Project Horizon

Cloud-Native Application Protection Platform (CNAPP) Implementation Report

Project Title: Project Horizon - Unified Zero Trust Architecture
Document Type: Technical Implementation Report

Executive Summary

The objective of this solution was to establish a **secure, monitored, and policy-driven K3s Kubernetes cluster** by integrating multiple security and observability tools into a cohesive Cloud-Native Application Protection Platform (CNAPP). The implementation focuses on securing the **entire container lifecycle** from infrastructure and image scanning, to admission control, runtime monitoring, and centralized logging using open-source technologies aligned with modern Kubernetes security best practices.

The deployed solution combines **Falco, Kyverno, Cosign, Checkov, Trivy, a private container registry, and the Elastic Stack** to deliver layered protection across build, deploy, and runtime stages:

- **Checkov** performs continuous Infrastructure as Code (IaC) scanning for misconfigurations in Terraform and Helm templates, shifting security left in the development pipeline.
- **Cosign** signs container images to ensure integrity, while **Kyverno** enforces require-image-signature and verify-signed-images policies, preventing the deployment of unsigned or tampered images.
- **Trivy** scans container images for known vulnerabilities prior to deployment, ensuring that only compliant, secure images are admitted.
- **Falco** provides runtime security detection, monitoring container behavior for anomalies such as unauthorized shell access or unexpected API server contact.
- **Elastic Agent** forwards Falco, Kyverno, Checkov, and Trivy logs into the **ELK Stack**, enabling centralized visualization, correlation, and detection engineering.

This architecture enforces **pre-deployment security (Checkov, Trivy, Cosign)**, **admission control (Kyverno)**, and **runtime defense (Falco)** all feeding into a **centralized SIEM (ELK)** for monitoring and investigation. The solution successfully detected and logged key security events such as "Terminal shell in container" and policy violations, verified signature enforcement, and ingested security telemetry for analysis through Kibana dashboards.

The result is a **comprehensive CNAPP foundation** that integrates DevSecOps workflows with runtime protection and centralized analytics. It demonstrates that an open-source toolchain, when properly configured and integrated, can deliver enterprise-grade visibility and control over Kubernetes workloads without proprietary dependencies.

1. Introduction

Requirements

Industry required the deployment of a **cloud-native security and monitoring solution** for its Kubernetes environment that would deliver visibility, policy enforcement, vulnerability management, and runtime threat detection across the **entire container lifecycle**. The solution needed to align with modern **Cloud-Native Application Protection Platform (CNAPP)** principles and address the following key requirements:

- **Infrastructure as Code (IaC) Security:** Detect misconfigurations and policy violations in Terraform and Helm templates early in the development process to reduce downstream risk.
- **Image Assurance:** Enforce strict image signing and verification policies to ensure that only trusted, signed container images are deployed in the cluster.
- **Vulnerability Scanning:** Continuously scan container images for known vulnerabilities prior to deployment to maintain a secure software supply chain.
- **Policy Enforcement:** Apply admission-control policies to block non-compliant workloads and enforce cluster security baselines.
- **Runtime Threat Detection:** Monitor cluster activity at runtime to detect suspicious behaviors such as container breakout attempts, unauthorized access, and anomalous API activity.
- **Centralized Logging & Visibility:** Ingest security logs from all tools into a centralized SIEM for visualization, analysis, and detection engineering.

- **Open-Source & Modular Architecture:** Implement the entire solution using open-source components to maximize flexibility, transparency, and cost-effectiveness.

Solution Implemented

To meet these requirements, a **K3s cluster security and monitoring stack** was designed and deployed using the following integrated components:

- **Checkov – IaC Security Scanning**
  Checkov was installed and configured to scan Terraform and Helm charts for security misconfigurations. This enabled early detection of policy violations in the build pipeline. IaC scanning was successfully performed, identifying insecure configurations as expected.
- **Cosign + Kyverno – Image Signing & Policy Enforcement**
  - Cosign was used to sign container images before deployment.
  - Kyverno was configured with require-image-signature and verify-signed-images policies to ensure only signed images could run in the cluster.
  - Both policies were verified to be in the READY state, and attempts to deploy unsigned or misconfigured images were successfully blocked.
- **Trivy – Vulnerability Scanning**
  Trivy was deployed to scan container images for known vulnerabilities prior to deployment. Vulnerability scanning was integrated into the image pipeline, ensuring that insecure images were flagged before reaching production.
- **Falco – Runtime Threat Detection**
  Falco was installed in the K3s cluster to provide real-time runtime security. It was configured with Falcosidekick to forward alerts to Elasticsearch.
  - Alerts such as “Terminal shell in container” and “Contact K8s API Server from container” were successfully generated and indexed.
- **Private Container Registry**
  A private Docker registry was deployed at 10.43.204.82:5000 to host signed images, ensuring controlled and secure image distribution.
- **Elastic Agent + ELK Stack – Centralized Logging & Monitoring**

- The Elastic Agent was deployed in the cluster to collect logs from Falco, Kyverno, Checkov, and Trivy.
- Logs were successfully ingested into Elasticsearch, creating dedicated index patterns such as falco-*.
- Kibana dashboards were used to visualize runtime alerts, policy violations, IaC misconfigurations, and vulnerability scan results.

Outcome

This integrated solution provides **end-to-end container lifecycle protection**:

- **Pre-Deployment:** IaC misconfiguration detection (Checkov), vulnerability scanning (Trivy), and image signing (Cosign).
- **Admission Control:** Policy enforcement through Kyverno, blocking unauthorized or unsigned images.
- **Runtime:** Behavioral monitoring and anomaly detection through Falco.
- **Centralized Monitoring:** ELK Stack for visualization, log correlation, and future detection rule development.

The implemented CNAPP solution significantly enhances the **security posture of the K3s environment**, establishing a strong foundation for scalable, policy-driven, and monitored Kubernetes operations using open-source technologies.

2. Scope

Current Scope

The current CNAPP implementation focuses on securing and monitoring a **K3s Kubernetes cluster** by integrating **pre-deployment security**, **admission control**, **runtime threat detection**, and **centralized logging**. The scope covers the **entire container lifecycle**, ensuring that workloads are scanned, validated, monitored, and logged using open-source tools.

Key functional areas within the scope include:

- **Infrastructure as Code (IaC) Scanning**
    - Continuous scanning of Terraform and Helm configurations using Checkov to detect security misconfigurations before deployment.

    - Detection of policy violations and insecure configurations during build-time testing.

- **Image Assurance & Policy Enforcement**
    - Signing of container images using Cosign to ensure image authenticity and integrity.
    - Enforcement of Kyverno policies (require-image-signature, verify-signed-images) to block unsigned or tampered images at the admission controller level.
- **Vulnerability Management**
    - Image scanning with Trivy to identify known vulnerabilities (CVEs) prior to pushing or deploying images to the private registry.
    - Integration of scanning into the container build workflow for early visibility and remediation.
- **Runtime Threat Detection**
    - Deployment of Falco for real-time detection of runtime anomalies inside Kubernetes containers and nodes.
    - Verification of alerts for events such as “Terminal shell in container” and “Contact K8s API Server from container”.
- **Private Registry Security**
    - Deployment of a private Docker registry at xx.xx.xx.xx:5000 to securely store and distribute signed container images within the cluster.
- **Centralized Logging & Monitoring**
    - Elastic Agent deployed to forward Falco, Kyverno, Checkov, and Trivy logs to Elasticsearch.
    - Kibana used to visualize runtime events, policy violations, misconfigurations, and vulnerability findings.
    - Logs stored in dedicated indices (e.g., falco-*) for organized search and detection engineering.

This scope demonstrates a **full CNAPP security pipeline**, covering detection and enforcement from build to runtime, and ensuring unified observability through ELK integration.

Constraints

The current CNAPP prototype was designed to prioritize functional coverage and tool integration over enterprise-scale deployment. Key constraints include:

- **Single-Cluster Environment:**
  The solution is deployed on a single K3s cluster without multi-cluster federation or high availability.
- **Manual Pipeline Triggers:**
  IaC scanning and image scanning are currently triggered manually; CI/CD integration is not fully automated.
- **Limited Retention & Scaling:**
  Elasticsearch uses default retention settings; long-term log retention and hot–warm–cold architecture are not yet implemented.
- **Basic Access Controls:**
  Dashboard access relies on basic authentication; advanced RBAC, SSO, and MFA are not configured.
- **Policy Coverage:**
  Kyverno policies focus primarily on image signature verification; broader security baselines (e.g., network policies, RBAC hardening, pod security) are not yet enforced.
- **Alert Correlation:**
  Security events are ingested but not yet correlated across sources (e.g., linking Falco runtime alerts to Trivy scan results or Checkov findings).

Gaps and Future Enhancements

To evolve this prototype toward a production-grade CNAPP platform, several enhancements have been identified:

- **CI/CD Integration:**
  Automate Checkov and Trivy scans within CI pipelines to enforce security checks earlier in the development lifecycle.

- **Extended Policy Enforcement:**
  Expand Kyverno policy coverage to include Pod Security Standards (PSS), RBAC restrictions, and network policy enforcement.
- **Scaling & Retention:**
  Implement Elasticsearch clustering and lifecycle policies for scalable log ingestion and long-term security data retention.
- **Advanced Correlation & Alerting:**
  Introduce cross-source correlation rules to link IaC, image scanning, runtime, and policy violations for improved detection fidelity.
- **Identity & Access Hardening:**
  Enable fine-grained RBAC for Kibana, integrate SSO/MFA, and apply strict access control for registry and cluster components.
- **Continuous Compliance:**
  Extend IaC scanning and policy enforcement to continuously assess compliance against CIS Kubernetes Benchmarks and organizational security standards.

Assumptions

The prototype assumes the following operating conditions:

- The K3s cluster is stable and has network connectivity for log forwarding and image pulling.
- Elasticsearch and Kibana are available and properly configured to receive and index logs from all integrated tools.
- Security scanning tools (Checkov, Trivy) have up-to-date vulnerability databases and policy sets.
- Cosign keys are securely managed and available for image signing and verification.
- Analysts and stakeholders have appropriate access to Kibana dashboards for monitoring and investigation.

This scope clearly defines the **functional coverage**, **limitations**, and **evolution path** for the CNAPP implementation. It establishes a solid foundation for container lifecycle security, while outlining concrete steps to scale toward enterprise maturity.

3. Tools & Solution Overview

The CNAPP implementation integrates multiple **open-source security and observability tools** to provide comprehensive coverage across the **container lifecycle**: from IaC scanning and image signing, to admission control, runtime threat detection, and centralized monitoring. The toolchain was selected to align with modern Kubernetes security practices while maintaining modularity and cost efficiency.

The table below summarizes the core components and their roles:

Component	Purpose	Key Functions
Checkov	IaC security scanning	Detect misconfigurations in Terraform and Helm templates before deployment
Cosign	Image signing	Sign container images to ensure authenticity and integrity
Kyverno	Policy enforcement	Enforce image signature verification and security baselines at admission
Trivy	Vulnerability scanning	Scan container images for known vulnerabilities (CVEs) pre-deployment
Falco	Runtime threat detection	Detect anomalous container behaviors at runtime (e.g., shell in container, API contact)
Falcosidekick	Alert forwarding	Forward Falco runtime events to Elasticsearch for centralized analysis
Private Docker Registry	Image storage	Secure, controlled registry for signed container images
Elastic Agent	Log forwarding	Collect Falco, Kyverno, Checkov, and Trivy logs and send to ELK Stack
Elasticsearch / Kibana	Centralized SIEM & visualization	Index, store, and visualize security telemetry; build dashboards and detections

Checkov – IaC Security Scanning

- **Role:** Shifts security left by scanning Terraform and Helm charts during the build phase.
- **Key Features:**
    - Detects policy violations and insecure configurations (e.g., open security groups, weak RBAC, public S3 buckets).

- Provides structured scan outputs for integration with ELK.

- **Outcome:** IaC scans were successfully executed, detecting expected misconfigurations and ensuring early visibility into configuration risks.

Cosign & Kyverno – Image Signing & Policy Enforcement

- **Cosign:**
  - Used to sign container images stored in the private Docker registry.
  - Ensures supply chain integrity by verifying image signatures before deployment.
- **Kyverno:**
  - Applied require-image-signature and verify-signed-images policies.
  - Blocks unsigned or tampered images during admission control.
  - Both policies were verified to be in READY state, confirming proper enforcement.
- **Outcome:** The admission controller reliably rejected non-compliant workloads, ensuring only trusted, signed images could run inside the cluster.

Trivy – Vulnerability Scanning

- **Role:** Identifies known vulnerabilities in container images before they are pushed or deployed.
- **Key Features:**
  - Scans OS packages and application dependencies.
  - Provides structured JSON reports suitable for ingestion into ELK.
- **Outcome:** Trivy successfully detected CVEs in test images, demonstrating its ability to enforce pre-deployment security gates.

Falco & Falcosidekick – Runtime Threat Detection

- **Falco:**

    - Monitors container and node activity in real time, using a ruleset to detect suspicious behaviors.
    - Alerts tested include “Terminal shell in container” and “Contact K8s API Server from container.”
- **Falcosidekick:**
    - Forwards Falco alerts to Elasticsearch for centralized logging.
    - Configured to generate daily indices (e.g., falco-YYYY.MM.DD) for organized storage.
- **Outcome:** Runtime security monitoring was fully functional, and Falco alerts were successfully indexed and visualized in Kibana.

Private Docker Registry

- **Role:** Provides a controlled environment for storing and distributing signed container images.
- **Configuration:**
    - Hosted at 10.43.204.82:5000.
    - Integrated into the signing workflow to ensure only trusted images are pushed and pulled.
- **Outcome:** Successfully tagged, signed, and pushed test images; verified by Kyverno signature enforcement during deployment.

Elastic Agent & ELK Stack – Centralized Monitoring

- **Elastic Agent:**
    - Deployed in the K3s cluster to collect logs from Falco, Kyverno, Checkov, and Trivy.
    - Configured for reliable forwarding to Elasticsearch.
- **Elasticsearch & Kibana:**
    - Elasticsearch serves as the central log repository, storing structured security telemetry.

    - Kibana provides dashboards for visualization of IaC findings, vulnerability reports, Falco runtime alerts, and policy enforcement events.

- **Outcome:** Logs were successfully ingested and visualized, with dedicated index patterns for each data source, enabling detection engineering and investigation.

Integrated Security Workflow

The CNAPP toolchain creates a **layered security pipeline**:

1. **Pre-Deployment** – IaC scanning (Checkov), image signing (Cosign), vulnerability scanning (Trivy).
2. **Admission Control** – Policy enforcement with Kyverno to block unsigned or vulnerable images.
3. **Runtime** – Continuous behavioral monitoring and anomaly detection using Falco.
4. **Centralized Monitoring** – Elastic Agent collects logs, and ELK Stack enables visualization and alerting.

This integrated workflow provides end-to-end coverage across the container lifecycle, significantly enhancing the security posture of the Kubernetes environment.

4. Configurations & Customizations

The CNAPP implementation involved several targeted configurations and integrations to ensure that each security tool operated reliably, produced structured telemetry, and integrated cleanly into the centralized monitoring stack. These configurations spanned across **IaC scanning**, **image signing & policy enforcement**, **vulnerability management**, **runtime detection**, and **logging infrastructure**.

4.1 Checkov – IaC Security Scanning

Custom Configurations:

- Checkov was configured to scan both **Terraform** and **Helm** directories for security misconfigurations.
- Policy packs were updated to include checks for:
    - Unrestricted security groups

    - Publicly exposed storage buckets
    - Weak RBAC roles and bindings
    - Missing encryption settings for cloud resources
- Output was set to **JSON format** to facilitate ingestion into Elasticsearch.

Purpose:
These custom checks align the IaC scanning pipeline with cloud security best practices and allow structured, automated analysis of misconfigurations during development.

4.2 Cosign – Image Signing

Custom Configurations:

- Cosign signing keys were generated and stored securely for use in the image signing pipeline.
- A signing workflow was established to sign images before pushing them to the private registry:
- cosign sign --key cosign.key 10.43.204.82:5000/nginx:test
- Verification steps were incorporated during deployment using Kyverno policies.

Purpose:
These configurations ensure that only **cryptographically signed container images** are permitted within the cluster, mitigating supply chain tampering risks.

4.3 Kyverno – Policy Enforcement

Custom Policies Applied:

- **require-image-signature**
    - Enforces that all deployed images must contain valid Cosign signatures.
- **verify-signed-images**
    - Performs signature verification against the Cosign public key before allowing pod admission.

Additional Customizations:

- Policies were applied at the **cluster-wide level** to cover all namespaces.
- Kyverno policy reports were forwarded to Elasticsearch via the Elastic Agent for centralized monitoring.
- Policy states were verified to be in **READY** status using:
- kubectl get clusterpolicy

Purpose:
These policies provide **admission control** at the Kubernetes API server level, blocking unauthorized or unsigned images early in the deployment process.

4.4 Trivy – Vulnerability Scanning

Custom Configurations:

- Trivy was set to **scan images before deployment**, integrated into the build workflow.
- Configured to output results in JSON for structured ingestion.
- Vulnerability databases were updated to ensure accurate CVE detection:
- trivy image --format json --output report.json 10.43.204.82:5000/nginx:test

Purpose:
This ensures that container images are scanned for known vulnerabilities before reaching production, closing a critical gap in the container supply chain.

4.5 Falco & Falcosidekick – Runtime Detection

Falco Configuration:

- Installed as a DaemonSet to monitor all nodes in the K3s cluster.
- Enabled detection rules for:
    - **Terminal shell in container**
    - **Contact K8s API server from container**
    - **Privilege escalation attempts**
- Log output configured in JSON for compatibility with Falcosidekick.

Falcosidekick Configuration:

- Forwarding configured to Elasticsearch via HTTP:
- FALCOSIDEKICK_ELASTICSEARCH_HOST=http://elasticsearch:9200
- Index naming pattern customized to falco-YYYY.MM.DD for organized storage.

Purpose:

These configurations enable **real-time behavioral monitoring** and ensure Falco alerts are available centrally for investigation and detection engineering.

4.6 Private Docker Registry

Custom Configurations:

- Hosted on 10.43.204.82:5000.
- Configured for **authenticated push/pull** operations.
- Integrated with Cosign signing to ensure only signed images are pushed and deployed.

Purpose:

Provides a **controlled image distribution environment**, reducing the risk of deploying untrusted external images.

4.7 Elastic Agent & ELK Stack

Elastic Agent Configuration:

- Deployed as a Kubernetes DaemonSet to collect logs from Falco, Kyverno, Checkov, and Trivy.
- Configured inputs for:
    - Falco logs via Falcosidekick
    - Kyverno admission and policy logs
    - Checkov IaC scan outputs
    - Trivy vulnerability scan reports

Elasticsearch Configuration:

- Created dedicated index patterns:
    - falco-* for runtime alerts

    - kyverno-* for policy enforcement events
    - checkov-* for IaC misconfigurations
    - trivy-* for image vulnerabilities
- Basic authentication enabled for Kibana dashboards.

Kibana Configuration:

- Dashboards were customized to display:
    - Real-time Falco alerts and severity breakdowns
    - IaC misconfiguration trends by resource type
    - Vulnerability distribution by CVE severity
    - Kyverno policy violation counts and types

Purpose:
These configurations ensure **reliable telemetry collection, structured indexing, and actionable visualization**, creating a central investigation hub for the CNAPP stack.

4.8 Security & Operational Controls

- **Logrotate** was configured for Falco and Elastic Agent logs to prevent uncontrolled disk usage.
- **Cosign keys** stored securely with restricted permissions to prevent tampering.
- **Access Controls:** Kibana access restricted to SOC users with basic auth; future plan includes SSO + RBAC.
- **Telemetry Validation:** Routine checks on Elasticsearch indices ensured ingestion pipelines remained healthy.

Summary

These configurations transformed the individual tools into a **cohesive CNAPP pipeline**. Each stage **build**, **admission**, **runtime**, and **monitoring** was tailored for structured telemetry, security policy enforcement, and centralized visibility. The result is a **robust, defense-in-depth architecture** for Kubernetes workloads.

5. Architecture & Data Flow Diagram

The CNAPP implementation follows a **multi-layered security architecture** designed to protect Kubernetes workloads throughout their lifecycle from **build and deployment** to **runtime monitoring and centralized analysis**. Each component in the toolchain plays a distinct role, and the data flow between them is orchestrated to ensure **real-time visibility**, **policy enforcement**, and **threat detection**.

5.1 High-Level Architecture

The architecture consists of four major security layers:

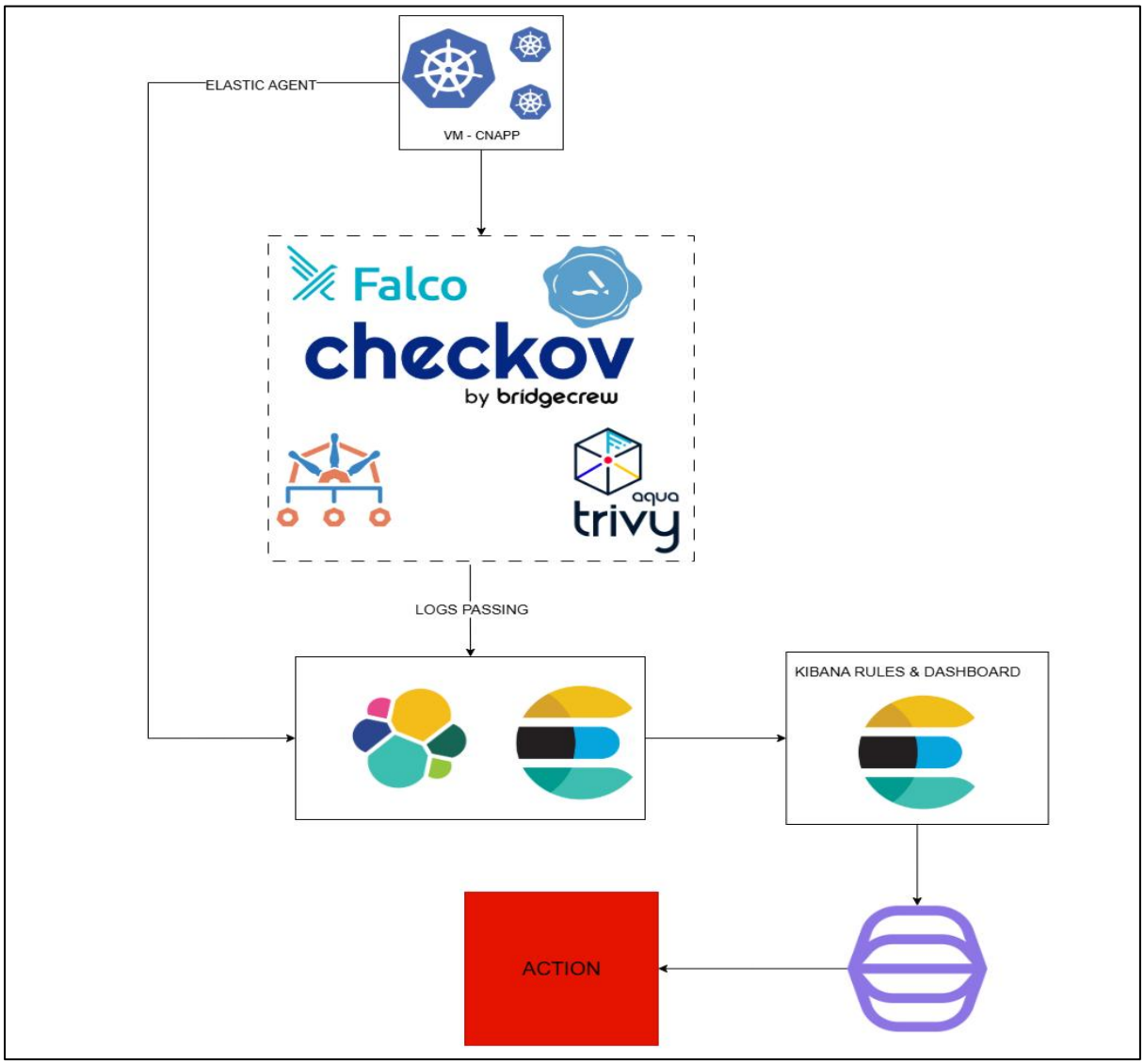
1. **Pre-Deployment Layer**
    - **Tools**: Checkov, Trivy, Cosign
    - **Purpose*:*** Detect misconfigurations, scan images for vulnerabilities, and sign trusted images prior to deployment.
2. **Admission Control Layer**
    - **Tools:** Kyverno + Cosign
    - **Purpose:** Enforce security policies at the Kubernetes API server level, blocking non-compliant workloads.
3. **Runtime Detection Layer**
    - **Tools:** Falco + Falcosidekick
    - **Purpose:** Monitor container and cluster activities for behavioral anomalies and security violations in real time.
4. **Centralized Monitoring & SIEM Layer**
    - **Tools:** Elastic Agent, Elasticsearch, Kibana
    - **Purpose:** Aggregate telemetry from all security tools, index it for search and detection, and visualize through dashboards.

5.2 Data Flow Description

The end-to-end data flow proceeds as follows:

1. **IaC & Image Scanning (Build Stage)**
    - Developers commit Terraform and Helm configurations.
    - Checkov scans IaC files for security misconfigurations.
    - Trivy scans container images for CVEs.
    - Cosign signs approved container images before pushing them to the private registry.
    - Scan results are exported as JSON and forwarded to Elasticsearch.
2. **Admission Control (Deploy Stage)**
    - When a new Pod is created, Kyverno applies require-image-signature and verify-signed-images policies.
    - Unsigned or vulnerable images are rejected immediately at the API server.
    - Kyverno generates admission reports that are collected by Elastic Agent and stored in Elasticsearch.
3. **Runtime Detection (Execute Stage)**
    - Falco runs as a DaemonSet across the cluster, monitoring kernel and container syscalls.
    - Suspicious behaviors (e.g., shell in container, API server contact, privilege escalation) trigger Falco alerts.
    - Falcosidekick forwards these alerts to Elasticsearch using structured HTTP payloads.
4. **Centralized Monitoring (Analyze & Respond Stage)**
    - Elastic Agent collects logs from Falco, Kyverno, Checkov, and Trivy.
    - All logs are sent to Elasticsearch and indexed with specific patterns (falco-*, checkov-*, trivy-*, kyverno-*).
    - Kibana dashboards visualize security events, IaC misconfigurations, vulnerability trends, and runtime alerts.
    - Analysts use these dashboards for detection engineering, triage, and response.

5.3 Data Flow Diagram



5.4 Key Architectural Strengths

- **End-to-End Security Coverage:** The architecture secures every stage build, deploy, and runtime without relying on external proprietary systems.
- **Defense-in-Depth:** Multiple independent layers (scanning, policy enforcement, runtime detection) reduce single points of failure.
- **Centralized Telemetry:** All security-relevant events flow into Elasticsearch, enabling unified analytics and detection engineering.
- **Scalability:** The design supports scaling across multiple namespaces and workloads with minimal modifications.

- **Modularity:** Each component can be upgraded or replaced without affecting the entire pipeline.

6. Evidence & PoC

The following evidence validates the successful deployment, configuration, and integration of all CNAPP components, demonstrating their operational effectiveness across scanning, enforcement, runtime detection, and centralized monitoring.

6.1 Checkov – IaC Misconfiguration Detection

Evidence:

- Terraform and Helm configuration directories were scanned using Checkov.
- Detected issues included insecure security groups, public buckets, and missing encryption on cloud resources.
- Scan outputs were generated in JSON format and ingested into Elasticsearch.
- Kibana dashboards visualize IaC misconfiguration trends over time.

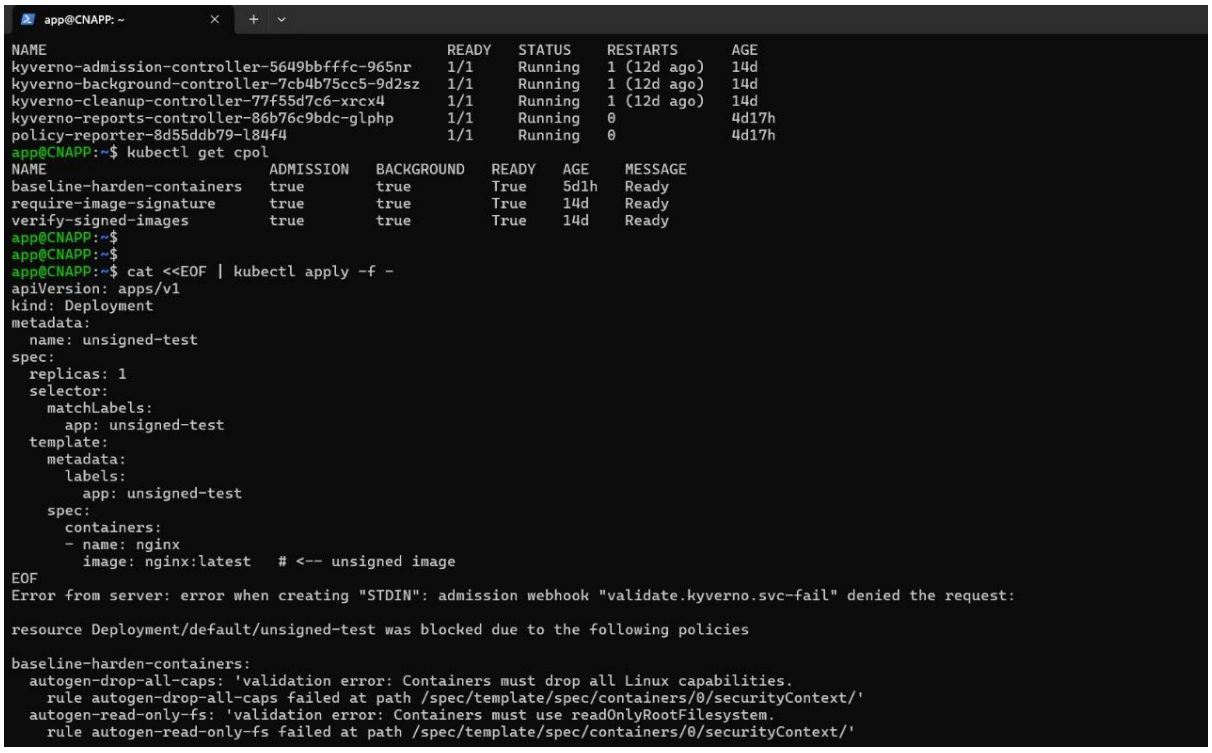
Validation: Misconfiguration alerts were visible in checkov-* index patterns and dashboards in Kibana.

6.2 Cosign & Kyverno – Image Signing and Policy Enforcement

Evidence:

- Container images were signed with Cosign before pushing to the private registry.

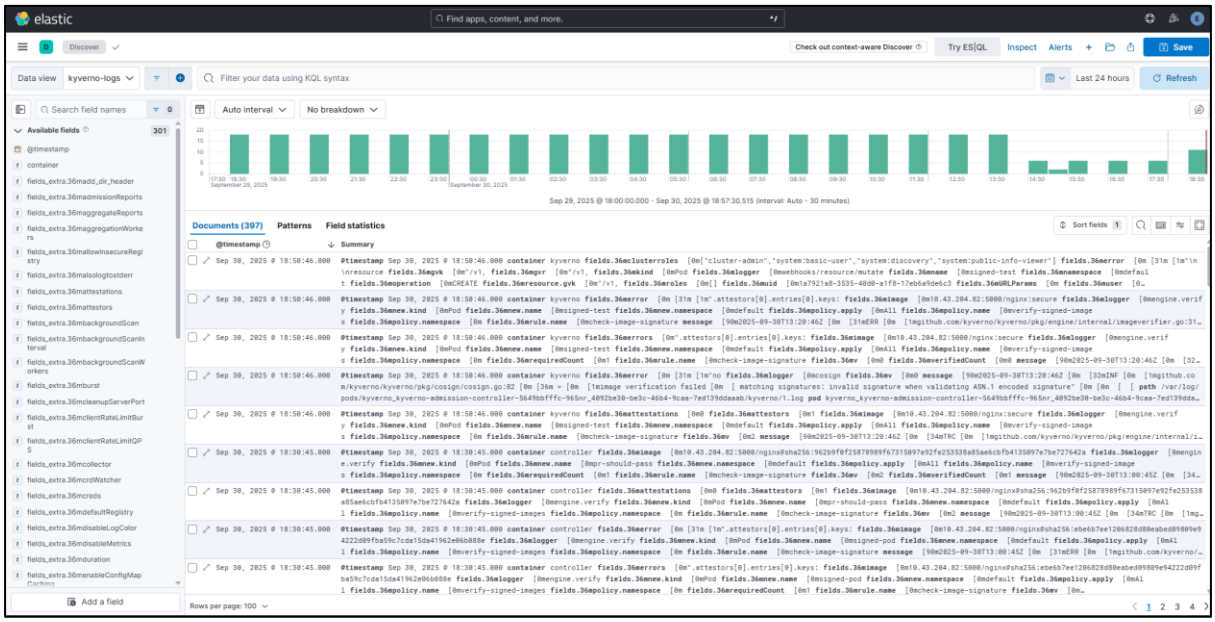
- Kyverno policies require-image-signature and verify-signed-images enforced signature verification during admission.



Validation:

- Unsigned images were correctly rejected by Kyverno during kubectl apply.
- Signed images were allowed, confirming policy enforcement.

- Kyverno admission logs were ingested into Elasticsearch and visualized through kyverno-* dashboards.



6.3 Trivy – Vulnerability Scanning

Evidence:

- Trivy was used to scan container images in the private registry.

```
app@CNAPP:~$ trivy image --scanners vuln 10.43.204.82:5000/nginx:secure
2025-09-30T13:31:06Z    INFO    [vuln] Vulnerability scanning is enabled
2025-09-30T13:31:10Z    INFO    Detected OS     family="debian" version="12.12"
2025-09-30T13:31:10Z    INFO    [debian] Detecting vulnerabilities...   os_version="12" pkg_num=149
2025-09-30T13:31:10Z    INFO    Number of language-specific files       num=0
2025-09-30T13:31:10Z    WARN    Using severities from other vendors for some vulnerabilities. Read https://trivy.dev/v0.66/docs/scanner/vulnerability#severity-selection for details.

Report Summary
```

Target	Type	Vulnerabilities
10.43.204.82:5000/nginx:secure (debian 12.12)	debian	139

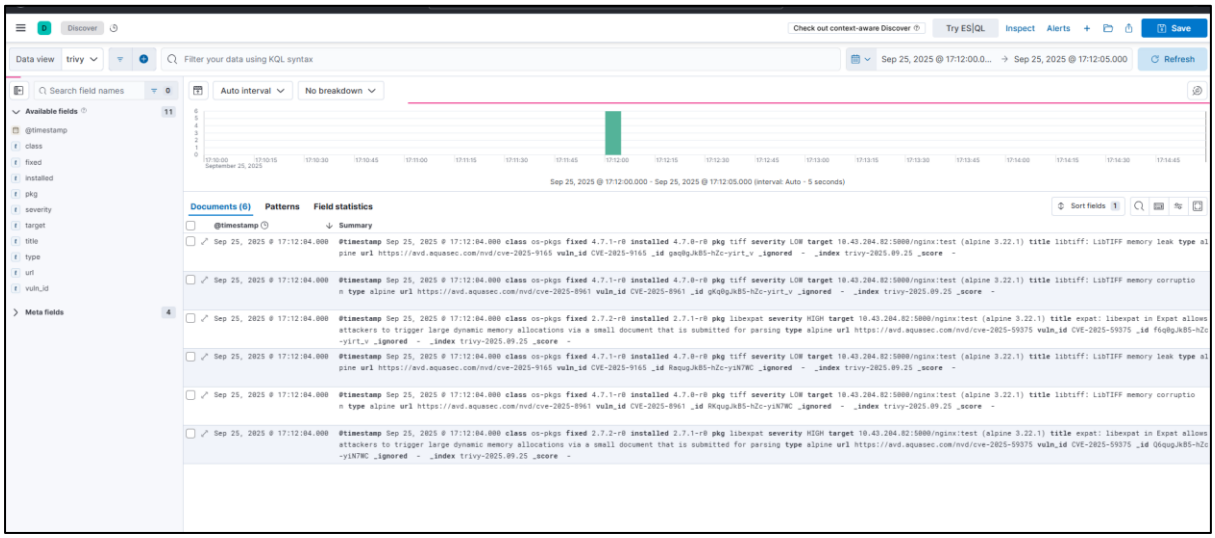
```
Legend:
- '-': Not scanned
- '0': Clean (no security findings detected)


10.43.204.82:5000/nginx:secure (debian 12.12)

Total: 139 (UNKNOWN: 0, LOW: 105, MEDIUM: 22, HIGH: 10, CRITICAL: 2)
```

Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
apt	CVE-2011-3374	LOW	affected	2.6.1		It was found that apt-key in apt, all versions, do not correctly... https://avd.aquasec.com/nvd/cve-2011-3374
bash	TEMP-0841856-B18BAF			5.2.15-2+b9		[Privilege escalation possible to other user than root] https://security-tracker.debian.org/tracker/TEMP-0841856-B18BAF
bsdutils	CVE-2022-0563			1:2.38.1-5+deb12u3		util-linux: partial disclosure of arbitrary files in chfn and chsh when compiled... https://avd.aquasec.com/nvd/cve-2022-0563
coreutils	CVE-2016-2781		will_not_fix	9.1-1		coreutils: Non-privileged session can escape to the parent session in chroot https://avd.aquasec.com/nvd/cve-2016-2781
	CVE-2017-18018		affected			coreutils: race condition vulnerability in chown and chgrp https://avd.aquasec.com/nvd/cve-2017-18018
	CVE-2025-5278					coreutils: Heap Buffer Under-Read in GNU Coreutils sort via Key Specification https://avd.aquasec.com/nvd/cve-2025-5278
curl	CVE-2025-10148	MEDIUM	will_not_fix	7.88.1-10+deb12u14		curl: predictable WebSocket mask https://avd.aquasec.com/nvd/cve-2025-10148
	CVE-2025-9086		affected			curl: libcurl: Curl out of bounds read for cookie path https://avd.aquasec.com/nvd/cve-2025-9086
	CVE-2024-2379	LOW				curl: QUIC certificate check bypass with wolfSSL https://avd.aquasec.com/nvd/cve-2024-2379
	CVE-2025-0725					libcurl: Buffer Overflow in libcurl via zlib Integer Overflow https://avd.aquasec.com/nvd/cve-2025-0725
dpkg	CVE-2025-6297			1.21.22		It was discovered that dpkg-deb does not properly sanitize directory p https://avd.aquasec.com/nvd/cve-2025-6297

- Detected vulnerabilities were logged in structured JSON format and ingested into Elasticsearch.
- Dashboards were built to visualize vulnerability distributions by CVE severity and component.

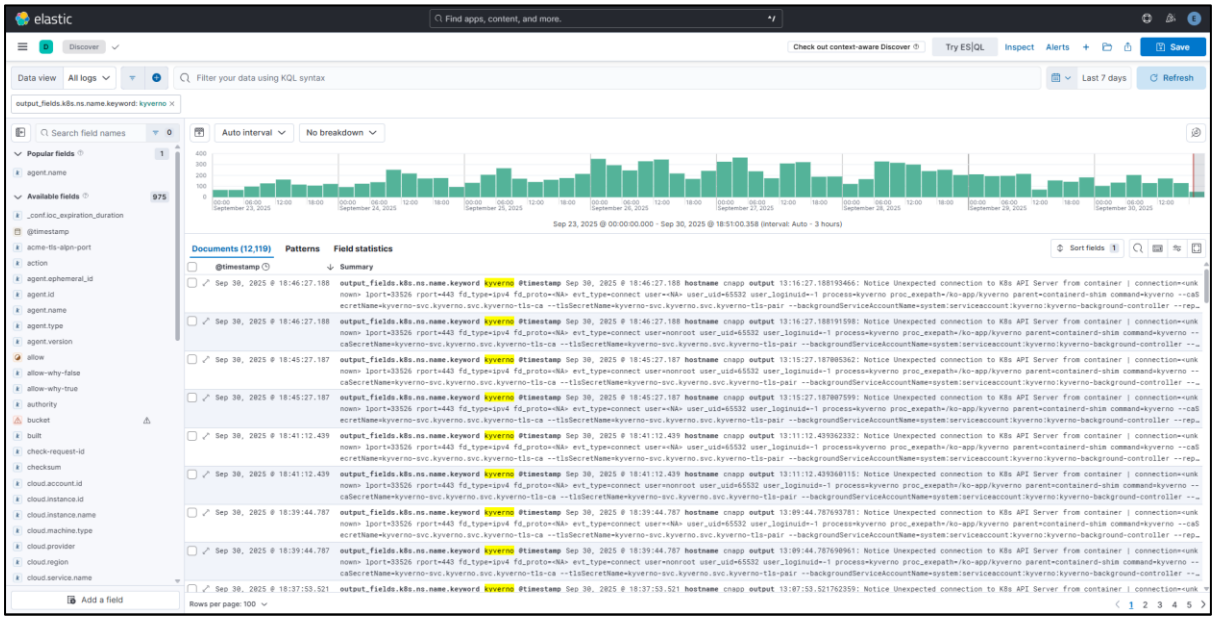


Validation: Vulnerability reports appeared in trivy-* index patterns and Kibana dashboards.

6.4 Falco & Falcosidekick – Runtime Threat Detection

Evidence:

- Falco generated alerts for anomalous runtime activities, including terminal access to containers and API server contact.
- Falcosidekick forwarded these alerts to Elasticsearch in real time.



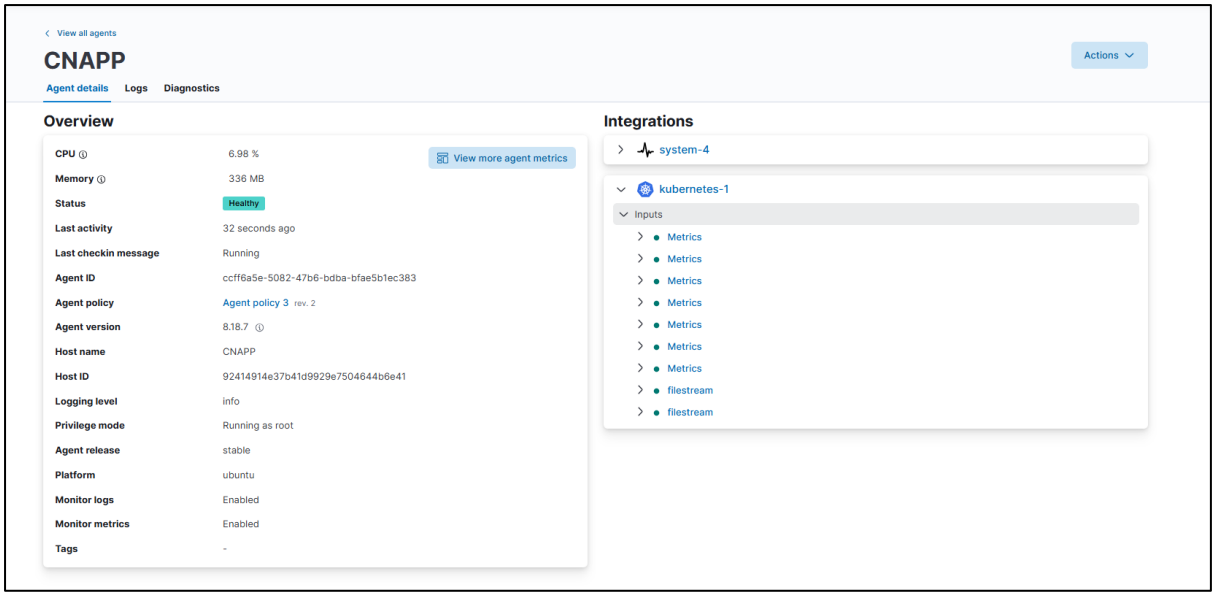
Validation:

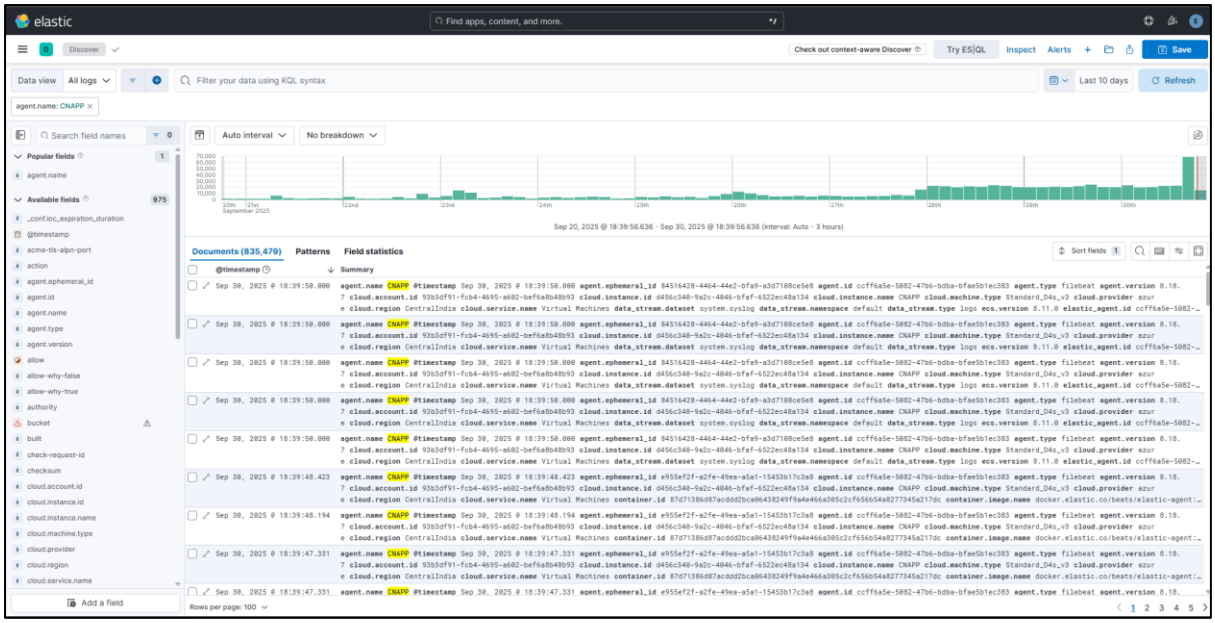
- Alerts appeared in the falco-* indices in Elasticsearch.
- Kibana dashboards displayed runtime threat timelines and severity distribution.
- Analysts confirmed Falco rules triggered during simulated container shell access.

6.5 Elastic Agent & ELK Stack – Centralized Telemetry

Evidence:

- Elastic Agent successfully collected logs from Checkov, Kyverno, Trivy, and Falco.
- Data was indexed in Elasticsearch and visualized through custom Kibana dashboards.
- Log pipelines were validated using GET _cat/indices API.

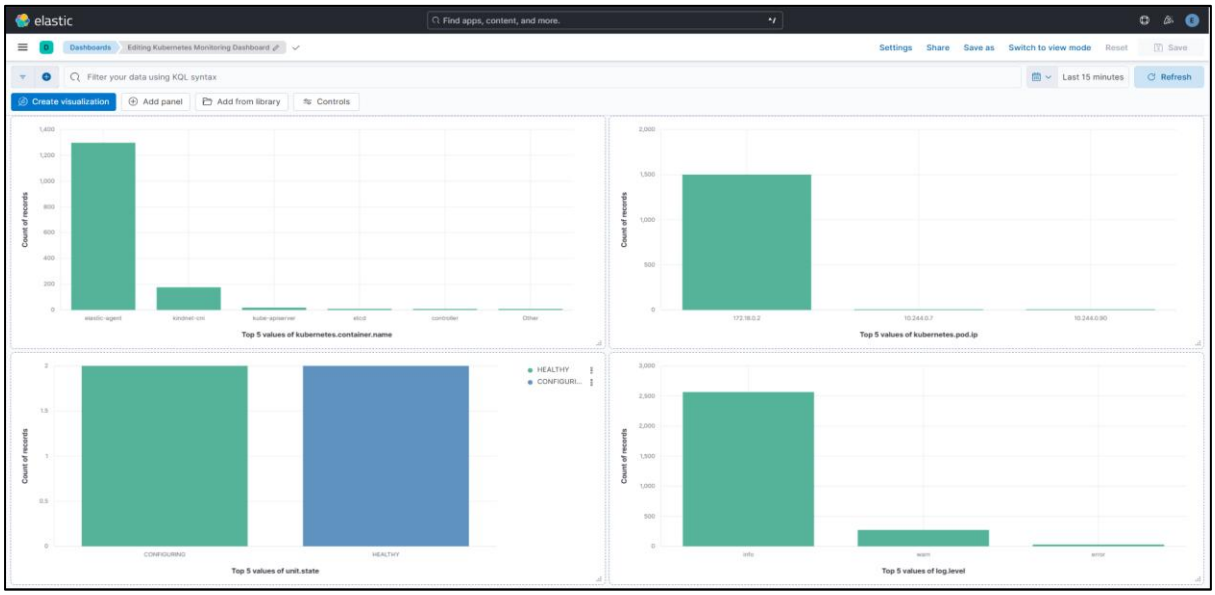


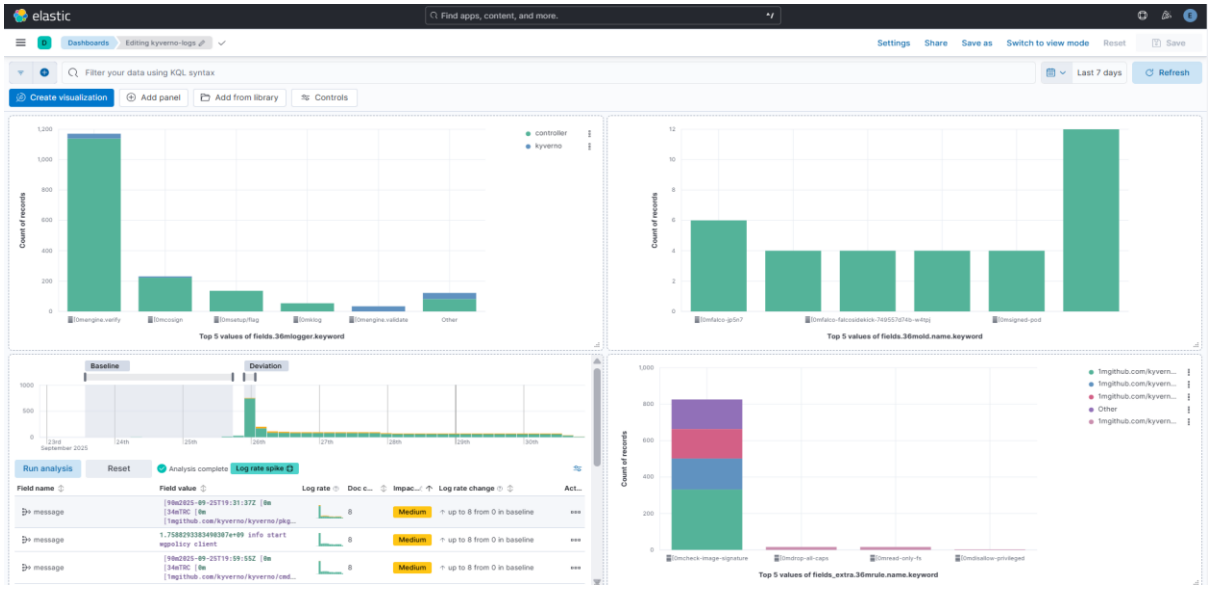


Validation:

- All components' telemetry was flowing correctly into Elasticsearch.

- Kibana dashboards presented correlated views across IaC misconfigurations, vulnerabilities, policy violations, and runtime detections.





6.6 End-to-End PoC Validation

Scenario:

A test container image with known vulnerabilities and unsigned signature was deployed to validate the end-to-end security pipeline.

Steps & Observations:

1. **Checkov** detected IaC issues during the build stage.
2. **Trivy** flagged vulnerabilities in the image.

3. **Kyverno** rejected the image due to missing signature.
4. A deliberately compromised container triggered **Falco** runtime alerts.
5. All logs were captured and correlated in Elasticsearch, and dashboards displayed real-time insights.

Outcome:
The complete CNAPP pipeline functioned as expected, providing **detection, enforcement, monitoring, and visibility** across the container lifecycle.

7. Requirement ↔ Evidence Mapping Table

The following table maps the **security and operational requirements** for the CNAPP implementation to the **corresponding technical evidence** collected during deployment and testing. This ensures complete alignment between expected outcomes and actual delivered capabilities, and provides clear traceability for SOC teams, auditors, and stakeholders.

Requirement	Solution Component	Implementation Summary	Evidence Reference
IaC misconfigurations must be detected before deployment	Checkov	IaC scanning on Terraform and Helm templates with misconfiguration policy packs	Checkov CLI scan output, JSON reports in checkov-* index, Kibana IaC dashboard
Only trusted and signed container images should be allowed in the cluster	Cosign + Kyverno	Images signed using Cosign; Kyverno policies require-image-signature and verify-signed-images enforced at admission	Cosign signing logs, Kyverno policy status (kubectl get clusterpolicy), rejection of unsigned images
Container images must be scanned for vulnerabilities prior to deployment	Trivy	Pre-deployment vulnerability scans with CVE detection and Elasticsearch ingestion	Trivy JSON reports, Kibana vulnerability dashboards (trivy-* index)

Unsigned or vulnerable images must be blocked at deployment time	Kyverno	Cluster-wide enforcement of image signature verification and policy checks at the API server level	Kyverno admission control logs, policy READY status, rejection events
Runtime security monitoring must detect anomalous container behavior	Falco + Falcosidekick	Falco DaemonSet monitoring runtime syscalls; Falcosidekick forwarding alerts to ELK	Falco alert JSON logs (falco-* index), simulated “shell in container” alerts
All security telemetry must be ingested into a centralized SIEM for monitoring	Elastic Agent + Elasticsearch	Elastic Agent forwarding logs from Falco, Kyverno, Checkov, and Trivy to Elasticsearch	_cat/indices output showing active indices, Elastic Agent configuration, successful ingestion
Security events must be visualized in dashboards for analysis	Kibana	Custom dashboards built for IaC, vulnerabilities, policies, and runtime alerts	Kibana dashboard screenshots (Falco alerts, Checkov misconfigs, Trivy CVEs, Kyverno policies)
End-to-end security pipeline must enforce DevSecOps principles	Full CNAPP stack	Integrated scanning, signing, policy enforcement, runtime detection, and SIEM monitoring	PoC validation scenario with test image, sequential detection logs, correlated dashboard views
Open-source tools should be used wherever possible to minimize licensing costs	Checkov, Trivy, Kyverno, Cosign, Falco, ELK	All components deployed were open-source with no proprietary licensing required	Tool version lists, deployment manifests, evidence of functional integrations

8. Conclusion

8.1 Achievements

The CNAPP implementation successfully delivered a **comprehensive, open-source–based container security pipeline** that spans the entire Kubernetes workload lifecycle. Key accomplishments include:

- **Shift-Left Security Enablement:**
    - Integrated **Checkov** and **Trivy** to enforce early-stage IaC and image vulnerability scanning, ensuring misconfigurations and known CVEs are identified before deployment.
- **Supply Chain Integrity Enforcement:**
    - Implemented **Cosign** for image signing and **Kyverno** for admission control, enabling cryptographic verification of container images and preventing unauthorized deployments.
- **Runtime Threat Detection:**
    - Deployed **Falco** with Falcosidekick to monitor container activities, successfully detecting anomalous behaviors such as container shell access and API server interaction.
- **Centralized SIEM Integration:**
    - Unified telemetry collection from all security tools through **Elastic Agent** into **Elasticsearch**, and developed **Kibana dashboards** for misconfigurations, vulnerabilities, policy violations, and runtime events.
- **End-to-End DevSecOps Workflow:**
    - Demonstrated a fully integrated CNAPP pipeline using only open-source components, delivering visibility and control across build, admission, runtime, and monitoring phases.

8.2 Gaps & Limitations

While the solution met core requirements, several gaps and limitations were identified that can be addressed in future iterations:

- **Lack of Automated Policy Updates:**
    - Current Kyverno policies and Checkov rules require manual updates; there is no CI/CD integration for dynamic policy synchronization.
- **Limited Alert Correlation:**
    - Although logs are centralized, correlation between IaC misconfigurations, vulnerabilities, and runtime events is basic and lacks advanced SOAR workflows.
- **Authentication & Access Control in Kibana:**
    - Basic authentication is currently in use. Fine-grained RBAC or SSO integration for SOC users is not yet implemented.
- **Scaling Considerations:**
    - The current setup is validated on a single K3s cluster. Multi-cluster deployments would require additional coordination for log shippers and policy replication.
- **No Active Response Automation:**
    - While detections are in place, no automated remediation (e.g., workload isolation, alert escalation) has been implemented in response to Falco or policy violations.

8.3 Roadmap & Future Enhancements

To further mature the CNAPP solution and align it with enterprise-grade SOC capabilities, the following enhancements are recommended:

Roadmap Item	Description	Expected Benefit
SOAR Integration	Integrate with Tines or a similar orchestration platform to automate alert enrichment and response.	Faster incident response, reduced analyst workload

Advanced Correlation Rules	Develop correlation logic in Elasticsearch/Kibana to link IaC, vulnerability, and runtime findings into unified incident views.	Improved detection fidelity and investigation efficiency
RBAC & SSO for Kibana	Integrate Kibana with Keycloak for role-based access control and SSO.	Strengthened access security and auditability
Policy-as-Code Automation	Implement GitOps workflows for Checkov/Kyverno rulesets using ArgoCD or Flux.	Faster policy updates, reduced manual overhead
Multi-Cluster Scaling	Extend Elastic Agent and Falco deployments to multi-cluster environments.	Broader coverage for distributed workloads
Remediation Playbooks	Add automated actions (e.g., blocking pods, revoking credentials) triggered by Falco or policy violations.	Reduced mean time to respond (MTTR)

8.4 Strategic Impact

By adopting this CNAPP architecture, the gains:

- **Continuous visibility** across build, deploy, and runtime phases.
- **Reduced attack surface** through enforced policies and vulnerability controls.
- **Faster detection and triage**, leveraging centralized telemetry and dashboards.
- **Cost efficiency**, as all components are open-source with no licensing overhead.
- **A modular security foundation** that can scale with the organization's Kubernetes footprint and integrate with advanced SOC automation platforms.