Final Year Project

Kumon: Performance Monitoring Tool

Final Report

FYP Team

Abdullah Waqar 20I-0615

Ahsan Iqbal 20I-1814

Rida Kamal 20I-2496

Supervised by

Dr. Muhammad Arshad Islam

Department of Computer Science

National University of Computer and Emerging Sciences

Islamabad, Pakistan

Spring 2024

Anti-Plagiarism Declaration

This is to declare that the above FYP report produced under the:

Title: **Kumon A Performance Monitoring Tool**

is the sole contribution of the author(s) and no part hereof has been reproduced on as it is basis (cut and paste) which can be considered as Plagiarism. All referenced parts have been used to argue the idea and have been cited properly. I/We will be responsible and liable for any consequence if violation of this declaration is determined.

Date: 09-04-2024

Student 1

Name: Abdullah Waqar

Signature:

Student 2

Name:Ahsan Iqbal

Signature:

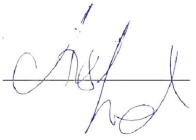
Student 3

Name: Rida Kamal

Signature:

Supervisor (Faculty)

Name: Dr. Muhammad Arshad Islam



Signature:

Authors’ Declaration

This states Authors’ declaration that the work presented in the report is their own, and has not been submitted/presented previously to any other institution or organisation.

Table of Contents

Abstract

Performance monitoring systems have gained more prominence in today's cloud-native era. KuMon uses new advanced eBPF technology and its design is supported by a wide range of literature and focuses on critical measures derived from academic research. Utilising various key resource efficiency, performance indicators, and scaling dimensions based on peer-reviewed research, KuMon will provide targeted and efficient monitoring mechanisms. KuMon follows intermediate levels of granularity, inspired by top researchers' advice on granular insights. This is a balance of approach capturing some fundamental dynamics and manageable overhead. The primary goal of KuMon is to produce tracked data at the system-call level, providing more detail to administrators while avoiding excessive strain on the infrastructure. KuMon has shifted from the traditionally efficiency oriented monitoring systems with improved precision and adaptability. In line with scholarly suggestions on metric priority and precision, KuMon seeks to offer a monitoring tool that is viable to guide administrators towards taking actions necessary for running robust and competent digital structures. We have managed to integrate Cilium, Prometheus and Grafanato to build an advanced performance monitoring tool.

1. Introduction

In today's world of fast paced cloud computing, the effective management of Kubernetes clusters is a significant challenge, along with the ever-increasing volume and complexity of data. This is where our Final Year Project, KuMon, comes into play. KuMon is an advanced performance monitoring tool, specifically engineered for Kubernetes environments. At its core, KuMon leverages the Extended Berkeley Packet Filter (eBPF) technology, a choice that emphasises our commitment to delivering a tool that is both powerful and precise in its functionality.

The main goal of building KuMon is to serve the cloud engineers, DevOps persons and system admin to escalate their work not only by making it easy for them but also by giving the insights of the system which they used. In the big world where information is big, and the room for mistake minimal, KuMon provides an accurate means to trace and display performance trends as they occur. It is critical in supporting decisions based on concrete information that has a direct positive effect on productivity and system performance. KuMon serves a significant gap in the present cloud computing ecosystem – a real-time, intuitive, one-stop digital solution that can take in large amounts of data and provide insights and remediation to problems such as those regarding performance lags or network configurations. KuMon is critical to be used by professionals engaged in the management of cloud structures as it provides key information for the utilization and performance of cloud technologies

1.1 Unpacking eBPF

eBPF, short for Extended Berkeley Packet Filter is developed from Unix operating systems, but has mutated into a dynamic and flexible virtual machine. It is primarily designed to provide a safe and effective means for logging and tracking events in a system as they unfold. eBPF is connected in the Linux kernel and contains small programs, called probes, which allow the gathering of application and system data from a range of areas, including networks, system calls, and events. Some recent works have reported that with eBPF-based monitoring tools, it is possible to get high fidelity with marginal performance overhead Joao et al. , 2016 [1] These probes run eBPF programs which are similar to small scripts running on specific events to implement the core function of KuMon. KuMon is a performance monitoring tool that integrates with eBPF technology and Kubernetes clusters. Targeted at cloud engineers, DevOps personnel, system administrators, and application developers, it is designed to afford profound intelligence for real-time allocation of resources and identification of network problems.

2. Literature Review

Beyer and Ewaschuk [2] emphasize CPU, memory, and storage metrics as fundamental for understanding system resource consumption. Merkel [3] underscores the need for effective resource management in containerized environments like Docker. Liu and Wood [4] identify latency, throughput, and error rates as critical performance indicators, reflecting system responsiveness and reliability. Nam & Kim [5] highlight the importance of monitoring process creation, termination, and system scalability for maintaining a dynamic environment.

Deri et al. [6] emphasize the significance of granular network metrics in understanding network traffic behaviors and congestion sources. Packet-level monitoring aids in understanding network behavior comprehensively without overwhelming the monitoring system. For KuMon, a medium level of granularity is targeted, revealing patterns effectively without causing excessive system overhead. At the system call level, traced information will cover critical tasks with minimal overhead, ensuring comprehensive understanding of system behavior.

Suo et al. [7] present efficient packet tracing methods that assist in selecting essential metrics for KuMon, enhancing its effectiveness in monitoring cloud-native environments. Scholz et al. [8] stress the importance of detailed data collection, aligning with KuMon's need for comprehensive monitoring capabilities.

eBPF's key advantage in performance monitoring lies in its ability to probe various kernel subsystems without requiring modifications to applications or introducing significant overhead Radchenko et al., 2017. [9]

Cilium, a leading open-source project, leverages eBPF to deliver advanced networking, security, and observability for Kubernetes environments. Built from the ground up on eBPF, Cilium offers rich network insights due to its service-identity-aware monitoring capabilities Matei et al., 2018 [10] This eliminates the need for complex sidecar proxies fostering a lightweight and scalable approach.

3. Project Vision

3.1 Problem Statement

In today's world of cloud computing, a pressing need for efficient performance monitoring has emerged within Kubernetes clusters. While traditional Application Performance Management (APM) tools and sidecar proxies have proven helpful, they also come with limitations in this particular context. APM causes overhead as each application has its own APM. A better alternative for APMs are sidecars. Although as seen in figures below in context to Kubernetes, sidecars are a good replacement for APMs as now each application doesn't require a performance management agent. However, this still causes latency and configuration overhead. These restrictions result in delayed identification of issues, inadequate information for resolving problems, and the added complication of managing extra components in an already intricate Kubernetes environment. Consequently, system administrators, cloud engineers, and DevOps experts face significant challenges in optimising their systems this is where KuMon aims to serve as an enhanced tool overcoming these drawbacks.

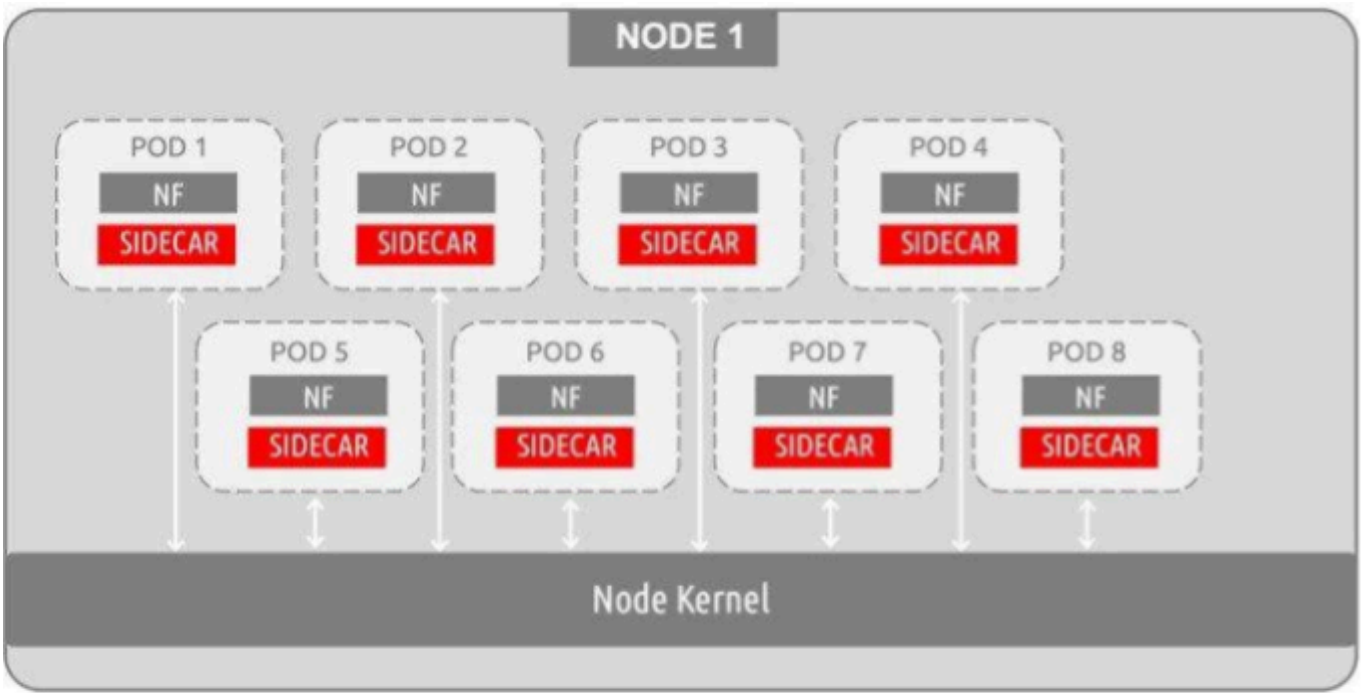


Figure 1: Sidecar configured on each pod

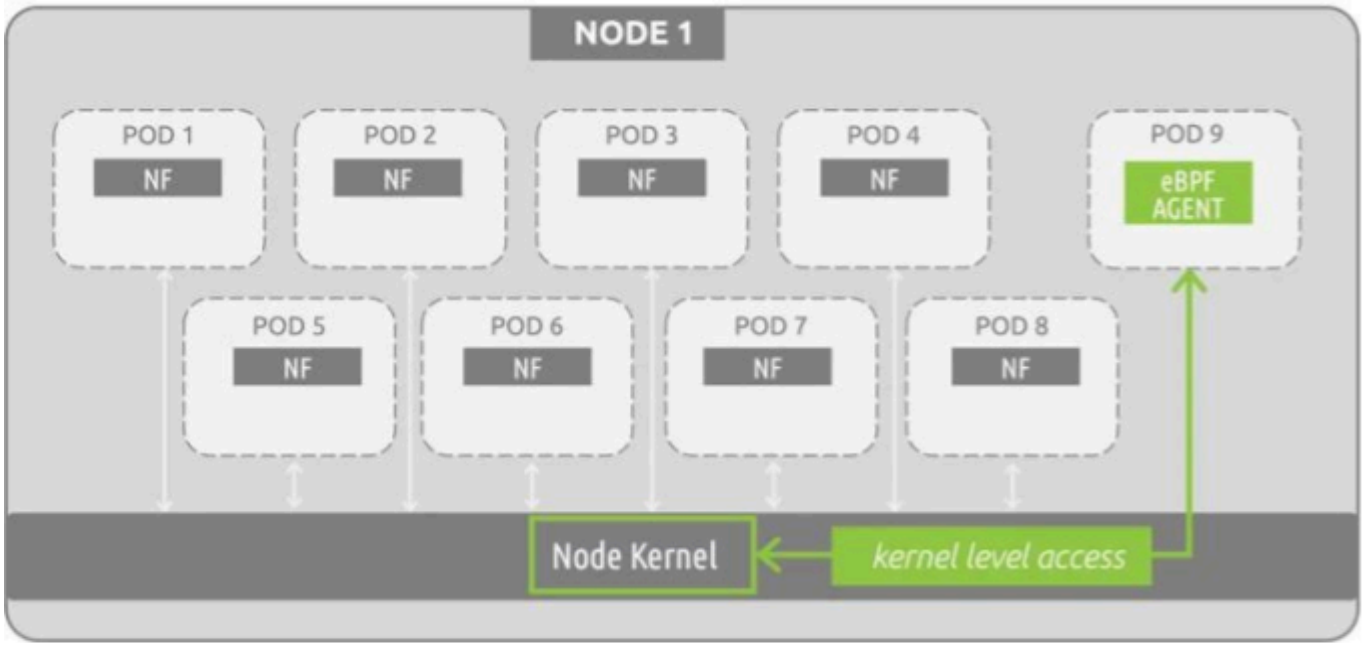


Figure 2: A single pod configured with ebpf monitors all pods in a cluster

Figure 1. illustrates the traditional sidecar approach, which often complicates Kubernetes monitoring by introducing an additional layer. Sidecars, while useful, can increase the complexity and resource consumption of the cluster.

Figure 2. highlights configuration of Application Performance Management (APM) tools. This setup has cumbersome configuration processes typically associated with APMs, offering a less efficient and intrusive method of monitoring Kubernetes clusters.

Kumon tackles these obstacles head-on by providing a tailored solution for Kubernetes clusters. This sets it apart from traditional APM tools by seamlessly integrating and minimising intrusion and overhead. By utilising eBPF technology, it effectively avoids the complexities and limitations of sidecars, instead directly tapping into the data streams of the cluster. This real-time data collection and monitoring enables efficient resource allocation by providing detailed insights of the system at kernel level. With its focus on the distinctive requirements of Kubernetes, Kumon ensures that professionals can effectively monitor and optimise their systems.

3.2 Business Opportunity

KuMon represents a significant business opportunity, catering to organizations operating in cloud native environments. This tool's capability to enhance performance, optimize resources, and ensure cost effectiveness and user satisfaction makes it a great addition to their cloud native toolset available. In an era where digital transformation and cloud adoption are at what organizations are really capitalizing on KuMon offers varied capabilities like:

Optimizing Resource Utilization: One of the key features of cloud-native environments is that resources are allocated and deallocated as needed. By collecting data and analyzing it, KuMon helps organizations understand how resources are being utilized. This way, they can allocate them better and keep costs low.

Cost Effectiveness: It is important to note that, in cloud native corporations, effectiveness must be cost-effective due to efficient utilization of resources and fast detection and resolution of performance issues. Thus, by averting costly performance bottlenecks, KuMon ensures that firms remain cost effective.

User Satisfaction: In the digital age, user satisfaction is a primary metric for success. Frustrating users with slow or unreliable applications would not however lead to success because real time monitoring and alerting capabilities by KuMon.

3.3 Objectives

1. Develop a Kubernetes-Specific Monitoring Tool
2. Utilise eBPF for Efficient Data Collection
3. Provide Customizable User Interfaces
4. Enable Historical Data Analysis
5. Facilitating quick detection and reporting of system issues

This way, Kumon simplifies and improves the task of monitoring Kubernetes clusters by leveraging effective data collection techniques and including useful desirable characteristics that would make the monitoring project more effective for users, Overall, Kumon provides a clear and efficient view of how the cluster is performing. This not only enhances possibilities of fixing problems easily, But also user-centric approaches that enable them to monitor their systems and sustain optimal functionality. These goals are in sync with the need for a tool that is specifically designed for Kubernetes and the enhancement of various factors like providing reliability to software systems in the cloud.

3.4 Constraints

The development and implementation of KuMon are subject to several constraints, which include:

1. Time Constraints: Given that KuMon is a Final Year Project, it is bound by academic timelines, which may limit the scope of development and thorough testing.

2. Hardware Limitations: Our ability to test and deploy KuMon is constrained by the hardware available to us. This limitation might affect the tool's scalability and performance testing in large-scale Kubernetes environments.

3. Data Collection and Processing Challenges: While eBPF technology enhances data collection, processing this data in real-time can be challenging, especially when dealing with large volumes of performance metrics.

4. Integration Complexities: Integrating KuMon with diverse Kubernetes setups presents a major challenge due to their unique configurations. KuMon must navigate these complexities carefully to seamlessly adapt to various architectures, versions, and deployment scenarios. Overcoming this demands precise testing, compatibility checks, and tailored solutions for a user-friendly experience without compromising monitoring efficiency.

3.5 Stakeholders Description

3.5.1 Stakeholders Summary

KuMon caters to a diverse group of stakeholders, including:

- Cloud Engineers
- DevOps Professionals
- System Administrators
- Application Developers

3.5.2 Key High Level Goals and Problems of Stakeholders

Cloud Engineers: Aspirations should continue to be strong healthy cloud answers and many programs such as KuMon to enable real-time access to applications and resource. Their challenge is staked on the vast potential that needs to be managed and supervised, which is distributed across numerous cloud environments.

DevOps Professionals: Concentrate efforts on efforts to enhance efficiency and productivity of processes and applying resources. Development and operation teams present challenges when it comes to the effectiveness of development and the configuration of operations; for example, real-time access to data for making effective preventive decisions.

System Administrators: Strive to provide stability and ensure operational efficiency on systems. One of their major problems is to accomplish the continual juggling between resource utilization and business speed.

Application Developers: Seek effectiveness and optimized cycles , to yield high performing applications. One challenge they face is the difficulty of gaining access to real-time performance data that is critical in optimization of operations of the applications.

3.6 Project Scope

KuMon is specifically designed to monitor and analyse performance metrics in Kubernetes clusters, focusing on aspects such as resource utilisation and network efficiency. KuMon will provide detailed, real-time insights and alerts. The current scope of the project is limited to monitoring and reporting, requiring manual intervention for problem resolution. Additionally, KuMon's capabilities are centred around Kubernetes environments; hence, it may not be applicable for other types of cloud or container orchestration systems.We use diverse cilium metrics ranging from those that monitor endpoint metrics to cluster health metrics like those for API calls, file descriptors etc. Hence comprehensive data is visualised which can be used to identify overshoots and bottlenecks as performance issues occur.

4. Software Requirement Specifications

4.1 List of Features

KuMon's feature set comprises the following key components:

1. **eBPF-based Data Collection:** Utilises Extended Berkeley Packet Filter technology for efficient and detailed real-time data collection from Kubernetes cluster with the help of various eBPF tools to enrich its monitoring capabilities and provide a more comprehensive view of system performance.

2. **Data Visualization with Kibana:** Offers intuitive visualisation of collected data through Kibana, enabling easy interpretation and analysis of performance metrics.

3. **Customizable Dashboards:** Provides flexible dashboard configurations, allowing users to tailor the monitoring interface to their specific needs and preferences.

4. **Real-Time Monitoring**: eBPF allows real-time monitoring in Kubernetes; it offers quick and deep insights into the system's state and performance without causing significant overhead.

5. **Alerting and Notifications System**: Features a robust alerting mechanism to promptly notify users about significant events or performance anomalies within the cluster.

4.2 Functional Requirements

4.2.1 Data Collection

Functional Requirement 1: Data Collection Using eBPF

KuMon shall be capable of collecting real time performance data from Kubernetes clusters using eBPF probes.

Bottlenecks:

- Cloud Engineers, DevOps Professionals, and System Administrators face difficulties in collecting real time performance data efficiently.
- Inability to collect granular performance data hampers their ability to identify and address performance issues.

Functional Requirement 2: Data Sampling

Bottlenecks:

- High data collection rates can lead to resource overhead, making it challenging to monitor applications efficiently.
- Low data sampling rates may miss critical performance events.

4.2.2 Data Visualization

Functional Requirement 3: Integration with Kibana

KuMon will integrate with Kibana for data visualization.

Bottlenecks:

- Customized visualizations can be time consuming and technically challenging to create.

Functional Requirement 4: Custom Dashboards

KuMon will allow users to create custom dashboards to monitor specific aspects of their Kubernetes clusters.

Bottlenecks:

- Stakeholders often need to monitor different metrics specific to their applications and environments, which standard dashboards may not address.

Functional Requirement 5: Visualization Types Provided

KuMon shall support multiple data visualization types, including line charts, bar charts, heatmaps, and tables for data representation.

Bottlenecks:

- Diverse data types require different visualization formats, and the lack of options may lead to misinterpretation of data.

4.2.3 Alerting and Notifications

Functional Requirement 6: Alerting Mechanism

KuMon shall implement customizable alerting mechanisms to notify stakeholders when predefined thresholds or conditions are met.

Bottlenecks:

- Not being alerted about critical performance deviations can lead to downtime and application issues.

Functional Requirement 7: Notification Channels

KuMon shall support various notification channels, such as email, SMS, and integrations with collaboration platforms (e.g., Slack).

Bottlenecks:

- Stakeholders rely on multiple communication channels, and failing to provide alerts through these channels can hinder timely responses.

4.2.4 Customizable Dashboards

Functional Requirement 8: Dashboard Customization

KuMon shall enable users to create, modify, and share dashboards that suit their specific monitoring needs.

Bottlenecks:

- Generic dashboards may not meet the unique requirements of stakeholders, leading to inefficiency in monitoring.

4.3 Non Functional Requirements

Efficient Data Processing: It will be important to ensure that the data is processed and analyzed effectively, and this is something that the KuMon will be able to accomplish thanks to its high processing power.

User-Friendly: The system will be optimized whereby one can learn it easily and operate without much difficulty.

Real-Time Data Processing: In order to manage all these flows the system will be able to process real time data.

Scalability: KuMon is optimistically designed to be very scalable, and this means that it can easily scale up to support the increased data, users, and monitoring needs or requirements; KuMon is designed to support horizontal scalability that employs the use of distributed systems, which ensures that the system remains responsive and always reliable, no matter the level of demand however as mentioned large data can be disadvantageous if not handled correctly, and with careful planning the company can overcome

Security: KuMon can be further tuned and augmented to further give focus to data security at different levels, where the use of encryption techniques, restriction to data access and using reliable methods of authentication plays an important role. Specifically, it applies best industry practices on handling and protecting confidential information as well as achieving the triple-A objectives on data confidentiality, integrity, and availability.

5. Iteration Plan

Iteration	Time Frame	Deliverables to Cover
1	September 2023 - October 2023	☑ Research on eBPF and relevant technologies ☑ Prepare the development environment with the necessary tools and libraries for eBPF programming ☑ Implement basic eBPF probes for data collection
2	November 2023 - December 2023	☑ Filter and categorise the collected data. ☑ Configure ELK stack (Monitoring Dashboard) ☑ Achieve real-time data reporting ☑ Multiple dashboards for each performance metric
	February 2024 - March 2024	☑ Enhance integration with eBPF libraries, tools, and scripts to improve data processing ☑ Configure Kubernetes cluster with Kumon
4	April 2024 - May 2024	☑ Work on alerting mechanisms for a customizable notification system, including support for various notification channels (e.g., email, SMS) ☑ Configured Cilium, Hubble, and Grafana to enable comprehensive visualisation

5.1 Iteration 1: Data Collection and Foundational Elements

Objective:

- Implement eBPF probes for data collection.
- Set up initial Kibana dashboards.
- Basic alerting functionality.

Key Deliverables:

- KuMon's data collection infrastructure.
- Initial integration with Kibana for data visualization.
- Basic alerting mechanisms.
- A solid foundation for future iterations.

Description:

Iteration 1 lays the groundwork for KuMon's core functionality. It focuses on implementing eBPF probes to collect real time performance data from Kubernetes clusters. The initial Kibana dashboards will provide basic data visualisation, while alerting mechanisms will be put in place to notify users of critical events.

5.2 Iteration 2: Data Storage and Real Time Monitoring

Objective: .

- Integrate with eBPF libraries, tools, and scripts.
- Enhance real time monitoring capabilities.

Key Deliverables:

- Historical data storage with retention options
- Integration with eBPF libraries and tools
- Live graphs showing real-time insights

Description:

This iteration of KuMon focuses on long term data management with historical data storage and configurable retention periods. It strengthens the tool's integration with eBPF libraries, tools, and scripts, ensuring seamless data processing. The real time monitoring capabilities will be enhanced to provide users with a comprehensive view of system performance as it happens.The flow of our project can be shown in figures from 5 to 8. The main components of Kumon is eBPF, Fileabeat, Elasticsearch, Logstash, Kibana. As shown in figure 5 the data is generated when applications/ micro-services are run and they trigger events. These events are attached to specific system calls in the kernel which is shown in figure 6. The contextual data that the eBPF scripts generate are stored in kernel maps, which are located in the kernel space. We access this contextual data in the user space and store in csv files after processing the data. As shown in figures 7 and figure 8 the data is transfered by filebeat which is installed and configured on the host machine. Filebeat makes sure to stream any new changes to Logstash using a filewatcher, this allows us to achieve real-time data transfer. Elasticsearch, Logstash and Kibana are hosted on a remote server. Where Logstash will receive data from Filebeat and create indices and pass to Elasticsearch. Elasticsearch, a search engine, will pass the indices to Kibana, where it will be visualized.

5.3 Iteration 3: Cluster Health Monitoring

Objectives

- Enhance monitoring capabilities with detailed metric visualisation using Grafana
- Develop a robust alerting mechanism for issue detection with Prometheus
- Implement user-specific dashboard in Grafana

Key Deliverables

- Detailed metric visualisation through enhanced Grafana dashboards, leveraging Cilium and Hubble metrics
- Alerting mechanism integrated with Prometheus, utilising Cilium and Hubble data
- User-specific dashboard in Grafana, incorporating Cilium and Hubble metrics

Description

Following the progress laid in the prior iterations, the current Iteration III is aimed at enhancing the monitoring foundation through the integration of Cilium, Hubble, and Prometheus, along with Grafana. These technologies support the objective to improve our ability to monitor the Kubernetes clusters and gain a better understanding of the situation. The Cilium monitoring holds a central status of the monitoring stack enabling syntactic deep K8s visibility with Hubble. Only, cilium offers the performance enhancement and networking visibility whereas hubble comes with full visibility to network traffic. In this iteration, we are using Cilium and Hubble in order to broaden the base of Grafana metrics. It will enable us to gain comprehensive information about the overall state of the network and the cluster as a whole.

Cilium & Hubble Metrics

Cilium provides us with a wealth of metrics related to network performance and security within our Kubernetes clusters. Some of the key metrics we are incorporating into our Grafana dashboards include the following-

- Network traffic volume visualises the volume of network traffic within the cluster helps us identify potential bottlenecks or anomalies

  ```
  sum(rate(cilium_network_bytes_total{direction="inbound"}[5m]))
  by (namespace, pod).
  ```

- Monitoring latency metrics helps us ensure that network performance meets our expectations. A query like this calculates the 99th percentile of network latency per namespace and pod over a 5-minute window.

  ```
  histogram_quantile(0.99,sum(rate(cilium_network_latency_bucket{}
  [5m])) by (le)) by (namespace, pod)
  ```

Hubble complements *Cilium* by providing additional insights into network performance and connectivity within Kubernetes clusters. Some of the key metrics we are leveraging from Hubble include the following-

- Visualising flow telemetry data helps us understand the flow of traffic within our clusters, enabling us to understand network performance. A query like this would calculate the rate of flow telemetry events per source and destination IP address within the default namespace over a 5-minute window.

  ```
  sum(rate(hubble_flow_total{namespace="default"}[5m])) by (source, dest)
  ```

- Monitoring DNS resolution metrics helps us ensure that DNS requests are handled efficiently within the cluster. This query calculates the rate of DNS queries per query name within the default namespace over a 5-minute window.

```
sum(rate(hubble_dns_query_total{namespace="default"}[5m]))by
(query_name)
```

Metric	Labels	Description
ipam_available	N/A	The total number of interfaces with addresses available
ipam_nodes_at_capacity	N/A	The total number of nodes in which the Operator is unable to allocate IP addresses
endpoint_state	`state`	The total number of endpoints on a node at a given point in time
bpf_map_ops_total	`map_name`, `operation`, `outcome`	The total number of eBPF map operations performed
endpoint_regenerations_total	`outcome`	The total number of completed endpoint regenerations

Table 1: shows the endpoint health metrics

Metric	Labels	Description
http_requests_total	`method`, `protocol`	The total number of HTTP requests
dns_queries_total	`rcode`, `qtypes`, `ips_returned`	The total number of DNS queries
drop_count_total	`reason`, `direction`	The total number of dropped packets
http_responses_total	`method`, `status`	The total number of HTTP responses
dns_responses_total	`rcode`, `qtypes`, `ips_returned`	The total number of of DNS responses
http_request_duration_seconds	`method`, `protocol`	The duration of an HTTP request in seconds

Table 2: shows the network performance metrics

By leveraging the capabilities of Grafana, we are empowered to transform raw numerical data into comprehensible visual representations. These visualizations, which can take various forms such as graphs or charts, enable us to gain deeper insights into the behavior and performance of the system under scrutiny. The following examples illustrate the types of metrics that can be effectively visualized through Grafana

1. **File Descriptors**

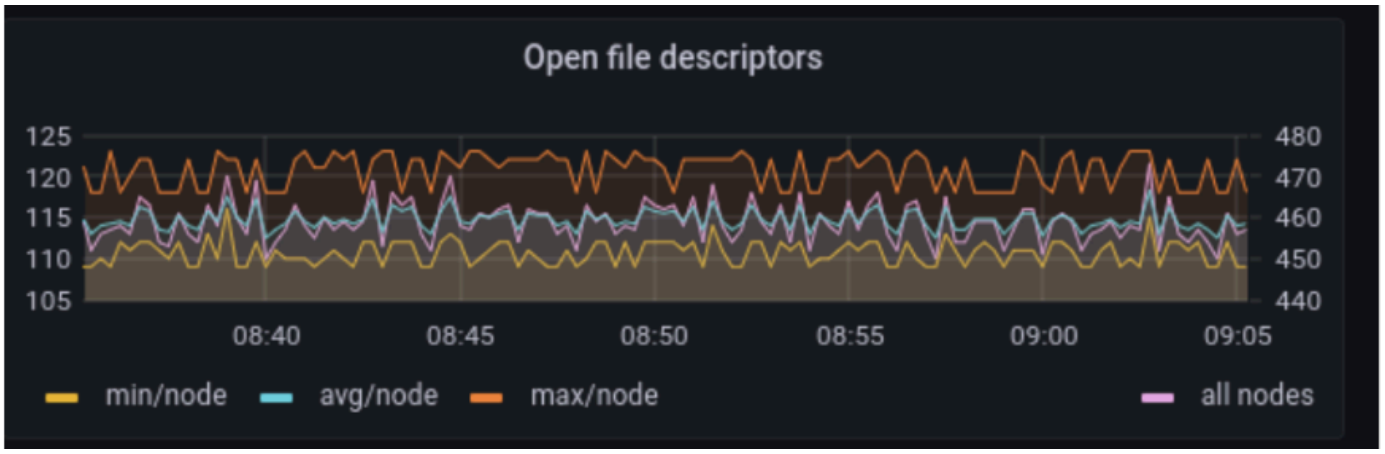


Figure 3: the graph depicts **cilium_endpoint_openfiles**, which tracks the number of open file descriptors per endpoint.

2. **API Call Latency**

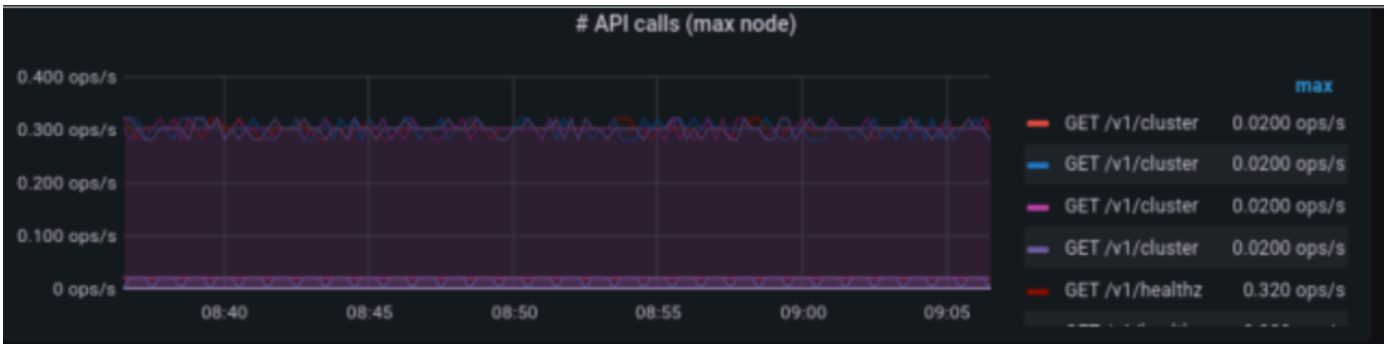


Figure 4: uses metrics like **cilium_api_duration**, which measures the latency of Cilium API calls

3. Forward Vs. Dropped

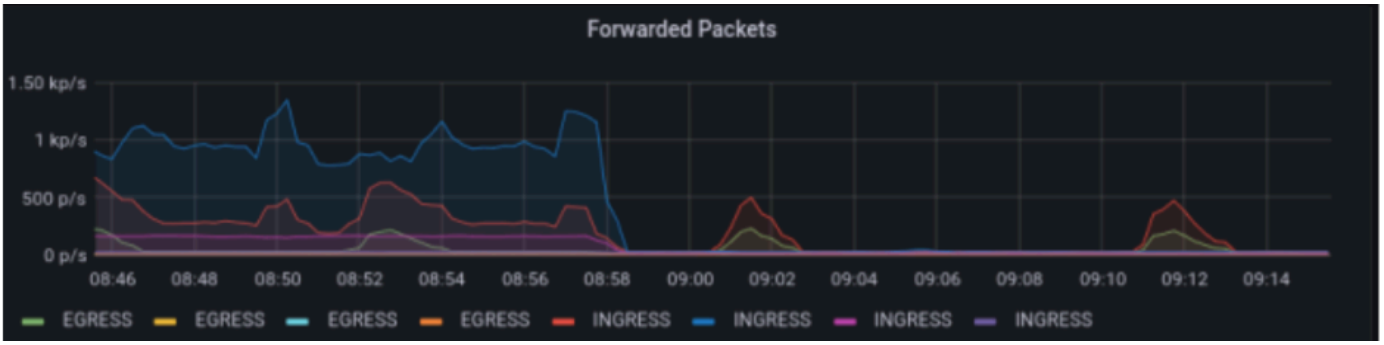


Figure 5: shows graph which uses **flows_verdict etc.** categorized by the verdict label

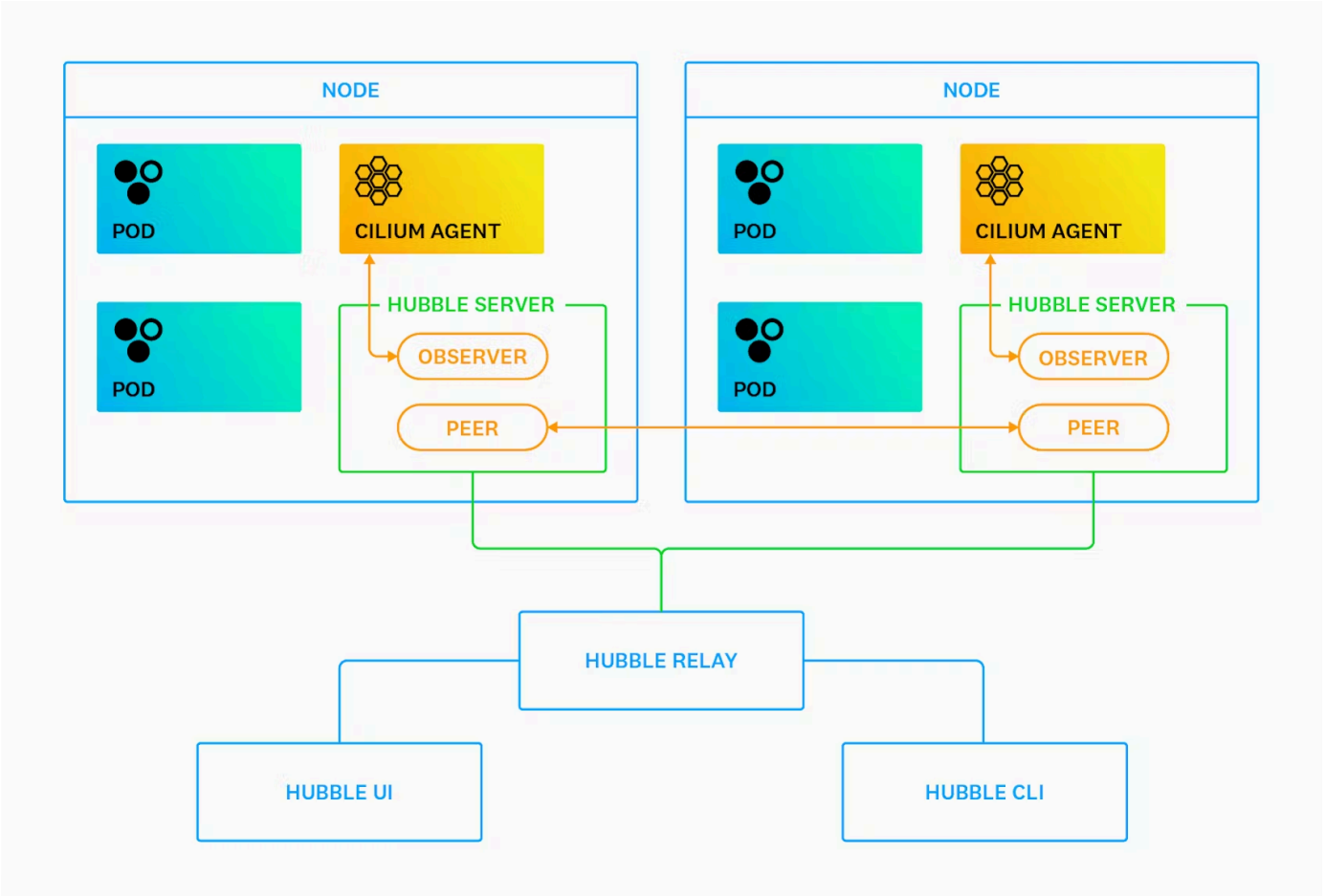


Figure 6: illustrates the key components of the Hubble ecosystem.

- **Pods:** These are the deployment units of the application. They contain one or more containers that share storage and network resources. In the diagram, there are four pods, two on the left and two on the right.
- **Cilium Agent:** This is a containerized networking daemon that provides secure east-west traffic between pods on the same cluster. There is a Cilium agent co-located with each pod.
- **Hubble:** This is an open-source observability platform for cloud-native applications. It provides monitoring and troubleshooting for service meshes. The diagram shows Hubble server and Hubble UI which could be used to visualize data.
- **Peer:** Peers are used for communication between Pod resources across different clusters. The architecture diagram shows a peer on either side.
- **Hubble Relay:** This is an optional component that can be deployed to aggregate metrics and events from multiple Hubble servers and send them to a central monitoring system. The architecture shows a Hubble Relay in the center.

Prometheus Alerting Mechanism

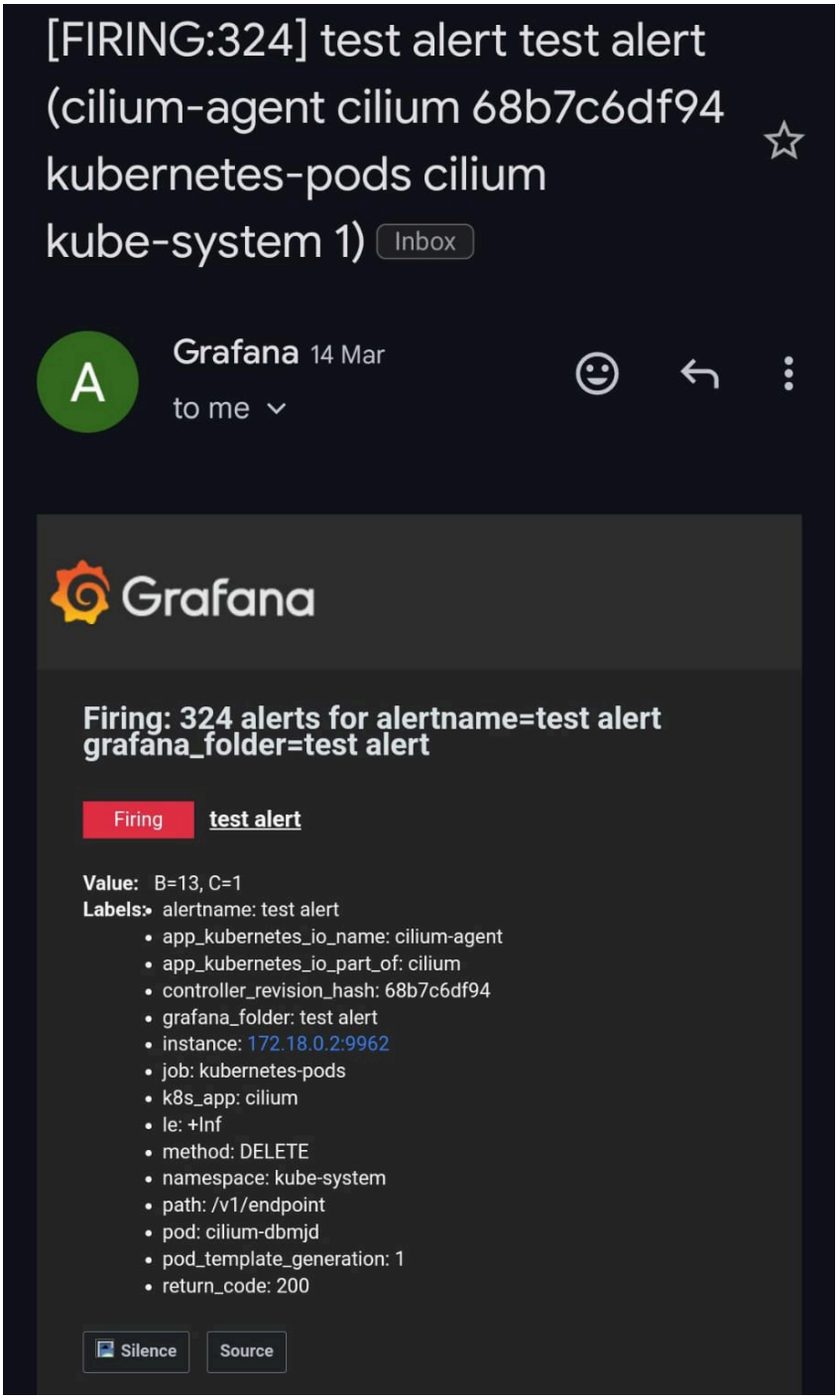


Figure 7: shows example of Prometheus alerting

Prometheus has been reliably employed in this iteration to oversee an effective alerting solution. This mechanism will be used to analyze the data collected by Cilium and Hubble to identify problems that exist and notify the users immediately. Through establishing metrics with certain values and designing workflows that will be able to track these metrics, it is possible to detect problem areas before they get out of hand, be alerted about them via Email and Slack etc. hence reducing their impact on the infrastructure of the networks.

Utilisation of Inspektor Gadget

Observing Kubernetes clusters requires awareness of their behavior and activity to support proper management processes. The principal character showed here is Inspektor Gadget; he is a significant figure in this regard as he facilitates our monitoring tool with a vast array of tools to get the job done as intended above. Through the strategic deployment of Inspektor Gadget's capabilities, we are able to gather in-depth data pertaining to various aspects of the cluster, including:Through the strategic deployment of Inspektor Gadget's capabilities, we are able to gather in-depth data pertaining to various aspects of the cluster, including:

- Network Policies: This kind of approach to analyzing the policies helps us to get a better picture of how traffic in the cluster is controlled.
- Resource Profiles: Having detailed resource profiles of applications made it easy to fine tune resource usage by analyzing the consumption patterns and looking for the ways to improve them.
- Snapshots: These periodic updates remove the de facto state from the system, and make it much easier to perform a retrospective analysis of a certain event that may have hindered the cluster.
- System Events: One performs a detailed analysis of system events which is useful to estimate the cluster health and its resistance to disturbances.

Additionally, Inspektor Gadget helps to filter relevant useful information in a simple and clear way from the Multi Tiered Kubernetes space. Coupled with the automatic mapping capability, you bring together low-level Kernel primitives and high-level Kubernetes resources. Democratically, this graceful solution makes our monitoring tool capable of quickly detecting various performance problems and issues in a cluster so as to be able to act proactively in order to maintain the right running of a cluster.Indeed, with the help of various gadgets provided by Inspektor Gadget , we can easily organize our monitoring tool in a way that will fit specific needs for monitoring the performance of various Kubernetes deployments that we own. This assurance of a highly optimized granular and strategic view of the internal system processes will indeed help in the development of better and efficient systems in cloud-native clusters.

6. Implementation Details

6.1 Data Collection Module

- **Deployment of eBPF Probes:**

  KuMon leverages eBPF probes for the extraction of performance data at the kernel level. These probes are strategically deployed within Kubernetes clusters to capture relevant data points and events.

- **Custom eBPF Programs:**

  Custom eBPF programs are developed to target specific performance metrics, such as CPU usage, memory consumption, and network activity. These programs are loaded into eBPF probes and executed in response to predefined events.

- **Real-Time Data Collection:**

  KuMon ensures that data collection is continuous and real-time. This approach guarantees that performance metrics are consistently updated and readily available for analysis. Importantly, eBPF probes perform this data capture with minimal overhead on the cluster.

6.2 Data Visualization and Analysis Module

- **Integration with Kibana**

  KuMon seamlessly integrates with Kibana, a powerful open-source data visualisation tool. Kibana serves as the user interface for monitoring data, providing a versatile range of visualisation options.

- **Custom Dashboards**

  KuMon allows users to create custom dashboards within Kibana to monitor specific aspects of their Kubernetes clusters. These dashboards are fully customizable, empowering users to tailor their monitoring experience according to their unique needs.

6.3 Alerting and Notification

- **Alerting Mechanisms**

  KuMon features customizable alerting mechanisms that notify stakeholders when predefined thresholds or conditions are met. These alerts can be configured based on specific thresholds via Prometheus

- **Notification Channels**

  To ensure widespread and timely communication, KuMon supports various notification channels. These channels encompass email, SMS, and integrations with collaboration platforms like Slack.

- **Cilium and Hubble**

  Cilium, with its Hubble observability layer, provides detailed network metrics that are crucial for Kubernetes networking which are then visualised in Grafana.

6.4 Iteration 2: Updated Use Cases, System Sequence Diagrams, & Sequence Diagrams

6.4.1 Use Cases

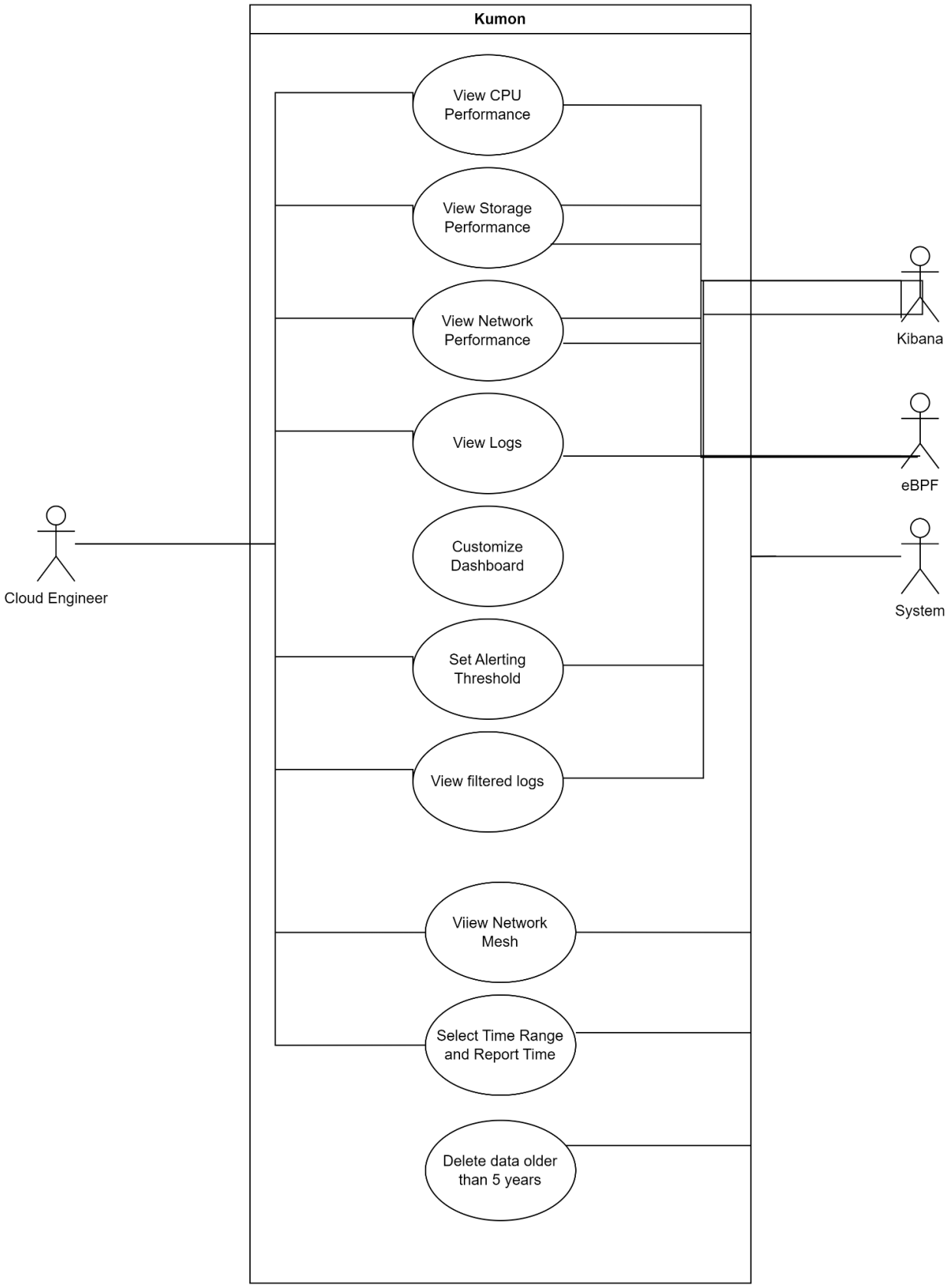


Figure 8: KuMon Use Case Overview

6.5 High Level Use Cases

6.5.1. CPU Monitoring

Use Case	CPU monitoring
Actor	Cloud Engineer
Type	Primary
Description	User should be able to view the CPU performance insights

6.5.2. Storage Monitoring

Use Case	Storage monitoring
Actor	Cloud Engineer
Type	Primary
Description	User should be able to view the storage performance insights

6.5.3. Network Monitoring

Use Case	Network monitoring
Actor	Cloud Engineer
Type	Primary
Description	User should be able to view the network performance insights

6.5.4. Audit Logs

Use Case	Audit Logs
Actor	Cloud Engineer
Type	Primary
Description	User should be able to view CPU, storage and network logs

6.5.5. Send Alerts

Use Case	Send Alerts
Actor	Cloud Engineer
Type	Primary
Description	User should be able to receive automatic alerts for storage performance issues.

6.5.6. View Filtered Logs

Use Case	View Filtered Logs
Actor	Cloud Engineer
Type	Primary
Description	User should be able to filter and view logs based on specific criteria.

6.5.7. Customise Dashboard

Use Case	Customize Dashboard
Actor	Cloud Engineer
Type	Primary
Description	User should be able to customize dashboard to monitor relevant insights

6.5.8. Delete Old Data

Use Case	Delete Data Older Than 5 Years
Actor	Cloud Engineer
Type	Primary
Description	User should be able to delete data that is older than five years from the storage system.

6.5.9. Network Observability through Cilium

Use Case	Network observability through Cilium
Actor	Cloud Engineer
Type	Primary
Description	Provides deep visibility into network traffic within the Kubernetes cluster.

6.5.10. Cluster Health Monitoring

Use Case	Prometheus cluster health monitoring
Actor	Site Reliability Engineer (SRE), DevOps Engineer
Type	Primary
Description	Scrapes metrics on application performance, resource utilization, and Kubernetes cluster health from Cilium

6.5.9. Network Mesh visibility through Hubble

Use Case	Network observability through Hubble
Actor	Cloud Engineer
Type	Primary
Description	Extends Cilium's functionality to provide deeper insights into network flows.

6.6 Extended Use Cases

6.6.1. CPU Monitoring

<table>
<tr><td>Use case</td><td>CPU Monitoring</td></tr>
<tr><td>Scope</td><td>The system under design</td></tr>
<tr><td>Primary Actor</td><td>Cloud Engineer</td></tr>
<tr><td>Stakeholders</td><td>Cloud Engineer</td></tr>
<tr><td>Post-Conditions</td><td>Show CPU performance with updated values</td></tr>
<tr><td>Main Success Scenario</td><td><table><tr><td>Actor</td><td>System</td></tr><tr><td>1. Selects “CPU” from dropdown

3. Selects filters</td><td>2. All update CPU graphs shown

4. Filtered graphs shown</td></tr></table></td></tr>
<tr><td>Extensions</td><td>1. Blank graphs when Kumon is run for the first time</td></tr>
</table>

6.6.2. Storage Monitoring

<table>
<tr><td>Use case</td><td>Storage Monitoring</td></tr>
<tr><td>Scope</td><td>The system under design</td></tr>
<tr><td>Primary Actor</td><td>Cloud Engineer</td></tr>
<tr><td>Stakeholders</td><td>Cloud Engineer</td></tr>
<tr><td>Post-Conditions</td><td>Show storage performance with updated values</td></tr>
<tr><td>Main Success Scenario</td><td><table><tr><td>Actor</td><td>System</td></tr><tr><td>1. Selects “Storage” from dropdown

3. Selects filters</td><td>2. All update Storage graphs shown

4. Filtered graphs shown</td></tr></table></td></tr>
<tr><td>Extensions</td><td>1. Blank graphs when Kumon is run for the first time</td></tr>
</table>

6.6.3. Network Monitoring

Use case	Storage Monitoring
Scope	The system under design
Primary Actor	Cloud Engineer
Stakeholders	Cloud Engineer
Post-Conditions	Show storage performance with updated values
Main Success Scenario	<table><tr><th>Actor</th><th>System</th></tr><tr><td>1. Selects “Network” from dropdown 3. Selects filters</td><td>2. All update Network graphs shown 4. Filtered graphs shown</td></tr></table>
Extensions	1. Blank graphs when Kumon is run for the first time

6.6.4. Audit Logs

Use case	Audit Logs
Scope	The system under design
Primary Actor	Cloud Engineer
Stakeholders	Cloud Engineer
Post-Conditions	Display a table of CPU, Storage, and network logs
Main Success Scenario	<table><tr><th>Actor</th><th>System</th></tr><tr><td>1. Select any performance metric from dropdown</td><td>2. Show the logs of the selected performance metric</td></tr></table>
Extensions	1. Blank table when Kumon is run for the first time

6.6.5. Send Alerts

<table>
<tr><td>Use case</td><td>Send Alerts</td></tr>
<tr><td>Scope</td><td>The system under design</td></tr>
<tr><td>Primary Actor</td><td>Cloud Engineer</td></tr>
<tr><td>Stakeholders</td><td>Cloud Engineer</td></tr>
<tr><td>Post-Conditions</td><td>User notified about performance issues via a notification</td></tr>
<tr><td>Main Success Scenario</td><td><table><tr><td>Actor</td><td>System</td></tr><tr><td>1. The user receives alerts</td><td>2. System identifies a performance issue</td></tr></table></td></tr>
<tr><td>Extensions</td><td>1. Alerts not sent if feature disabled
2. Delayed alerts if system undergoing maintenance</td></tr>
</table>

6.6.6. View Filtered Logs

<table>
<tr><td>Use case</td><td>View Filtered Logs</td></tr>
<tr><td>Scope</td><td>The system under design</td></tr>
<tr><td>Primary Actor</td><td>Cloud Engineer</td></tr>
<tr><td>Stakeholders</td><td>Cloud Engineer</td></tr>
<tr><td>Post-Conditions</td><td>View filtered logs with the help of a query</td></tr>
<tr><td>Main Success Scenario</td><td><table><tr><td>Actor</td><td>System</td></tr><tr><td>1. Filter logs based on several conditions</td><td>2. Show filtered logs</td></tr></table></td></tr>
<tr><td>Extensions</td><td>1. No logs match the condition
2. Error in query</td></tr>
</table>

6.6.7. Customize Dashboard

Use case	Customize Dashboard
Scope	The system under design
Primary Actor	Cloud Engineer
Stakeholders	Cloud Engineer
Post-Conditions	Dashboard gets updated with changes made by user
Main Success Scenario	<table><tr><th>Actor</th><th>System</th></tr><tr><td>1. Add, remove or edit element in the dashboard</td><td>2. Saves the changes and updates the dashboard</td></tr></table>
Extensions	

6.6.8. Delete Data Older than 5 Years

Use case	Delete 5 year old data
Scope	The system under design
Primary Actor	Cloud Engineer
Stakeholders	Cloud Engineer
Post-Conditions	5 year old data gets deleted
Main Success Scenario	<table><tr><th>Actor</th><th>System</th></tr><tr><td></td><td>1. System identifies 5 year old data 2. Identified data gets deleted</td></tr></table>
Extensions	

6.6.9. Network Observability through Cilium

<table>
<tr><td>Use Case</td><td>Network Observability through Cilium</td></tr>
<tr><td>Scope</td><td>The system under design</td></tr>
<tr><td>Primary Actor</td><td>Cloud Engineer</td></tr>
<tr><td>Stakeholders</td><td>Cloud Engineer</td></tr>
<tr><td>Post-Conditions</td><td>There is enhanced network visibility, secure communication between containers</td></tr>
<tr><td>Main Success Scenario</td><td>
<table>
<tr><th>Actor</th><th>System</th></tr>
<tr><td>2. The Cloud Engineer gets improved network visibility</td><td>1. Enforced network policies, and secure communication</td></tr>
</table>
</td></tr>
<tr><td>Extensions</td><td>Integration with security information and event management (SIEM) tools for advanced threat detection.</td></tr>
<tr><td>Frequency of Occurrence</td><td>As performance issues occur.</td></tr>
</table>

6.6.9. Network Observability through Cilium

<table>
<tr><td>Use Case</td><td>Network Observability through Hubble</td></tr>
<tr><td>Scope</td><td>The system under design</td></tr>
<tr><td>Primary Actor</td><td>Cloud Engineer</td></tr>
<tr><td>Stakeholders</td><td>Cloud Engineer</td></tr>
<tr><td>Post-Conditions</td><td>Deeper understanding of network behavior</td></tr>
<tr><td>Main Success Scenario</td><td>
<table>
<tr><th>Actor</th><th>System</th></tr>
<tr><td>2. Gain insights into application network communication patterns</td><td>1. Quickly identify the root cause of network latency or connectivity problems affecting applications.</td></tr>
</table>
</td></tr>
<tr><td>Extensions</td><td>Traffic mirroring for capturing and analyzing network packets in real-time, integration with network visualization tools for a more comprehensive view.</td></tr>
<tr><td>Frequency of Occurrence</td><td>As performance issues occur.</td></tr>
</table>

6.6.9. Cluster Health Monitoring

<table>
<tr><td>Use Case</td><td>Application and Cluster Performance Monitoring with Prometheus</td></tr>
<tr><td>Scope</td><td>The system under design</td></tr>
<tr><td>Primary Actor</td><td>Cloud Engineer</td></tr>
<tr><td>Stakeholders</td><td>Cloud Engineer</td></tr>
<tr><td>Post-Conditions</td><td>Consolidated view of application and cluster health, ability to set alerts and trigger notifications for performance issues.</td></tr>
<tr><td>Main Success Scenario</td><td>
<table>
<tr><th>Actor</th><th>System</th></tr>
<tr><td>2.Gain insights into application behavior and resource consumption for better planning</td><td>1. Identify application performance issues</td></tr>
</table>
</td></tr>
<tr><td>Extensions</td><td>Monitoring custom metrics specific to your applications, integration with chat platforms for real-time alerts.</td></tr>
<tr><td>Frequency of Occurrence</td><td>As performance issues occur.</td></tr>
</table>

6.7 ERD

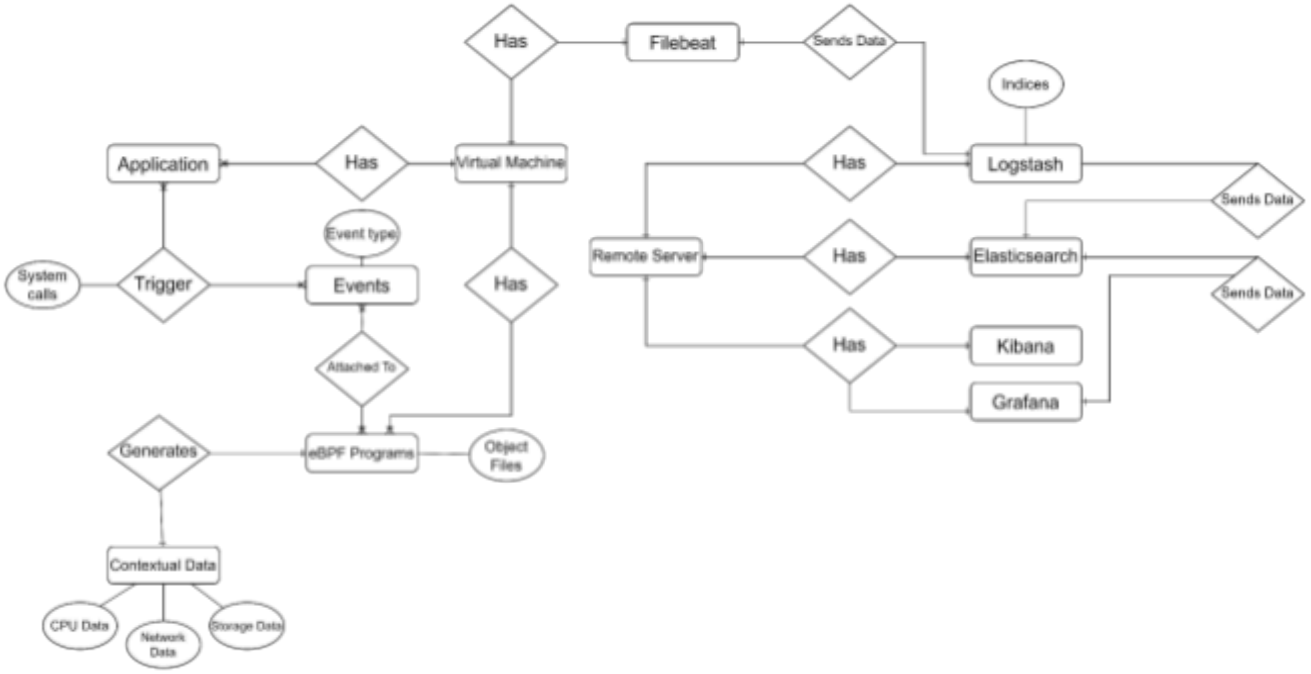
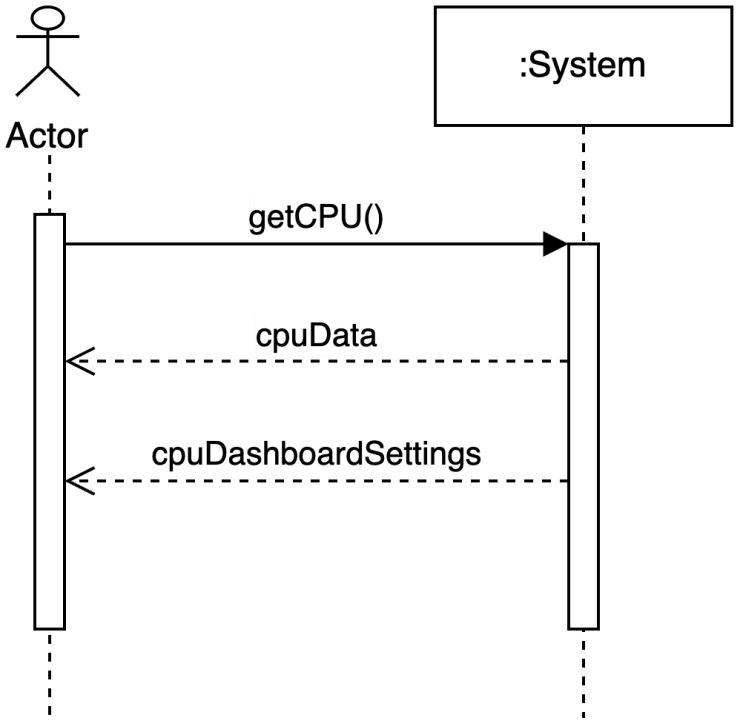


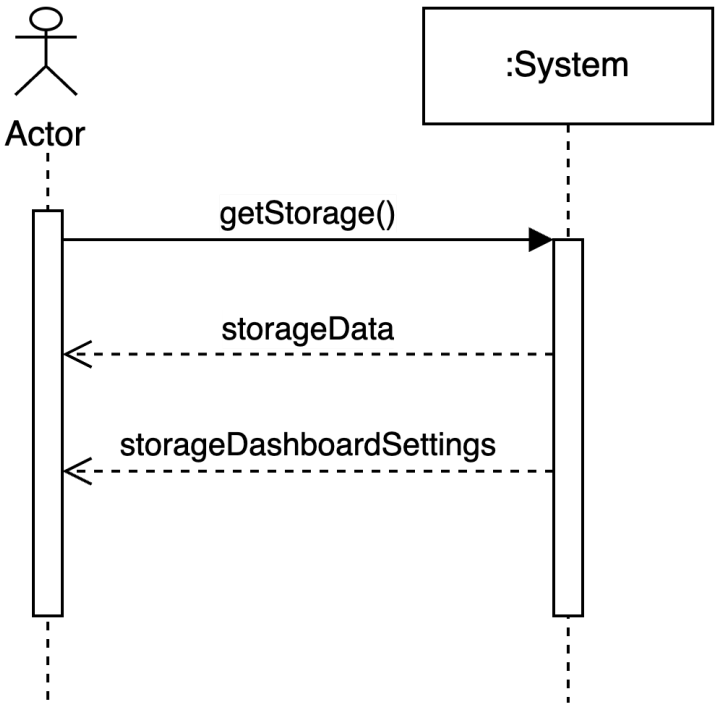
Figure 9: Entity Relation Diagram

6.8 SSD

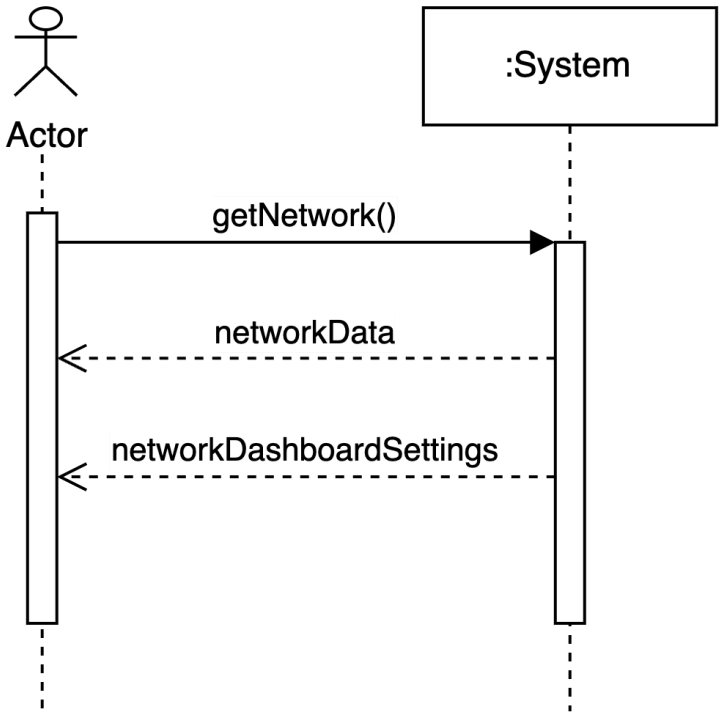
6.8.1. View CPU Performance



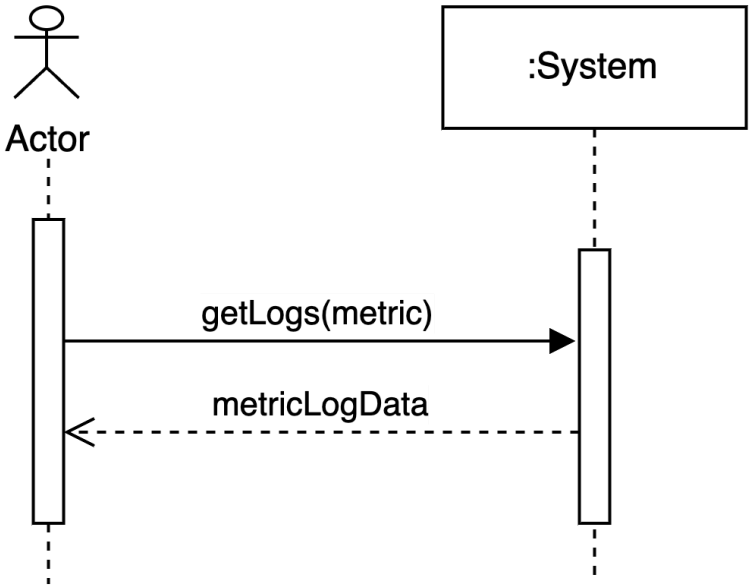
6.8.2. View System Storage Performance



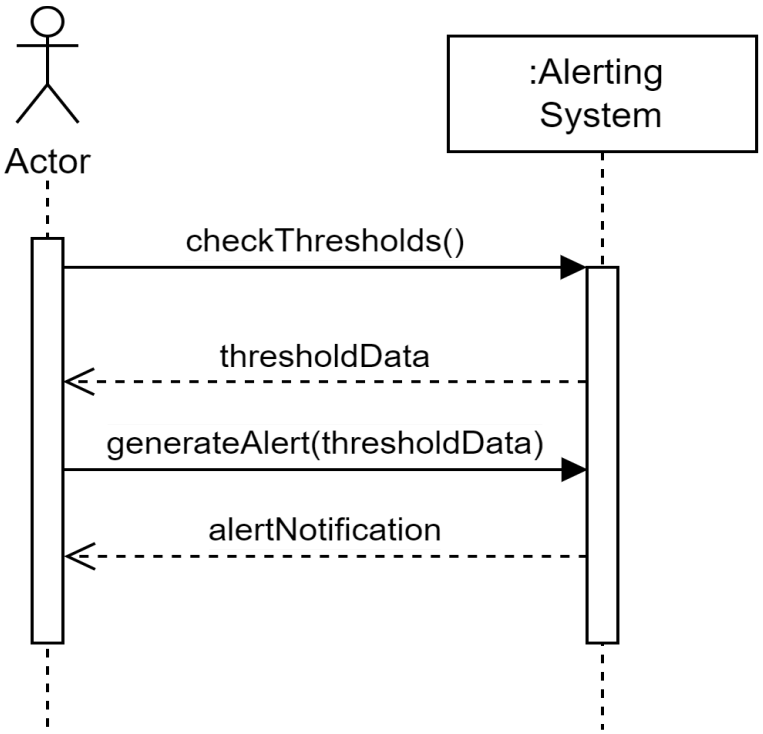
6.8.3. View Network Performance Details



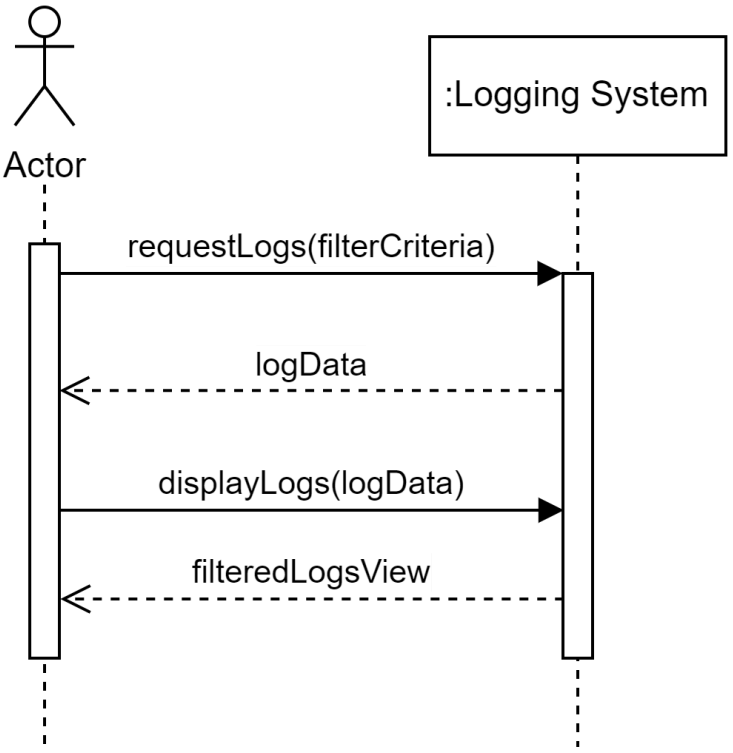
6.8.4. Audit System Logs



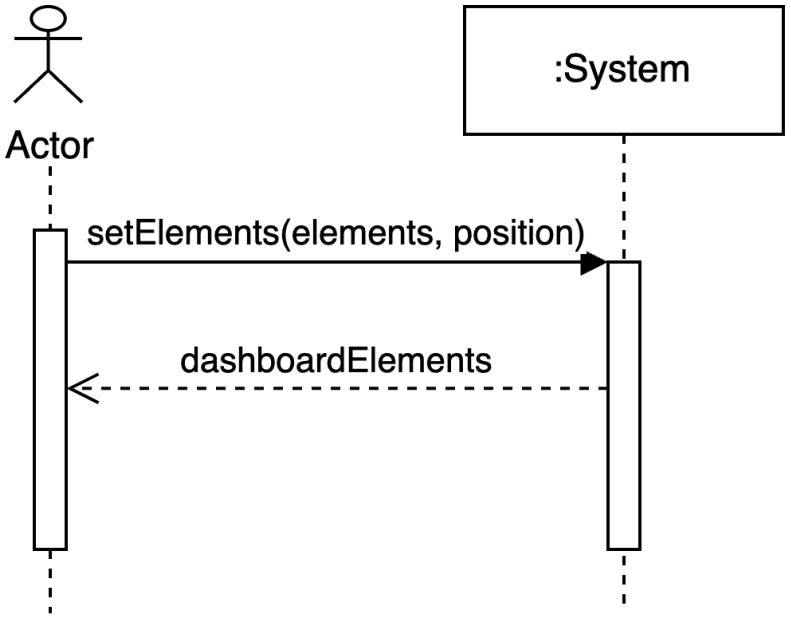
6.8.5. Send Performance Alerts via Prometheus



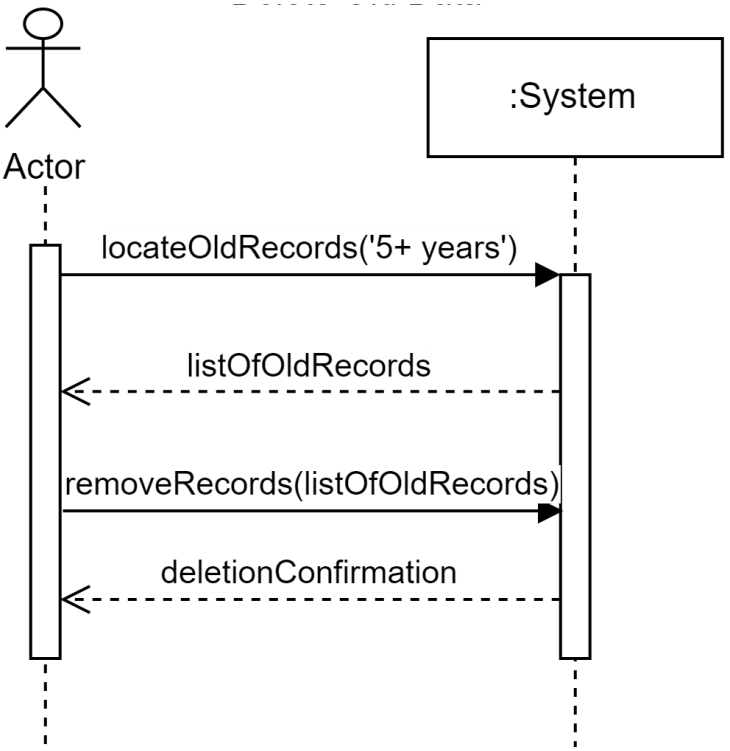
6.8.6. View Filtered Logs



6.8.7. Customise Dashboard

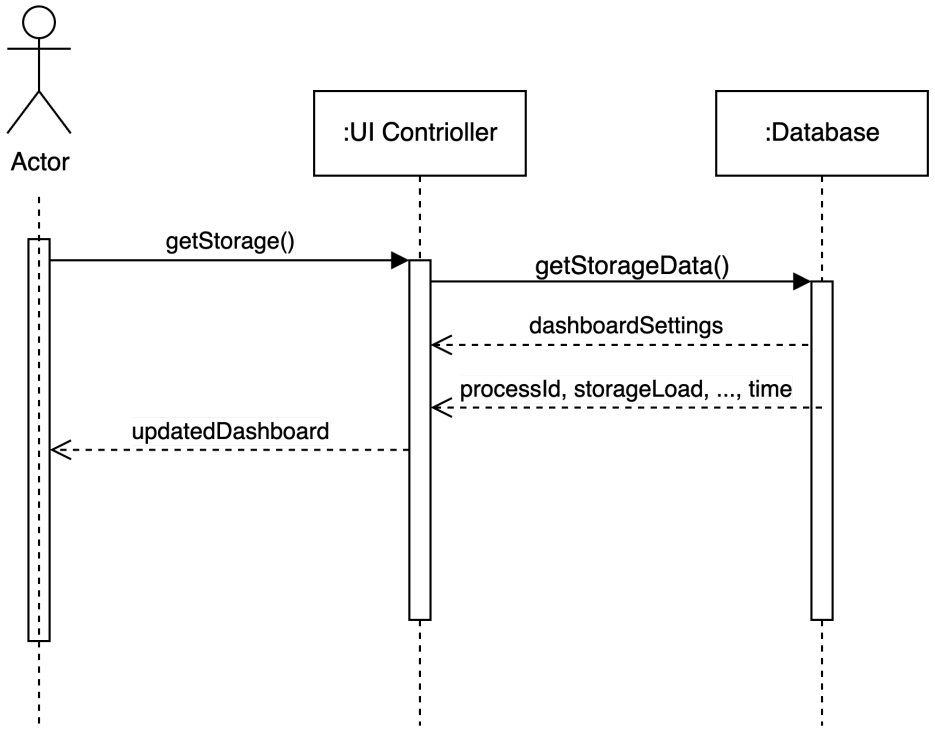


6.8.8. Delete Old Data



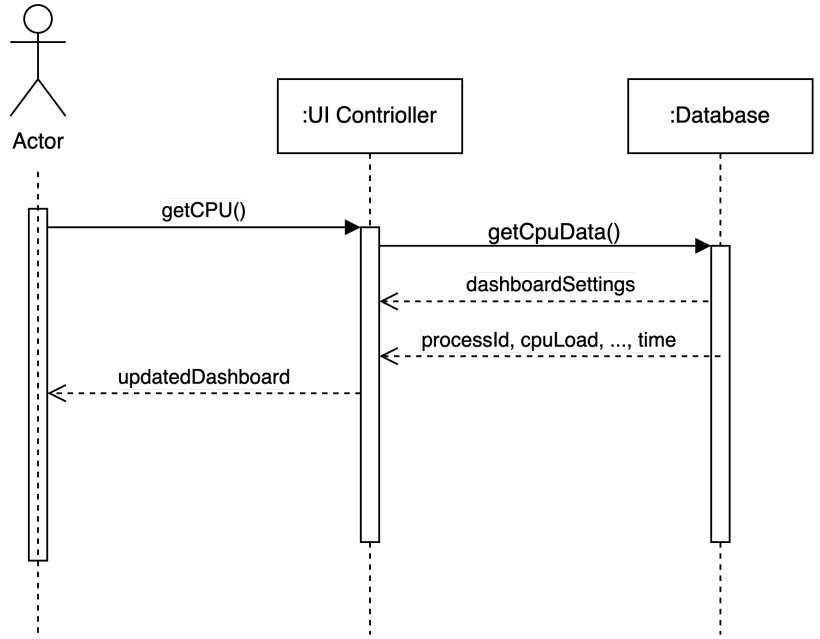
6.9 Sequence Diagrams

6.9.1 View System Storage Performance Details

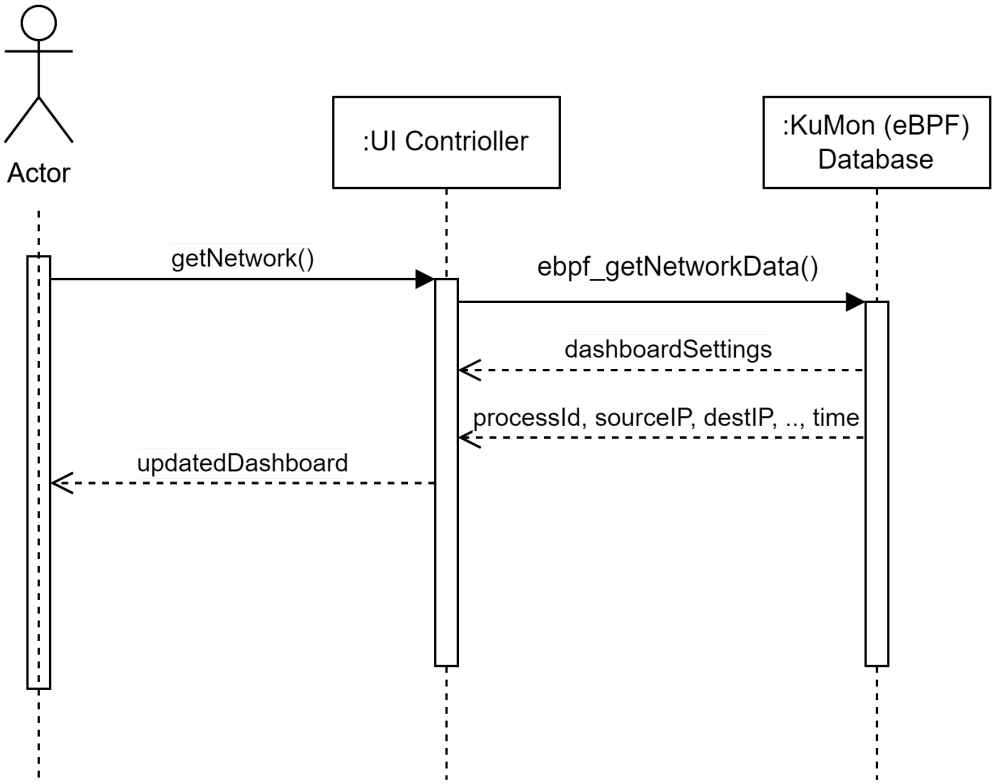


6.9.2 View System CPU Performance Details

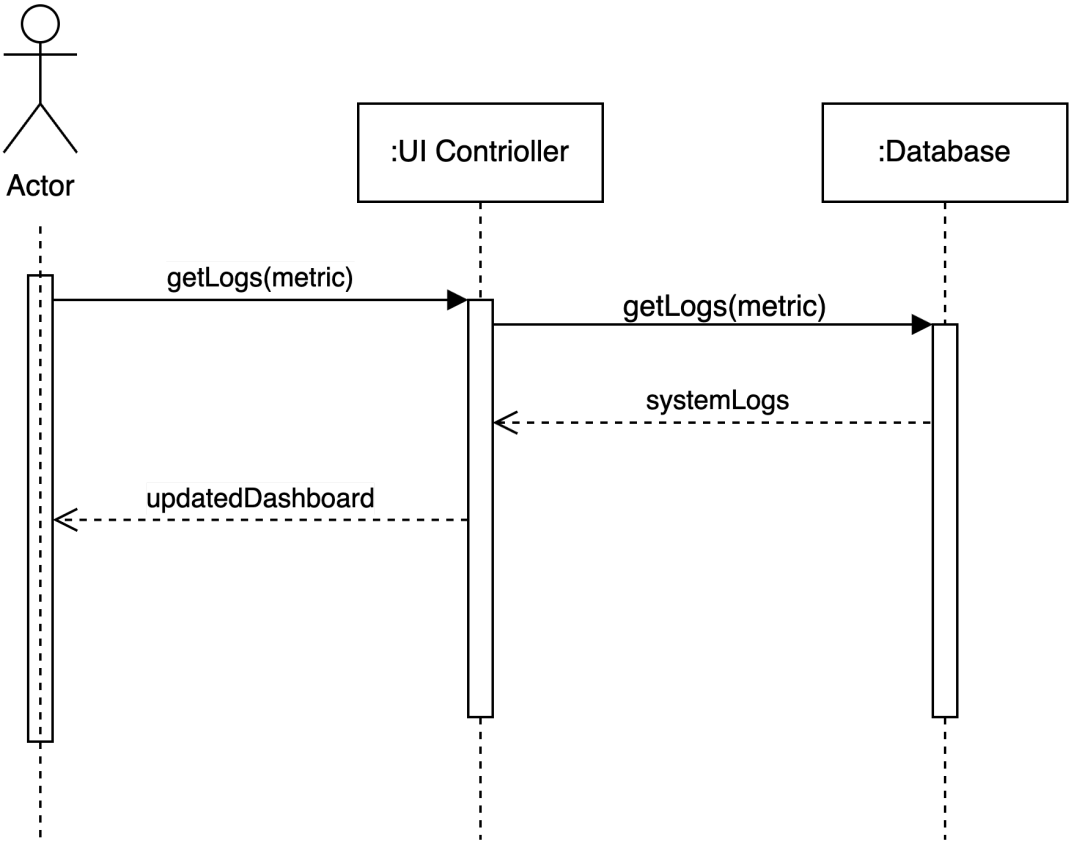
View system CPU performance details



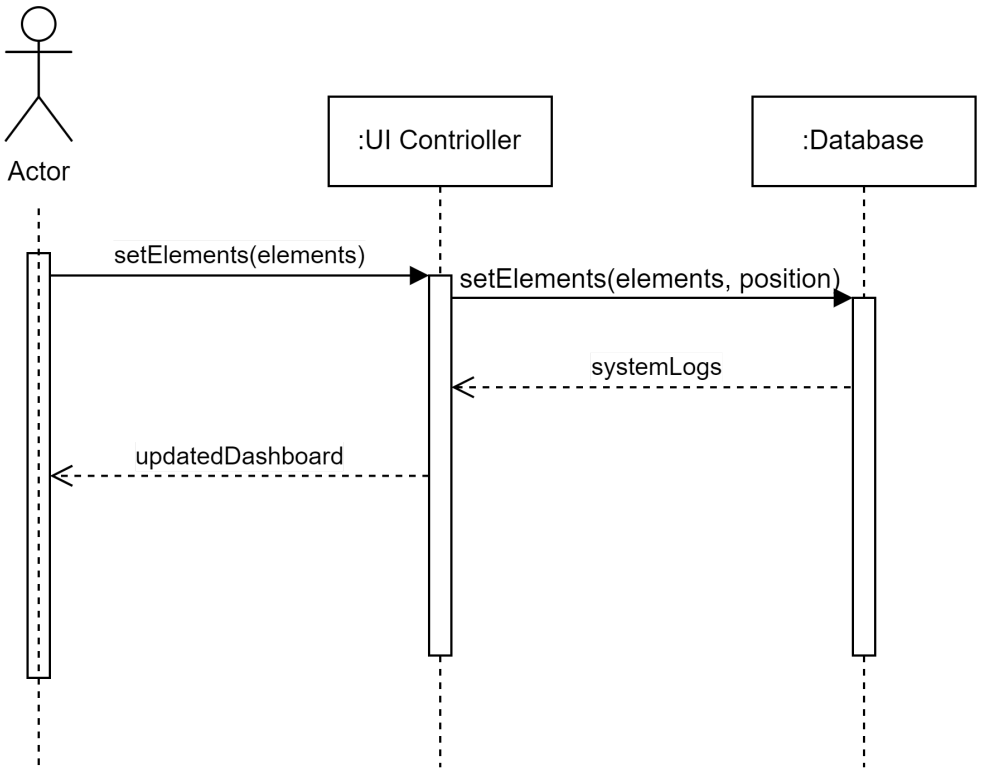
6.9.3 System Network Performance



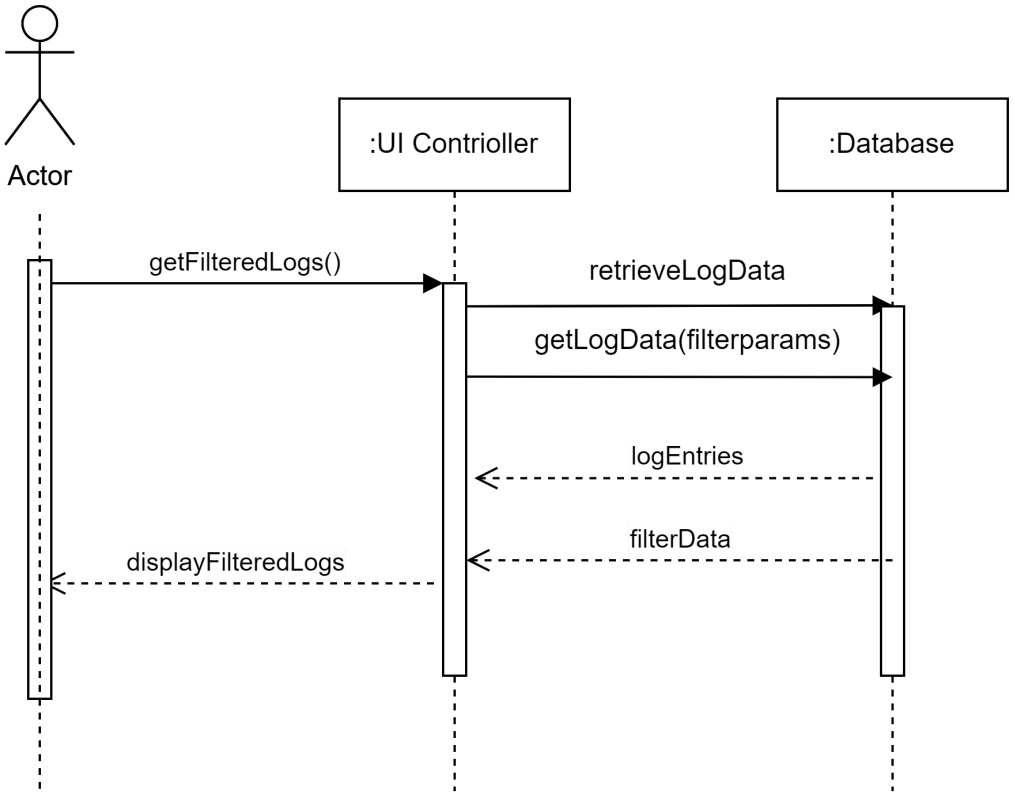
6.9.4 View System Logs



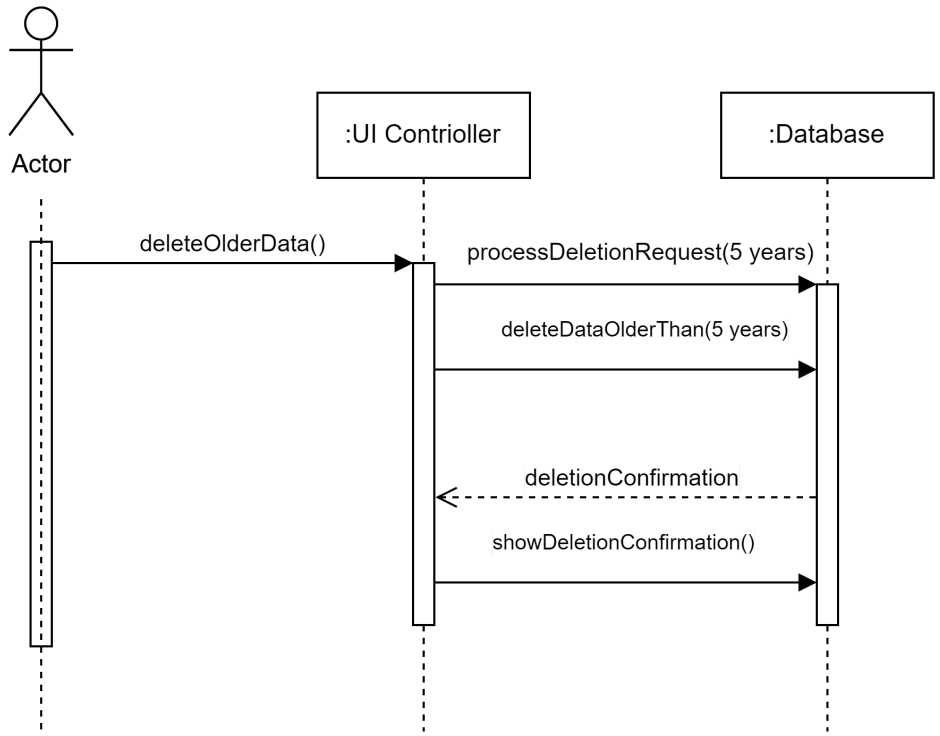
6.9.5 Customise Dashboard



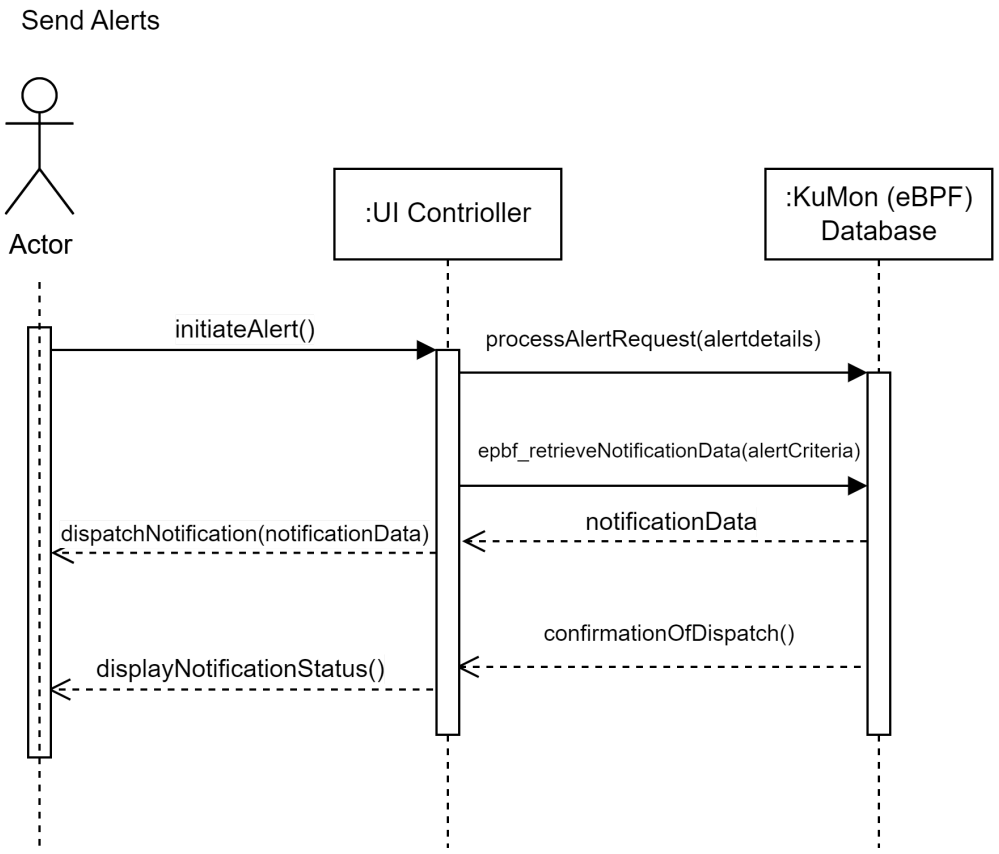
6.9.6 View Filtered Logs



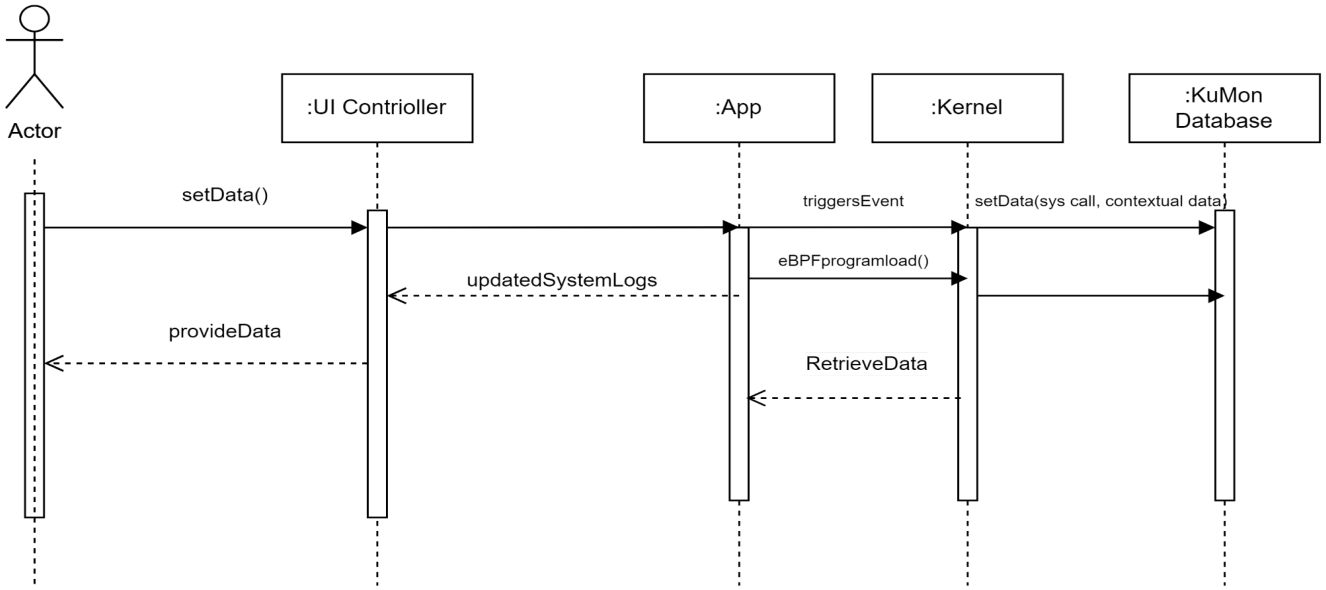
6.9.7 Delete Data Older than 5 years



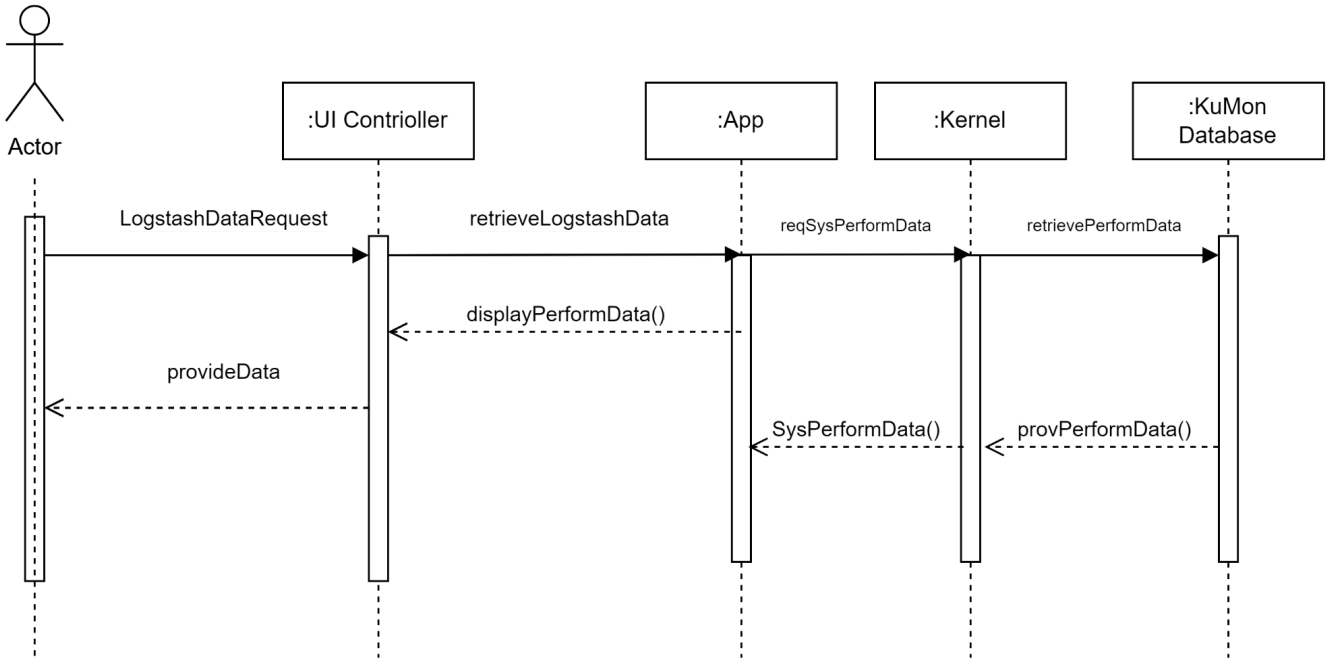
6.9.8 Send Alerts



6.9.9 Data Generation through system calls



6.9.10 Data Integration through Logstash



6.10 Data Flow Diagram

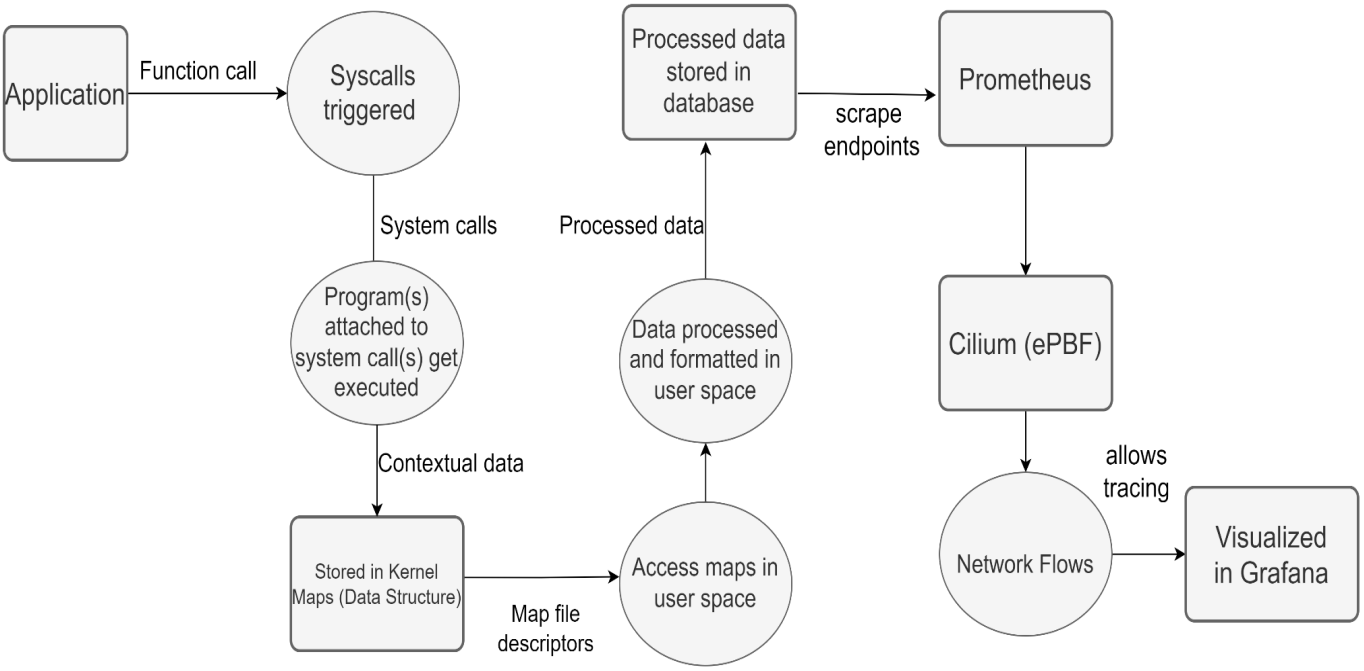


Figure 10: KuMon Data Flow Diagram

6.11 Kumon Architecture

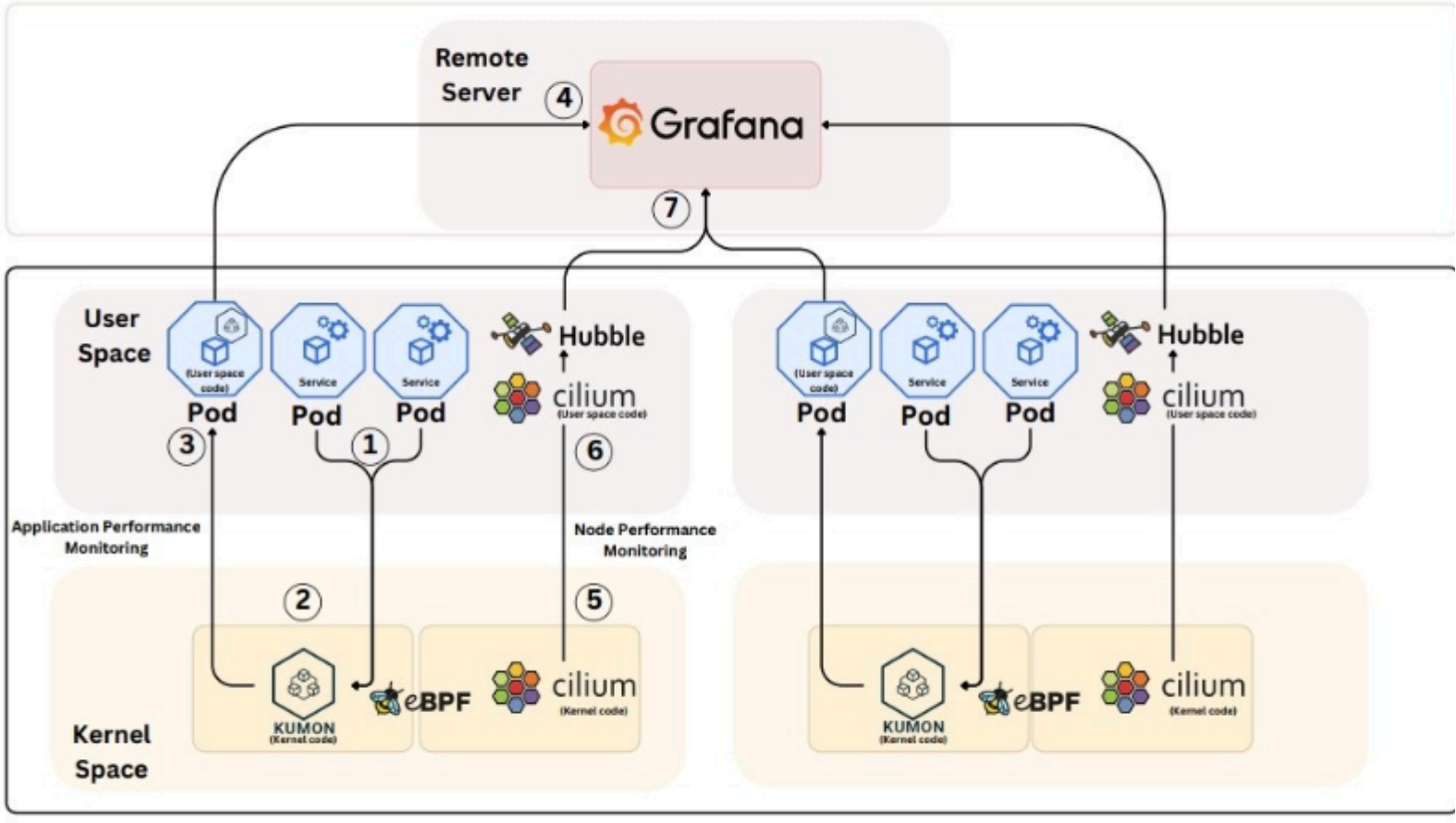


Figure 11: Kumon Architecture

6.12 Process Diagram

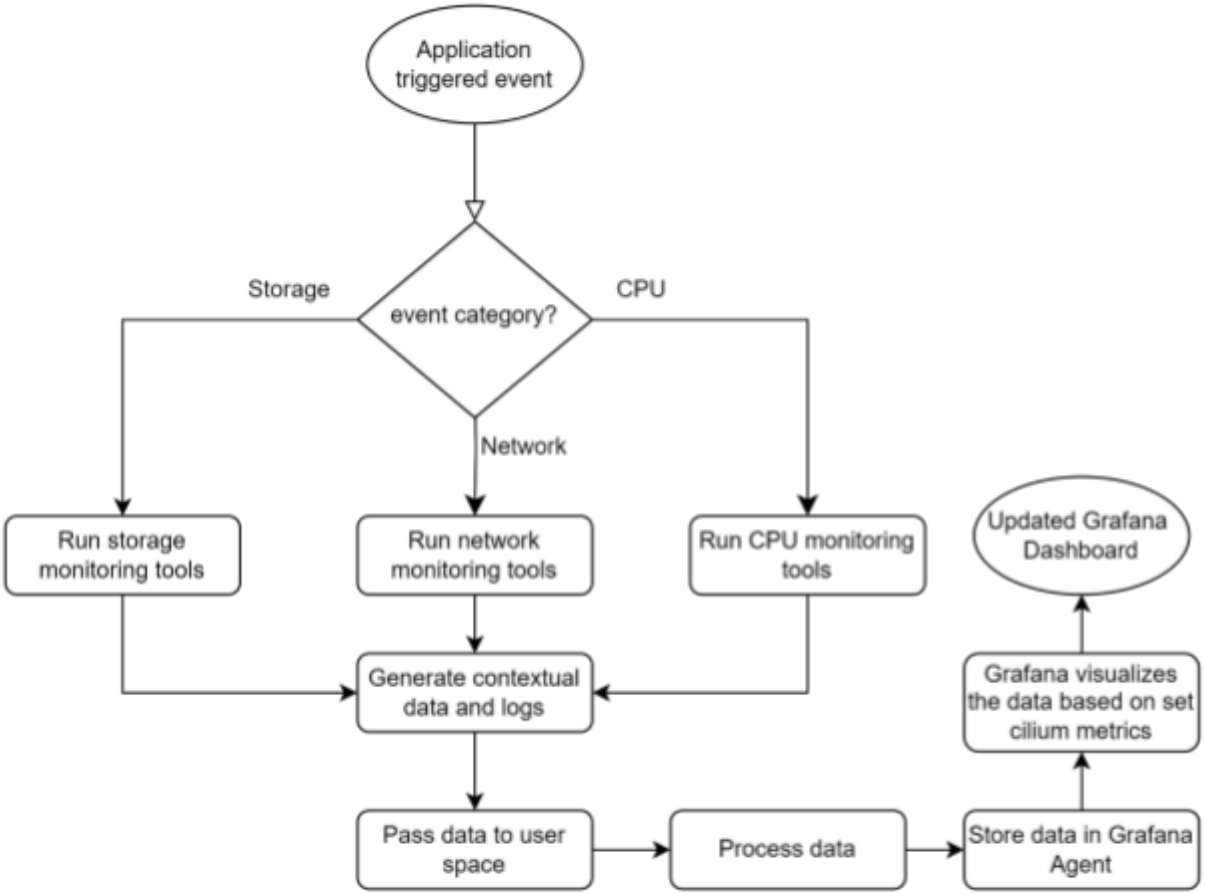


Figure 12: Process Diagram

6.13 Overall implementation

As shown in Figure 11 Kumon architecture is designed to ensure efficient monitoring and optimization across two critical spaces: User Space and Kernel Space. In the User Space, we find several “Pods” (containers or components) responsible for running user-facing applications. These Pods are connected to a central component labeled “Hubble.” The role of the “Hubble” component is to bridge communication between the User Space and a remote “Grafana” server. Within this space, the focus is on Application Performance Monitoring, ensuring that user applications perform optimally. The Kernel Space deals with low-level system operations. Here, two essential components reside, eBPF which allows efficient packet filtering and tracing within the Linux kernel. It plays a crucial role in monitoring network traffic and system behavior.

Cilium contributes to overall system mesh visibility and performanc metrics. In the Kernel Space, the emphasis is on Node Performance Monitoring, ensuring that the underlying nodes (servers or hosts) operate smoothly. The architecture seamlessly integrates both spaces. Grafana, hosted remotely, aggregates data from the User Space and Kernel Space. Operators and administrators can analyze metrics, visualize trends, and make informed decisions based on the insights provided by Grafana.

Data Collection Module:

We have used multiple ebpf scripts for this module. With the help of these scripts we have obtained the following data:

- on and off-CPU time per task
- Trace new processes via exec() syscalls
- Run queue (scheduler) latency
- Trace long process scheduling delays
- Trace TCP sessions and summarize lifespan
- Show per process I/O size
- Summarize block device I/O latency
- Trace the lifespan of short-lived files
- Trace why file is missing (deleted or renamed)
- Trace MySQL/PostgreSQL queries slower than a threshold

eBPF Integration:
KuMon will rely on eBPF (Extended Berkeley Packet Filter) technology to monitor and collect data at the kernel level.
This approach ensures low overhead and high-performance data collection without the need for invasive kernel modifications.

Deployment and Privileges:
KuMon will be deployed with root privileges, as it needs access to kernel system calls for monitoring and data extraction.
It is essential to exercise caution and maintain the security of the system when using root privileges.

System Call Monitoring:
KuMon will register specific system calls within the kernel to extract contextual data. These system calls may include those related to process management, network activity, and resource utilisation.
By intercepting these system calls, KuMon can capture pertinent information during various system events.

Data Mapping:
Extracted data from the registered system calls will be mapped into a custom output format. This custom output may include structured data, such as csv, to facilitate efficient processing and analysis.

Kernel-Space Data Transfer:
KuMon will transfer the custom output data directly from the kernel space to an external server or endpoint. This is a crucial step in ensuring real-time data transmission without involving user space.

Data Visualization Module:

Elasticsearch Integration:
Elasticsearch engine configured to receive data from logstash and transfer to Kibana

Kibana Integration:
Configured to receive data from Elasticsearch for visualization

Logstash Integration:
On the external server, KuMon can use Logstash, a popular log processing tool, to collect, filter, and transform the incoming data streams.
Logstash will be configured to receive the data sent by Filebeat

Filebeat Integration:
Collect data generated by ebpf scripts
Send the data to Logstash in real-time with the help of filewatchers

System Security, Monitoring, and Scalability Measures

User Access and Monitoring:
Access to the Kibana dashboard is restricted to authorized users. Security measures, such as authentication and authorization, are implemented to ensure that sensitive system data is protected.

Monitoring and Maintenance:
Fileabeat runs into an error if **ELK (Elasticseach, Logstash, Kibana)** is not running
Continuous monitoring is ensured with the help of a script that halts Fileabeat if ELK is stopped due to some error
Filebeat is rerun after ELK starts working

Scalability:
The implementation should consider scalability to handle larger volumes of data and more extensive system environments as needed.

By following these implementation details, KuMon will effectively leverage eBPF to provide in-depth, real-time system and network utilization monitoring, facilitating the quick identification of performance issues and high-utilization processes for DevOps and cloud engineers.

7. User Manual

1. Introduction

Welcome to KuMon, the comprehensive performance monitoring tool for Kubernetes clusters powered by eBPF technology. This user manual is your guide to effectively installing, configuring, and using KuMon to monitor your Kubernetes environments. Whether you're a cloud engineer, DevOps professional, system administrator, or application developer, KuMon provides the insights you need to optimize application performance, resource allocation, and network efficiency.

2. Installation Guide

Installation Steps	Detailed Instructions
Clone the Repository	Clone KuMon from GitHub to your local system using the commands: git clone https://github.com/kumon/kumon.git cd kumon
Build with Docker	Use Docker to build KuMon: docker-compose build
Deploy with Docker	Deploy KuMon with the Docker command: docker-compose up -d
Confirm Installation	Verify the installation by accessing the KuMon dashboard at the URL: https://<elastic-search-ip>:5601
General Settings	Configure general preferences in the configuration file in your KuMon installation directory.
Data Collection	Specify the metrics for system calls, network activity, and resource utilization within your Kubernetes cluster.
Data Visualization	Set up Kibana settings to connect with your Kibana instance for data visualization.
Alerting and Notifications	Define thresholds for alerts and configure notification channels like email, SMS, or Slack.
Custom Dashboards	Tailor dashboards to your monitoring needs by creating and modifying them in your KuMon setup.
Accessing the Dashboard	Enter the provided URL in your browser to log in to the KuMon dashboard with your credentials.
Viewing Metrics	View real-time metrics through various charts and tables on the dashboard.
Creating Custom Dashboards	Monitor specific Kubernetes cluster aspects by customizing dashboards.
Setting Alerts	Be notified of performance thresholds with configured alerts.
Analyzing Historical Data	Review historical data for past performance trends and optimization.
Troubleshooting	Consult the troubleshooting section for common installation and usage issues.
Support	For further assistance, contact our support team at i202496@nu.edu.pk.

Table 3: Installation Steps Summarized

Installation Steps:

Clone the Repository

git clone https://github.com/abwqr/ebpf-kumon.git

Setup Instructions

1. Setup ELK Stack with Docker

Navigate to the `docker-elk` directory and run the following commands:

cd docker-elk

docker-compose up setup

docker-compose up

2. Setup Filebeat

Navigate to the `filebeat` directory in the project folder and run the following command. The current image is amd64 compatible.

cd ../filebeat

docker-compose up

3. Install Required Packages

Navigate to the `commands/setup` directory and run the `install.sh` script to install all the required packages.

cd ../../commands/setup

bash install.sh

KuMon is now ready for monitoring your Kubernetes cluster.

Configuration:

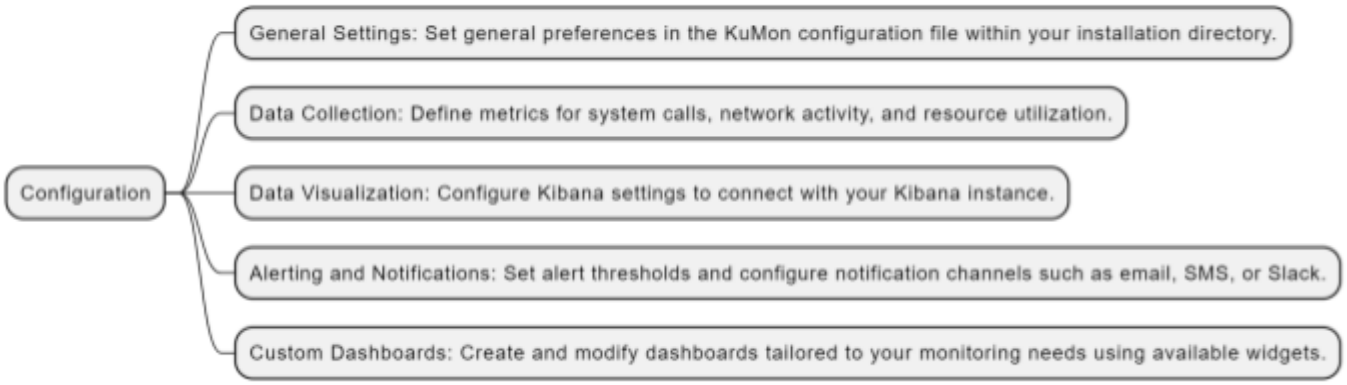


Figure 13: Configuration at a glance

├— **General Settings:** Table 1 above provides a preview of the steps, however to amend the settings at a more granular, specific level based on desired preferences, an editor in 'KuMon' config file can be accessed in your installation directory. This file usually include general features of the system, information related with its operation, as well as preferences from user's side and levels of logging. It is advisable that the specified settings are tweaked according to the needs of the monitoring system in order to maximize the effectiveness of the KuMon system.

├— **Data Collection:** Define general metrics and specific parameters on what to log across different contexts like system calls, network processes and consumed resources. To set up certain parameters for monitoring in KuMon, it is possible to access the options in the Configuration section for setting up specific metrics for monitoring, setting up the intervals of sampling for each parameter, as well as specifying the data sources to be monitored. It is positively flexible in a way that users can adjust this data collection process in anticipation to the monitoring aspects that you consider important.

├── **Data Visualization:** Located in the KuMon > Configuration > Settings tab you should set up the connection between KuMon and your Kibana instance. It involves entering the required connection parameters like the server addresses, the authentication credentials, and the pattern of indexing of supervised data so that the data is easily mapped and visualized in Kibana. Update that mappings and indices are correctly created in the Kibana so that the gathered metrics could be presented and analyzed properly.

├── **Alerting and Notifications:** Implement threshold values instruments within KuMon to establish conditions that correlate to alarms and notifications with defined criteria. Adjust these thresholds based on performance normalization factors and tolerances for critical thresholds of your system. Also, under KuMon, you may suggest the notification channels such as emails, SMS or Slack to require immediate notification as soon as the set filters reach a certain limit. Tweak these notifications so that only the people holding the stakes that demand their attention are notified every time a significant system incident arises.

└── **Custom Dashboards:** You can create new KuMon dashboards to monitor the aspects that is most important to you or modify KuMon dashboards to your specific requirements. Utilize the best features on the available widgets, graphs, and all the other pieces of the display to develop an all-encompassing dashboard that can accommodate and track metrics appropriate to the performance of the system. Organise and structure all of these factors in a way, which would be most comfortable and useful for distinct user types in your organisation, while enabling comprehensive tracking and evaluation.

Usage:

Accessing the Dashboard: Log in to the KuMon dashboard using the URL provided.

Viewing Metrics: Observe real-time metrics on the dashboard.

Creating Custom Dashboards: Customise dashboards for specific cluster monitoring.

Setting Alerts: Get notified about performance thresholds.

Analysing Historical Data: Examine historical data for performance trends.

Troubleshooting and Support

Refer to the troubleshooting section for common issues with installation or usage.

For further assistance, contact the support team at i202496@nu.edu.pk.

Note: The KuMon Installation Guide outlined above, with detailed guide linked in the GitHub repository, allow for the user as show in the Figure 9 access a User Interface for Installation Guide which then redirects to Kibana Home, and the relevant Kibana Dashboard pertinent to requirements of user specified will be displayed. Some of the basic dashboard visualisations can be in the form of bar charts, heat maps and can be customised further with widgets and add ons being displayed in the utility bar on Kibana Section 8. Further showcases some interface examples that show test data visualisation.

8. Kumon User Interface

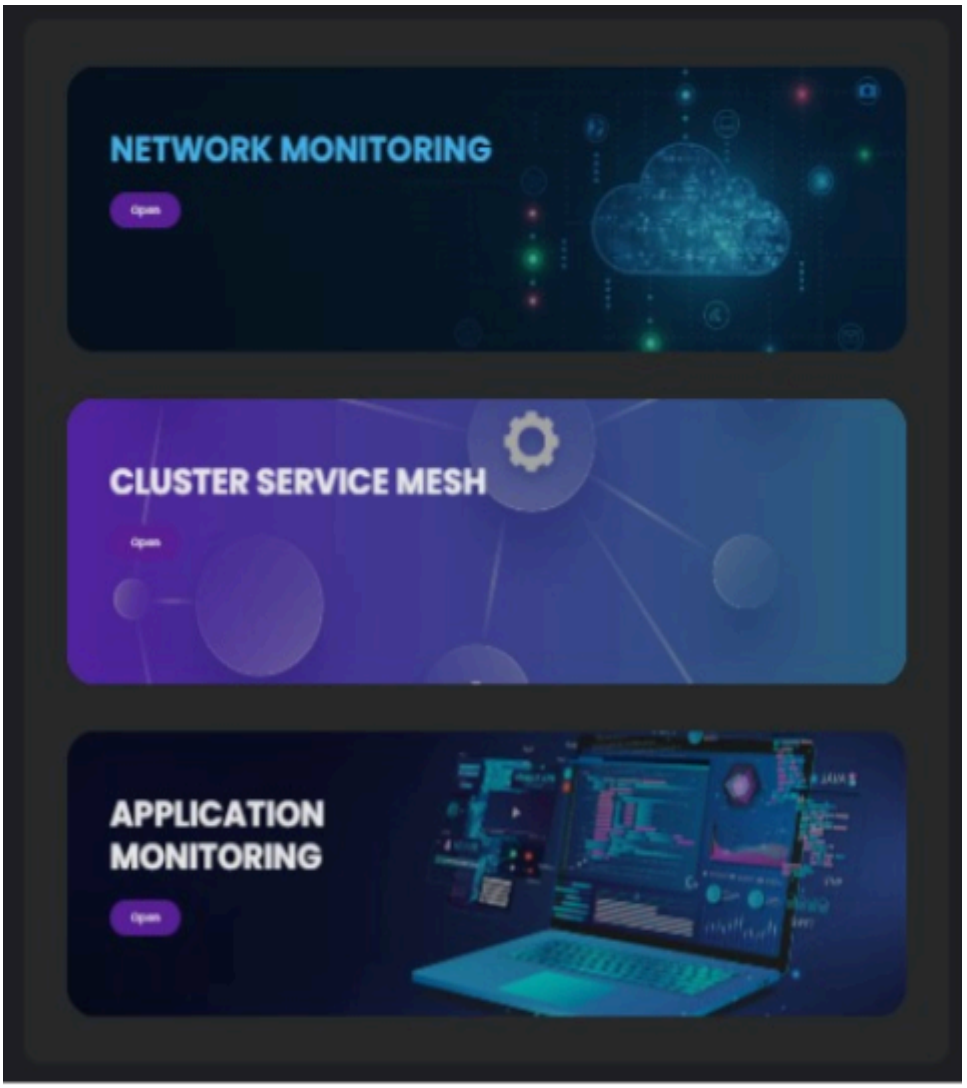


Figure 14. Kumon UI (The respective button redirects to Dashboards)

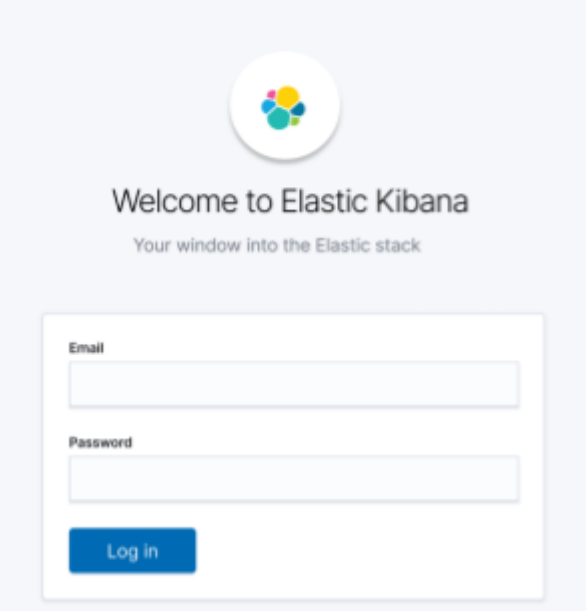


Figure 15: Login Page for Kibana

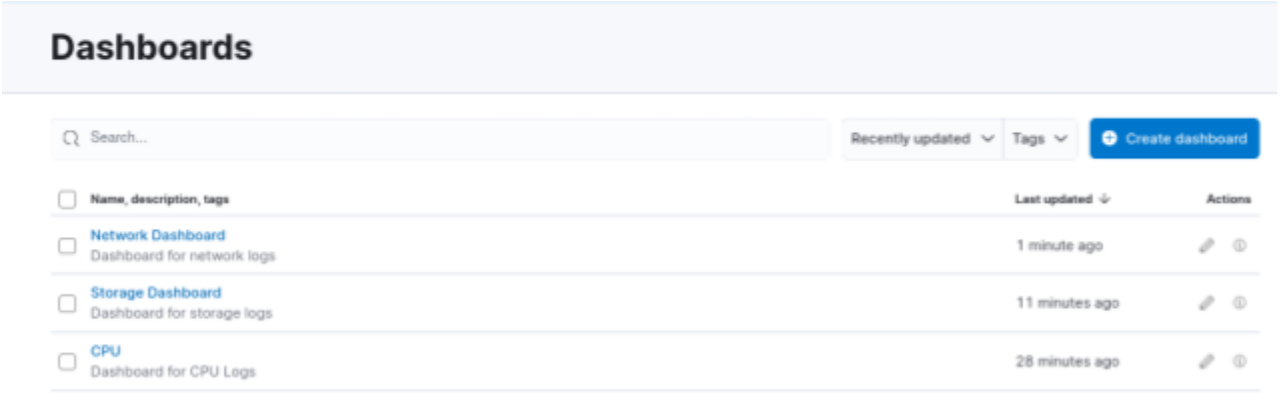


Figure 16: Landing page showing all dashboards

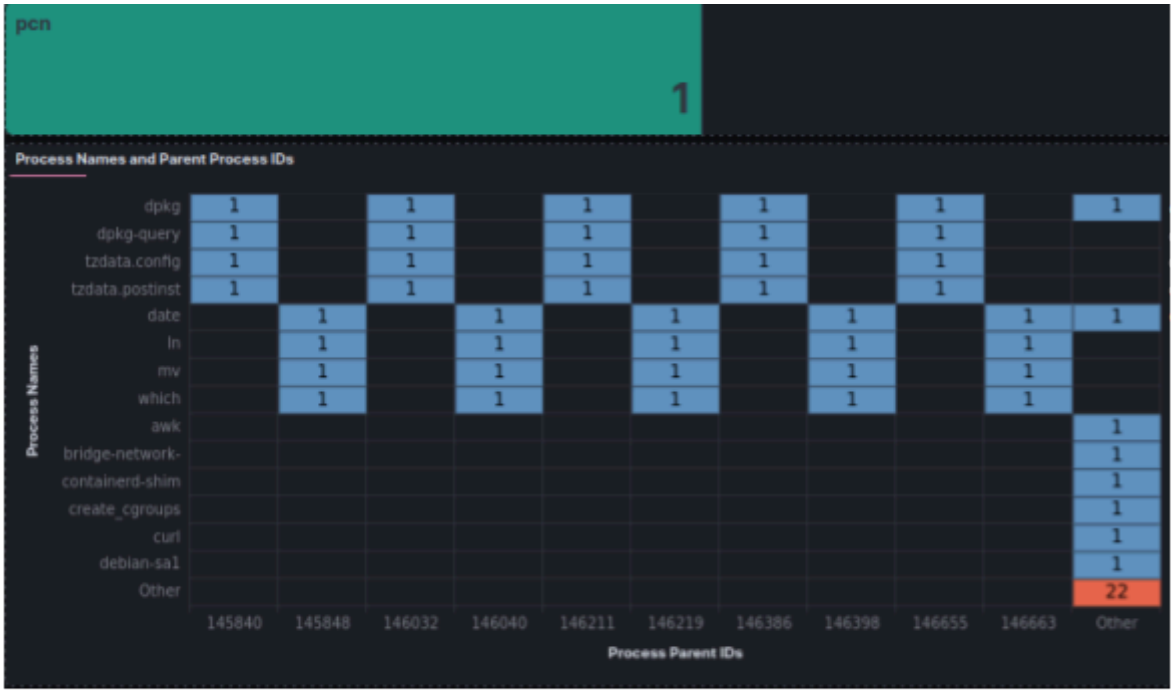


Figure 17: Kibana Visualization for Kumon showing process Names and Parent Process IDs

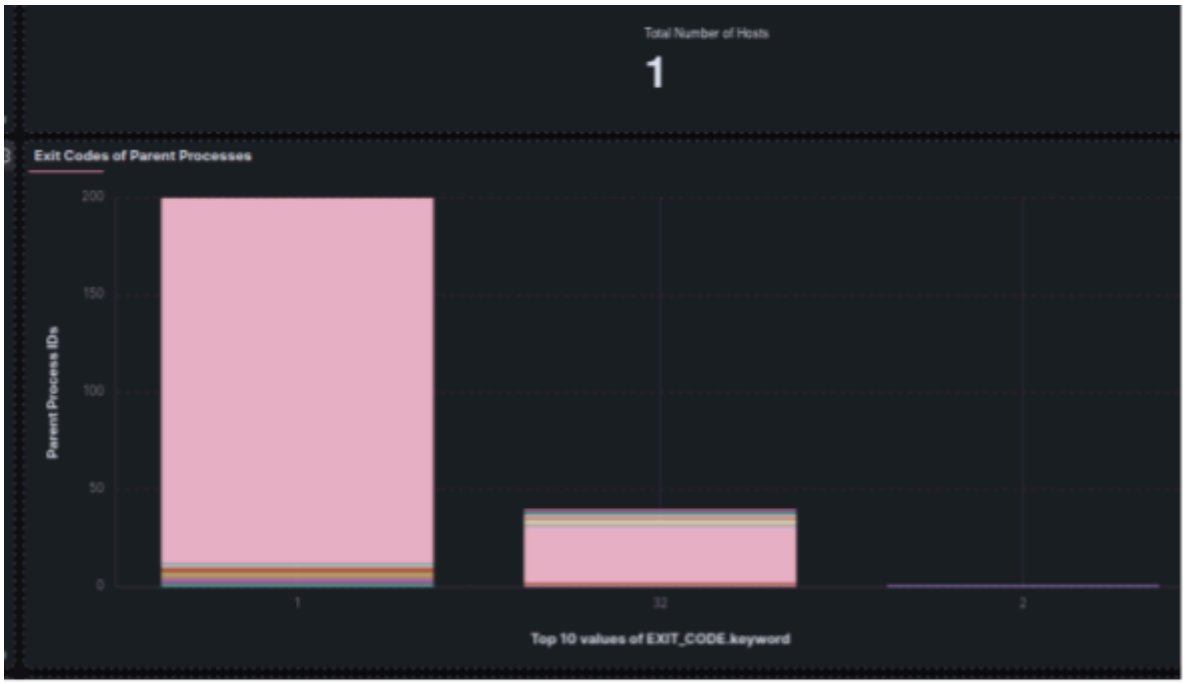


Figure 18: Kibana Visualization for Kumon showing Exit Codes of Parent Processes

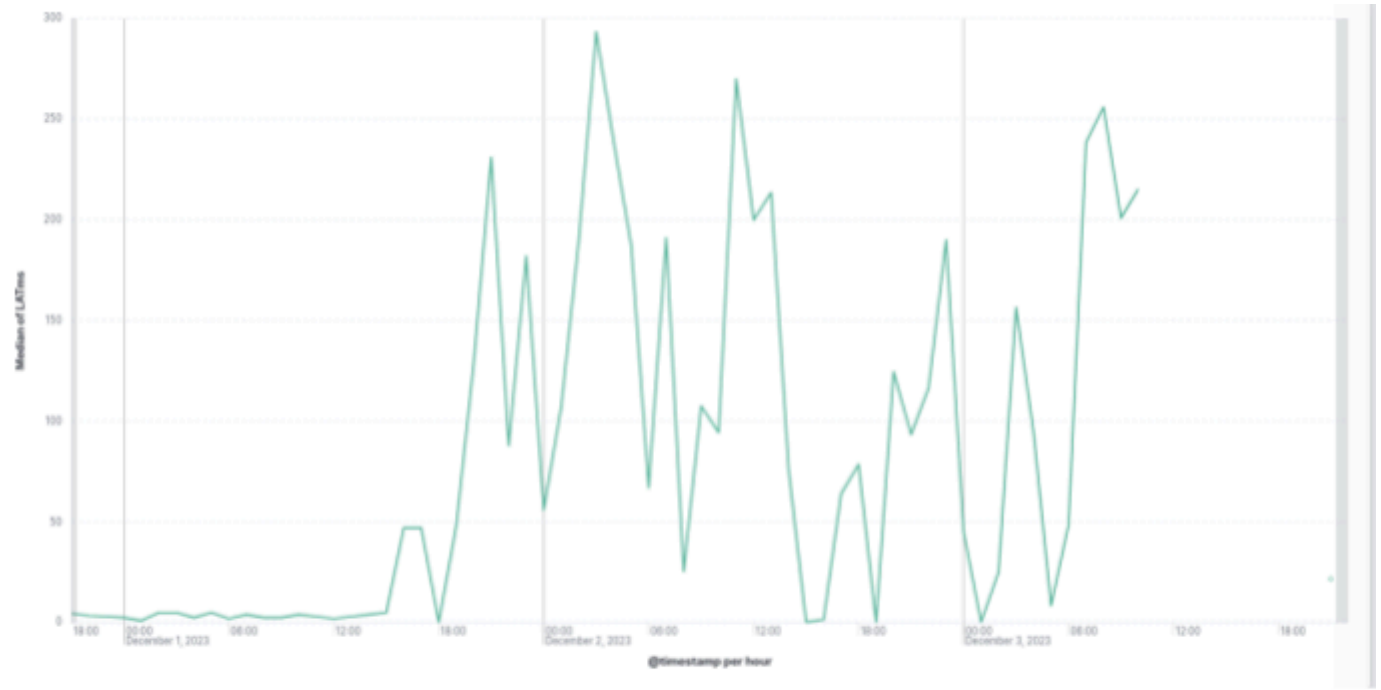


Figure 19: Kibana Visualization for Kumon showing average of latency as a time-series graph

Visualising Key Performance Metrics in Kumon

Figures 21 to 24 as shown below collectively provide a comprehensive view of critical performance metrics within the Kumon system, focusing on data transfer sizes, process times, and read/write operations. Figure 21 depicts the median value of data transfer sizes in bytes, offering insight into typical data loads. Figure 22 s illustrates the average duration of various processes, essential for identifying bottlenecks and inefficiencies. Figure 23 shows the volume of data involved in write operations, crucial for assessing data flow and storage needs. Figure 24 displays the data volume associated with read operations, helping to understand retrieval patterns and resource utilization. Together, these figures enable effective monitoring and optimization of the Kumon system.

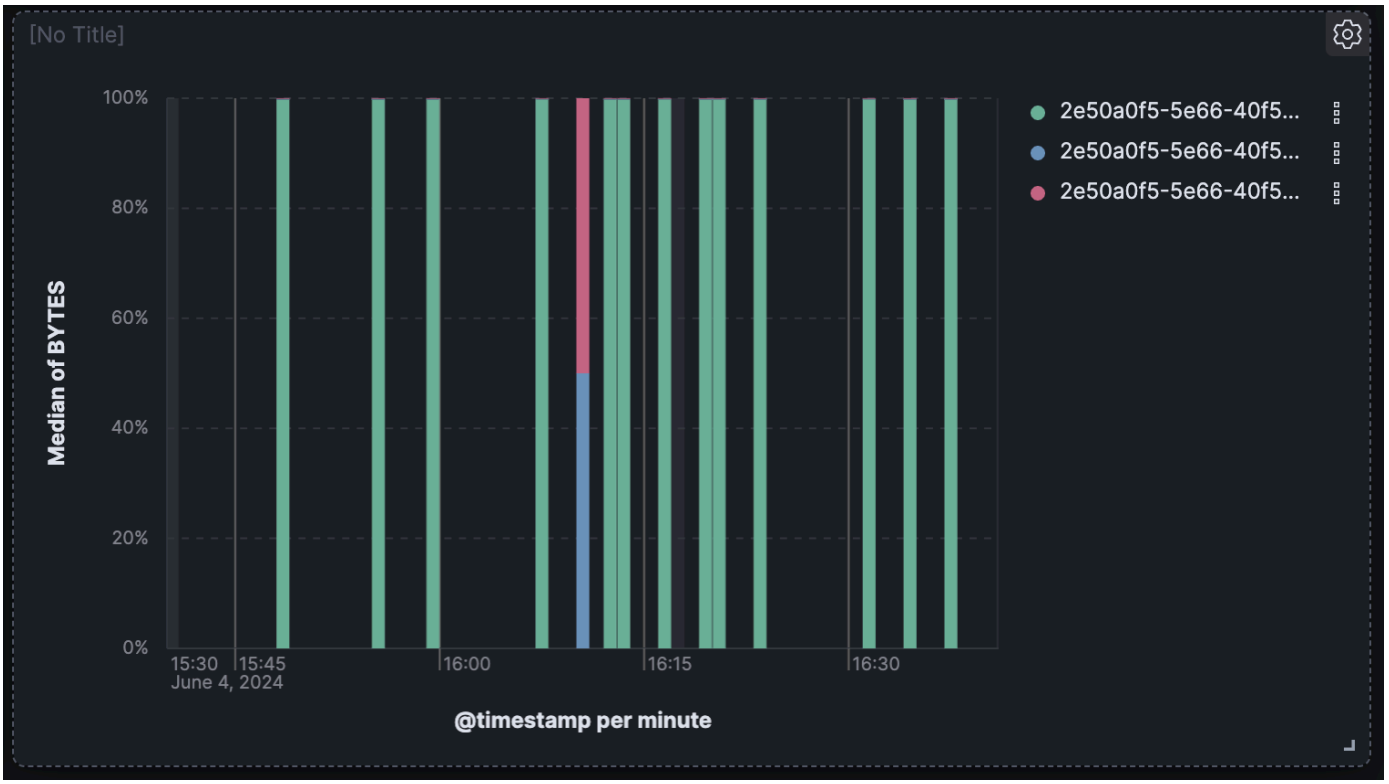


Figure 20: Kibana Visualization for Kumon showing median of BYTES

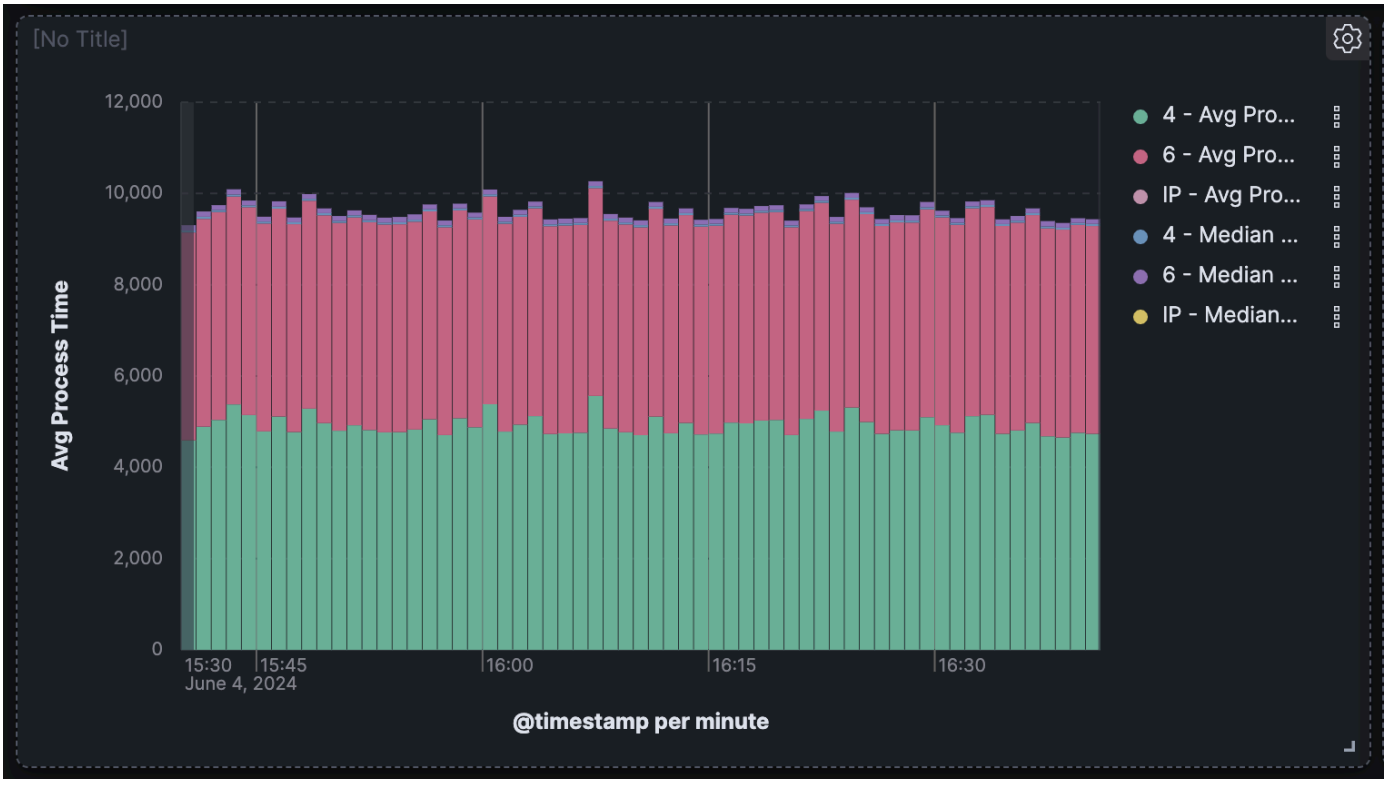


Figure 21: Kibana Visualization for Kumon showing average process time

Cluster Mesh Monitoring

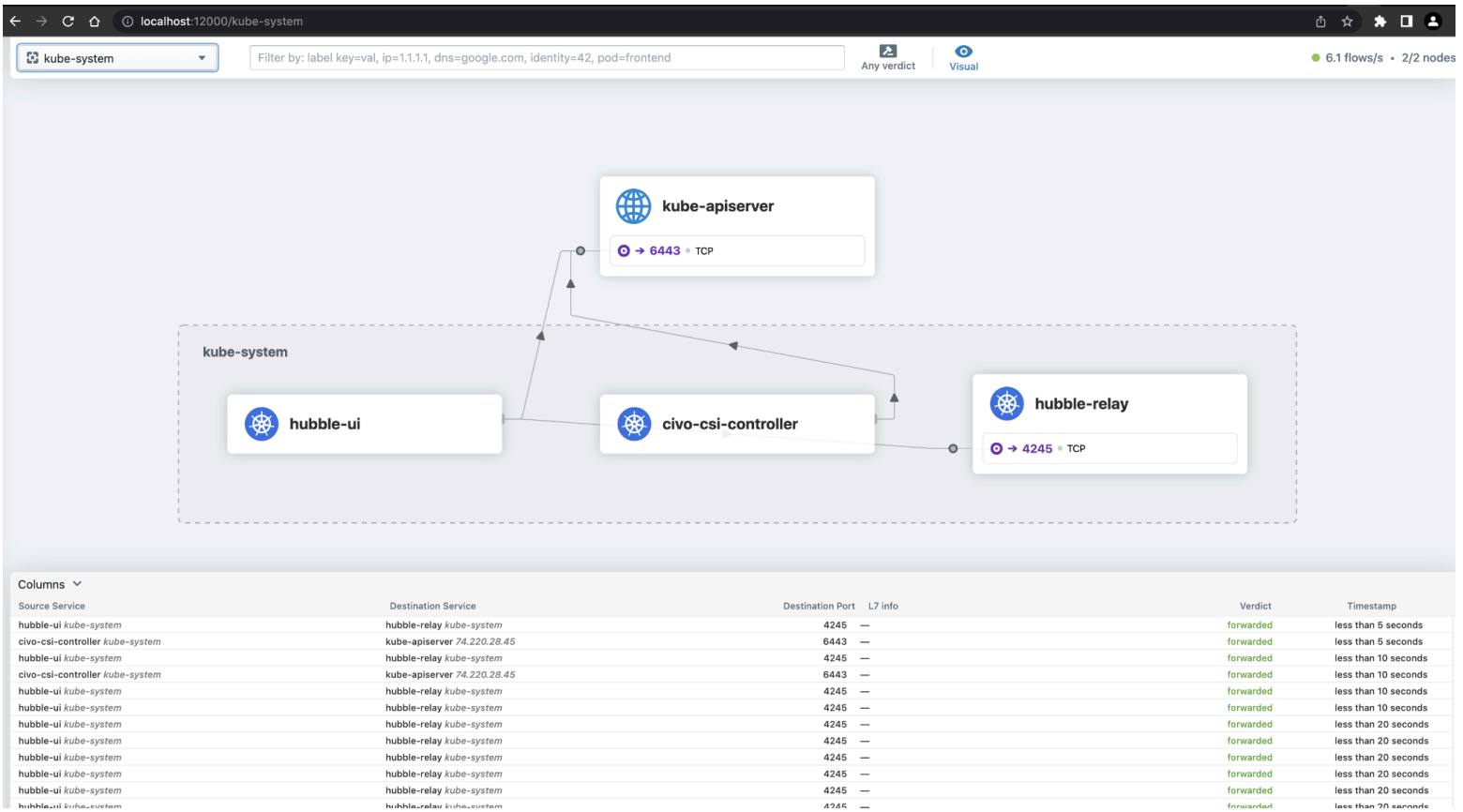


Figure 22:Cluster Mesh highlighting connections along with the data exchange and coordination

The service mesh in the Kumon architecture visualizes the communication within the cluster, detailing pod-level connections. It shows whether connections between pods are successfully established or not, providing a clear view of the interactions and dependencies between different pods. This comprehensive monitoring of pod connections and communications enhances the ability to troubleshoot issues, ensure reliable data flow, and maintain robust system performance.

Cluster Monitoring

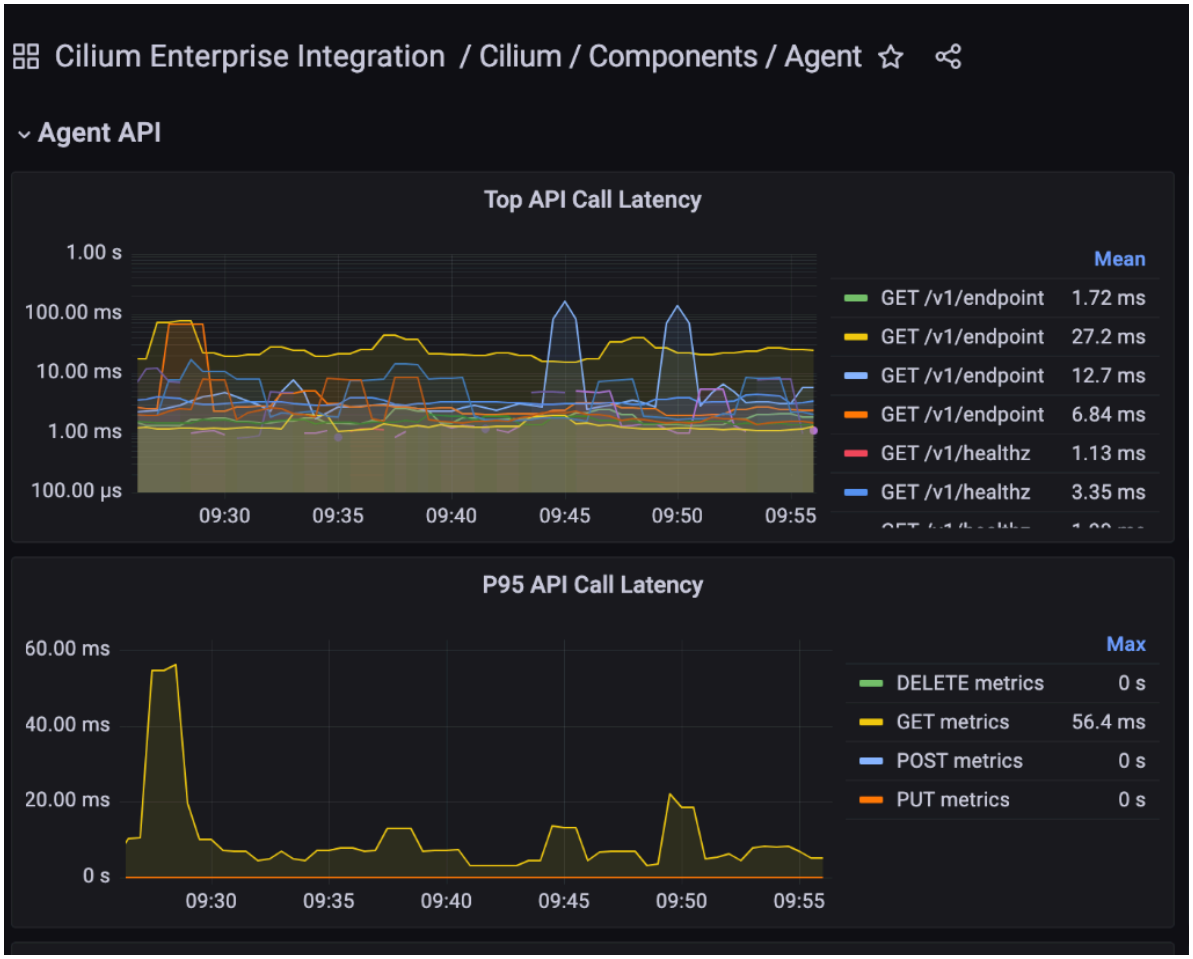


Figure 23: Grafana Visualization showing Cilium Metrics graphs

Cilium metrics visualized in Grafana provide detailed insights into cluster monitoring, enhancing the observability of network and security performance. Cilium, integrates deeply with the cluster, offering metrics such as network flow, latency, API call durations, and endpoint status. These metrics help in understanding how network policies are enforced, monitoring traffic patterns, and identifying potential security threats. Grafana dashboards present this data in an accessible and actionable manner, allowing operators to quickly detect and respond to anomalies

Conclusion

In conclusion, KuMon emerges as a integral tool in the landscape of cloud computing, addressing the upcoming and growing demands of Kubernetes cluster monitoring. By integrating eBPF technology, KuMon not only enhances performance monitoring for its users but also provides real-time, actionable insights, crucial for cloud engineers, DevOps professionals, and system administrators. The precision and depth of data offered by KuMon ensure that users can make informed decisions, leading to improved productivity in organisations. KuMon stands as a testament to the innovative fusion of cutting-edge technology with practical application, filling a significant void in the current cloud ecosystem.

Future Work

Looking ahead, there are several avenues for the future development of KuMon. Firstly, expanding the tool's compatibility to support a wider range of Kubernetes distributions and cloud platforms will enhance its accessibility and utility. Next trying to incorporating machine learning algorithms for predictive analytics could further augment KuMon's capabilities, enabling proactive management and optimization of cloud resources. Finally, continuous collaboration with the cloud computing community for feedback and feature requests will ensure that KuMon evolves to meet emerging challenges and user needs, solidifying its role as an indispensable tool in the realm of cloud infrastructure management.

9. References

[1] Joao, M., Sinnl, M., Schmid, S., & Warneke, A. (2016, April). Efficient Network Traffic Monitoring with eBPF. In Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies (pp. 157-170)

[2] Beyer, B., & Ewaschuk, R. (2016). Monitoring Distributed Systems. O'Reilly Media, Inc. (pp. 100-105)

[3] Merkel, D. (2014). Docker: Lightweight Linux containers for consistent development and deployment. Linux J., 239(3).

[4] Graf, T. (2018). Accelerating Envoy with the Linux kernel. In CloudNativeCon Europe and KubeCon Europe.

[5] Nam, T., & Kim, J. (2017). Open-source IO Visor eBPF-based packet tracing on multiple network interfaces of Linux boxes. In 2017 International Conference on Information and Communication Technology Convergence (ICTC) (pp. 324–326).

[6] Deri, L., Sabella, S., & Mainardi, S. (2019). Combining system visibility and security using eBPF. In Proceedings of the Third Italian Conference on Cyber Security (ITASEC) (Vol. Vol-2315, pp. 50–62).

[7] Suo, K., Zhao, Y., Chen, W., & Rao, J. (2018). vNetTracer: Efficient and programmable packet tracing in virtualized networks. In 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS) (pp. 165–175).

[8] Scholz, D., Raumer, D., Emmerich, P., Kurtz, A., Lesiak, K., & Carle, G. (2018). Performance implications of packet filtering with Linux eBPF. In 2018 30th International Teletraffic Congress (ITC 30) (Vol. 01, pp. 209–217).

[9] Liu, Y., Xu, B., Zhao, Y., Guo, Z., Li, H., & Wen, Z. (2014, March). Safe and Efficient Network Traffic Monitoring with eBPF. In Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI '14) (pp. 461-474).

[10] Radchenko, A., Sivathanuayut, S., & Agrawal, G. (2017, April). Scalable eBPF-based Network Traffic Analysis Using Hardware Offloading. In Proceedings of the 12th ACM Conference on Emerging Networking Experiments and Technologies (pp. 429-442).

[11] Joao, M., Sinnl, M., Schmid, S., & Warneke, A. (2016, April). Efficient Network Traffic Monitoring with eBPF. In Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies (pp. 157-170)

[12] Matei, I., Popoviciu, A., Le Guen, T., & Felber, P. A. (2018, April). Cilium: Network Security for the Cloud Native Era. In Proceedings of the 14th ACM Conference on Embedded Network Computing (pp. 10-21).

Appendix A: Illustrative Materials and Visual Representations

Note: This appendix provides an in-depth explanation of the referenced diagrams and figures within the report.

Figure 1: Sidecar Configuration on Each Pod

Each pod in the Kumon architecture is equipped with a sidecar, which enhances monitoring and data collection within the pod. This setup allows for better observability and performance tracking at the pod level.

Figure 2: Single Pod Configured with eBPF Monitors on Cluster

A specific pod within the cluster is configured with eBPF monitors. These monitors provide detailed insights into network traffic, system behavior, and performance, aiding in deeper analysis and troubleshooting.

Figure 3: Cilium Endpoint Open Files

The graph depicts the number of open file descriptors per endpoint. Monitoring this metric helps identify potential resource leaks or inefficiencies, ensuring optimal resource usage within the system.

Figure 4: Cilium API Duration

This metric measures the latency of Cilium API calls. It provides information about the responsiveness and efficiency of Cilium's communication, highlighting any potential performance bottlenecks.

Figure 5: Flows Verdict Graph

Categorised by the verdict label, this graph visualizes network flows and their associated verdicts. It helps track network policies and security decisions, providing insight into network traffic patterns and policy enforcement.

Figure 6: Key Components of the Hubble Ecosystem

This figure illustrates the essential components within the Hubble ecosystem. These components work together to provide comprehensive monitoring and observability, facilitating efficient system management.

Figure 7: Prometheus Alerting

Showcasing the configuration of Prometheus alerts, this figure highlights how alerts notify operators of critical events or anomalies within the system, enabling prompt response and resolution.

Figure 8: KuMon Use Case Overview

Providing an overview of how KuMon is applied in real-world scenarios, this figure highlights the value and impact of performance monitoring, showcasing practical applications and benefits.

Figure 9: Entity-Relationship Diagram (ERD)

The ERD visually represents the relationships between different entities or data tables within the KuMon system, aiding in the understanding of data structure and interactions.

Figure 10: Data Flow Diagram

Outlining the flow of data and interactions between various components in the KuMon architecture, this diagram helps in comprehending the overall data movement and process logic.

Figure 11: Kumon Architecture

This figure presents the overall architecture of the Kumon system, emphasizing its key building blocks and how they interconnect to form a cohesive monitoring solution.

Figure 12: Process Diagram

The process diagram visually explains the sequence of steps involved in monitoring and analyzing performance metrics, providing a clear understanding of the workflow and procedures.

Figure 13: Configuration at a Glance

Summarizing critical configuration settings or parameters within KuMon, this figure provides a quick reference to essential configuration details.

Figures 14 and 15: User Interface (UI)

These figures showcase the user interface for interacting with KuMon, including login screens and dashboard layouts, highlighting the user-friendly design and navigation features.

Figure 16: Landing Page and Dashboards

The landing page displays all available dashboards, allowing users to navigate and explore different metrics and visualizations, providing a centralized view of monitoring data.

Figures 17-19: Kibana Visualizations

These figures demonstrate various Kibana visualizations related to KuMon, such as process names, exit codes, latency, and data transfer metrics, offering detailed insights into system performance and behavior.

Figure 20: Median of BYTES

This visualization represents the median (middle value) of some metric related to data transfer or size, likely denoted in bytes. The median provides insight into the typical or central value, which can be useful for understanding the distribution of data sizes across different processes or endpoints.

Figure 21: Average Process Time

This graph shows the average time taken by various processes within the Kumon system. Monitoring process time helps identify bottlenecks, inefficiencies, or performance variations.

Figure 22: Hubble Connections

The visualization displays the connections between pods etc. showing service mesh visualization

Figure 23: Grafana Graphs showing Cilium Metrics

Shows the Grafana graphs showcasing Cilium Metrics like top API call latency.