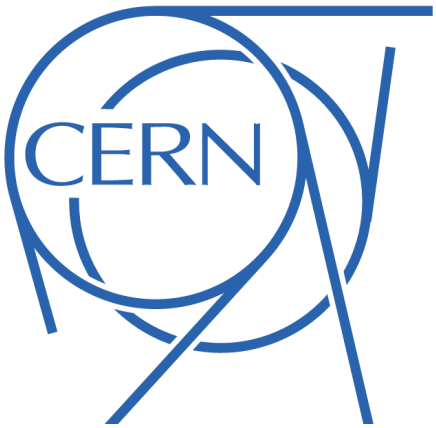

Conseil Européen pour la Recherche Nucléaire

European Organization for Nuclear Research

Research & Computing Sector
Scientific Information Service Department

PROJECT REPORT

CIMo
Cloud Infrastructure Monitoring

Mihai-Cristian Brezniceanu
University POLITEHNICA of Bucharest

Supervisor: Jimil Desai
Co-Supervisor: Benjamin Bergia

CERN Summer Student Program
CH-1211 Geneve 23, Suisse
September 5th, 2025

Abstract

The project focuses on developing an internal monitoring solution to process and display the operational parameters of applications built by CERN's Scientific Information Service (SIS) team. The system integrates with Grafana monitoring and operates across multiple dedicated Kubernetes clusters hosted within the OpenStack cloud infrastructure. It leverages a complete pipeline based on widely adopted, industry-standard tools (Prometheus, Mimir), enabling distribution as a free, plug-and-play solution with a high level of automation. Custom dashboards were designed to visualize cluster and application health and performance, enhancing infrastructure visibility and simplifying issue detection and resolution.

Acknowledgments

I would like to express my deepest appreciation and sincere gratitude to my supervisors—my friends—for their invaluable guidance and support throughout this project. I am especially thankful for their warm welcome, constant encouragement, and the thoughtful advice they shared during my stay.

I am equally grateful to the Library & Archive Section for their kindness and for sharing with me the living history they preserve.

This journey would not have been complete without the friendships I built along the way—the people who made each day truly unforgettable.

Contents

1 Group and Project Presentation

The Tools and Services (TS) section of CERN's Scientific Information Service[1] group designs, develops, and maintains a variety of software applications to support CERN researchers, experiments, and the wider particle physics community. These applications run on Kubernetes clusters hosted within CERN's OpenStack-based cloud infrastructure.

To ensure reliability and availability, we continuously collect metrics with Prometheus, visualize them through Grafana dashboards, store them for further querying in Mimir, and configure automated alerts for specific thresholds. These monitoring practices provide clear insights into system performance and enable the rapid detection of technical issues, helping to sustain high service availability and minimize downtime.

2 Objectives and Introduction to Monitoring

The project was guided by several key objectives:

✓ Develop multiple Grafana dashboards to visualize critical components through various metrics (application health, memory usage, storage volumes, node status, etc.).

✓ Configure an efficient alerting system capable of generating meaningful alerts to detect major anomalies and resource saturation.

[1] Info SIS Organizational Structure

✓ Document the entire process to support future improvements and reuse.

Moreover, to achieve these goals, we first analyzed the existing infrastructure to identify the data sources, the metrics provided to interpret their meaning, and to determine how they could be processed and visualized. Given the scale of the current time-series database, not all metrics are relevant, especially since they originate from different tools and services such as HAProxy, cAdvisor, or Kubelet (node agent). Furthermore, all the metrics are stored in a Prometheus-specific format and to retrieve them we need to use a specific querying language, PromQL.

3 SIS Monitoring Architecture

To get deeper into the current structure of the infrastructure, we have to present the major components as the figure below.

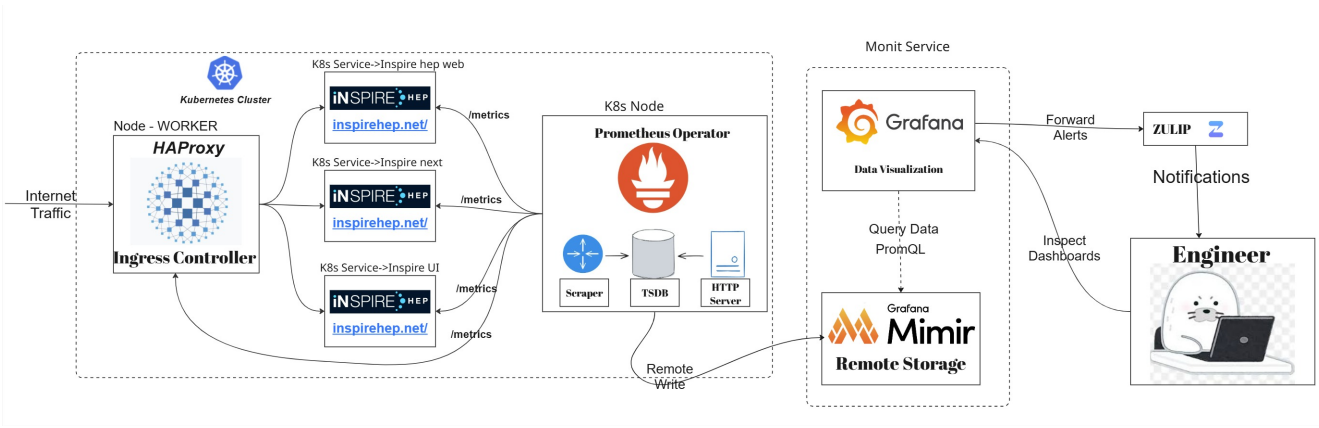

Figure 1: SIS Monitoring Overview

Each application (Inspire, HEPData, CAP, SCOAP³, OAPEN) runs in its own dedicated Kubernetes cluster across multiple nodes. These clusters are made accessible to end users through an Ingress Controller, which handles key aspects such as load balancing, secure connections, and metrics exposure.

Use the Prometheus Operator to collect metrics from the clusters and forward them to Mimir, a time-series database (TSDB) designed for large-scale metrics retention and optimized querying over extended periods. Prometheus is an open-source monitoring system with a multi-dimensional model, an efficient time-series database, a flexible query language (PromQL), and a modern alerting mechanism.

PromQL allows real-time selection and aggregation of time-series data based on specific labels, which can then be visualized through Grafana.

Grafana provides efficient processing and real-time visualization of these metrics, supporting a wide range of display formats and time-series analyses for rapid updates and comprehensive monitoring.

4 Grafana, dashboards and alert

Grafana has access to multiple data sources and according with this our dashboards are making full use of them to provide a quick health check of the applications and the underlying infrastructure. Each layer will allow a deeper dive into the technical details of current issues we face, helping to identify the actual cause for a fast and efficient solution.

4.1 HTTP 5xx Ratio per Proxy (Last 5m vs Prev 5m)

1. Overall goal

Quickly assess the health of the entire workflow between user and application to ensure high availability. Additionally, monitor for sudden spikes in server errors across applications or services.

2. Query Documentation

```
(
 sum(rate(haproxy_server_http_responses_total{code=~"5..", proxy="$proxy"}[5m]))
  / sum(rate(haproxy_server_http_responses_total{proxy="$proxy"}[5m])) + 1e-9
)
/
(
  sum(rate(haproxy_server_http_responses_total{code=~"5..", proxy="$proxy"}[5m] offset 5m))
  / sum(rate(haproxy_server_http_responses_total{proxy="$proxy"}[5m] offset 5m)) + 1e-9
) > 0
```

Note: This approach was chosen to measure the increase in 5XX error rates relative to total responses, accounting for fluctuations in both requests and errors over a 5-minute period.

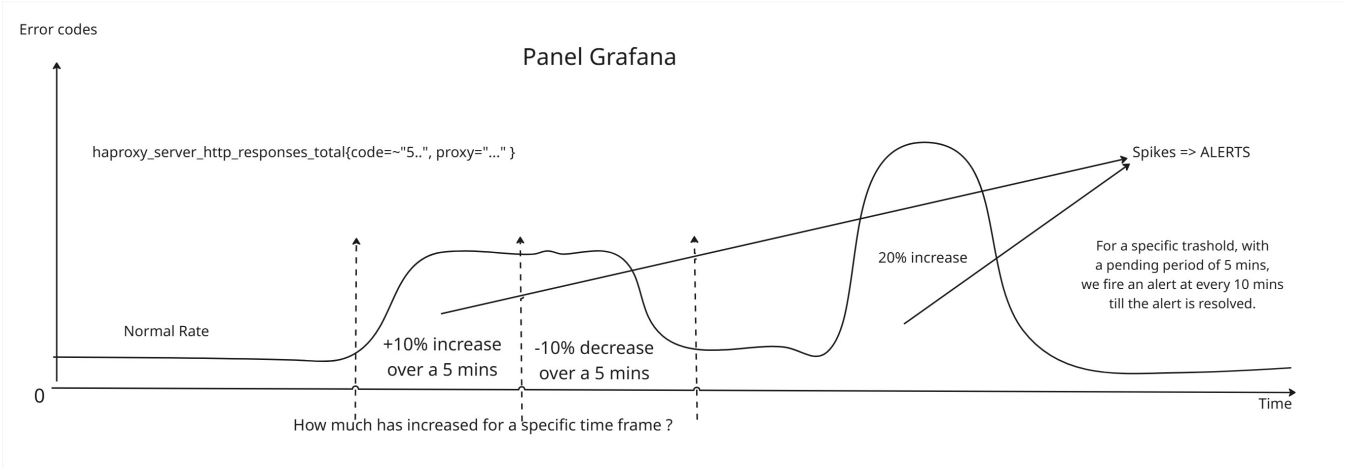

Figure 2: Alert Description

✓ **haproxy_server_http_responses_total** it's a counter metric which provides the total number of HTTP responses with status 1xx-5xx returned since the worker process started

✓ the **rate** function returns the specific rate for over the last **5 minutes** and is taking care of **counter resets**

✓ make a selection after the **proxy** env variable, what will take the total of **5XX codes** with a regex, **divided** by the **total number of codes** returned

✓ repeat the same pattern with second term of the fraction, but with an **offset of 5min**, to compare it

✓ divide them, (**current error rate**) / (**previous error rate**), and apply a filter for a value higher than 0 (zero) to ensure that this expression only returns if both error rates exist (removes any NaN or zero divisions)

✓ add a small number, **+1e-9,** to handle **zero denominators**

✓ for example

- ✓ In the last 5 minutes: 10 errors out of 1000 requests → error rate = 1%
- ✓ 5 minutes before that: 2 errors out of 1000 requests → error rate = 0.2%
- ✓ Then (1%) / (0.2%) = 5 times increase

By following a specific threshold, decide if our application flow is healthy reported to the current implementation and given parameters.

4.2 Restarts per pod due to the OOMKilled

1. Overall goal

This dashboard monitors pod restarts caused by OOMKilled events within a specified

time frame, helping teams proactively detect and address memory-related issues, especially under the current lenient memory limits.

2. Query Documentation

```
changes(
  container_oom_events_total{namespace="$namespace"}[$__range]
) > 0
```

✓ **container_oom_events_total** it's a counter metric which sums out all the memory events observed for a container item[✓] make the label selection after the namespace env variable

✓ use **changes** function, to get the total number of restarts for container due to the OOMKilled reason

✓ filter the output of metrics to display only containers that have these problems and set the threshold value bigger than zero

✓ the current implementation was extended to every Data Source, with a small adjustment on the **namespace** variable

4.3 Percentage of total storage used

1. Overall goal

Display the percentage of total storage used versus available per namespace, including indicators for nearly full PVs.

2. Query Documentation

```
sort_desc(
  max(
    (
    kubelet_volume_stats_used_bytes{exported_namespace="$namespace"}
      /
      (kubelet_volume_stats_capacity_bytes{exported_namespace="$namespace"} + 1e-9)
    ) * 100
  ) by (persistentvolumeclaim)
)
```

Note: For PVCs using more than 90%, allocate additional storage and set specific alerts. For those using less than 10%, consider freeing up that space or allocate much less space.

✓ **kubelet_volume_stats_used_bytes** is a gauge metric that exposes the number of used bytes in the volume

✓ **kubelet_volume_stats_capacity_bytes** is a gauge metric that exposes capacity in bytes of the volume

✓ the same PVC can appear across multiple time series due to different labels (instance in our case) and **to aggregate** them, use **max(...) by (persistentvolumeclaim)** to reduce all these time series into a single number per PVC and get the maximum value (the values are the same for all instances so there is no max and minimum in this case, so we are not losing data)

✓ **sort_desc()** function to present the volume storage in decreasing order

Figure 3: Percentage of total storage used

With the monitoring infrastructure in place, the next step was to configure alerting to detect issues before they escalate. Grafana's alerting system enables continuous monitoring of metrics and sends notifications when anomalies occur. This setup involved creating alert rules that specify when notifications should be triggered based on query results. With a dedicated contact point for each team, allows alerts to be received via Zulip or other incoming webhooks.

5 Project Outcome

The results confirm that the project meets the initial objectives. The monitoring system provides real-time visibility into infrastructure performance, supports early detection of anomalies (e.g., node failures, high error rates, OOMKilled pods, storage exhaustion), and improves response times by combining metric analysis with automated alerts.

Notifications are delivered to Zulip, ensuring the team receives timely warnings about potential issues. All monitoring components are integrated and collect data continuously. Including the documentation created, the setup provides a solid foundation for future development, which will be by real impact in maintaining highly available RCS-SIS-TS services.

Bibliography

[1] *Grafana variables at a glance*. Accessed: 2025-08-22. URL: https://volkovlabs.io/blog/grafana-variables-at-a-glance-environment-data-source-explained-102b8e05e1a5/.

[2] *HAProxy Kubernetes Ingress Controller*. Accessed: 2025-08-23. URL: https://www.haproxy.com/documentation/kubernetes-ingress/.

[3] *How to Monitor Disk Usage*. Accessed: 2025-08-12. URL: https://betterstack.com/community/questions/monitor-disk-usage-in-kubernetes-persistent-volumes/.

[4] *Kube state metrics*. Accessed: 2025-08-25. URL: https://github.com/kubernetes/kube-state-metrics.

[5] *Kubernetes Metrics List*. Accessed: 2025-09-02. URL: https://www.juniper.net/documentation/us/en/software/cn-cloud-native22/cn-cloud-native-feature-guide/cn-cloud-native-network-feature/topics/ref/cn-cloud-native-k8-metric-list.html.

[6] *Monitoring Kubernetes layers*. Accessed: 2025-08-27. URL: https://grafana.com/blog/2023/01/25/monitoring-kubernetes-layers-key-metrics-to-know/.

[7] *Prometheus Documentation*. Accessed: 2025-09-01. URL: https://prometheus.io/docs/introduction/overview/.

[8] *Understanding Kubelet*. Accessed: 2025-08-29. URL: https://blog.devops.dev/understanding-kubelet-94868f3adc07.

[9] *Vector Matching*. Accessed: 2025-08-17. URL: https://iximiuz.com/en/posts/prometheus-vector-matching/.