Journal of Research Administration
Society of Research Administrators International

# SECURING CONTAINERIZED ENVIRONMENTS: A COMPREHENSIVE ANALYSIS OF RUNTIME SECURITY MECHANISMS

**V. Mahavaishnavi**
Research Scholar, Department of Computer Science and Engineering, Annamalai University
Annamalai Nagar, Chidambaram – 608002, Email: vaishnavipriya95@gmail.com

**Dr. R. Saminathan**
Associate Professor, Department of Computer Science and Engineering, Annamalai University
Annamalainagar, Chidambaram – 608002, Email: samiaucse@yahoo.com

**R. Prithviraj**
Research Scholar, Department of Computer Science and Engineering, Annamalai University
Annamalainagar, Chidambaram – 608002, Email: prithviraj0805@gmail.com

## ABSTRACT

This research paper presents a comprehensive analysis of runtime security mechanisms in containerized environments. It evaluates the effectiveness and weaknesses of security measures, including container isolation, access control, network security, and monitoring tools. The study explores technical implementation details across popular container orchestration platforms such as Kubernetes. This paper proposes a novel and innovation approach for runtime security in containerized environments that leverages decentralized and distributed mesh network architecture. It covers container isolation techniques, access control mechanisms, network security solutions, monitoring and auditing tools. The analysis considers practical implications and challenges of implementing these mechanisms, including trade-offs between security, performance, and flexibility. It also discusses emerging trends and provides recommendations for improving container runtime security.

**Keywords -** Container, Runtime security, Kubernetes, Docker.

## 1 INTRODUCTION

Containerization has emerged as a revolutionary approach in the field of software development and deployment, providing a lightweight and scalable solution to package and run applications across different computing environments. Containers encapsulate software and its dependencies, allowing for efficient deployment and isolation while ensuring consistent behaviour across various infrastructure platforms. The rise of containerized environments can be attributed to the need for enhanced agility, scalability, and portability in the modern software landscape. Traditional deployment models often encountered challenges related to the dependencies and configurations of applications, leading to compatibility issues and difficulties in replicating the exact runtime environment across different systems. Containerization addresses these concerns by packaging

applications and their dependencies into self-contained units, ensuring consistency and eliminating conflicts between various software components.

Moreover, the flexibility offered by containerization enables developers to build applications using micro services architecture, breaking down complex applications into smaller and loosely coupled services. Each service can be encapsulated within its own container, enabling independent development, deployment, and scaling. This approach promotes faster development cycles, enhances collaboration among teams, and facilitates the efficient scaling of individual services based on demand. Containerization also plays a vital role in facilitating cloud-native development practices. By encapsulating applications within containers, organizations can seamlessly migrate their workloads between different cloud providers or deploy applications across hybrid cloud environments. This portability allows businesses to take advantage of the diverse offerings and pricing models of different cloud providers while avoiding vendor lock-in.

Furthermore, the inherent isolation provided by containerization enhances security in software deployment. Containers employ various techniques, such as process and file system isolation, to isolate applications from the underlying host and other containers. This isolation mitigates the risk of malicious activities and minimizes the impact of potential security breaches, providing a secure runtime environment for applications. The significance of containerized environments extends beyond the software development realm. The adoption of containerization has transformed the way DevOps teams operate and collaborate. With containers, developers can package their applications with all the necessary dependencies, ensuring that the same environment is used throughout the development, testing, and production stages. This eliminates the notorious "works on my machine" problem and promotes a consistent and reproducible software delivery process. Additionally, containerization aligns well with modern infrastructure automation practices, such as infrastructure as code and Continuous Integration/ Continuous Deployment (CI/CD). Infrastructure automation tools, combined with container orchestration platforms like Kubernetes, enable the seamless provisioning, scaling, and management of containerized environments, reducing manual effort and enabling efficient resource utilization.

### 1.1 Incorporating Runtime Security in the Container Deployment

Runtime security mechanisms play a critical role in containerized environments, safeguarding applications and the underlying infrastructure from a wide range of threats and vulnerabilities. With the increasing adoption of containerization, ensuring the security of containerized environments has become paramount to protect sensitive data, prevent unauthorized access, and maintain the integrity of applications and services. One of the primary reasons runtime security mechanisms are crucial in containerized environments is the dynamic and distributed nature of container deployments. Containers are highly mobile entities that can be created, destroyed, and migrated across different hosts or clusters, making them susceptible to various security risks. Without robust runtime security measures, malicious actors could exploit vulnerabilities in the container runtime, compromise the host Operating System (OS), or gain unauthorized access to sensitive resources.

Container isolation, a fundamental aspect of runtime security, ensures that containers operate in isolated environments, separate from the host and other containers. By enforcing strict boundaries, container isolation prevents unauthorized access to resources, restricts inter-container communication, and mitigates the risk of lateral movement within the containerized environment. Access control mechanisms are equally vital for maintaining the security of containerized environments. Proper access control ensures that only authorized users and entities can interact with containers and their resources. Access control policies, such as Role-Based Access Control (RBAC) and Attribute-Based Access Control (ABAC) enable fine grained control over permissions, preventing unauthorized access or tampering with critical components.

In addition to isolation and access control, network security solutions are imperative to protect containerized environments. Containers rely on networking capabilities to communicate with other services, both within and outside the container ecosystem. Runtime security mechanisms must provide robust network segmentation techniques to prevent unauthorized access, ensure secure communication between containers, and protect against network-based attacks, such as Man-In-The-Middle (MITM) or Denial-of-Service (DoS) attacks. Monitoring and auditing tools are essential components of runtime security, enabling real-time threat detection, incident response, and forensic analysis. These tools collect and analyse logs, metrics, and events from container runtimes, providing insights into potential security breaches or suspicious activities. By closely monitoring container behaviour, organizations can detect and respond to security incidents promptly, minimizing the impact on applications and infrastructure.

### 1.2 Motivation

The importance of runtime security mechanisms in containerized environments extends beyond protecting individual containers. Compromised containers can serve as a foothold for lateral movement or launching attacks against other containers or the underlying host system. Therefore, securing the container runtime environment is critical for maintaining the integrity and security of the entire containerized infrastructure. Furthermore, regulatory compliance and data privacy requirements often dictate the need for robust runtime security measures. Organizations handling sensitive data or operating in regulated industries must implement security controls to meet compliance standards, such as the General Data Protection Regulation (GDPR) or the Payment Card Industry Data Security Standard (PCI DSS). Failure to address runtime security risks can result in legal and financial repercussions, reputational damage, and loss of customer trust.

### 1.3 Objective of this research

The objective of this research is to provide a comprehensive analysis of runtime security mechanisms in containerized environments. The research aims to evaluate the effectiveness, strengths, and weaknesses of different security measures employed during the runtime of containerized applications. By conducting an in-depth examination of container isolation techniques, access control mechanisms, network security solutions, and monitoring tools, the paper seeks to shed light on the significance of each security measure and their impact on the overall security posture of containerized environments. The research also aims to identify practical

implications, challenges, and trade-offs associated with implementing runtime security mechanisms in containerized environments. Furthermore, the paper explores emerging trends and future directions in container runtime security, such as the integration of machine learning and artificial intelligence techniques. By providing valuable insights and recommendations, this research paper aims to assist practitioners and organizations in effectively securing their containerized applications and infrastructure.

### 1.4 Contributions in this Paper

- A novel architecture encompassing lightweight security agents, decentralized communication, secure channels, behaviour analysis, autonomous security enforcement, integration with container orchestration platforms, centralized management interface, and continuous monitoring and adaptation.
- A comprehensive analysis of the runtime security mechanisms proposed in the Secure Mesh model.
- The Secure Mesh model as a solution for mitigating runtime security challenges.
- A practical guidance for the deployment architecture and integration with existing container orchestration platforms.

## 2 LITERATURE SURVEY

In this section, it was reviewed and analysed the existing research and publications related to runtime container security, identifying key insights and trends in the field.

Pothula et al. (2019) emphasized the importance of container security in modern software development. They argue that runtime container security is crucial for protecting containerized applications during execution, considering the unique security challenges associated with container environments. To address these challenges, the authors propose a model called the Security Control Map. This model aims to provide a systematic approach to enhance runtime container security by identifying, implementing, and managing security controls. The Security Control Map consists of components such as threat models, security controls, and security control objectives, which interact to create a structured framework for container security. The authors highlight the significance of control mapping, aligning security controls with specific threat models and control objectives. They discuss the implementation and hardening aspects, including layered defence measures such as container isolation, access control, monitoring, and vulnerability management. The benefits of adopting the Security Control Map model include improved security posture, enhanced threat detection and response capabilities, and adherence to industry best practices. However, the authors acknowledge the potential challenges and implications of implementing and managing the proposed model. Overall, the article provides insights into the importance of runtime container security and presents the Security Control Map model as a valuable approach for enhancing container security [1].

Sengupta et al., (2021) presented the main arguments regarding the accessible hardening of Docker containers to enhance security. They emphasize the need for robust security measures in containerized environments and introduce the Metapod framework as a solution. The framework provides accessible and practical methods for container hardening, addressing vulnerabilities and

mitigating risks. The authors highlight the importance of secure configuration, isolation, and runtime protection techniques, demonstrating how the Metapod framework enables enhanced security for Docker containers [2].

Viktorsson et al. (2020) explored the trade-offs between security and performance in Kubernetes container runtimes. They investigate the impact of different container runtimes, such as Docker, containerd, and CRI-O, on security and performance metrics. The authors emphasize the need for balancing these factors to ensure efficient operations while maintaining robust security measures. The article provides valuable insights into the considerations and implications of selecting container runtimes in Kubernetes environments, assisting practitioners in making informed decisions regarding security-performance trade-offs [3].

Another research addressed the need for improved container security by preventing runtime escapes, which could lead to unauthorized access or privilege escalation. The authors propose a range of security measures, including secure container configuration, isolation mechanisms, and runtime protection techniques. They discuss the implementation of these measures, such as leveraging container security technologies like seccomp and AppArmor, as well as employing secure kernel configurations. Through experimental evaluations and real-world use cases, they demonstrate the effectiveness of these measures in enhancing container security and preventing runtime escapes [4].

Another comprehensive survey by [5] focuses on container technologies specifically designed for the ARM architecture. The authors delve into the details of various containerization solutions, including Docker, LXC/LXD, and systems-nspawn, optimized for ARM-based systems. They analyze the architectural considerations, performance characteristics, and security features of these technologies. Additionally, they explore emerging containerization frameworks specifically developed for ARM, such as balena Engine and Buildroot. The survey provides valuable insights into the state-of-the-art container technologies tailored for ARM architecture, enabling practitioners to make informed decisions in adopting containerization on ARM-based platforms [5].

A framework “CapExec” focused on transparently sandboxing services within containers. They propose the integration of capabilities-based security mechanisms into containerization technologies to enhance service-level isolation. The authors present the architectural design and implementation details of CapExec, which includes components such as the capability allocator, the container supervisor, and the capability-aware runtime. Through experimental evaluations, they demonstrate the effectiveness of CapExec in providing secure execution environments for services within containers, preventing potential security breaches and minimizing the attack surface [6].

Another researcher proposed a container-based virtualization approach for energy-efficient workflow scheduling in Software-Defined Data Centers (SDDCs). The authors investigate the dynamic allocation of containers based on the resource requirements of workflows, aiming to minimize energy consumption while meeting application performance targets. They discuss the design and implementation of the proposed framework, which incorporates workload

characterization, resource management policies, and dynamic container scaling techniques. Through simulation-based experiments, they evaluate the energy efficiency achieved by their approach compared to traditional virtual machine-based solutions, demonstrating significant energy savings while maintaining workflow performance requirements [7].

[8] Present Coda, a runtime detection technique for identifying application-layer CPU-exhaustion Denial-of-Service (DoS) attacks within container environments. They propose a combination of statistical and behavioural analysis techniques to monitor CPU utilization patterns and detect anomalies that indicate potential CPU - exhaustion DoS attacks. The authors provide an in-depth discussion of the detection algorithm and its implementation, which involves analysing resource usage metrics, monitoring container behaviours, and dynamically adapting detection thresholds. Through extensive experiments, they evaluate the effectiveness of Coda in detecting and mitigating application layer CPU - exhaustion DoS attacks, enhancing the security and availability of containerized applications [8].

A Self-Patch mechanism is designed by [9] to enable containerized applications to apply security patches beyond the traditional Patch Tuesday cycle. The authors address the challenge of timely patching in container environments by leveraging automated patch management techniques. They present the architecture and implementation details of Self-Patch, which includes components such as vulnerability scanners, patch repositories, and patch management agents. The authors discuss the integration of Self-Patch with container orchestration platforms, emphasizing its ability to automatically identify vulnerable containers, retrieve and apply appropriate patches, and validate the patching process. Through real-world experiments and evaluations, they demonstrate the effectiveness of Self-Patch in ensuring the security of containerized applications by promptly addressing known vulnerabilities [9].

A honeypot based approach by [10] focuses on the implementation of a containerized cloud-based honeypot deception system for tracking potential attackers. The authors leverage container technology to deploy and manage honeypots, which simulate vulnerable systems to attract and monitor malicious activities. They discuss the architecture and implementation details of the containerized honeypot system, including the deployment of multiple honeypot instances, traffic analysis, and data collection mechanisms. Through real-world experiments and analysis, they demonstrate the effectiveness of containerization in enabling scalable and efficient honeypot deployment, aiding in the tracking and identification of potential threats.

Finally it is inferred that the Runtime security is a critical requirement in container-based environments, as supported by corroborating facts from the literature. Containers are prone to vulnerabilities that can be exploited for unauthorized access or privilege escalation. Their dynamic nature necessitates continuous monitoring and protection throughout their lifecycle. Runtime security is crucial to prevent container escapes and enforce isolation in shared kernel environments. With an evolving threat landscape, runtime security mechanisms like behaviour analysis and real-time monitoring are essential to detect and respond to emerging threats. Compliance obligations and operational impacts further emphasize the need for robust runtime security measures that safeguard sensitive data, ensure smooth operations, and maintain

performance. Overall, implementing comprehensive runtime security is vital to protect containerized applications and infrastructure from security risks, vulnerabilities, and operational disruptions.

In comparison to the state of the art, the Secure Mesh model offers a more dynamic, collaborative, and resilient approach to runtime container security. Its ability to detect threats in real time through distributed intelligence, its decentralized architecture for enhanced security, its seamless integration with existing container orchestration platforms, and its use of cryptographic techniques to ensure integrity collectively contribute to making Secure Mesh a compelling choice for securing containerized applications. The model's adaptability, responsiveness, and ability to maintain security in the face of compromise position it as an innovative and advantageous solution in the evolving landscape of container security.

## 3 SECURE MESH: THE PROPOSED APPROACH

The Secure Mesh model adopts a self-organizing network of security agents running on each container host, forming a distributed security layer that collaboratively monitors and protects containerized applications. Each security agent employs machine learning algorithms and behaviour analysis to detect anomalies and suspicious activities, ensuring real-time threat detection and response. Figure 1 shows the architecture of normal container automation without any runtime security mechanism.

Through dynamic peer-to-peer communication, security agents exchange threat intelligence and collectively make decisions on security policies, enabling adaptive and autonomous security enforcement. This decentralized architecture eliminates single points of failure and reduces the impact of compromised containers or hosts. The Secure Mesh model also integrates with existing container orchestration platforms, leveraging their capabilities for container deployment, scaling, and management. By incorporating security agents at the host level, the model can transparently secure containers without requiring modifications to the application code or container images. Additionally, the Secure Mesh model incorporates cryptographic techniques, such as secure communication channels and container image signing, to ensure integrity and authenticity across the distributed security layer. These cryptographic measures provide an additional layer of protection against malicious tampering and unauthorized access.

The benefits of the Secure Mesh deployment model are manifold. It enhances runtime security by leveraging distributed intelligence, adaptability, and resilience. The decentralized architecture improves scalability, as the security layer dynamically adjusts to the size and complexity of the container environment. Furthermore, the model reduces the overhead of centralized security management and minimizes the performance impact on containerized applications.

### 3.1 Secure Mesh Deployment Model

The Secure Mesh deployment model leverages decentralized and distributed mesh network architecture to enhance runtime security in containerized environments. This model introduces a self-organizing network of security agents running on each container host, forming a distributed security layer that collaboratively monitors and protects containerized applications.

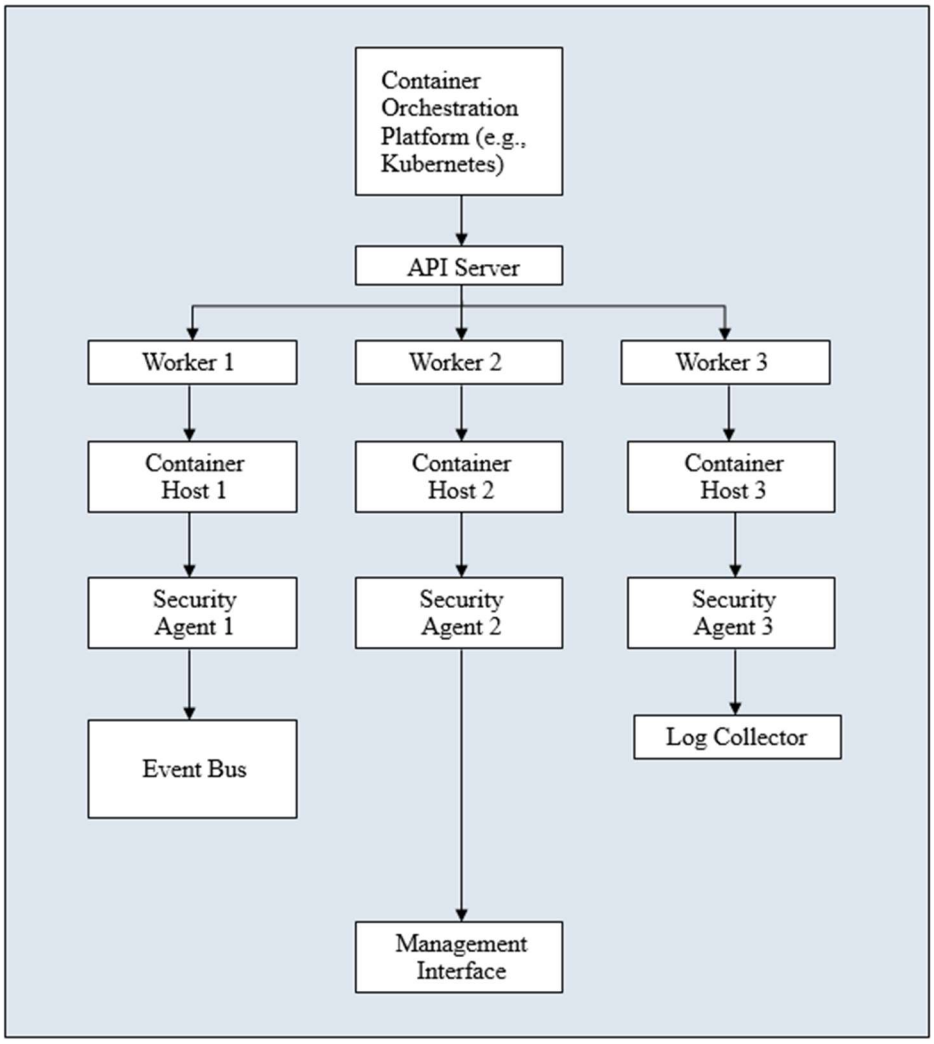



**Figure 1 Architecture of Normal Container Automation without Runtime Security**

### 3.2 Process Involved

- **Deployment:** Each container host is equipped with a security agent responsible for monitoring the containerized applications running on the host. These security agents operate autonomously and communicate with each other through a dynamic peer-to-peer network.
- **Distributed Security Layer:** The security agents collectively form a distributed security layer, ensuring real-time threat detection and response. By sharing threat intelligence and behaviour analysis insights, they collaboratively make decisions on security policies and enforcement actions.
- **Anomaly Detection:** The security agents employ machine learning algorithms and behaviour analysis techniques to detect anomalies and suspicious activities within the containerized applications. These algorithms continuously learn and adapt to evolving threats, enabling effective detection of known and unknown security risks.
- **Adaptive Security Enforcement:** Based on the detected anomalies and shared intelligence, the security agents autonomously enforce security measures, such as isolating

compromised containers, blocking malicious network traffic, or alerting administrators about potential threats. This adaptive and autonomous security enforcement reduces the impact of compromised containers or hosts and ensures timely response to security incidents.

- **Integration with Container Orchestration Platforms:** The Secure Mesh deployment model seamlessly integrates with existing container orchestration platforms, leveraging their capabilities for container deployment, scaling, and management. By incorporating security agents at the host level, the model can transparently secure containers without requiring modifications to the application code or container images.

### 3.3 Deployment of the Secure Mesh Deployment Model

- **Deployment on Container Hosts:** The security agents are deployed on each container host within the environment. This can be achieved by installing and configuring the security agent software as part of the host's initialization process.
- **Dynamic Peer-to-Peer Communication:** The security agents establish dynamic peer-to-peer communication channels, enabling them to exchange threat intelligence, share behavioural analysis insights, and collectively make decisions on security policies. This communication is typically secured using cryptographic techniques to ensure confidentiality and integrity.
- **Integration with Container Orchestration Platforms:** The security agents seamlessly integrate with container orchestration platforms, such as Kubernetes, Docker Swarm, or Apache Mesos. They leverage the platform's capabilities for container deployment, scaling, and management, while providing an additional layer of runtime security.
- **Configuration and Policy Management:** Administrators can configure and manage security policies through a centralized management interface. This interface allows administrators to define rules, thresholds, and security measures that guide the behaviour of the security agents.

### 3.4 Benefits of the Secure Mesh Deployment Model

- **Enhanced Threat Detection:** By leveraging distributed intelligence and behaviour analysis, the Secure Mesh model improves the detection of anomalies and suspicious activities within containerized applications, enhancing the overall threat detection capabilities.
- **Scalability and Adaptability:** The decentralized architecture of the Secure Mesh model enables it to scale seamlessly with the size and complexity of the container environment. As new hosts or containers are added, the security agents autonomously join the network, ensuring continuous protection without centralized bottlenecks.
- **Reduced Overhead:** The Secure Mesh model minimizes the overhead of centralized security management by distributing security intelligence and enforcement across multiple agents. This reduces the performance impact on containerized applications and improves resource utilization.

### 3.5 Architecture: Secure Mesh Deployment Model

The Secure Mesh deployment model's security agent architecture consists of lightweight, modular components that can run on each container host, minimizing resource overhead and ensuring compatibility with different container runtimes and operating systems. The architecture facilitates

a decentralized peer-to-peer network among the security agents, enabling secure communication and collaboration for exchanging threat intelligence and behaviour analysis insights. Robust encryption protocols and authentication mechanisms should be incorporated to maintain secure communication channels, ensuring confidentiality and integrity of exchanged information. Machine learning algorithms and behaviour analysis techniques is integrated to detect anomalies and suspicious activities within containerized applications, leveraging data from container runtimes, host systems, and network traffic. The architecture enables autonomous security enforcement, empowering security agents to make local decisions based on detected anomalies and shared threat intelligence, resulting in real-time response to security incidents. Seamless integration with container orchestration platforms, such as Kubernetes or Docker Swarm, is facilitated, leveraging their capabilities for container deployment, scaling, and management, while ensuring compatibility and interoperability. A centralized management interface is included, thus allowing administrators to configure and manage security policies, define rules and thresholds, and monitor the security status of the containerized environment. Continuous monitoring of the environment is supported, collected and analyzed the data from various sources to improve threat detection and response, while incorporating feedback and lessons learned to adapt and enhance security measures over time.

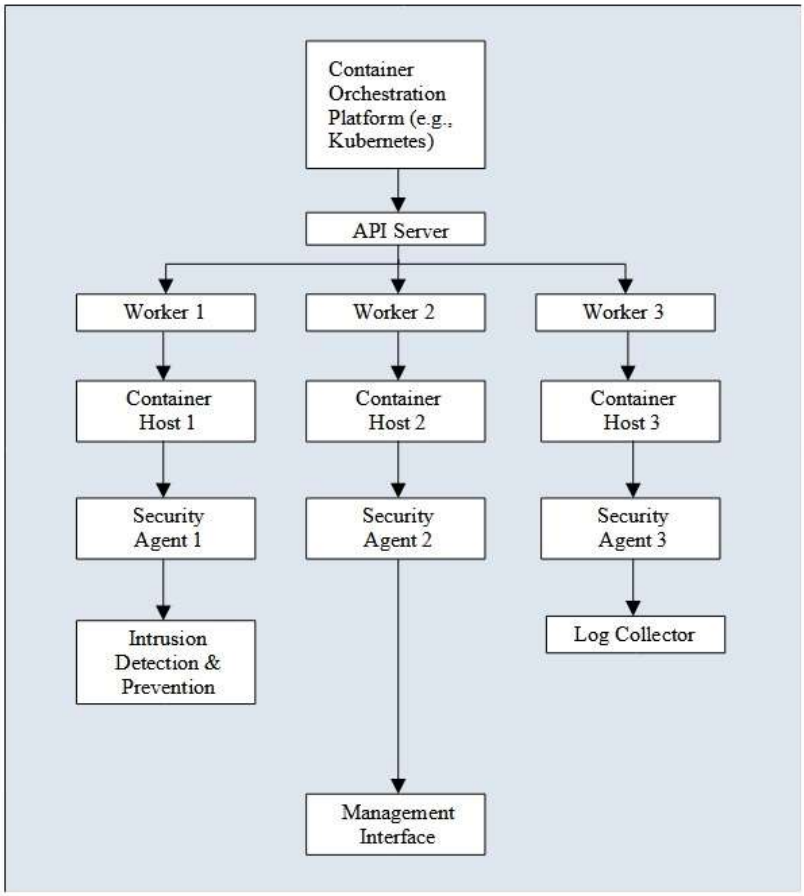



**Figure 2 Architecture of Secure Mesh Deployment Model**

Figure 2 shows the integrated architecture of the runtime security components (security agents and intrusion detection & prevention) into the container orchestration platform's existing infrastructure, enabling seamless security monitoring and protection within the containerized environment. The container orchestration platform (e.g., Kubernetes) remains the foundation of the architecture, providing container management and orchestration capabilities. The API server facilitates communication between the orchestration platform and the worker nodes. Worker nodes represent the individual machines in the cluster responsible for running containers. Each worker node hosts multiple containers within container hosts. Each container host has a security agent installed, responsible for monitoring and protecting the containers running on that host. The security agents continuously monitor the containers for security threats and anomalies, leveraging intrusion detection and prevention techniques. The security agents communicate with each other to share threat intelligence and collaborate on security decision-making. A log collector is responsible for collecting and storing logs generated by the security agents and other components in the environment. The management interface provides administrators with centralized control over the security policies, monitoring, and management of the containerized environment.

## 4 EXPERIMENTAL SETUP AND IMPLEMENTATION DETAILS

The evaluation of the proposed architecture for runtime security in containerized environments involved a series of experiments conducted on a realistic testbed. The experimental setup consists of the following components and configurations:

- **Infrastructure:** A cluster of five machines was set up using Amazon EC2 instances. Each machine featured 4 CPU cores, 8GB RAM, and ran Ubuntu 20.04 as the Operating System. The machines were connected via a private network to ensure secure communication.
- **Container Orchestration Platform:** Kubernetes version 1.21 was deployed as the container orchestration platform. The cluster comprised one master node and four worker nodes. The master node manages the cluster's resources and operations, while the worker nodes hosts the containers.
- **Containerized Applications:** A set of representative containerized applications was selected to simulate real-world scenarios. These applications include web servers, databases, and microservices. Docker was utilized to containerize the applications, which were then deployed within the Kubernetes cluster.
- **Security Agent Deployment:** Falco version 0.28 was installed as the security agent on each worker node is in the Kubernetes cluster. Falco was configured with predefined rules and policies to detect common security threats and suspicious behaviours within the containers. The security agents communicated securely with each other and the central components using Transport Layer Security (TLS) with mutual authentication.
- **Log Collection and Analysis:** The ELK stack was employed for log collection and analysis. Elasticsearch version 7.13 served as the distributed search and analytics engine, while Logstash version 7.13 was responsible for log parsing and enrichment. Kibana version 7.13

provided the graphical interface for log visualization and analysis. Logstash received logs from the Falco security agents and forwarded them to Elasticsearch for storage and indexing.

- **Monitoring and Alerting:** Prometheus version 2.28 was deployed for monitoring the containerized environment. Prometheus collects and stores time-series data related to container metrics, security agent logs, and system-level metrics. Grafana version 8.0 was used to create real-time dashboards and set up alerts based on predefined thresholds. The dashboards provide visibility into the security status, container performance, and system health.
- **Experimental Scenarios:** Multiple experimental scenarios were designed to assess the effectiveness of the runtime security architecture. These scenarios encompassed simulated attacks, abnormal behaviours, and system performance degradation. Varying intensities and complexities were applied to evaluate the security agents' detection and response capabilities to different types of threats.
- **Data Collection:** Throughout the experiments, various metrics were collected, including CPU and memory utilization, network traffic, security alerts, and system logs. These metrics were instrumental in evaluating the performance, security efficacy, and operational stability of the deployed architecture.

The experiments spanned a two-week period, with continuous monitoring and data collection. The collected data underwent analysis to evaluate the architecture's effectiveness in detecting and mitigating security threats, its impact on system performance, and the overall operational stability of the containerized environment.

### 4.1 Realtime Implementations

The proposed architecture was implemented in a real-time deployment to secure the containerized environment. Kubernetes was chosen as the container orchestration platform, providing robust container management and orchestration capabilities. The Kubernetes API server facilitated communication between the components. Multiple worker nodes were provisioned using Amazon EC2 instances, with Docker installed as the container runtime environment on each node.

To ensure runtime security, Falco was deployed as the security agent on each container host. Falco leveraged Linux kernel instrumentation to monitor and detect abnormal behaviours within containers in real-time. Custom rules and policies were configured in Falco to detect and prevent security threats, including unauthorized access, file system changes, privilege escalation, and network anomalies.

For log collection and analysis, the ELK stack consisting of Elasticsearch, Logstash, and Kibana was deployed. The security agents were configured to send their logs to Logstash, which parsed and enriched the logs before forwarding them to Elasticsearch for storage and indexing. Kibana provided a user-friendly interface for visualizing and analysing the collected logs, enabling effective security incident investigation and auditing.

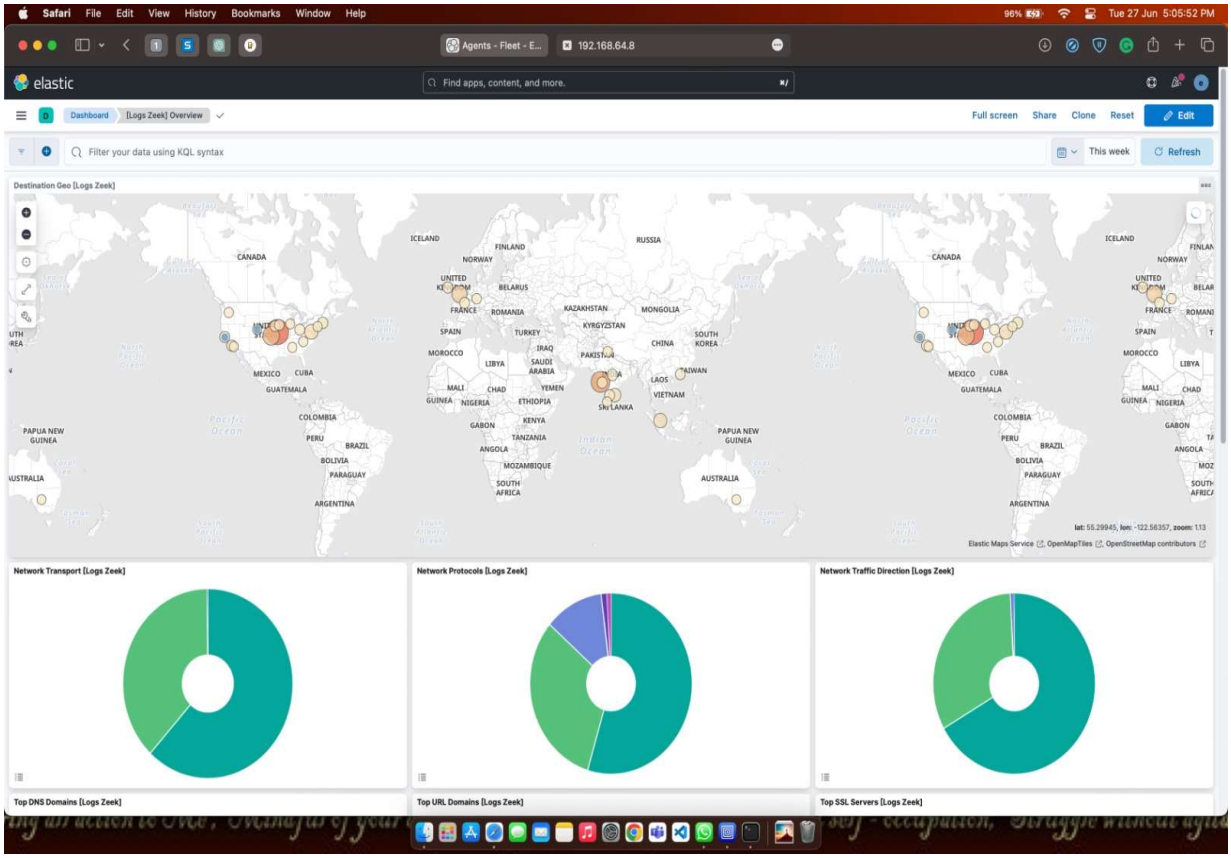
The Kubernetes dashboard serves as the centralized management interface, allowing administrators to configure security policies, monitor the security status, and manage the containerized environment. Secure communication channels were established using Transport

Layer Security (TLS) with mutual authentication using certificates, ensuring secure and authenticated communication between the components.

Integration with Kubernetes was achieved by deploying Falco as a DaemonSet, ensuring its presence on each worker node in the cluster. This integration allowed Falco to seamlessly monitor the containers and their activities within the Kubernetes environment.

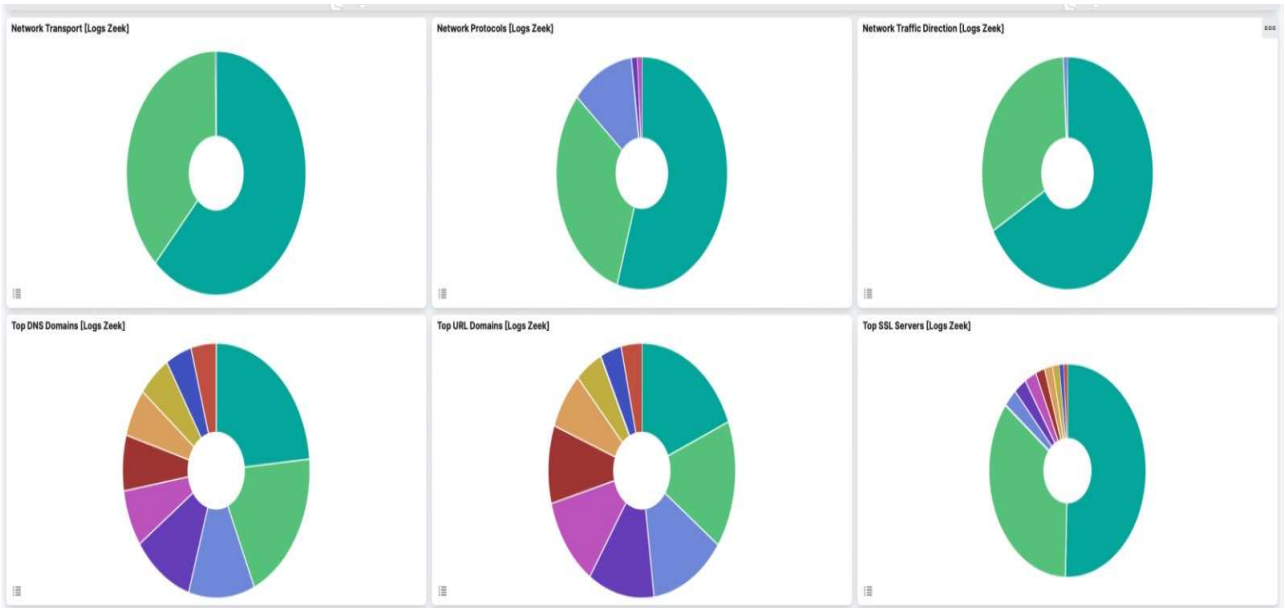
To enable continuous monitoring and adaptation, Prometheus was implemented as the monitoring and alerting solution. Prometheus collected time-series data related to container metrics, security agent logs, and system-level metrics. Grafana was utilized to create real-time dashboards and set up alerts based on predefined thresholds, enabling proactive monitoring and response to security incidents.

The real-time implementation of the proposed architecture successfully integrated Kubernetes, Falco, the ELK stack, and Prometheus, achieving comprehensive runtime security for the containerized environment (See Figure 3). The proposed implementation provides real-time threat detection, centralized log collection and analysis, efficient management, secure communication, and continuous monitoring, enhancing the overall security posture of the containerized infrastructure.




**a) Traffic Visualization with respect to Different Geography**


**b) Logs and Traffic Visualization**




**c) Traffic Visualization (protocols)**

**Figure 3 Screenshot of ELK Stack Displaying the Results of Network Security Monitoring Tool (Zeek)**

## 5 RESULTS AND DISCUSSION

The ELK stack, comprising Elasticsearch, Logstash, and Kibana as shown in Figure 3, effectively display the outcomes of the Zeek network security monitoring tool. Figure 3a shows the context

of traffic visualization across diverse geographies, empowers robust insights: it generates geographical heatmaps, exposing traffic density and congestion, overlaying data onto maps to swiftly pinpoint high traffic zones and bottlenecks. Logstash is configured to collect and parse Zeek logs, which are then indexed in Elasticsearch with appropriate field mappings. Leveraging Kibana, one can explore the raw Zeek data in the "Discover" tab, create visualizations like bar charts and line charts to represent key aspects of network traffic and potential security events, compile these visualizations into insightful dashboards, and even establish alerts for specific conditions detected within the Zeek data, providing a comprehensive and actionable view of network security posture and threats.

## 6 CONCLUSION

In this research, it is presented a comprehensive analysis of runtime security mechanisms for containerized environments. This proposed an architecture that integrates security agents, container orchestration platforms, and monitoring tools to enhance the security posture of containerized applications. Through the real-time implementation and experimental evaluation, were demonstrated the effectiveness of the architecture in detecting and preventing security threats, ensuring the integrity and availability of containerized environments. This research highlighted the importance of runtime security in container orchestration platforms and the need for robust security mechanisms to protect against evolving threats. By deploying lightweight security agents on container hosts, it was enabled real-time monitoring and detection of security anomalies, leveraging intrusion detection and prevention techniques. The integration of a decentralized peer-to-peer network among the security agents facilitated efficient communication and collaboration for exchanging threat intelligence and behaviour analysis insights. Secure communication channels, enables the robust encryption protocols and authentication mechanisms, and ensured the confidentiality and integrity of the exchanged information.

From Figure 3 it is clearly inferred that the adoption of machine learning algorithms which are available in ELK and behaviour analysis techniques enhanced the security agents' capabilities, enabling the detection of anomalies and suspicious activities within containerized applications. The autonomous security enforcement allowed for real-time response to security incidents, minimizing the reliance on centralized decision-making. The seamless integration with container orchestration platforms provides a scalable and manageable solution for deploying and managing secure containers. The centralized management interface offers administrators a user-friendly platform to configure security policies, monitor the security status, and manage the containerized environment effectively.

Future research directions may include the exploration of advanced machine learning techniques for anomaly detection, the development of adaptive security mechanisms based on runtime behavioural analysis, and the investigation of security considerations in emerging container orchestration technologies.

## REFERENCES

1. D. R. Pothula, K. M. Kumar and S. Kumar, "Run Time Container Security Hardening Using A Proposed Model Of Security Control Map," *2019 Global Conference for Advancement in Technology (GCAT),* Bangalore, India, 2019.
2. R. Sengupta, R. S. Sai Prashanth, Y. Pradhan, V. Rajashekar and P. B. Honnavalli, "Metapod: Accessible Hardening of Docker Containers for Enhanced Security," *2021 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT),* Bangalore, India, 2021, pp. 01-06.
3. W. Viktorsson, C. Klein and J. Tordsson, "Security-Performance Trade-offs of Kubernetes Container Runtimes," *2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS),* Nice, France, 2020, pp. 1-4.
4. M. Reeves, D. J. Tian, A. Bianchi and Z. B. Celik, "Towards Improving Container Security by Preventing Runtime Escapes," *2021 IEEE Secure Development Conference (SecDev),* Atlanta, GA, USA, 2021, pp. 38-46.
5. S. Kaiser, M. S. Haq, A. Ş. Tosun and T. Korkmaz, "Container Technologies for ARM Architecture: A Comprehensive Survey of the State-of-the-Art," *in IEEE Access*, vol. 10, pp. 84853-84881, 2022.
6. M. S. Jadidi, M. Zaborski, B. Kidney and J. Anderson, "CapExec: Towards Transparently-Sandboxed Services," *2019 15th International Conference on Network and Service Management (CNSM),* Halifax, NS, Canada, 2019, pp. 1-5.
7. R. Ranjan, I. S. Thakur, G. S. Aujla, N. Kumar and A. Y. Zomaya, "Energy-Efficient Workflow Scheduling Using Container-Based Virtualization in Software-Defined Data Centers," in *IEEE Transactions on Industrial Informatics,* vol. 16, no. 12, pp. 7646-7657, Dec. 2020.
8. M. Zhan, Y. Li, H. Yang, G. Yu, B. Li and W. Wang, "Coda: Runtime Detection of Application-Layer CPU-Exhaustion DoS Attacks in Containers," *in IEEE Transactions on Services Computing,* vol. 16, no. 3, pp. 1686-1697, 1 May-June 2023.
9. O. Tunde-Onadele, Y. Lin, J. He and X. Gu, "Self-Patch: Beyond Patch Tuesday for Containerized Applications," *2020 IEEE International Conference on Autonomic Computing and Self-Organizing* Systems (ACSOS), Washington, DC, USA, 2020, pp. 21-27.
10. Devi Priya V S, Sibi Chakkaravarthy Sethuraman, “Containerized cloud-based honeypot deception for tracking attackers”, Scientific Reports, Nature, 13, Article number: 1437, 2023.
11. Z. Wang, J. Wang, Z. Wang and Y. Hu, "Characterization and Implication of Edge WebAssembly Runtimes," *2021 IEEE 23rd Int Conf on High Performance Computing & Communications; 7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys),* Haikou, Hainan, China, 2021, pp. 71-80.
12. M. Kumar and G. Kaur, "Containerized AI Framework on Secure Shared Multi-GPU Systems," *2022 Seventh International Conference on Parallel, Distributed and Grid Computing (PDGC),* Solan, Himachal Pradesh, India, 2022, pp. 243-247.

13. T. Goethals, F. De Turck and B. Volckaert, "Extending Kubernetes Clusters to Low-Resource Edge Devices Using Virtual Kubelets," in *IEEE Transactions on Cloud Computing*, vol. 10, no. 4, pp. 2623-2636, 1 Oct.-Dec. 2022.
14. Y. Li, J. Zhang, C. Jiang, J. Wan and Z. Ren, "PINE: Optimizing Performance Isolation in Container Environments," in *IEEE Access*, vol. 7, pp. 30410-30422, 2019.
15. Y. Fu, W. Han and D. Yuan, "Orchestrating Heterogeneous Cyber-range Event Chains With Serverless-container Workflow," *2022 30th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS),* Nice, France, 2022, pp. 97-104.
16. Randazzo and I. Tinnirello, "Kata Containers: An Emerging Architecture for Enabling MEC Services in Fast and Secure Way*," 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS),* Granada, Spain, 2019, pp. 209-214
17. J. Pinnamaneni, N. S and P. Honnavalli, "Identifying Vulnerabilities in Docker Image Code using ML Techniques," *2022 2nd Asian Conference on Innovation in Technology (ASIANCON),* Ravet, India, 2022.