studies such as [12] and [13] emphasize that centralized IAM systems reduce security management overhead in large enterprises.

4.3. GitOps and policy enforcement

GitOps frameworks, like ArgoCD and Flux, have become cornerstones of enterprise Kubernetes management. By treating infrastructure as code, GitOps ensures that configuration changes are declarative, version-controlled, and auditable. Enterprises can rapidly recover from misconfigurations and maintain consistent cluster states across environments.

Integrating policy enforcement tools such as Open Policy Agent (OPA) and Kyverno within GitOps pipelines enhances security by automatically validating configurations against predefined policies before deployment. Liu et al. [9] emphasize that combining GitOps with policy enforcement significantly reduces human errors and configuration drift, enabling compliance and security consistency [11, 12].

4.4. GitOps and policy enforcement

The dynamic nature of Kubernetes environments necessitates the use of specialized tools to address security aspects. Kubernetes-native tools like Trivy, Falco, and Kyverno tackle specific areas:

- Trivy: Scans container images, repositories, and Infrastructure-as-Code (IaC) files for vulnerabilities.
- Falco: Monitors runtime system calls for suspicious activity, such as privilege escalations.
- Kyverno: Enforces policy compliance, including RBAC rules and network policies.

Integrating these tools into CI/CD pipelines and runtime monitoring workflows provides a layered approach to security. Recent advancements in tools like Falco's integration with eBPF have enhanced runtime observability.

4.5. High availability and disaster recovery

Enterprise-grade Kubernetes infrastructures must be resilient against failures and disasters. High availability (HA) involves deploying control plane nodes and worker nodes across multiple availability zones to ensure continuous operation during outages. Managed services like GKE and EKS provide out-of-the-box HA setups, but on-premises environments require manual configuration of load balancers, etcd clusters, and backup solutions [13, 14].

Disaster recovery strategies should include regular backups of etcd, container images, and application data. Tools like Velero facilitate automated backup and restore processes for Kubernetes resources. Incorporating multi-region failover mechanisms further strengthens an organization's disaster recovery posture [15, 16].

4.6. Secure development lifecycle in Kubernetes

Security in Kubernetes starts during the development phase. Implementing a Secure Development Lifecycle (SDLC) that integrates security checks early in the CI/CD pipeline minimizes vulnerabilities. Practices include:

- Code Scanning: Tools like SonarQube identify insecure coding patterns.
- Image Scanning: Solutions like Trivy detects vulnerabilities in base images.
- Policy as Code: Embedding security policies in configuration files ensures adherence to organizational standards.

Developers should also receive regular training on container security best practices. Studies like [17, 18] indicate that container security and threat detection are key components of enterprise Kubernetes security.

4.7. Zero trust architecture in Kubernetes

Zero Trust Architecture (ZTA) has emerged as a critical paradigm for securing modern Kubernetes environments. ZTA operates on the principle of "never trust, always verify," ensuring that no entity—whether inside or outside the network—is inherently trusted.

In Kubernetes, ZTA implementation includes:

- Strong Authentication: Using multi-factor authentication (MFA) and short-lived certificates for API server access.
- Microsegmentation: Isolating workloads at the pod level using network policies and service meshes like Istio.
- Continuous Verification: Monitoring access and actions in real-time using tools like OPA.

Recent research, including Ali et al. [10], demonstrates the effectiveness of ZTA in reducing attack surfaces and mitigating lateral movement within clusters [19, 20].

4.8. Multi-cluster observability and threat detection

Observability and threat detection become increasingly complex in multi-cluster environments [21, 22]. Enterprises must invest in centralized monitoring and real-time threat detection to ensure security at scale.

- **Centralized Logging and Metrics:** Platforms like Prometheus, Grafana, and Loki enable the aggregation of logs and metrics across clusters. Combining these tools with centralized SIEM solutions, such as Splunk or Elastic Security, provides actionable insights into security events [23, 24].
- **AI-Driven Threat Detection:** Machine learning models can analyze patterns in logs to detect anomalies and potential threats. For example, tools like ThreatMapper leverage AI to identify and prioritize vulnerabilities in Kubernetes workloads [25, 26].
- **Incident Response Automation:** Integrating security orchestration, automation, and response (SOAR) platforms with Kubernetes environments enables rapid mitigation of security incidents.

Research by Malul et al. [2] demonstrates that enterprises employing AI-driven threat detection reduce incident response times by 45%.

5. Experiments

Experiments play a critical role in evaluating and refining Kubernetes security mechanisms. This chapter outlines practical experiments designed to enhance cluster security by implementing advanced configurations and observing their impact in controlled environments. Each experiment is grounded in best practices and supported by real-life Kubernetes deployment scenarios.

5.1. Deploying images only from trusted sources

Ensuring that only trusted container images are deployed is fundamental to maintaining a secure Kubernetes environment. This experiment involves configuring admission controllers to enforce image source validation.

Setup:

- Enable Kubernetes' *Validating Admission Webhook* feature.
- Deploy an image policy webhook, such as Portieris, to validate image signatures and enforce policies.
- Configure policies to allow images only from signed and verified registries, such as Docker Hub or private repositories.

Execution:

- Attempt to deploy an unsigned or untrusted image and observe the rejection by the admission controller.
- Deploy a signed image from a trusted source and verify its acceptance.

Expected Outcome:

- Only signed images from trusted registries are permitted.
- Rejections are logged with detailed reasons for audit purposes.

```

apiVersion: v1
kind: Pod
metadata:
  name: nginx-proxy-exp
  labels:
    app: nginx-proxy-exp
spec:
  containers:
    - name: nginx
      # dockerhub registry disallowed for the experiment
      image: nginx
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop:
            - ALL
        runAsNonRoot: true
        runAsUser: 1000
        runAsGroup: 1000
        readOnlyRootFilesystem: true
        seccompProfile:
          type: RuntimeDefault

```

Figure 1: Configuring Kubernetes cluster to disallow public docker hub registry

```

Error from server: error when creating
"nginx-proxy-exp.yaml": admission webhook
"validate.kyverno.svc-fail" denied the request:

resource Pod/service/nginx-proxy-exp was blocked due
to the following policies

disallow-non-harbor-images:
  disallow-non-harbor-images: 'validation failure:
  Images may only come from the following
  trusted harbor registries: harbor.faizul.in'

```

Figure 2: Expected error complaining about invalid registry

5.2. Dropping unnecessary container capabilities

By default, containers run with a broad set of Linux capabilities, many of which are unnecessary and pose security risks. This experiment focuses on enforcing policies to drop all unnecessary capabilities for every workload.

Setup:

- Deploy a Kubernetes *PodSecurityPolicy* (PSP) or its successor, Pod Security Admission, to enforce dropping all capabilities by default using the *Drop*: [“ALL”] directive.
- Create a validating webhook to reject any pod definition that does not explicitly drop all capabilities.

Execution:

- Attempt to deploy a pod without specifying *Drop*: ALL in the *securityContext* and verify it is rejected.
- Deploy a compliant pod with *Drop*: ALL and test its functionality.

Expected Outcome:

- Only pods explicitly dropping all capabilities are successfully deployed.
- Logs and webhook responses provide clear feedback on rejected attempts.

```

apiVersion: v1
kind: Pod
metadata:
  name: nginx-proxy-exp
  labels:
    app: nginx-proxy-exp
spec:
  containers:
    - name: nginx
      image: nginx
      securityContext:
        allowPrivilegeEscalation: false
        # drop all capabilities disabled for the experiment
        capabilities:
          drop:
            - ALL
        runAsNonRoot: true
        runAsUser: 1000
        runAsGroup: 1000
        readOnlyRootFilesystem: true
        seccompProfile:
          type: RuntimeDefault

```

Figure 3: Enforcing “Drop ALL” capabilities

```

Error from server: error when creating
"nginx-proxy-exp.yaml": admission webhook
"validate.kyverno.svc-fail" denied the request:

resource Pod/service/nginx-proxy-exp was blocked due
to the following policies

disallow-capabilities-strict:
  require-drop-all: 'validation failure:
Containers must drop 'ALL' capabilities.'

```

Figure 4: Expected error complaining about “Drop ALL” capabilities

5.3. Enforcing rootless mode

Running containers as non-root users significantly mitigates the impact of potential container breakouts. Containers operating with root privileges inherently pose a risk because a vulnerability within the containerized application could be exploited to gain access to the host system or escalate privileges within the cluster. By enforcing rootless mode, organizations can limit the damage caused by such exploits, as non-root containers lack the administrative privileges required to perform critical actions on the host. This experiment demonstrates the configuration and benefits of enforcing rootless mode for workloads by strictly prohibiting deployments with *runAsNonRoot* set to *false*. It also emphasizes the importance of explicitly defining *runAsUser* and *runAsGroup* fields to ensure that workloads are running under specific, non-root identities. Such configurations provide an additional layer of security by mapping the container’s user and group IDs to restricted permissions, effectively adhering to the principle of least privilege. By validating these settings during deployment, Kubernetes administrators can ensure that even if a containerized application is compromised, the attack surface remains minimal.

Setup:

- Analyze SLAM algorithms, their advantages and disadvantages.
- Create a Kubernetes admission controller webhook to enforce security contexts requiring:
 - *runAsNonRoot* set to true.
 - *runAsUser* and *runAsGroup* explicitly defined.
- Update Kubernetes manifests to reflect these requirements in their securityContext.
- Configure the cluster’s Pod Security Standards (PSS) to disallow any pods without a properly configured securityContext.

Execution:

- Attempt to deploy a pod with runAsNonRoot set to false and verify that it is rejected.
- Deploy a pod with runAsNonRoot: true, specify runAsUser: 1000 and runAsGroup: 1000, and ensure it functions as expected.
- Simulate potential security breaches, such as attempting privilege escalations, and observing the cluster's responses.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-proxy-exp
  labels:
    app: nginx-proxy-exp
spec:
  containers:
    - name: nginx
      image: nginx
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop:
            - ALL
        runAsNonRoot: false # Should be true
        runAsUser: 1000
        runAsGroup: 1000
        readOnlyRootFilesystem: true
        seccompProfile:
          type: RuntimeDefault
```

Figure 5: Container requires root privileges

```
Error from server: error when creating
"nginx-proxy-exp.yaml": admission webhook
"validate.kyverno.svc-fail" denied the request:

resource Pod/service/nginx-proxy-exp was blocked due
to the following policies

require-run-as-nonroot:
  run-as-non-root: 'validation error:
  Running as root is not allowed.
  Either the field spec.securityContext.runAsNonRoot
  must be set to 'true', or the fields
  spec.containers[*].securityContext.runAsNonRoot,
  spec.initContainers[*].securityContext.runAsNonRoot, and
  spec.ephemeralContainers[*].securityContext.runAsNonRoot
  must be set to 'true'. rule run-as-non-root[0]
  failed at path
  /spec/containers/0/securityContext/runAsNonRoot/
  rule run-as-non-root[1] failed at path
  /spec/containers/0/securityContext/runAsNonRoot/'
```

Figure 6: Error on requiring root privileges

Expected Outcome:

- Pods without runAsNonRoot: true or missing runAsUser/runAsGroup are rejected at deployment.
- Workloads run with non-root privileges and adhere to the principle of least privilege.
- Logs and monitoring tools confirm enforcement and capture rejection details.

5.4. Enforcing read-only root filesystem

Ensuring that containers operate with a read-only root filesystem is a critical measure to minimize the impact of potential exploits. By restricting the filesystem to read-only mode, administrators can prevent attackers from modifying or overwriting files within the container, which is a common tactic used to escalate privileges, install malicious software, or exfiltrate sensitive data. This approach aligns with the principle of immutability, where containers are treated as static artifacts that should not change after deployment. This experiment demonstrates how to enforce the *readOnlyRootFilesystem* setting to prevent unauthorized modifications to container filesystems. By doing so, organizations can ensure that even if an application vulnerability is exploited, the attacker is unable to persist changes or disrupt critical container operations. Additionally, the enforcement of a read-only filesystem reduces the attack surface, as it limits writable paths and ensures that application dependencies, configurations, and binaries remain intact and secure.

Setup:

- Update the securityContext of your pod specifications to include `readOnlyRootFilesystem: true`.
- Implement a validating admission webhook to enforce the use of `readOnlyRootFilesystem` across all workloads in the cluster.
- Use Kubernetes Pod Security Standards (PSS) to define and enforce policies disallowing writable root filesystems.

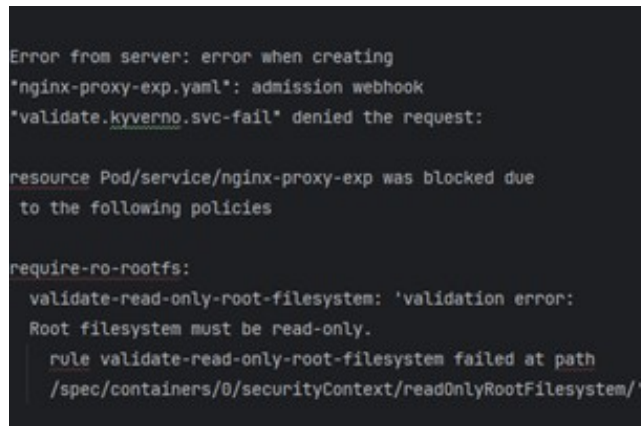
```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-proxy-exp
  labels:
    app: nginx-proxy-exp
spec:
  containers:
    - name: nginx
      image: nginx
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop:
            - ALL
        runAsNonRoot: true
        runAsUser: 1000
        runAsGroup: 1000
        readOnlyRootFilesystem: false # Should be true
        seccompProfile:
          type: RuntimeDefault
```

Figure 7: Container with `readOnlyFilesystem` set to false

Execution:

- Attempt to deploy a pod without `readOnlyRootFilesystem: false` and observe the rejection by the webhook or policy enforcement.
- Deploy a pod with the `readOnlyRootFilesystem` setting enabled and validate its functionality by attempting to write to the root filesystem.

- Simulate a potential attack scenario, such as injecting malicious scripts, and observe the system's inability to persist these changes.



```
Error from server: error when creating
"nginx-proxy-exp.yaml": admission webhook
"validate.kyverno.svc-fail" denied the request:

resource Pod/service/nginx-proxy-exp was blocked due
to the following policies

require-ro-rootfs:
  validate-read-only-root-filesystem: 'validation error:
  Root filesystem must be read-only.
  rule validate-read-only-root-filesystem failed at path
  /spec/containers/0/securityContext/readOnlyRootFilesystem/'
```

Figure 8: Container rejection with read Only Root File System set to false

Expected Outcome:

- Pods without readOnlyRootFilesystem: true are rejected at deployment.
- Workloads operate with immutable root filesystems, ensuring unauthorized changes cannot be made.
- Logs and monitoring tools capture details of policy violations and deployment rejections.

5.5. Limiting resource usage

Limiting resource usage is a fundamental practice in maintaining a stable and secure Kubernetes cluster, especially in multi-tenant environments. Resource constraints help prevent a single workload from monopolizing CPU or memory resources, which could otherwise lead to Denial of Service (DoS) scenarios or adversely impact the performance of other workloads. These constraints also protect against malicious actors or misconfigured applications that might attempt to overwhelm cluster resources, either intentionally or unintentionally. By enforcing resource limits, Kubernetes administrators can ensure a fair and predictable allocation of resources across all deployed applications, promoting better overall cluster performance and stability. This experiment explores the practical steps to configure and monitor resource quotas and limits effectively. It highlights how resource limits can act as a safeguard against cascading failures by isolating resource-intensive workloads and ensuring that other services remain unaffected.

Setup:

- Define resource quotas and limits in the namespace using ResourceQuota objects.
- Set resource requests and limits for individual pods in the resources section of their manifests.

Execution:

- Deploy workloads with and without resource constraints.
- Simulate resource-intensive operations and observe the impact on cluster performance.

Expected Outcome:

- Pods exceeding their allocated resources are throttled or evicted, preventing disruption to other workloads.

- Monitoring tools like Prometheus or Grafana provide insights into resource utilization trends.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-proxy-exp
  labels:
    app: nginx-proxy-exp
spec:
  selector:
    matchLabels:
      app: nginx-proxy-exp
  template:
    metadata:
      labels:
        app: nginx-proxy-exp
    spec:
      containers:
        - name: nginx
          image: nginx
          imagePullPolicy: Always
          resources: # Limiting Requests and Limits
            requests:
              cpu: "1"
              memory: 16i
            limits:
              cpu: "2"
              memory: 26i
```

Figure 9: Limiting container resources

```
Error from server: error when creating
'nginx-proxy-exp.yaml': admission webhook denied
the request: Requests and/or limits for container
nginx are missing. Please set requests and limits for
all the containers according to their average usage.
```

Figure 10: Error message regarding missing requests and limits

Conclusions

Kubernetes has revolutionized the way organizations deploy and manage applications, offering unparalleled scalability and flexibility. However, its adoption at scale introduces complex security challenges that demand thoughtful solutions and proactive measures. This paper has explored the multifaceted landscape of Kubernetes security, from identifying challenges to implementing key security pillars and designing practical experiments to validate these measures.

The critical challenges highlighted include ensuring secure configurations, mitigating risks associated with insider threats and supply chain attacks, and maintaining robust runtime security. The key pillars of Kubernetes security, such as identity and access management, network policy enforcement, workload isolation, and observability, provide a comprehensive framework for addressing these challenges. Practical experiments detailed in this work, such as enforcing image

provenance, dropping unnecessary container capabilities, and implementing runtime security tools, offer actionable steps for organizations to strengthen their Kubernetes environments.

As Kubernetes continues to evolve, so do the threats targeting its ecosystems. Organizations must remain vigilant and adaptable, leveraging the latest tools, practices, and research to safeguard their infrastructure. Future work in Kubernetes security should focus on emerging technologies like service meshes, advanced threat detection mechanisms, and automated policy enforcement driven by machine learning.

By adopting a holistic and experimental approach to Kubernetes security, organizations can not only secure their clusters but also build a culture of continuous improvement and resilience. In an era where cybersecurity threats are ever-present, the insights and strategies discussed in this paper serve as a foundation for enterprises striving to achieve robust, scalable, and secure Kubernetes deployments.

Declaration on Generative AI

While preparing this work, the authors used the AI programs Grammarly Pro to correct text grammar and Strike Plagiarism to search for possible plagiarism. After using this tool, the authors reviewed and edited the content as needed and took full responsibility for the publication's content.

References

- [1] M. U. Haque, M. M. Kholoosi, M. A. Babar, Kgsecconfig: A knowledge graph based approach for secured container orchestrator configuration, in: IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), 2022, 420–431.
- [2] E. Malul, et al., GenKubeSec: LLM-based Kubernetes misconfiguration detection, localization, reasoning, and remediation. arXiv, 2024. doi:10.48550/arXiv.2405.19954
- [3] Red Hat. The State of Kubernetes Security in 2024, 2024. URL: <https://www.redhat.com/en/blog/state-kubernetes-security-2024>
- [4] M. S. I. Shamim, F. A. Bhuiyan, A. Rahman, XI Commandments of Kubernetes security: A systematization of knowledge related to Kubernetes security practices, in: IEEE Secure Development (SecDev), 2020, 58–64.
- [5] SentinelOne, Kubernetes security companies in 2025, 2025. URL: <https://www.sentinelone.com/cybersecurity-101/cloud-security/kubernetes-security-companies/>
- [6] Google Cloud, GKE security overview. URL: <https://cloud.google.com/kubernetes-engine/docs/concepts/security-overview>
- [7] Microsoft, Azure security baseline for Azure Kubernetes service (AKS), 2023. URL: <https://learn.microsoft.com/en-us/security/benchmark/azure/baselines/azure-kubernetes-service-aks-security-baseline>
- [8] E. Song, et al., Canal mesh: A cloud-scale sidecar-free multi-tenant service mesh architecture, in: Proceedings of the ACM SIGCOMM 2024 Conference, 2024, 860–875.
- [9] C. Liu, et al., A protocol-independent container network observability analysis system based on eBPF, in: IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS), 2020, 697–702.
- [10] A. Ali, et al., Implementation of new security features in CMSWEB Kubernetes Cluster at CERN, in: EPJ Web of Conferences, vol. 295, 2024.
- [11] Check Point, Kubernetes security: Challenges and best practices, 2023. URL: <https://www.checkpoint.com/cyber-hub/cloud-security/what-is-kubernetes/kubernetes-k8s-security/>
- [12] M. Gupta, Deployment of identity and access management solutions (IAM) using DevOps environment, 2023.

- [13] S. Yi, Secure IAM on AWS with multi-account strategy. arXiv, 2025. doi:10.48550/arXiv.2501.02203
- [14] A. Eldjou, et al., Enhancing container runtime security: A case study in threat detection, in: Tunisian-Algerian Joint Conference on Applied Computing, vol. 3642, 2023, 55–69.
- [15] A. Poniszewska-Marańda, E. Czechowska, Kubernetes cluster for automating software production environment, *Sensors*, 21(5) (2021).
- [16] I. Dronjuk, M. Nazarkevych, O. Troyan, The modified amplitude-modulated screening technology for the high printing quality, in: International Symposium on Computer and Information Sciences, 2016, 270–276. doi:10.1007/978-3-319-47217-1_29
- [17] K. Senjab, et al., A survey of Kubernetes scheduling algorithms, *J. Cloud Comput.* 12(1) (2023).
- [18] V. Hrytsyk, M. Nazarkevych, Real-time sensing, reasoning and adaptation for computer vision systems, in: Lecture Notes in Computational Intelligence and Decision Making: 2021 International Scientific Conference, Intellectual Systems of Decision-making and Problems of Computational Intelligence, vol. 77, 2022, 573–585.
- [19] P. Skladannyi, et al., Improving the security policy of the distance learning system based on the zero trust concept, in: Cybersecurity Providing in Information and Telecommunication Systems, vol. 3421 (2023) 97–106.
- [20] R. Syrotynskyi, et al., Methodology of network infrastructure analysis as part of migration to zero-trust architecture, in: Cyber Security and Data Protection, vol. 3800 (2024) 97–105.
- [21] Z. Rejiba, J. Chamanara, Custom scheduling in Kubernetes: A survey on common problems and solution approaches, *ACM Comput. Surv.* 55(7), (2022) 1–37.
- [22] I. Dronyuk, M. Nazarkevych, Z. Poplavska, Gabor filters generalization based on ateb-functions for information security, in: International Conference on Man-Machine Interactions, 2017, 195–206.
- [23] V. Sokolov, P. Skladannyi, N. Korshun, ZigBee Network resistance to jamming attacks, in: IEEE International Conference on Information and Telecommunication Technologies and Radio Electronics (UkrMiCo), 2023, 161–165.
- [24] V. Sokolov, P. Skladannyi, A. Platonenko, Jump-stay jamming attack on Wi-Fi systems, in: IEEE 18th International Conference on Computer Science and Information Technologies (CSIT), 2023, 1–5.
- [25] V. Senkivskyy, et al., Modeling of alternatives and defining the best options for websites design, in: 2nd International Workshop on Intelligent Information Technologies and Systems of Information Security, vol. 2853, 2021, 259–270.
- [26] I. Tsmots, et al., Intelligent motion control system for the mobile robotic platform, in: 7th International Conference on Computational Linguistics and Intelligent Systems, vol. 3403, 2023, 555–569.