National College of Ireland

Project Submission Sheet

Student Name:	Balamuralikrishna Raju		
Student ID:	23326701		
Programme:	MSc in cloud computing	**Year:**	2025 - 2026
Module:	Cloud Computing Research Project		
Lecturer:	Shaguna Gupta		
Submission Due Date:	11/12/2025		
Project Title:	Comparative Analysis of Open-Source Kubernetes Runtime Security Tools in Cloud Platforms of AKS and EKS		
Word Count:	8285		

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the references section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.

Signature: Balamuralikrishna Raju

Date: 11/12/25

PLEASE READ THE FOLLOWING INSTRUCTIONS:

1. Please attach a completed copy of this sheet to each project (including multiple copies).
2. Projects should be submitted to your Programme Coordinator.
3. **You must ensure that you retain a HARD COPY of ALL projects**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. Please do not bind projects or place in covers unless specifically requested.
4. You must ensure that all projects are submitted to your Programme Coordinator on or before the required submission date. **Late submissions will incur penalties.**
5. All projects must be submitted and passed in order to successfully complete the year. **Any project/assignment not submitted will be marked as a fail.**

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

AI Acknowledgement Supplement

[Insert Module Name]

[Insert Title of your assignment]

Your Name/Student Number	Course	Date

This section is a supplement to the main assignment, to be used if AI was used in any capacity in the creation of your assignment; if you have queries about how to do this, please contact your lecturer. For an example of how to fill these sections out, please click here.

AI Acknowledgment

This section acknowledges the AI tools that were utilized in the process of completing this assignment.

Tool Name	Brief Description	Link to tool

Description of AI Usage

This section provides a more detailed description of how the AI tools were used in the assignment. It includes information about the prompts given to the AI tool, the responses received, and how these responses were utilized or modified in the assignment. **One table should be used for each tool used**.

[Insert Tool Name]	
[Insert Description of use]	
[Insert Sample prompt]	[Insert Sample response]

Evidence of AI Usage

This section includes evidence of significant prompts and responses used or generated through the AI tool. It should provide a clear understanding of the extent to which the AI tool was used in the assignment. Evidence may be attached via screenshots or text.

Additional Evidence:

[Place evidence here]

Additional Evidence:

[Place evidence here]

National College of Ireland

Comparative Analysis of Open-Source Kubernetes Runtime Security Tools in Cloud Platforms of AKS and EKS

MSc Research Project
Cloud computing

Balamuralikrishna Raju
Student ID: 23326701

School of Computing
National College of Ireland

Supervisor: Shaguna Gupta

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Balamuralikrishna Raju
Student ID:	23326701
Programme:	MSc cloud computing **Year:** 2025 – 2026
Module:	Cloud Computing Research Project
Supervisor:	Shaguna Gupta
Submission Due Date:	11/12/2025
Project Title:	Comparative Analysis of Open-Source Kubernetes Runtime Security Tools in Cloud Platforms of AKS and EKS
Word Count	……8285……**PageCount**……23…….

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Balamuralikrishna Raju

Date: 11/12/2025……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

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

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

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

Comparative Analysis of Open-Source Kubernetes Runtime Security Tools in Cloud Platforms of AKS and EKS

ABSTRACT

An increasing use of cloud native adoption, significantly raised the adoption of containerised microservices, one such effective container orchestration platform is Kubernetes. Though the cloud managed Kubernetes services like AWS EKS (elastic Kubernetes service) Azure AKS (azure Kubernetes service) made the deployment and scaling in and out of the microservices easier, it has introduced significant runtime security challenges that the cloud providers inadequately addressed. Some of the open-source security tools like Kube armour, tetragon, Tracee, Falco are helpful in detecting the runtime security challenges, but their comparative efficacy and operational impact in multi cloud environments is largely unaddressed. The previous work done by malik and vugt (2023) gives a valuable information but these tests were conducted on a in a controlled hardware testbed environment, this left a critical gap of knowing their performance on different clouds platforms. This research conducts a comparative analysis of these tools by deploying them in EKS and AKS cluster, which hosts a set if microservice application. By applying a simulates cyber-attack and setting up monitoring tools like Prometheus and Grafana, this study evaluates each tool's detection accuracy latency resource usage like CPU and memory. These findings will give insights of trade-offs between security, observability and system overhead in major cloud native Kubernetes environments, which guides in required tool selection based on specific cloud context and performance requirement.

1 INTRODUCTION

The rapid migration of on-premises to cloud services has grabbed significant attention among the stake holders, it helps in reducing the initial setting up infrastructure cost, post maintenance cost and everything. With the advent of this, cloud orientated microservice architecture rose into prominence from the conventional monolithic architecture, this led to microlithic architecture which in turn induced micro services-oriented container orchestration, though there are some popular container orchestration tools are available in the market, among them Kubernetes emerged as popular container orchestration for scale in/out the services according to the demand. Though Kubernetes has many advantages it has some limitations like runtime security, observability and policy enforcement. The cloud managed Kubernetes services like elastic Kubernetes service (EKS) and azure Kubernetes service (AKS) secures the cluster by managing the master control plane, scheduler etc, but it comes with certain issues to be addressed like container runtime security within the cluster.

The important things to be addressed here is runtime security of the container. To mitigate these concerns deploying security tools like Falco Kube Armour, and Tetragon to detect and block the threats for ensuring container runtime security. Though these tools are known for identifying and blocking attacks by using the customized policies, there is no concrete evidence these tools behaviour in cloud platforms and their resource utilization in AKS and EKS.

Earlier researchers like Thomas Martijn van Vugt and Tania Malik (2023) used these tools in two contrast hardware lab environments which had given remarkable output and set a benchmark. Indeed, the need to evaluate the tools in respect of real-world micro service application in public cloud platform is largely remain unaddressed.

This research is driven towards addressing these gaps in cloud platforms. To understand the performs of these security tools, sample application is deployed in the Kubernetes cluster. Security tools are also deployed using Helm chart in the cluster. The aim is to see those tools effectiveness in terms of its detection, prevention of unauthorized access and its management of cluster resource utilization across two major cloud Kubernetes platforms. By injecting shell attacks, evaluation of each tool's response time, accuracy is determined in the two different cloud platforms.

1.1 Background of the problem:

Security challenges in Kubernetes are multivariant. The static analysis of manifest of files for misconfiguration and image vulnerability scanning in the pre deployment stage are essential but are in effective in detecting threats during the runtime of the application. Runtime security focuses on while the application is executing, which is crucial in containerized application, if a attacker breaches one container, it would be easy to take over the entire cluster. Native Kubernetes mechanisms like pod security standards and admission controllers offer certain protection, but they are limited to API request level, which introduce latency and lack of visibility of actual container behaviour in post-deployment.

This led way to dynamic tracing tools which use eBPF, a technology in Linux kernel that ensures safe and efficient program execution in kernel without changing the kernel source code. These tools like Falco, Tracee, Tetragon uses eBPF to trace system calls and network activity providing real time insights into anomalous behaviour. Whereas Kube armour combines eBPF and Linux security modules to enforce security policies at container level. The research done by malik and vugt (2023) were held controlled hardware test beds which doesn't reflect the managed service integrations of public cloud Kubernetes like EKS and AKS. The performance of these tools can be significantly differed in a cloud platform due to factors like virtualized hardware, cloud specific network plugins.

1.2 Motivation of the research

The clear and critical gap of the lab-based evaluations of Kubernetes security tools and their practical implementation in cloud environments. The findings of this research will be relevant for teams taking strategic decisions of their Kubernetes stack.

To bridge multi cloud knowledge gap: There is no proper evidence of how these security tools perform on the containerized environment of the public cloud. This study will compare the efficiency of these tools in EKS and AKS, which gives insights into how cloud context influences security outcomes.

Real world tool selection: To show the trade-off involved to the organisation while selecting between the tool to determine whether Kube armour dealing with higher resource cost in cost sensitive environment compared to Falco's equally rule based detection perform equally well in terms of networking.

Integration into CI/CD pipelines: It will possibly offer valuable insights for seamlessly integrating runtime security into devsecops pipeline for EKS and AKS

1.3 Problem statement:

Current lab-based performance attestations of Kubernetes security tools are useful as benchmarks but their findings are based on controlled testbeds of hardware and do not provide practical context-rich information on production-scale cloud infrastructures. These tools may be affected by the underlying cloud-specific infrastructures and integrations of managed services into other platforms, like AWS Elastic Kubernetes Service (EKS) and Azure Kubernetes Service (AKS), which can have a significant impact on the performance, the detection accuracy, and the resource overhead of the tools. This research paper fills this important gap by comparing the performance of the top open-source security tools in these most common cloud-native Kubernetes environments systematically, thus producing the much-needed data on how tools can be used to select and implement in the real world.

1.4 Research question

What is the impact of open-source Kubernetes security tools on observability, threat detection and resource usage when deployed in AWS EKS and Azure AKS?

Problem solution: In order to test these Kubernetes tools in cloud, we will be creating the EKS and AKS cluster in AWS and Azure respectively, deploy micro service application through a CI/CD pipeline, along with these tools using helm charts. Inject cyber-attacks and to see how these open-source tools perform in detection of cyber-attacks, its response time and resource usage in the cluster. Its performance is monitored using Prometheus and Grafana.

1.5 Research objective: Primary aim of the research is to give a comparative performance analysis of open-source Kubernetes security tools in AWS EKS and Azure AKS environments.it is achieved by

1. Create a standard EKS and AKS cluster and deploy a microservice application and selected security tools using helm chart.
2. Evaluate the performance of each tool (Falco, Tracee, Tetragon, Kube Armor) in detecting and alerting on set of simulated runtime attacks. Each tool will determine detection accuracy, latency.
3. Will assess each tool's system usage like CPU and memory consumption which impacts cluster performance on the whole.
4. To know about cross cloud performance metrics of tools between AWS and Azure to identify any major variations were there because of the underlying cloud providers infrastructure and services.
5. Based on the findings give practical recommendations to DevOps teams in using the optimal security tools to match their priorities such as maximizing detection, minimizing system usage and cloud specific integration.

1.6 Limitations

1. The study conducted with respect to two major cloud provider like AWS and Azure, it doesn't deal with other cloud providers.
2. The simplified attack like reverse shells, cryptojacking may not encompass the full multi stage attack techniques.
3. The microservice application used for testing might have less traffic when compared to the real enterprise grade systems. So, the resource overhead and tools performance might vary under high volume of load.
4. The Kubernetes and these tools version may change in future, so the efficacy of the results may also change with change in versions
5. The whole experiment will be conducted in a small cluster setup i.e. the number of worker nodes in the cluster will be less, duration of the network traffic and testing will be also be in less time to minimize the cost.

1.7 STRUCTURE OF THE REPORT

Literature review: This section deals with related previous works, what they did with respect to their agenda, what is the gap the they left in the research and how this research address those critical gaps.

Research methodology: This section deals with overall methodology used to achieve the desired results, what are the security tools used to meet the requirement of the agenda and it basically contains the blue print of the work.

Design specification: This section specifies the architectural diagram and the cloud platform, security, monitoring tools used for this purpose.

Implementation: This stake deals how all the infrastructure set up were set up in both the cloud platforms as well, how the security and monitoring tools were deployed in the environment.

Evaluation: This section deals with based on the results acquired from both the cloud platforms, how it is analysed with respect to each tool, in each cloud, and cross cloud evaluation, also compared these results with base paper also discussed.

Conclusion: This section deals with overall analysis of the result in both the cloud platforms and along with base paper, what is inferred.

Future work: It address the limitation or some potential gap of the research results and how it can be taken it forward for the future research with some hints.

2 LITERATURE REVIEW

This paper is a systematic review of Kubernetes security literature in four areas, namely pre-deployment security, runtime detection tools, service mesh security and security provisioning. Although the available literature thoroughly addresses the technique of the static analysis and lab-based tests it indicates the immediate need of lack of knowledge regarding the performance of open-source applications of runtime tools in a cloud setup such as the AWS EKS and Azure AKS. The preexisting literature work does not provide comparative data on the accuracy of detection, overhead of resource, and the effect of the use of tools such as Falco, Tetragon and Kube Armour in the cloud managed Kubernetes services. This critical gap forms the basis of research of our comparative analysis of these tools in multi-cloud environments.

2.1 HANDLING OF PRE-DEPLOYMENT SECURITY AND CONFIGURATION MANAGEMENT IN KUBERNETES

The research by Kapetanidou (2025) selected six security scanning tools to give practical need for tools selection, which contributes in a significant performance variation and capabilities between the static tools and workload scanning approaches. However, their research primarily focused on misconfiguration detection and vulnerability check, leaving a crucial gap in runtime security protection evaluation. Our research addresses this gap by focussing on deploying the Kubernetes security tools like Kube armour, tetragon and Falco in public cloud Kubernetes services like EKS and AKS and measuring their performance outcomes.

Mahajan and Mane (2022) have tried to mitigate the problem of container security with respect to the crucial aspect of misconfigurations, which occurred during the deployment phase. They implemented a model of pre-runtime configuration checks, which checks the configuration of Docker containers with respect to a set security standard, such as the CIS Docker Benchmark, before allowing the deployment to happen. Their analysis of often occurring threats to containers in a systematic manner and designed a model of identifying configuration flaws in images and Docker files is a significant shift-left security measure. Nevertheless, the model they suggest is only workable at the pre-deployment stage, which leaves a considerable gap since it does not provide the safeguard against the threats that can appear when containers are deployed successfully and are in the running state. The gap that this research addresses is a critical lack of comparative studies on the work of open-source runtime security tools (Falco, Kube Armor, Tracee, Tetragon) on cloud Kubernetes infrastructures (EKS and AKS). The point at which Mahajan and Mane leave deployment validation is where this research starts with regards to runtime in terms of how these tools identify active threats, the effect on performance, and their effectiveness in a real-life cloud environment, thus giving a full picture of any security assessment of the entire lifecycle of containers.

Rahman et al. (2023), conducted an empirical study on Kubernetes manifests and identified eleven common security misconfigurations like having hardcoded secrets, missing resource usage limitation, and use of insecure http, he measured their frequency by using SLI-KUBE tool. He himself mentioned that they lack in detecting the runtime security checks. Our research greatly addresses this gap by implementing Kubernetes runtime security tools EKS and AKS of public cloud and generating a cyber-attack against a micro service to see their performance of these tools.

Nerella and Puvvada (2024) addresses the existing Kubernetes security problems from the enterprise point of view and recommends comprehensive preventive control across eight security domains. Their work highly revolves around infrastructure hardening, policy as code, access management, network policies, encryption and secrets management etc. Though they acknowledge runtime threats, they didn't deep dive into it, like how these measures perform against active runtime attacks, their work predominantly focusses on pre-deployment and configuration security.

The work by Shamim, Bhuiyan and Rahman (2020) they identified and categorized eleven best security practices like applying pod and network security policies, implementing RBAC, and comprehensive logging etc. It identifies the security requirements but doesn't evaluate technical solution or tools which can be used to achieve the security of the micro service application. This paves way for the logical pathway for our research, to address these issues we will be using opensource tools like Falco, Tetragon, Tracee, and Kube Armour in the public cloud Kubernetes environments like AKS and EKS.

2.2 RUNTIME THREAT DETECTION AND EBPF-BASED SECURITY TOOLS IN KUBERNETES

The research done by malik and vugt(2023) resulted in giving significant results. They tested these four open-source Kubernetes security tools like Falco, Kube Armour, Tracee, Tetragon in a confined environment. Though this research they found how these tools were effective in detecting the threats to the microservice application during runtime, with varied system resource usage for each tool. It is observed that Kube Armour offers policy enforcement at robust level, with the cost of using high CPU and memory, whereas eBPF based tools like Falco, and tetragon gives comprehensive observability with less resource foot print. However, this research was conducted on grid 5000 testbed and a private server. The performance and efficiency of the tools are not addressed in major public clouds, where factors like virtualized infrastructure and cloud native networking could profoundly alter tools behaviour. So, therefore this research gap enforces to test these Kubernetes tools in major AWS EKS and Azure AKS cloud platforms, to bridge the gap between lab-based benchmark and cloud native practice.

The eBPF runtime security is further exemplified by Gwak et al. (2023), he developed a tool called KRSIE Kubernetes runtime security instrumentation and enforcement, their work contributed toward dynamic security provided using eBPF technology, without container restarting and compared with tetragon and Kube armour. However, their work is limited to custom built tool KRSIE and didn't test in on any major cloud providers like AWS and Azure. Whereas our research work in carried out in public cloud with multiple tools like Falco, and multiple attack scenarios, and monitoring the resource usage of these tools as well.

Kubernetes runtime security is one of the most important critical issues, and container escape is the most dangerous types of threats, which eventually leads to compromise the whole cluster itself. Research was constantly evolved to address and mitigate these challenges. As an example, CPCED, a dedicated system implemented as a CNI plug, to identify container escapes in real-time by tracking interactions on the namespace interactions, is proposed by Hao et al., which proved to work highly in a controlled test setting (Hao et al., n.d.). Yet, though these academic prototypes can be effective in demonstrating the feasibility of a given concept, their implementation needs integration and sustainability which might not be applicable to every organisation. On the other hand, a number of more mature, open-source security tools such as Falco, Tetragon, and Kube Armor are also easily accessible and utilise technologies such as eBPF as a runtime protection tool. The gap that must be closed urgently is moving beyond the regulated testing of individual systems such as CPCED to the realisation of the trade-offs involved in the operational deployment of these trendy tools in production-grade, cloud-native applications. Earlier assessments, including one by Malik and van Vugt (2023) have been restricted to hardware testbeds alone with a large knowledge gap on their performance, resource consumption, and detection capabilities in a managed cloud Kubernetes service like AWS EKS and Azure AKS where underlying infrastructure and networking can drastically impact the behaviour of these security tools.

Recent studies have moved Kubernetes security into active deception techniques, such as the study conducted by Kahlhofer et al. (2025), their Koney framework, which is a cyber deception framework as code based on policy documents. Koney uses cloud-native tools, such as Tetragon with eBPF to keep vigil on file access to honeytokens and Istio service mesh to identify at the HTTP level, all this paved a way for use of an advanced security orchestration. Nonetheless, as Koney manages to show how these technologies can be used to deceive, but it lacks to discuss the performance features of them, their resource usage, and the response time in production clouds. This study seals that gap by performing a comparative study of these very tools - such as Tetragon and other eBPF based security packages such as Falco - in terms of their operational impact, detection accuracy, hence system overhead when implemented in AWS EKS and Azure AKS clusters and therefore provides the necessary performance data that deception frameworks like Koney utilises but does not itself provide.

The work by R. Bagnato (2025) emphasis on container security, by acknowledging runtime threats as critical challenge and identified technologies like eBPF as emerging solutions for mitigating the challenges. However, it’s a review paper, its scope was limited to identifying the problems and giving ideas to research direction, it doesn’t implement or provide any actual evaluated data on their comparative performance. This research fills the gap by implementing runtime threat detection using Kubernetes open-source security tool in major public clouds like AKS and EKS.

Kazenas and Ponomareva (2023) provides a valuable comparative survey of different commercial tools, and open security tools like sysdig, aqua security, Falco. It highlights their potential use of case of runtime detection and runtime protection features, and it finds Falco as potent tool for detecting the vulnerabilities and at the same time its lack of prevention capabilities. However their study was qualitative and feature based, they simply analysed by official documentation and marketing materials of what features these tools have, but doesn't implement how effectively and efficiently these tools will function in real time scenarios. This gap gives us an opportunity to test these Kubernetes tools in major public cloud like AWS and Azure environments.

2.3 SERVICE MESH AND POLICY-BASED SECURITY IN KUBERNETES

Ashraf et al (2025) used two approaches like using Istio for mTLS-based micro service-authentication and open policy agent to prevent un authorised access. Their work largely deals with breaches through strict access control and TLS based encryption, but it did not address runtime concerns that bypass these controls. This research directly addresses these gaps by detecting and preventing by using tools like Kube armour, Falco and Tetragon. Thus, providing a very much needed second line of defence, which detects attacks not covered by their model of policy-based prevention.

The challenge of securing the intricate communication layer in microservice architecture seems to be a important issue of cloud-native enviornments. Alboqmi et al. address this gap in service mesh security, as it is stated that whilst platforms such as Istio offer core security by pre-established settings, they do not offer a dynamic, runtime evaluation of trust to prevent the application being compromised by a service that is now an attacker, issuing unauthorised API calls. To solve this, they introduce a Runtime Trust Evaluator (RTE) which is based on a Service Dependency Graph to identify the anomalous service-to-service requests, and imposes a Zero-Trust posture on the application network layer. Nevertheless, this emphasis on service mesh communication security coexists with the no less important requirement of container-level runtime security, protection against such threats as container escape and privilege escalation that have their origins inside a pod. The usability and effectiveness of the open-source tools that are aimed at this latter goal, like Falco, Tetragon, and Kube Armor, are not quantified in real-world and multi-cloud Kubernetes environments, which constitute a critical gap in the guidance that DevOps teams receive regarding taking a multi-layered approach to security.

Kermabon-Bobinne et al. (2025) deals with performance overhead generated by such policy-as-code tools as OPA/Gatekeeper in large Kubernetes clusters. They determine that reactive intercept-and-cheque procedures may introduce delays of up to 600 ms in an 800-pod cluster, and even result into security breaches because of delays in state replication (CVE-2021-43979). Their solution, PerfSPEC presents a proactive design which automates the profiling and ranking of security policies based on performance impact and predictive models (Bayesian Networks, n-grammes, LSTM) to pre-calculate verification outcomes which is less than 10 ms to enforce.

Although PerfSPEC has an innovative method of optimising overheads in policy enforcement, its performance, similar to much of the literature, is not explicitly tested regarding this performance in the large public cloud Kubernetes services (EKS, AKS). The effectiveness of such a proactive system might be greatly dependent on the performance attributes of the virtualized infrastructure and networking of the underlying cloud provider. Thus, our study supplements this study with the essential cloud-contextualised performance information on runtime security tools, which builds a more comprehensive view of the trade-offs in the operation of a multi-cloud Kubernetes stack in the real world.

2.4 PROVISIONING AND SCHEDULING OF SECURITY SERVICES IN KUBERNETES

Further going with the security aspects of Kubernetes, Sava Stanišić, Milan Vesković, Olga Ristić (2025) they listed out the potential security vulnerabilities in Kubernetes environments like kubelet Api misconfiguration, insecure network policies, unverified container images, a crucial one for the runtime tools to defend these threats. Their work is primarily diagnostic and recommending some of the mitigation strategies like regular scanning and updates of images, avoiding use of default service account, encryption of secrets etc. our research directly addresses these threats by taking up the vulnerabilities they mentioned such as container escape and privilege escalation and using them as base for the simulated attack. It measures the performance of these tools in AKS and EKS clusters.

In a 2022 paper by Doriguzzi-Corin et al., the authors identified a new application-conscious scheduler (PESS), which is intelligent to provision security service chains in Kubernetes to achieve both QoS and security goals, reflecting the drawbacks of scheduling in Kubernetes. Their work was effective to optimise the initial location of security functions, but only considered the provisioning stage and did not consider the production-ready security tools. My research is focussed on doing comparative analysis of these Kubernetes runtime security tools in both AWS and Azure real world cloud environments.

2.5 Summary

Author and year	Core focus	Key contribution	Identified Gap	Research's Response
Kapetanidou (2025)	Use of security scanning tools	Compared static & workload scanning tools for misconfigurations.	Lacks of runtime protection	Focus to post-deployment runtime security in the cloud platforms
Mane and Mahajan (2022)	Analysis of misconfiguration	Pre-runtime config checks by comparing with CIS benchmarks.	Runtime is largely unaddressed.	Taking forward from where they left, implanting runtime test
Rahman et al (2023)	Deals with empirical misconfiguration	Detected familiar security misconfigurations in manifests.	Admitted lack of runtime security checks.	Use of runtime tools to detect attacks by surpassing these misconfigurations.
Nerella and puvuuda (2024)	Focus on standard enterprise Security Practices	Comprehensive prevention controls in 8 security domains.	Mostly deals with pre-deployment focus; no deep dive into runtime attacks.	Gives the missing runtime attack performance data.
Shamim et al (2020)	Use of standard security best Practices	Formulated 11 Kubernetes security best use case	Identifies requirements but doesn't evaluate those technical tools.	Tests the tools that can operationalize these best practices during runtime.
Malik and van vugt (2023)	Analysis of runtime tools	Comparative analysis of Falco, KubeArmor, Tracee, and Tetragon on a hardware testbed.	Lab test only cloud performance unknown	Implement their methodology to AWS EKS & Azure AKS for cloud performance and validation.
Gawk et al (2023)	Use of Custom eBPF Tool (KRSIE)	Developed a custom runtime security tool using eBPF.	Not tested on cloud providers.	Tests established, community-adopted tools in public clouds.
Hao et al (2024)	Container Escape Detection	Proposed CPCED, a CNI plugin for real-time container escape detection.	Academic prototype; lacks maturity and integration.	Evaluates mature, open-source tools addressing similar threats.
Kalhofer et al (2025)	Used a deception Framework called (Koney)	Tetragon/Istio for advanced identification and orchestration.	Didn't evaluate performance overhead tools.	Performance metrics their framework depends on.

Kazenas & Ponomareva (2023)	Survey of qualitative tools	Compared commercial tools and open-source tools by its features.	There is no practical implementation	Noted missing empirical and performance-based data from real-cloud testing.
Bagnato (2025)	Just a review of challenges	Found runtime threats and mentioned eBPF as a solution.	Lacks of practical evaluation.	Implements the glided direction with experimental validation.
Ashraf et al. (2025)	Based on Policy Driven Security	Implement Istio & OPA for mTLS and access control.	Didn't address that runtime threats which bypass these policies.	configured runtime tools as a second defense layer against in pod threats.
Alboqmi et al (2023)	Runtime Trust on Service Mesh	Pitched Runtime Trust Evaluator for service-to-service communications.	Focused only on network layer, but ignored container-level security.	Evaluates container-level runtime tools for complementary protection.
Kermabon-Bobinne et al (2025)	Using Policy Performance (PerfSPEC)	Addressed OPA/Gatekeeper overhead by proactive profiling.	But not evaluated in cloud contexts (EKS/AKS).	Gives cloud-contextualized performance data for runtime security tools.
Stanisic et al (2025)	Vulnerability Catalog	Cited potential Kubernetes loopholes and mitigation strategies	Did only diagnostic analysis but no tool evaluation.	Uses their listed vulnerabilities for example container escape for simulated attacks.
Doriguzzi - corin et al. (2022)	Used security aware scheduler	Pitched PESS scheduler for provisioning secure service chains.	Focused on scheduling only didn't evaluate the deployed tools	Interlinked runtime tool evaluation within deployed security service chains.

2.6 Critical Analysis

The literature that is already present offers a good theoretical and laboratory-based background of Kubernetes runtime security. Nevertheless, the important discrepancy between the controlled experimental findings and the reality of the practical applications of production cloud environments is found to be constant and considerable.

Another resource by van Vugt and Malik (2023) are fundamental standard in this approach. Their comparative analysis of Falco, Kube Armour, and Tetragon was the first attempt to quantify such performance trade-offs as the strong-policy enforcement of Kube Armour at the expense of a larger CPU/memory utilisation compared to tools built on eBPF like Falco and Tetragon. Nevertheless, the main weakness of their study, and the feature that unites the vast majority of other similar studies (e.g., Gwak et al., 2023; Hao et al., n.d.; Doriguzzi-Corin et al., 2022), is the restriction to controlled hardware testbeds, such as Grid'5000 or personal servers. These platforms do have the virtualized infrastructure, cloud-native networking, and managed service integrations of platforms such as AWS EKS and Azure AKS. As a result, little is known and unproven of the performance and detection accuracy or resource overhead of such tools within the environments that are most in need of them.

Other researchers include Kazenas and Ponomareva (2023) who provide useful qualitative, feature-based comparisons, but do not present empirical implementation data. On the other hand, a prototype such as CPCED (Hao et al.) or KRSIE (Gwak et al.) have some conceptual viability, but there is a significant aspect of integration maturity and practicability with respect to the more established tools based on open-source.

Our study methodology is specifically aimed at overcoming such shortcomings. Comparative analysis of the same set of open-source tools (Falco, Kube Armour, Tetragon) in live, managed Kubernetes services (EKS and AKS) helps close the gap that is so acute between benchmarking in the laboratory and the practise of the cloud-native. By implementing these tools in the cloud platform and how it is accomplished in realistic conditions, our methodology goes beyond feature lists and controlled tests and allows us to do so. It also involves the quantification of the effect of cloud-specific variables on detection latency, resources utilisation, and operational overhead, hence offers the contextual and practical evidence that is lacking in the literature.

Limitations of the Field and the approach in General

Though this research fills an important gap, it is not completely free of limitations, which also reflect broader challenges in the field

Cloud Providers Scope: The research will be constrained to two providers, AWS and Azure and the research results might not be directly applicable to other providers such as Google GKE, Digital Ocean.

Simplified Attack Models: Simulated attacks are representative but need not reflect the much complex multi-stage, advanced persistent threats of the real world.

Scale and Traffic: The size of the test microservices application and the cluster is smaller than that of a large enterprise, therefore, the performance under using large number of microservice with real use case is an aspect of research to be carried out in the future.

Temporal Validity: Kubernetes and security tools have a very fast pace of development, so versions specific to the performance can vary over time.

Regardless of these constraints, this study is a necessary step towards legitimising Kubernetes security tools and provides an informed tooling choice with actual data that DevOps teams can make in their cloud environment.

3 RESEARCH METHODOLOGY

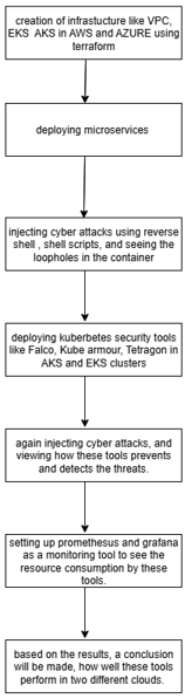


FIGURE 1: FLOW DIAGRAM OF THE METHODOLOGY

3.1 Infrastructure Creation

The first stage of methodology starts with resource creation in respective cloud platforms. We need to configure a private network in both cloud platforms like VPC/VNET in AWS and Azure respectively, in that we need to configure subnets to place our cluster node groups, which is basically a collection virtual machines, which host our test microservice like nginx and security tools which we going to deploy. The Kubernetes cluster can be created in number of ways here we use YAML file and Azure CLI to create the cluster in the respective clouds.

3.2 Microservices Deployment

After creating the necessary resources in the cloud, now it’s time to deploy test microservice like nginx in to the both Kubernetes cluster, to make that identical microservice were deployed in both the cluster, so that the baseline test scenario would be same in both cloud platforms. This microservice will act as a base to execute shell commands like reverse shell, privilege escalation, etc on it. The application will be deployed using a Kubernetes manifest file in both the cloud platforms.

3.3 Baseline Attack Injection

In order to test the deployed application against shell attack, we will be using set of shell commands like file tampering, privilege escalation, sensitive files, reverse shell entering into the container by executing kubectl commands, obviously these tests will get passed, because there no particular policy enforced at the moment detect or block all the events. The idea is to see how vulnerable the containers are in the runtime and without protection how it is get exploited.

3.4 Security Tools Deployment with Custom Policies:

Following that, open-source Kubernetes runtime security tools like Falco, Tetragon, Kube Armor are deployed in the cluster using respective helm charts in particular name space of the cluster. Now for the respective tools like Kube Armor, have to write custom policies which include which microservice it needs to monitor and what are the actions it has to block with respect to the microservice it has tied up. Similarly, tetragon and Falco will also have default rules and will custom rules to detect the shell attacks in the application and it will be stored as logs in the respective security tool container for future reference.

3.5 Post Deployment attack and detection assessment

As the security tools are deployed and configured with the policies, now it's time to run the shell command attacks against deployed microservice. As per the agenda of default rules of security tools, and custom polices the Falco and Tetragon will detect the attack and Kube armour will block the action as per the written policy. All these logs can be seen in the respective security pod logs. On the whole, need to analyse how well these security tools performs with detection and preventing the attacks with less latency.

3.6 Performance Monitoring Arrangement

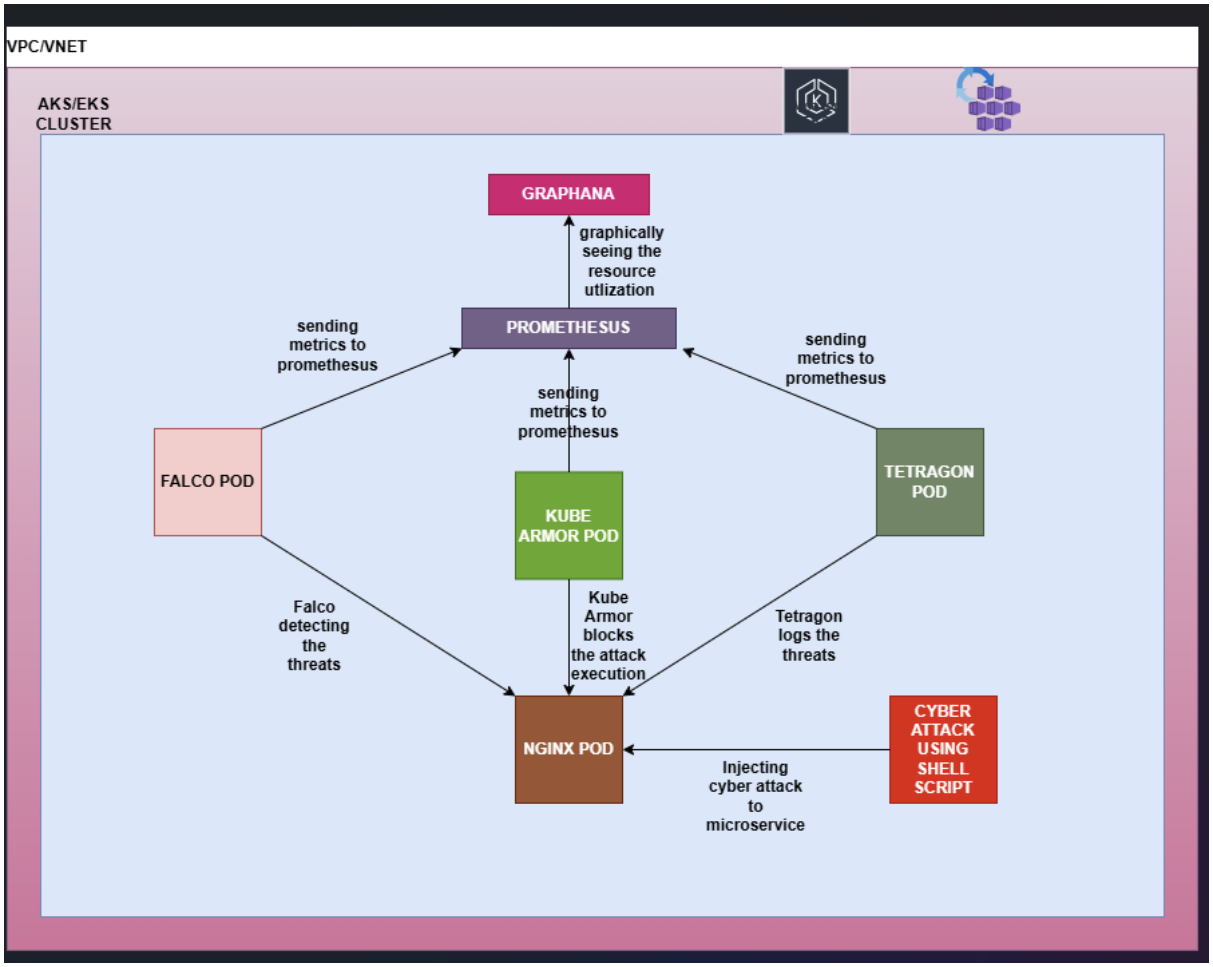
A complete performance monitoring of these security tools running inside a pod will be monitored by Prometheus and Grafana stack. Prometheus collects the metrics of the pod running in the cluster, and Grafana provides a dash board to view the resource utilization like CPU and Memory utilization of the respective tools in both cloud platforms.

3.7 Comparison and Conclusion

The last phase consists of analysis of all the data gathered from the two AKS and EKS cloud platforms, how well each tool behaved in detecting the altering about the attacks got executed in the container of the application. And comparing the resource usage in each cloud, like whether it used less or more resource in cloud when to compared to the other cloud. And at last, comparing the result with the base paper, to see whether these results align with the base paper or it gives different result altogether.

4 DESIGN SPECIFICATION

The architecture diagram consist of what are the tools which is required to carry out this experiment, for that which cloud platforms are used.



4.1 Cloud Platforms

To strive the implementation in AWS and Azure cloud, the first step is creating a separate isolated virtual network in both the cloud, i.e. VPC and VNET respectively. Following that subnets have to be configured to place the node groups which is primarily responsible for hosting the micro services and security tools.

4.2 Security Tools

Falco: using the kernel level monitoring it acts as cloud native runtime security analyses used in detecting the suspicious behaviour Kubernetes cluster. It continuously monitors the system call and container functions to detect abnormal activities like privilege escalation, file tampering attempt etc are made in running microservices.

Kube Armor: As a runtime security enforcement tool, it protects the Kubernetes cluster using kernel primitives like eBPF, App Armor, and BPF-LSM. Using default and custom Kube Armor policies we can block activities which is highly sensitive in a running container. It provides real time alerts and blocking of the incidents.

Tetragon: It is an eBPF based security observability and runtime protection tool, gives deep understanding of the process, network, files activity at the Kubernetes cluster level. It also has the ability to block certain incidents by writing and applying custom policies. It is helpful enhancing cluster security and compliance

4.3 Cluster Configuration

In both the cloud platforms, cluster creation is done using the respective CLI'S and node groups are created with different specifications, but ubuntu is chosen as OS for the both the clusters. Once the cluster is up and running, a sample nginx application is deployed in the cluster, which will be the target for our test case scenarios during shell script execution

4.4 Monitoring services

To ensure the overall monitoring of the cluster, Prometheus is used to read the scrape of the endpoints, which is critical in ensuring the overall cluster performance and health and Grafana dashboard is used to graphically view the CPU and Memory utilization of the pods running inside the cluster.

5 IMPLEMENTATION

This section deals with detailed steps of how the research is going to get implemented in a step-by-step approach by covering the technical setup, deploying a sample server application, and experimental process which is going to be used to evaluate Kubernetes security tools in AWS EKS and Azure AKS environments.

5.1 Infrastructure provisioning in cloud

A separate account cloud environment setup has been created in both AWS and Azure. Installed Command line interface, (CLI) for both AWS and Azure. Created respective IAM roles with appropriate permission for terraform operations to create VPC/VNET, EKS and AKS cluster in the respective clouds.

5.2 Cloud CLI Installation and Execution

Using respective AWS and Azure CLI created EKS and AKS cluster in AWS and Azure, installed kubectl a command line interface to interact with Kubernetes cluster respectively. Using terraform init and terraform apply command created the Kubernetes cluster in both AWS and Azure.

5.3 Deployment of microservices and security tools

Installed helm for deploying Kubernetes security tools like Falco, tetragon, Kube armour using helm charts and deployed microservices using CLI commands. Using CLI tool kubectl verified all the pods and containers are up and running.

5.4 Monitoring stack implementation

In the same EKS and AKS cluster deployed Prometheus and Grafana as a microservice, to monitor the traffic of the application, and how much the CPU and memory utilization have taken place for the respective security tool pod when it is functioning for preventing/detecting the threats.

5.5 Simulating cyber attacks

Two or three attack scenarios were implemented in this analysing phase, they are reverse shell attack, where it tries to maintain constant connection to the compromised container from the attacker's machine. Uses this compromised container as a foothold to enter into another container as well.

Next one is privilege escalation, where it tries to gain higher privileges than originally assigned, tries to gain root access of the container, next is the crypto jacking where it tries unauthorised use of computing resource.

5.6 Execution protocol

Initially the cyber-attack will be executed without deploying the security tools, later the attack reinjected by deploying the security tools and logged all the attack parameters and timestamp.

5.7 Shell scripts execution

The idea is to test the sample nginx microservice application against multiple cyber-attack scenarios like, privilege escalation, container escape, reverse shell, file tampering etc. To accomplish this task, shell script was used for all the attack scenarios against target attack pod nginx.

5.8 Response of security tool

Once the attack got generated in targeted nginx pod, the already deployed security pod like Falco triggers alert, that some security breach has occurred and it will be stored in the logs. Similarly based on the Kube Armor default and custom policy which is applied as rule in the cluster with respect to the target pod, will block the attempt like file tampering, reverse shell execution and that logs will be captured in the pod as well. Similarly, tetragon also identifies the violation and stores the logs for later analysis.

5.9 Detection metrics and resource utilization tracking

By using kubectl top nodes command where can find out which security tools container in a top consumes more resource utilization in the cluster. Detection of attack time is found by finding the difference between initiation of attack and alert generation. Metrics like CPU usage and memory consumption is collected in milli cores and megabyte respectively. It will be measured in both AWS EKS and Azure AKS Kubernetes clusters, and will find out which tool performs better in which cloud in the prevention and detection at the cost of resource utilization and cost optimization. It is also visually seen using Grafana dashboard.

6 EVALUATION

6.1 Kube Armor block result in AKS and EKS

The command is kubectl exec -it name of the pod then followed by respective execution

Scenario	Command executed	Policy triggered	Kube Armor action	Result	latency
Sensitive file access	cat /etc/shadow	Block sensitive file access	Block	Permission denied	0.12 secs
Shell attempt	touch /etc/blocked-file	Block shell attempt	Block	permission denied	0.16 secs
File tampering	sh -c "ls /etc"	Block file tampering	Block	Permission denied	0.14 secs

Latency = log time(alert) – log time(event). Here the latency is taken and measured from single runs of the each execution.

6.2 Falco detection result in AKS and EKS

Scenario	Command executed	Message in Falco detection	Action/result	Latency
Sensitive file access	cat /etc/shadow	Sensitive file accessed by unauthorized person	Warning/ detected	0.15 secs
File tampering	touch /etc/malicious-file	File tampering detection, unauthorized access	Warning/detected	0.10 secs
Shell spawned	bash -c “echo test”	Shell attempt is spawned inside the container	Warning/detected	0.12 secs

6.3 Tetragon detection result in AKS and EKS

Scenario	Command executed	Message in tetragon log detection	Action/result	Latency
Sensitive file access	cat /etc/passwd	Sensitive file accessed with policy name	Logged and detected	0.0004 secs
File tampering	Echo test > /etc/test file	File tampering with respective policy name	Logged and detected	0.0008 secs
Shell execution	bash -c whoami	Shell attempt with respective policy name	Logged and detected	0.0017 secs

6.4 Metrics of the Tools in AKS Cluster

Tools	Avg CPU consumption	Peak CPU consumption	Memory utilisation	Highlights
Kube Armor	1.5 – 2 %	2 %	330 – 350 mib	Light weight
Falco	2.2 – 2.6 %	2.8 %	420 mib	Higher CPU utilization
Tetragon	1 – 1.2 %	1.1 %	2.2 Gb	High memory usage and low cpu consumption

6.5 Metrics of the Tools in EKS Cluster

Tools	Total CPU consumption	Total Memory utilization	Highlights
Kube Armor	1.7 – 1.8 %	624 mib	High memory usage with relatively moderate CPU utilization
Falco	1.4 – 1.55 %	145 mib	Mid-level CPU usage whereas low memory consumption
Tetragon	0.73 %	400 mib	Lowest CPU usage and moderate memory use

6.6 Cross Cluster Comparison

Tool	CPU (AKS)	CPU (EKS)	Memory (AKS)	Memory (EKS)
Kube Armor	1.5 – 2 %	1.7 – 1.8 %	350 Mib	624 Mib
Falco	2.2 – 2.8 %	1.4 – 1.55 %	420 Mib	145 Mib
Tetragon	1 %	0.73 %	2.2 Gib	400 Mib

This section deals with performance and resource utilisation of the security tools deployed in the elastic Kubernetes service (EKS) and azure Kubernetes service (AKS). Though the performance of these tools is similar in both the clusters, there resource utilisation in the cluster varies according to their node types and kernel optimization.

6.7 Key observation

Kube Armor: Here CPU utilization is almost same in both the cluster, whereas memory consumption is almost double the size of AKS in EKS due to the deployment two BPF container pods by default.

Falco: In this when compared to the two clouds, the CPU utilisation drops significantly on EKS, and memory consumption also becomes the least when seen with other tools.

Tetragon: This tool consumed less CPU resource in both the cluster, much efficient when compared to other two tools in the clusters. But memory usage varies drastically, in EKS it used 400 mb, where as in AKS cluster it used very high memory of 2.2 Gb.

So, the overall evaluation would be tetragon most efficient in terms of CPU utilization in both the cluster, and Falco best for memory efficiency with respect to EKS, and more consistent is Kube Armor in both the cluster, whereas tetragon is highly sensitive in terms of memory use when compared with each cluster.

6.8 Comparison with Base Paper

CPU usage comparison

Tool	Paper (private host)	Paper (Grid 5000)	AKS	EKS
Kube Armor	16.02 %	6.55 %	1.5 – 2 %	1.7 – 1.8 %
Falco	7.3 %	2 %	2.2 – 2.8 %	1.4 – 1.55 %
Tetragon	7.36 %	1.75 %	1 %	0.73 %

Memory usage comparison

Tool	Paper (private host)	Paper (Grid 5000)	AKS	EKS
Kube Armor	13 Gb	19 Gb	350 Mib	624 Mib
Falco	11.5 Gb	17.8 Gb	420 Mib	145 Mib
Tetragon	11.55 Gb	17.25 Gb	2.2 Gib	400 Mib

6.9 Analysis with base paper

Microservice and cluster difference

The base paper deals with almost fifteen microservices which continuously interacting, whereas in this research we are just using one nginx microservice, as the agenda here is to estimate how these open source microservice security tools performs and its resource consumption in two different c loud platform only, so used a light weight microservice, which in turn leads fewer syscalls, fewer security events and lower eBPF activity.

Optimization of Cloud

In general, managed cloud platform Kubernetes services like AKS and EKS operates at low-level kernel, cgroup and runtime optimization which in turn leads to less resource utilization of these security tools like Falco, Kube Armor, Tetragon.

Cgroups reduces the resource overhead by throttling CPU efficiently, accounting the memory in better way, and distributes the memory smoothly which avoid any unwanted bursts. Another important factor is managed Kubernetes use hardened and optimized kernels, which is specifically designed for microservice environment. Some of common optimization include reduced kernel modules, optimized syscall dispatch paths and pre-enabled eBPF features. The next important factor is because of the eBPF JIT, as the three depend on eBPF for syscall/interception, without JIT eBPF bytecode is interpreted very slowly. So, with the use of JIT, it is translated into native machine code and got executed at the kernel level, which in turn reduces the time.

7 CONCLUSION

In conclusion the comparative analysis of the opensource security tools like Falco, Kube Armor, Tetragon, in managed Kubernetes cloud platforms like AKS and EKS yielded different results with respect to the resource utilization in different clouds, if one is performing in one aspect means, in other aspect it lags behind the other tool. But performance wise each behaved in a much similar way with respect to prevention and detection of each attack in both the cloud platforms. When compared to the base paper, it gives a different dimension altogether, though we used less microservice when compared to the controlled lab environment experiment, but the cloud managed Kubernetes service helps in less resource utilization of the security tool with its prebuilt kernel optimization, cgroups isolation, and enhanced eBPF JIT performance, which in turn reduces the syscall processing overhead and improves the enforcement latency.

8 FUTURE WORK

Considering the limited bandwidth of cost associated with the cloud platforms, the experiment was conducted using a limited microservice. So, in future this research has a technical scope of incorporating more microservices in to the cluster, like a full fudged application with micro architecture setup and deploying these security tools alongside would make the enhancement like a real-world testing. Also, these security tools logs can be stored in AWS S3 and Azure blob storage for future references, so that no security breach can be left unnoticed. Also, it has the potential to spread these experiments in multi cloud like GCP, Digital Ocean.

REFERENCES

Kermabon-Bobinnec, H., Bagheri, S., GholipourChoubeh, M., Majumdar, S., Jarraya, Y., Wang, L. and Pourzandi, M., 2024. PerfSPEC: Performance Profiling-Based Proactive Security Policy Enforcement for Containers. *IEEE Transactions on Dependable and Secure Computing*, *22*(2), pp.919-938.

Shamim, M.S.I., Bhuiyan, F.A. and Rahman, A., 2020. Xi commandments of kubernetes security: A systematization of knowledge related to kubernetes security practices. *2020 IEEE Secure Development (SecDev)*, pp.58-64.

Alboqmi, R., Jahan, S. and Gamble, R.F., 2023, August. A runtime trust evaluation mechanism in the service mesh architecture. In *2023 10th International Conference on Future Internet of Things and Cloud (FiCloud)* (pp. 242-249). IEEE.

Hao, Y., Zhang, X. and Wang, D., 2024, December. CPCED: a container escape detection system based on CNI plugin. In *2024 IEEE 23rd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)* (pp. 242-253). IEEE.

Kahlhofer, M., Golinelli, M. and Rass, S., 2025. Koney: A Cyber Deception Orchestration Framework for Kubernetes. *arXiv preprint arXiv:2504.02431*.

Mahajan, V.B. and Mane, S.B., 2022, June. Detection, analysis and countermeasures for container-based misconfiguration using docker and kubernetes. In *2022 International Conference on Computing, Communication, Security and Intelligent Systems (IC3SIS)* (pp. 1-6). IEEE.

Ashraf, B., Kumar, S. and Jawad, N., 2025, September. A Policy-Driven Approach for Securing Microservices Workflow in Kubernetes Cluster. In *2025 IEEE International Conference on Cluster Computing Workshops (CLUSTER Workshops)* (pp. 1-2). IEEE.

Doriguzzi-Corin, R., Cretti, S., Catena, T., Magnani, S. and Siracusa, D., 2022, June. Towards Application-Aware Provisioning of Security Services with Kubernetes. In *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)* (pp. 284-286). IEEE.

Bagnato, R., Notarianni, L., Sabatini, A. and Vollero, L., 2025, August. Security Challenges and Solutions in Containerized Environments: A Comprehensive Review. In *2025 IEEE International Conference on Cyber Security and Resilience (CSR)* (pp. 720-724). IEEE.

Nerella, H. and Puvvada, P.S., 2024, November. Reinforcing Kubernetes Security: A Gap Analysis and Modern Security Practices for Enterprise. In *2024 First International Conference on Data, Computation and Communication (ICDCC)* (pp. 780-785). IEEE.

Kapetanidou, I.A., Nizamis, A. and Votis, K., 2025, March. An evaluation of commonly used Kubernetes security scanning tools. In *Proceedings of the 2nd International Workshop on MetaOS for the Cloud-Edge-IoT Continuum* (pp. 20-25).

van Vugt, T.M. and Malik, T., 2023, November. A Practical Analysis of Open-Source Security Tools in Microservice Kubernetes Environments. In *2023 Cyber Research Conference-Ireland (Cyber-RCI)* (pp. 1-8). IEEE.

Stanišić, S., Vesković, M., Ristić, O. and Đorđević, B., 2025, March. Security Aspects of Container Orchestration in Kubernetes Environments. In *2025 24th International Symposium INFOTEH-JAHORINA (INFOTEH)* (pp. 1-5). IEEE.

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

Gwak, S., Doan, T.P. and Jung, S., 2023. Container Instrumentation and Enforcement System for Runtime Security of Kubernetes Platform with eBPF. *Intelligent Automation & Soft Computing*, *37*(2).

Presentation link: https://youtu.be/jrR-4THuD3Y