Observability for Cloud Native platforms

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Bachelor thesis

Degree Programme in Business Information Technology

2023

Marko Hartikainen

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutus | Tiivistelmä
Tekijä Marko Hartikainen | Vuosi 2023
Työn nimi Observability for Cloud Native platforms
Ohjaaja Ismo Turve

Opinnäytetyön tarkoituksena oli tutkia ja toteuttaa avoimen lähdekoodin havaittavuusratkaisu ensin kehitysympäristöön ja myöhemmin tuotantoympäristöön.

Opinnäytetyössä tarkasteltiin perinteisten valvontaratkaisujen vertailua havaittavuuden tarjoamiin etuihin. Tutkimuksessa tarkasteltiin, miten havaittavuus ylittää valvonnan kattamalla laajemman valikoiman työkaluja, joiden avulla saadaan syvällisempiä näkemyksiä järjestelmän toimintaan, suorituskykyyn ja mahdollisiin verkkoon liittyviin ongelmiin.

Opinnäytetyö keskittyi myös erilaisten kojetauluissa tarjottavien havaittavuuden visualisointiominaisuuksien vertailuun. Nämä kojetaulut ovat ratkaisevassa asemassa kerätyn datan esittämisessä helposti tulkittavassa muodossa, mikä mahdollistaa valvontatiimille päätöksenteon perustuen havaittuun järjestelmän toimintaan.

Opinnäytetyö tehtiin yhteistyössä Datalounges Oy:n kanssa, missä havaittavuusratkaisu otettiin käyttöön onnistuneesti omassa pilvi-infrastruktuurissa. Tämä käyttöönotto mahdollisti reaaliaikaisen valvonnan, järjestelmän suorituskyvyn ja toiminnan analysoinnin sekä hälytysten lähettämisen.

Avainsanat Havainnollistaminen
Pilvinatiivi
Lokienhallinta
Monitorointi

Sivut 25 sivua ja liitteitä 5 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology
Author Marko Hartikainen
Subject Observability for Cloud Native platforms
Supervisors Ismo Turve

Abstract
Year 2023

The purpose of the thesis was to explore and implement an open-source observability solution first, to the development environment, and later into production.

The thesis delved into the comparison between traditional monitoring solutions and the advantages offered by observability. It examined how observability goes beyond monitoring by encompassing a wider range of tooling to gain deeper insights into system behavior, performance and potential network issues.

The thesis also focused on the comparison between different dashboard solutions that offer visualization capabilities for observability. These dashboards played a crucial role in presenting the collected data in an easily format, allowing monitoring team make decisions based on the observed system behavior.

Thesis was conducted in collaboration with Datalounges Oy, where the observability solution was successfully deployed in the company's own cloud infrastructure. This deployment enabled real-time monitoring, analysis of the system's performance, behavior and alerting.

Keywords Observability
Cloud Native
Log Management
Cloud Monitoring

Pages 25 pages and appendices 5 pages

Glossary:

K8s	Kubernetes
K3s	Lightweight Kubernetes distribution
Agent	Monitoring software gathers system and application metrics from virtual machine instances
Helm	Kubernetes package manager
CRUD	Create, Read, Update, Delete operation
DevOps	Set of tools and practices that increases an organization’s ability to deliver applications and services more efficiently.
AWS	Amazon Web Services
CPU	Central Processing Unit
CSP	Cloud Service Provider
SaaS	Software as a Service
HLA	High-Level architecture
API	Application Programming Interface
JVM	Java virtual machine

Contents

Figures and tables

Appendices

1 Introduction

The world of IT -infrastructure is getting containerized. With containers, it is possible to make great business ideas into great digitalized services quickly, reliably and systematically. While containerized software is becoming the form factor of choice for IT industry, new technologies challenge how IT is consumed. They make cost effective multi- and hybrid cloud solutions possible and provide companies and organizations with unprecedent agility.

As these applications evolve to business critical assets, it also becomes increasingly important for organizations to stay up to date on the status of these applications. It is critical to understand how digital services behave under load, collect logs for analysis and get alerted in case of problems.

Such requirements are common and widely acknowledge for traditional datacenter infrastructure. Also virtual machines in cloud environments have solved this challenge as they do not significantly differ from the datacenter infrastructures when it comes to monitoring and log management requirements. Cloud Native is different, however. Ephemeral containers and self-adjusting applications that connect via API cannot be managed similarly to well-known virtual machines. For this purpose new technologies require new tooling.

In this thesis such methodologies and technologies are researched and put into practice by setting up an observability platform that enables a service provider organization to measure and monitor cloud native applications across different cloud platforms.

Thesis research questions are:

- How to build observability into Kubernetes environment?
- What factors influence the solution of the user interface?

2 Observability

Definition of observability can be summarized as follows: “Observability is the ability to measure a system’s current state based on the data it generates, such as logs, metrics, and traces.” (Livens, 2021)

Even though observability is a relatively new term in DevOps, its origin comes from 1960 by Rudolf E. Kálmán’s (1930-2016) Control Theory, where it was first intoduced to describe how well a system can be measure by its outputs. Observability is not a single technology but a practice that helps operations teams reach their service level targets, reduce time required for repairs and extend the mean time between failures. For developers observability is a vital tool to troubleshoot applications and create robust services. Observability helps both developer and operations teams to get a holistic picture of the system being observed. (Crowdstrike, 2021)

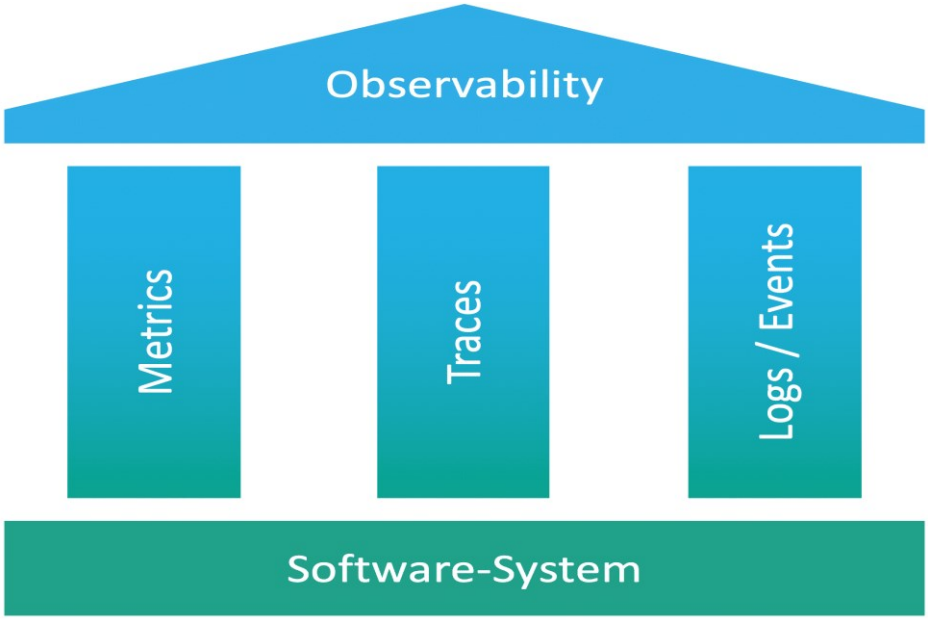
During these years, definition of observability has evolved into its latest form, the three pillars of observability. (Han, 2019)

2.1 Pillars of Observability

To distinguish the different technologies and their use cases, observability is split into three pillars, logs, metrics and traces as in figure 1. Although, having all of these does not make a system observable, it is a good start. The value of observability comes from combining these pillars and using the data to provide services to operations teams then to consume.

Together these pillars collect comprehensive information about the system and its behaviour, enabling operators to run systems reliably and respond to unexpected behaviour. Using metrics, an operator can identify when the system is operating slower than usual or if there is an anomaly in its behaviour. Traces help identify which part of the system is slower than usual or causing the anomaly and if it needs to be addressed. Finally, logs are there for errors and exceptions, and allow operators to carry out further analysis.

Figure 1. Three pillars of observability (Wert, 2019)



2.1.1 Metrics

Metrics are counters or measurements of a system characteristic during a time period. Metrics are numeric by definition and represent aggregated data. Examples of metrics can be average CPU usage per minute per server or the number of requests returning errors per JVM each day. Metrics can be collected from infrastructure, load balancers and even applications. (Ellingwood, 2017)

2.1.2 Logs

Logs are intended to leave clues on what part of the codebase a request has reached, and if the application encountered anything unexpected or abnormal in processing that request. Logs can also be used to capture access attempts, as in the case of access logs. Logs can be generated by the application responding to requests or by the operating system, example syslog or the Windows event log. (Sharif, Arfan, 2022)

2.1.3 Traces

A distributed trace, more commonly known as a Trace, records the paths taken by requests, made by an application or end-user as they propagate through multi-service architectures, like microservice and serverless applications. Without tracing, it is challenging to pinpoint the cause of performance problems in a distributed system. It improves the visibility of our application or system's health and lets us debug behavior that is difficult to reproduce locally. Tracing is essential for distributed systems, which commonly have nondeterministic problems or are too complicated to reproduce locally. (Bigelow, 2021)

Tracing makes debugging and understanding distributed systems less daunting by breaking down what happens within a request as it flows through a distributed system. A Trace is made of one or more Spans. The first span represents the root span, each root span represents a request from start to finish. The spans underneath the parent provide a more in-depth context of what occurs during a request or what steps make up a request. (Coralogix, 2022)

2.1.4 Events

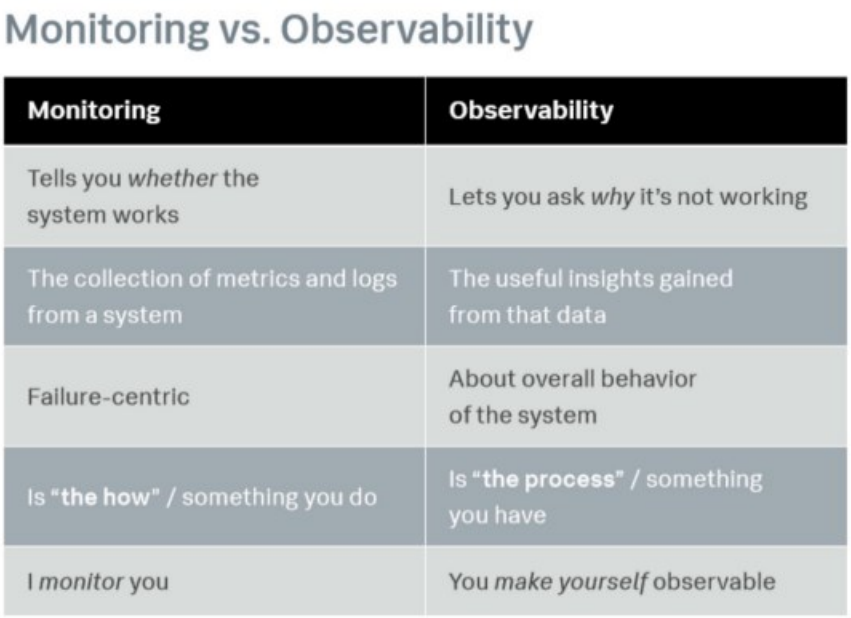
Alongside the three pillars, events can be utilized to enhance a system's observability. For instance, it can decide that every time an admin user executes a privileged task, the system registers an event in an observability tool. Events are registered with specific actions, the execution of a function, updating of a database record or an exception thrown by the code and are timestamped, unchangeable documentation that comes in three forms, plaintext, structured and binary. Analysed over time, events can help determine patterns and structured logs can also be used as low-level events. (Arfan Sharif, 2023; O'reilly, n.d.)

2.2 Traditional monitoring and observability

As cloud-native environments have become more complex and root cause of failures more difficult to pinpoint, companies have realized benefits of observability for their business. Traditionally monitoring and log management have been separate practices that have allowed visibility to system state. Monitoring is a tooling solution for teams to allow configuring dashboards and set alerts to inform team if anomaly is detected. All these are done with predefined metrics and logs. This is a working solution but rely on assumption that team acknowledges what is going to happen. (Livens, 2021)

Monitoring and observability are technical solutions utilized by teams to manage and troubleshoot their systems. While monitoring is focused on enabling teams to observe and comprehend the current state of their systems through the collection of predefined sets of metrics or logs. In observability, like figure 2 shows, observability allow you to ask questions why it is not working. Observability enables teams to proactively debug their system by exploring properties and patterns that are not predetermined. (Google, n.d.)

Figure 2. Differences between monitoring and observability. (A beginner's guide to observability, s. 6)



2.3 Benefits of observability

While running simpler systems, example virtual machine instance in AWS, monitoring resources such as cpu, memory and network is normally enough and monitoring team is able to see if instance requires more capacity. Distributed system on other hand, are more complex. There are workloads that are constantly updated, there might be internal network connection failures that causes application slowness and therefore more difficult to spot on and every change occur new type of failure. This might lead into unidentified information errors as “unknown unknown”, when simplier environment is expecting to build environment monitoring to things that are anticipated during project, called as “known unknown” type of errors. (Splunk, 2020; Padgett, 2021)

So why is observability important in this scenario?

- Cloud Native software solutions are complex. Traditional technologies do not meet the requirements of combining logs, traces and metrics to run and understand cloud native applications by combining all data sources and presenting them in an understandable format. (Parr, n.d.)

- Observability saves time and debugging effort when developers are solving software related problems. (Stripe, 2018)

- Get alerted for problems early and identify root causes of system behavior problems. (Splunk, 2020)

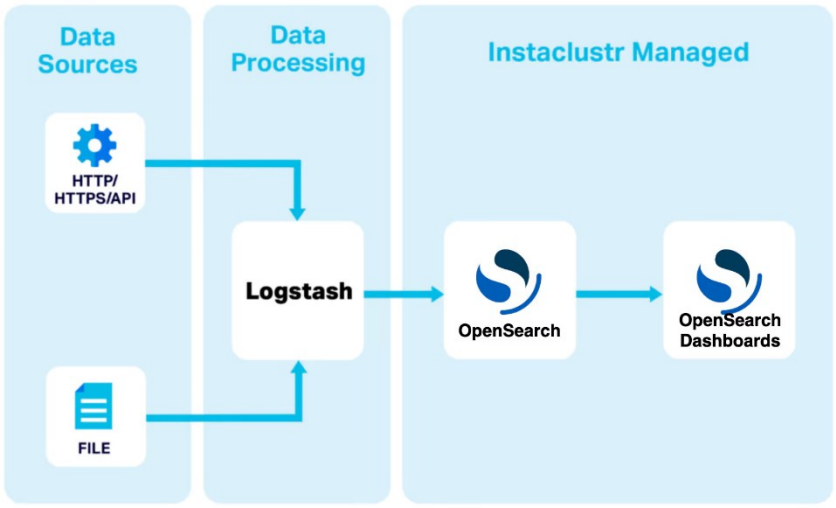
Observability offers several benefits to a system by enhancing its visibility, making monitoring safer, more effective and providing a full-scale view of events and performance. It not only identifies problems in real-time but also delivers data that allows the complete observation of the application flow, preventing failures in the future. (Seixas, 2021)

2.4 Components of observability solution

To make system observable, it needs tooling. Figure 3 illustrates this workflow. Logs, metrics and other events need to be collected from various systems. These are then stored and an analytics solution that has search capabilities is required so that dev and ops specialist can look for and filter the items they require for their tasks. A visualization service is a powerful extension of the tooling which enables users of the platform to quickly understand and measure the state of their systems. Traceabilty can be part of the technology stack or part of the practice. Also alerting for identified key events is a necessary component.

On top of data collection and analysis, it is important to create reports that allow different stakeholders to understand and explain the status of systems.

Figure 3. Components of observability solution. (Intraclustr, n.a)



There are various solutions in the market that have structured their offerings differently, but mainly they can be categorized as follows:

- **Data sources**. Typically agents or similar components that collect metrics, events and logs from data sources.

- **Data processing** or aggregation. These technologies collect the data from the data sources, normalize it and manage the data flow.
- **Indexing & storage.** Store data to allow monitoring, analysis and security use cases.
- **Visualization**. A dashboard solution that then provides views and visibility to collected data in human readable format. (Horovits, n.d.)

Together this structure sets up a foundation for services that make observability valuable. Collecting data from different sources, processing and storing it and visualizing into a desired format are steps that let organizations combine and analyse the operational data so that they can get alerted in case of anomalies and do tracing to find the actual root causes for problems.

2.4.1 Data sources

Log and metric management involve the continuous collection, storage, processing, and analysis of data from various programs and applications to improve system performance, resource management, security, and compliance. This practice can be categorized into different stages including collection, monitoring, analysis, retention, indexing or search and reporting. (Arfan Sharif, 2023)

Typically, in a standard setup, the system will generate logs and metrics that need to be shipped out from data source. A log shipper, agent, is then responsible for transporting these logs to their intended destination. (Elastic, n.d.)

2.4.2 Data processing

Data processing is the process of collecting and summarizing data from multiple sources to provide a comprehensive view of a system or network's performance. It involves defining the data sources, collecting raw data, processing, cleaning it and summarizing it into a more manageable form. The resulting dataset can be used to monitor network performance, identify issues, anomalies and generate reports for analysis and troubleshooting. (Paperduty, n.d.)

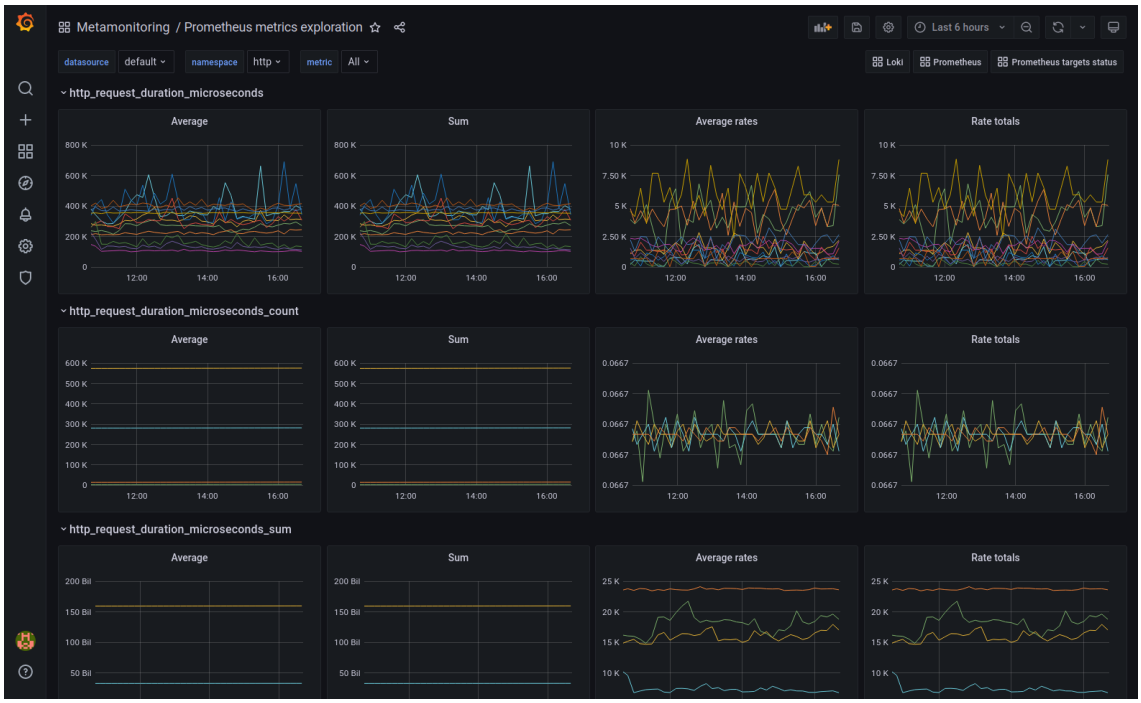
2.4.3 Indexing & storage

Indexing refers to the process of creating a searchable index of the data, which involves parsing and analysing the data and creating an index of terms that can be used to search the data. Once the data is indexed, search engines can then be used to search the data based on specific search criteria, such as keywords or phrases. (OpenSearch, n.d.)

2.4.4 Visualization dashboard

Dashboards are used to visualize the collected data. With various possibilities, it is possible to build visuals for different types of scenarios for different teams as in Figure 4. Dashboard solution makes it possible to navigate the data, build real time event visualization and monitor workloads across all environments. (Elastic, n.d.)

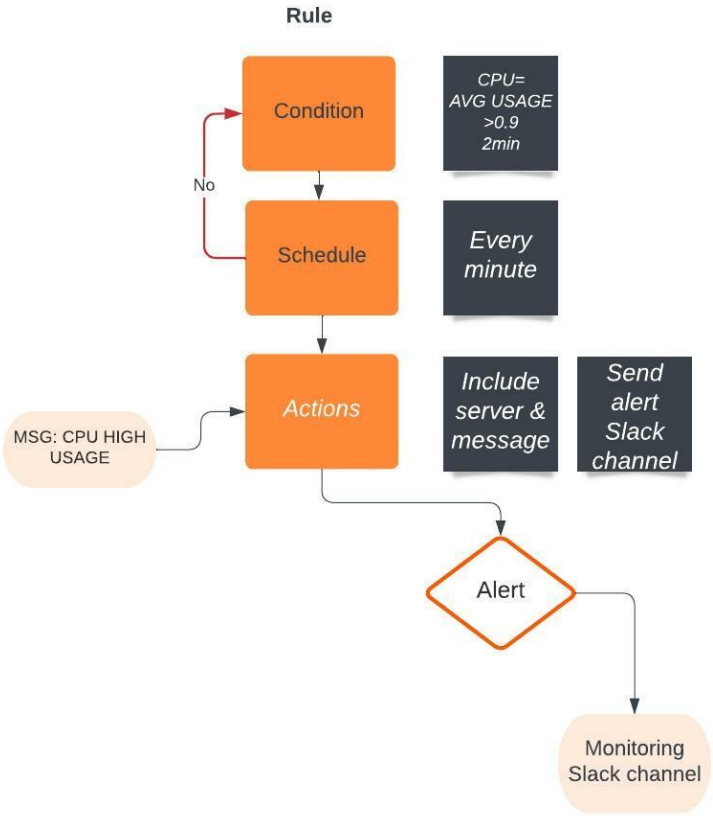
Figure 4. Visualization. (Verteuil, 2022)



2.4.5 Alerting

Figure 5 illustrates an alerting workflow, and it allows to define rules to detect different types of conditions and raise flags to the monitoring team when metric values fall outside predefined scope. System then triggers an action that notifies the monitoring team. Most commonly alerts are sent via email or sms, but it is also possible to send a message to Teams or similar messaging platform. (Elastic, n.d.)

Figure 5. Basic worklow for alerting



2.4.6 Tracing

Trace is a collection of associated events that occur as a result of an input, example a user's login. A trace includes numerous spans, each representing an operation carried out in response to the original request. Each span records start and end times for the operation, and other optional attributes like service instance identifiers. This method enables the identification of each request, call, and process involved in a particular transaction, the sequence of their occurrence, and the duration of each phase. (Coralogix, 2022)

3 Service provided requirements: CASE Datalounges

With Cloud Native computing and containerization, ideas can become great digital services. Datalounges, the Cloud Native platform company provides as-a-service solutions that enable observability across clouds, help customers better manage and secure cloud native applications and improves business continuity. With these services, Datalounges customers deliver both a developer experience and secure, reliable operations for cloud native applications.

In the following chapter, we meet cloud service provider requirements for observability solution and comparison for the dashboard solution.

3.1 Service Provider requirements

The goal of this project is to implement an observability solution for a cloud service provider. Datalounges Oy as a CSP (Cloud Service Provider), offers infrastructure for customers to deploy their applications to the cloud as well as a platform for development. With an Observability solution, service provider can stay on top of systems running in the infrastructure, get alerts for anomalies, extend mean time for failures and prevent downtime. As the number of systems that require observing is not known beforehand, the solution should be scalable to meet demand.

In order to ensure that this project's observability solution will be appropriate for its use cases, the CSP has set requirements in table 1 and optional requirements can be found in appendix 2. The requirements concern the system's operational environment, technical requirements and functionalities. They are further divided into mandatory and optional categories. Mandatory requirements need to be addressed on the end product and of the optional requirements, some are implemented and others discussed in the summary chapter of this thesis.

Table 1. CSP requirements

Mandatory Requirements		
Requirement	Explanation	Additional info
Must run in CSP's own cloud infrastructure	The CSP provides Kubernetes capacity as-a-service. Observability deployment must run in that capacity.	
Must have the following integrations available	OIDC Harbor GitLab Current Kubernetes version	
Must support Private and Public Cloud Kubernetes environments serviced by CSP	The solution must be able to collect logs, metrics and traces from AWS EKS, AKS, GKE and Rancher RKE2 Kubernetes environments	A common Kubernetes integration is required
Must have disaster recovery option	The solution must include a solution for disaster recovery so that customers using the service can be guaranteed for their data's availability	
Technical		
Must be containerized and run on Kubernetes	The services must be containerized and run in CSP's Kubernetes capacity	
Must be deployable as a Helm chart	Must be deployable as a Helm chart so that CSP's admins can deploy and redeploy the solution	
Must support the following use cases	Log management Metrics management Tracing Alerts generation	
Must support the following user tasks	Create views in the dashboard component as self-service Search logs and metrics Collect log item changes for alerts	

3.2 Observability solution comparison

Even if observability market as such is vast and there are technologies and vendors from small open-source projects to global software vendors, the requirements set several limitations that only a handful of providers can match. For example, Gartner Peer Insight lists 77 technologies that can be considered observability (Gartner) and G2 (G2, n.d) with 66 entries provide a wide foundation for comparisons. Many of the technologies have strong functionalities and a wide selection of features. A typical solution is a monitoring (Dynatrace, Microsoft) tool or a log management platform (Splunk, New Relic) that has expanded to include observability functionality. Then a different branch of solutions are the SaaS solutions that can be either standalone (Datadog) or extensions of the cloud platforms they run on (AWS, Google).

Key limits for selection turn out to be support for on-premise Kubernetes, open source codebase and deployment options. The main contenders in table 2 cover the minimum requirements set forth for Datalounges, have strong userbases and have received high rankings from users in public categorizations mentioned earlier in this chapter.

OpenSearch Dashboard is a user-friendly platform for managing and visualizing data in opensearch, an open source search and analytics engine. It allows users to create custom dashboards, monitor real-time metrics and perform complex searches.

Zabbix is an opensource monitoring software that can be used to track and analyze the performance of networks, servers, and applications. It features a web-based interface and supports multiple monitoring methods, such as SNMP, JMX, and ICMP.

Grafana is another opensource platform that provides data visualization and analytics for multiple data sources, including databases, cloud services, and IoT devices. It offers a range of data visualization tools, such as charts, tables and graphs. It can be integrated with other monitoring tools.

Each of these are opensource, can be deployed on required infrastructure, support private and public clouds, provide disaster recovery options and have a helm chart available for containerized deployment. A Helm is a package manager for Kubernetes, designed to simplify the deployment and management of the applications. It provides an efficient way to package, install and update applications on a Kubernetes cluster. Helm uses charts, which are packages containing all the necessary resources and configuration files required to deploy an application. Helm simplifies the deployment process by packaging all the required resources and configurations into a single chart. This makes it easier to deploy complex applications and ensures consistency across deployments.

Table 2. Top solutions

CSP Requirements			
Requirement	OpenSearch	Zabbix	Grafana-Prometheus
Opensource	Yes	Yes	Yes
Commercial Support	AWS support program (subscription)	Support subscription	Support subscription
Community Support	Strong. Community driven product e.g slack and git	Strong. Community, events, forum	Good, slack channel. Community forum, but not technical problem solving
Deployment helm chart	Yes	Yes	Yes
DR option	Yes	Limited	Yes
Integrations	Yes	Yes	Yes

3.3 Technology selection

Majority of the essential items in selection criteria can be found in each of the three solutions and the primary differences among them lie in the deployment options and plugin offerings. An excellent example of this is the tracing feature, which is already integrated into the OpenSearch platform. Overall, while all three tools have their strengths as seen in table 3, OpenSearch Dashboard stands out for its flexibility and ease of use, particularly for managing and visualizing data in OpenSearch and in deployment section and is selected as the prime candidate for deployment testing.

Table 3. Technology comparison

Technology comparison			
Requirement	OpenSearch	Zabbix	Grafana-Prometheus
Pros	Open source and highly customizable	Comprehensive monitoring capabilities	Wide range of data source integrations
	Powerful search and analytics capabilities	Supports various notification types and methods	User-friendly interface and easy to use
	Supports various data types and formats	Has a wide range of data source integrations	
Cons	Requires more advanced technical expertise to set up and maintain	Requires more resources than other solutions	Limited data transformation capabilities
		Steeper learning curve Requires additional plugins or integrations for full benefith	May not be suitable for highly complex data environments

4 Observability deployment: CASE Datalounges

This chapter discusses development infrastructure architecture and explains the concept of Opensearch deployment as well metric agent which is deployed into client machine running K3s. After deployments, we'll see components that creates observability.

Opensearch deployment will be made in Datalounges capacity as required. The infrastructure has some caveats when comparing to public cloud service providers and need to be considered as part of the planning. Another cause for concern is organizing data collection to gather metrics and logs. This will then enable setting up tracing and alerts.

Visualizations will include memory and cpu usage for pods, but mainly the use cases beyond the basic requirements will be decided once the requirements for deployments, data collection and capability to create visualizations have been finalized.

4.1 Infrastructure

OpenSearch is a cloud native application and requires Kubernetes infrastructure to run. Development cluster consists of 9 nodes as figure 6 describes.

Figure 6. Infrastructure HLA

Development cluster is installed using Canonical MAAS (metal as a service) deployment. Once the nodes have been installed and the network between them is operational the

system is ready for a Kubernetes deployment. Kubernetes services consists of masters, that maintain the state of the cluster and manage operations for the infrastructure, workers that are commanded by the masters and run the actual workloads and a storage solution provides persistent storage services for the applications. For this project a deployment tool is selected to deploy the cluster and its services. In the Rancher family, the management tool is called Rancher, which is used to create and manage the k8s cluster.

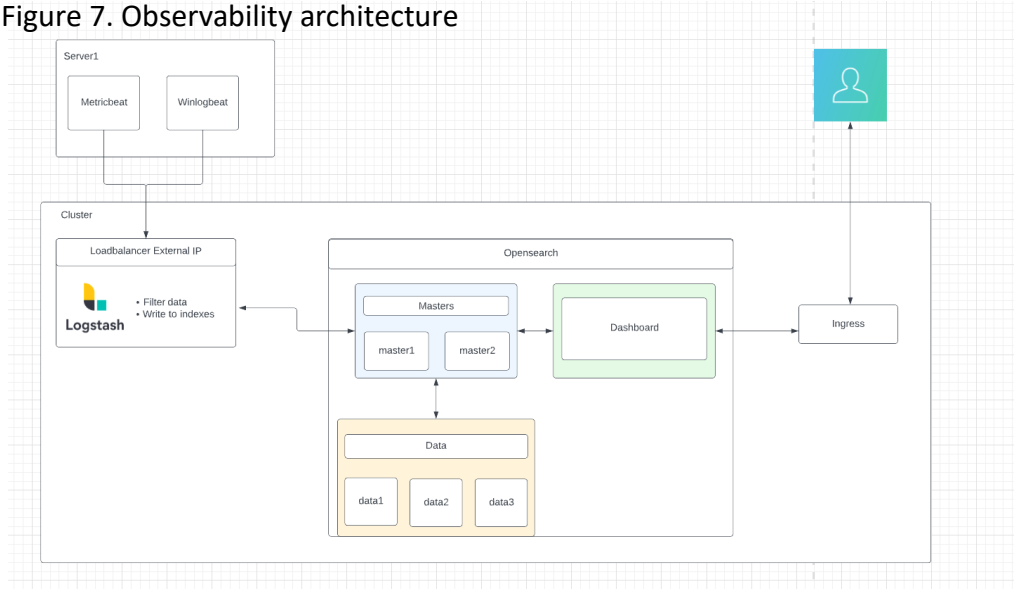
4.1.1 Observability deployment

Once infrastructure has been set up and a platform that provides the Kubernetes services have been installed, the next step is to consider how to run opensearch. In Kubernetes this means a deployment. A deployment provides declarative updates for Pods and ReplicaSets. In this case, deployments set the state for Opensearch pods and tell Kubernetes to keep them in that state. Development testing can be done with default values that the helm chart contains. Normally, testing can be done with a single node, but default configuration expects at least a three node setup.

Deployment consists three different parts: Master, data and dashboard. Overall result will follow architecture as in figure 7.

- Master role manages all critical tasks example, creating and deleting indexes.
- Data roles are responsible of the data and performing all data related tasks, searching, indexing and CRUD operation.
- Dashboard then helps visualize these's indexes and ables user search data with graphical user interface.

Figure 7. Observability architecture



The installation command for Logstash is as follows in figure 8 and complete deployment file can be found in appendix 1.

Figure 8. Deploy Logstash

```
# Deploying Logstash

kubectl apply -f logstash.yaml
```

Complete deployment commands are listed in figure 9. As mentioned, it is possible to use default chart in development environment, but one possible modification is to deploy the opensearch master and data components separately. This can be achieved by adjusting the configuration in the "values.yaml" file to specify separate deployments for the master and data nodes. By making these modifications, chart can be customized to meet specific requirements, ensuring that the master and data components are deployed independently for improved performance and scalability. This specific command retrieves the default values for a Helm chart called "opensearch/opensearch" The > symbol redirects the output of the "helm show values" command into specified file, in this case "values.yaml."

Figure 9. Complete deployment command list

```
# Add opensearch project into Helm

helm repo add opensearch https://opensearch-project.github.io/helm-project

# Retrieve default values into file

helm show values opensearch/opensearch > values.yaml

# Deploy opensearch master nodes

helm install opensearch-master-node opensearch/opensearch -n default -f
master_values.yaml

# Deploy opensearch data nodes

helm install opensearch-data-node opensearch/opensearch -n default -f
data_values.yaml

# Deploy opensearch dashboard

helm install opensearch-dashboard opensearch/opensearch-dashboards -n default
```

4.1.2 Agent installation

Agents are deployed using a helm installation as in figure 10 and installed into an environment which is running k3s lightweight kubernetes distribution. Agents are gathering metrics and logs only from containers and send these to Logstash for data processing. Logs and metrics from the node itself are not gathered.

Figure 10. Agent installation with helm

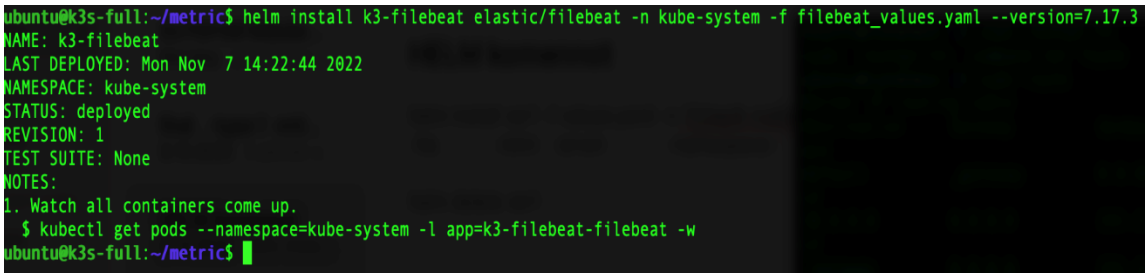


Figure 11 shows values file what can be updated before installation process. Values can also be edited after deployment is done, but this can lead other errors what deployment log may not recognize.

Figure 11. Agent values.

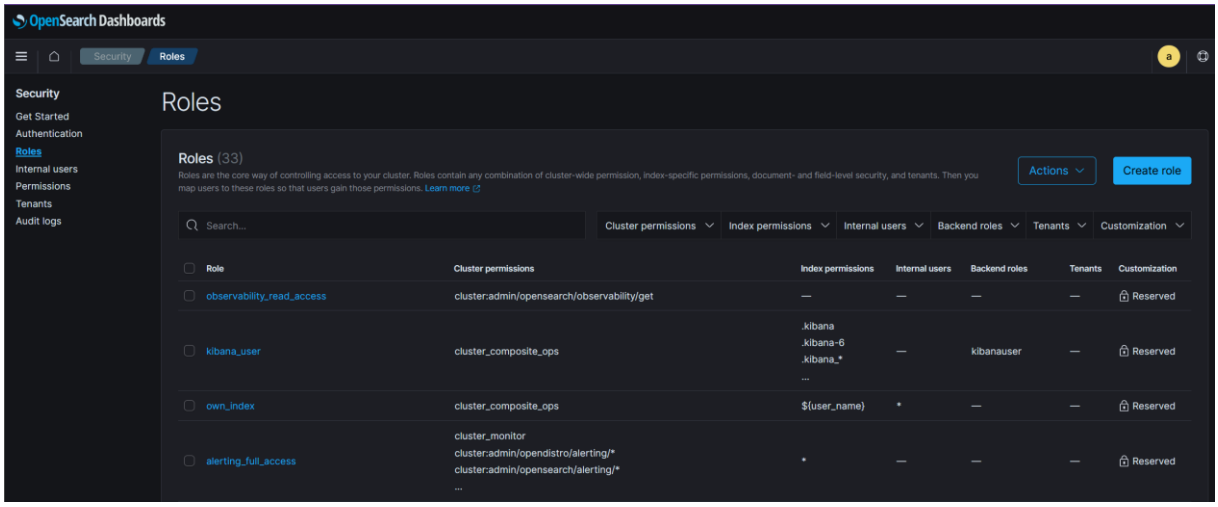
```
filebeatConfig:
  filebeat.yml: |
    filebeat.inputs:  # log location
    - type: container
      paths:
        - /var/log/containers/*.log
      processors:
      - add_kubernetes_metadata:
          host: ${NODE_NAME}
          matchers:
          - logs_path:
              logs_path: "/var/log/containers/"

    # Output to Datalounges R4DAR
    output.logstash:
      hosts: "URL or ip for logstash:5044"
      timeout: 10
      bulk_max_size: 512
      max_retries: 1
      ssl.enabled: true
      ssl.verification_mode: "none"
      ssl.certificate: "/usr/share/r4dar/certs/client.pem"
      ssl.key: "/usr/share/r4dar/certs/client-key.pem"
      ssl.certificate_authorities: "/usr/share/r4dar/certs/root-ca.pem"
```

4.2 Role based views

Role based access control comes with multiple default option as seen in figure 12, which can be pointed to a user or group. It can allow teams to access only their own indexes and dashboards or allow user or groups to access alert control, but not data itself.

Opensearch supports the required use cases for user management and roles put forward in the Datalounges requirements.

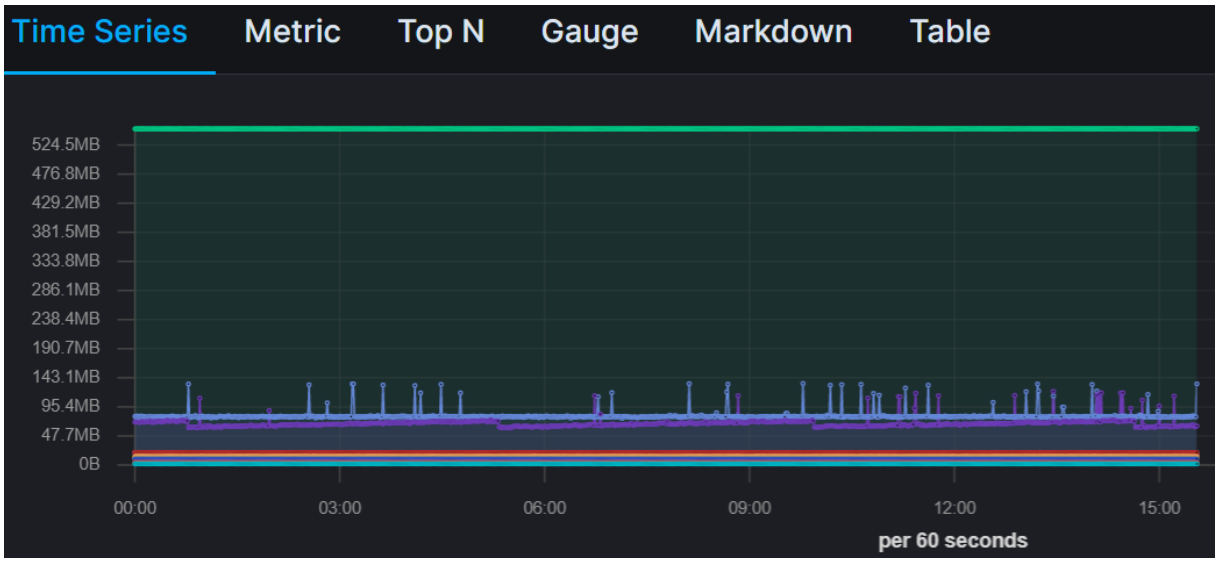
Figure 12. Opensearch roles



4.3 Observability

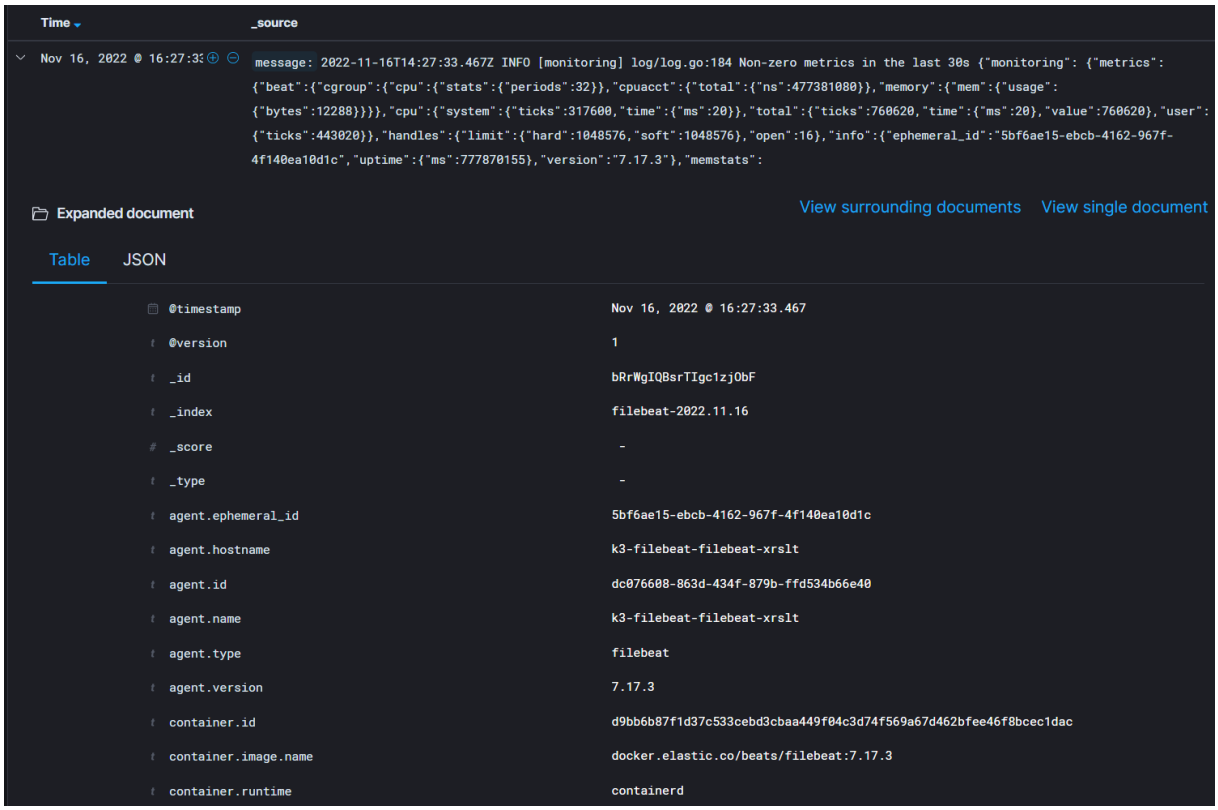
Visualizations of metrics are key components of the dashboards. A monitoring team can have multiple dashboards which are built for a single application to one quick scope view that summarizes status of the environment. Figure 13 shows example of max memory usage of all pods.

Figure 13. Pod's memory usage



Discover gives user easy access to logs using DQL (Dashboard Query Language) or a selection of ready fields where to find information what is needed. This is useful for single monitoring sessions and gives an ability to build own visualization views to track single id events. Figure 14 shows single metric event in discovery search.

Figure 14. Single event in discovery



Tracing is a default installed plugin in later Openseach Dashboard versions and allows application developers to have a better understanding of their applications behaviour and to see latencies directly in the application.

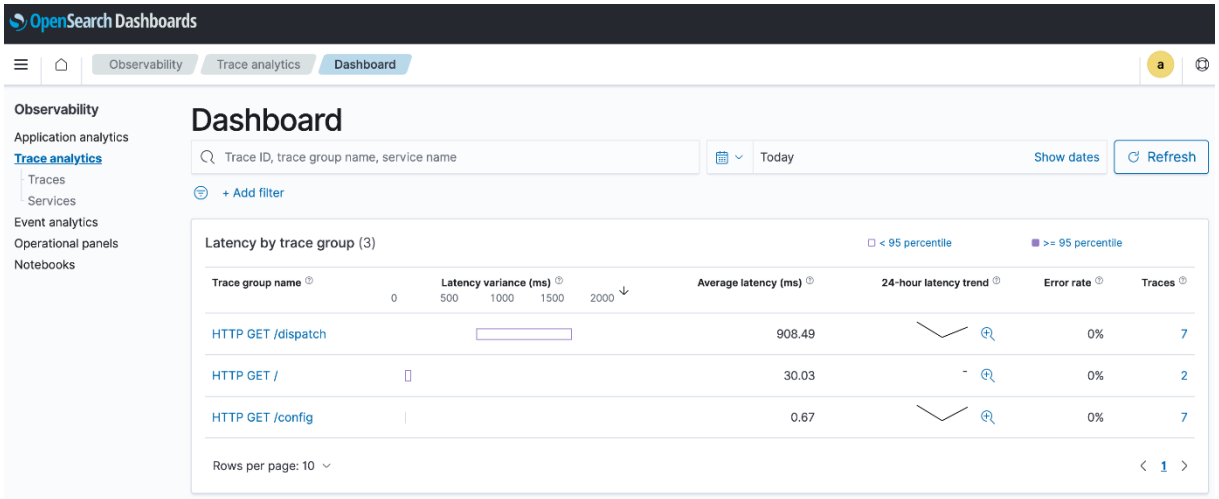
Trace requires adding instrumentation to application front end, example in figure 15.

Figure 15. Trace example in python

```
from opentelemetry import trace
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import
BatchSpanProcessor, ConsoleSpanExporter
from opentelemetry.sdk.resources import SERVICE_NAME,
Resource

# Service name is required for most backends,
# and although it's not necessary for console export,
# it's good to set service name anyways.
resource = Resource(attributes={
    SERVICE_NAME: "your-service-name"
})

provider = TracerProvider(resource=resource)
processor = BatchSpanProcessor(ConsoleSpanExporter())
provider.add_span_processor(processor)
trace.set_tracer_provider(provider)
```

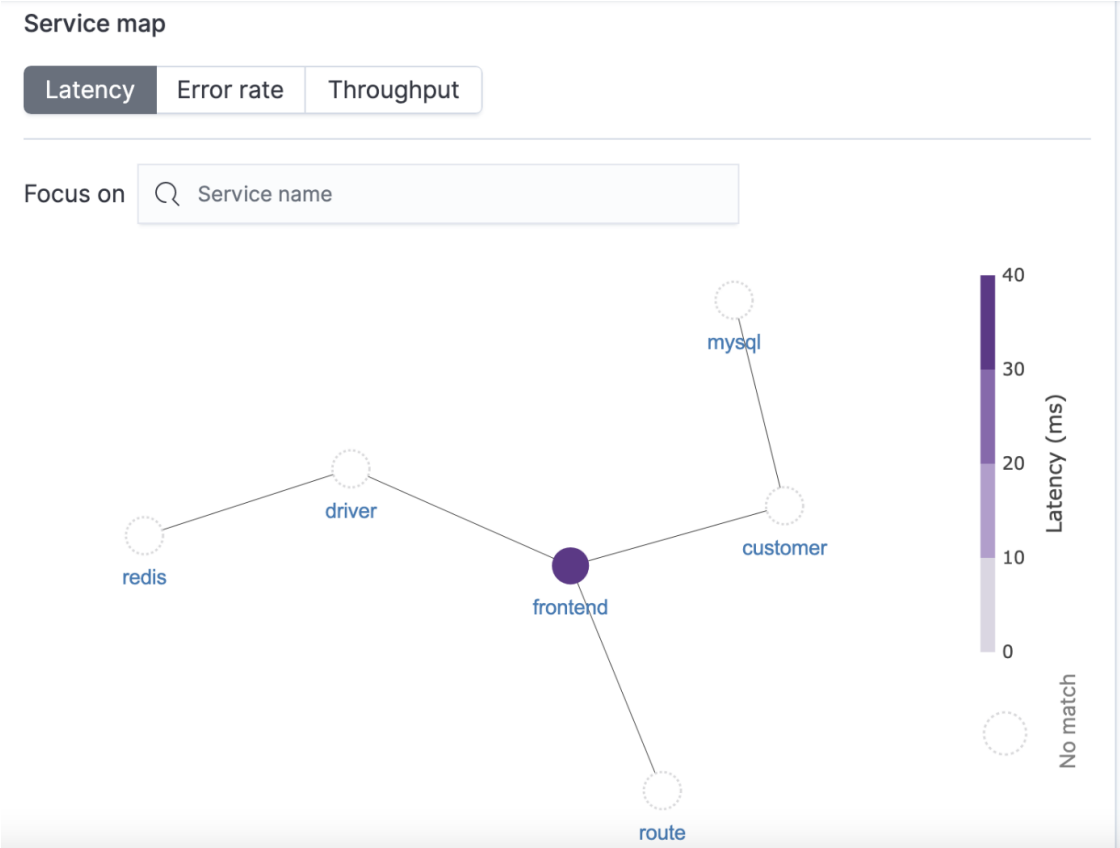
Trace analytics provides us valuable information of how our front-end application API responds to user selection. This helps developers understand response times better. Figure 16 shows example of trace analytics when web page is opened in K3s client machine.

Figure 16. Trace analytics view



In figure 17, service map draws visual representation of the various services and components that make up a software system, along with their relationships and dependencies. By tracing the interactions between these services and components, developers can identify potential bottlenecks, performance issues, or other problems that may affect the overall performance of the system.

Figure 17. Trace service map



5 Results

Finding the appropriate observability tooling posed a significant challenge in this project. Various factors were considered during the search for suitable solutions. These factors included comprehensive monitoring solutions that provide real-time insights, tools capable of collecting both infrastructure and application metrics, customizable dashboards and alerting mechanisms.

OpenSearch has many configuration options and configuring them correctly can be a daunting task. It's important to get the configuration right to ensure optimal performance and security. Inadequate resources can lead to poor performance, while over-provisioning can result in unnecessary costs and this needs to keep in mind if deployment is done into public cloud service provider.

Thesis first research question were how to achieve observability in Kubernetes, it is essential to find proper tooling and ensure a successful deployment. This involves selecting the right metric and logging collection, implementing tracing mechanisms and setting up alerts for proactive issue detection. The deployment was able to meet the requirements and performance was optimal and the application was able to handle the expected workload. Second research question was what factors influence the solution of the user interface. Considering the factors in table 3, OpenSearch dashboard stand out as the most favourable option for the user interface, offering a well-rounded combination of flexibility, ease of use, and maintainability.

In summary, the project of building observability and deploying it to a Kubernetes environment has been a great experience. Through this project, I gained hands-on experience in implementing observability tools, configuring monitoring solutions and building dashboard visualizations. This practical knowledge has equipped me with valuable expertise that can be applied to future projects and further advancements in Kubernetes and observability.

References

Arfan Sharif. (6. 2 2023). *Crowdstrike*. Noudettu osoitteesta https://www.crowdstrike.com/cybersecurity-101/observability/

Bigelow, S. J. (2021). *Techtarget*. Noudettu osoitteesta https://www.techtarget.com/searchitoperations/definition/distributed-tracing

Coralogix. (19. 7 2022). Noudettu osoitteesta https://coralogix.com/blog/what-is-tracing-everything-to-know/

Crowdstrike. (29. 6 2021). Noudettu osoitteesta https://www.humio.com/blog/observability-redefined/

Elastic. (n.d.). Noudettu osoitteesta https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-overview.html

Elastic. (n.d.). Noudettu osoitteesta https://www.elastic.co/guide/en/kibana/current/introduction.html

Elastic. (n.d.). Noudettu osoitteesta https://www.elastic.co/guide/en/kibana/current/alerting-getting-started.html

Ellingwood, J. (5. 12 2017). *DigitalOcean*. Noudettu osoitteesta https://www.digitalocean.com/community/tutorials/an-introduction-to-metrics-monitoring-and-alerting

G2. (n.d). Noudettu osoitteesta https://www.g2.com/categories/observability-solution-suites

Gartner. (n.d). *Gartner*. Noudettu osoitteesta https://www.gartner.com/reviews/market/application-performance-monitoring-and-observability

Google. (n.d.). *Google Cloud*. Noudettu osoitteesta https://cloud.google.com/architecture/devops/devops-measurement-monitoring-and-observability

Han, C. (25. 2 2019). *Medium*. Noudettu osoitteesta https://medium.com/hepsiburadatech/3-pillars-of-observability-d458c765dd26

Horovits, D. (n.d.). *Logz.io*. Noudettu osoitteesta https://logz.io/learn/complete-guide-elk-stack/

Intraclustr. (n.a). Noudettu osoitteesta https://www.instaclustr.com/support/documentation/opensearch/using-logstash/connecting-logstash-to-opensearch/

Livens, J. (1. 10 2021). *Dynatrace*. Noudettu osoitteesta https://www.dynatrace.com/news/blog/what-is-observability-2/

OpenSearch. (n.d.). Noudettu osoitteesta https://opensearch.org/docs/latest/im-plugin/index/

O'reilly. (n.d.). Noudettu osoitteesta https://www.oreilly.com/library/view/distributed-systems-observability/9781492033431/ch04.html

Padgett, C. M. (21. 6 2021). *Forbes*. Noudettu osoitteesta https://www.forbes.com/sites/forbesbooksauthors/2021/06/21/managing-known-and-unknown-unknowns/

Paperduty. (n.d.). Noudettu osoitteesta https://www.pagerduty.com/resources/learn/what-is-data-aggregation/

Parr, K. (n.d.). Noudettu osoitteesta https://venturebeat.com/data-infrastructure/introduction-to-observability-what-is-observability-and-why-is-it-important/

Seixas, V. (12. 7 2021). *Benefits of Observability*. Noudettu osoitteesta Azion: https://www.azion.com/en/blog/benefits-of-observability/

Sharif, Arfan. (21. 12 2022). Noudettu osoitteesta Crowdstrike: https://www.crowdstrike.com/cybersecurity-101/observability/log-file/

Splunk. (1. 3 2020). *Splunk*. Noudettu osoitteesta https://www.splunk.com/en_us/data-insider/what-is-observability.html#monitoring-and-observability

Splunk. (ei pvm). A beginner's guide to observability. s. 27.

Stripe. (2018). https://stripe.com/files/reports/the-developer-coefficient.pdf.

Verteuil, A. d. (6. 6 2022). Noudettu osoitteesta Grafana Labs: https://grafana.com/blog/2022/06/06/grafana-dashboards-a-complete-guide-to-all-the-different-types-you-can-build/

Wert, A. (27. 6 2019). *Novatec*. Noudettu osoitteesta https://www.novatec-gmbh.de/en/blog/5-reasons-why-opentelemetry-will-boost-observability-and-monitoring/

APPENDIX 1 LOGSTASH CONFIGURATION FILE

Manifest for deploying logstash

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: logstash-configmap
data:
  logstash.yml: |
    http.host: "0.0.0.0"
    path.config: /usr/share/logstash/pipeline
  logstash.conf: |
    input {
       beats {
         port => 5044
         ssl => true
         ssl_certificate_authorities => ["/usr/share/certs/root-ca.pem"]
         ssl_certificate => "/usr/share/certs/client.pem"
         ssl_key => "/usr/share/certs/client-key.pem"
         # ssl_verify_mode => "none"
       }
     }

    filter {
     }

     output {
        opensearch {
          user => "logstash"
          password => "U7KUqMIwabpjIt2" # "logstash"
          index => "%{[@metadata][beat]}-%{+YYYY.MM.dd}"
          hosts => ["https://opensearch-master:9200", "https://opensearch-data:9200"] #
["https://opensearch-master:9200","opensearch-cluster-data:9200"]
          cacert => "/usr/share/certs/root-ca.pem"
          ssl_certificate_verification => false
       }
      }

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: logstash
spec:
  replicas: 2
```

```
  selector:
    matchLabels:
      app: logstash
  template:
    metadata:
      labels:
        app: logstash
    spec:
      containers:
      - name: logstash
        image: opensearchproject/logstash-oss-with-opensearch-output-plugin:latest
        ports:
        - containerPort: 5044
        volumeMounts:
          - name: config-volume
            mountPath: /usr/share/logstash/config
          - name: logstash-pipeline-volume
            mountPath: /usr/share/logstash/pipeline
          - name: certs
            mountPath: /usr/share/certs
        resources:
            limits:
              memory: "8Gi"
              cpu: "2500m"
            requests:
              memory: "4Gi"
              cpu: "800m"
      volumes:
      - name: config-volume
        configMap:
          name: logstash-configmap
          items:
            - key: logstash.yml
              path: logstash.yml
      - name: certs
        secret:
          secretName: opensearch-certificates
      - name: logstash-pipeline-volume
        configMap:
          name: logstash-configmap
          items:
            - key: logstash.conf
              path: logstash.conf
---
kind: Service
apiVersion: v1
metadata:
```

```
  name: logstash
spec:
  selector:
    app: logstash
  ports:
  - protocol: TCP
    port: 5044
    targetPort: 5044
  type: LoadBalancer
```

Appendix 2 CSP requirements

Mandatory Requirements		
Requirement	Explanation	Additional info
Must run in CSP's own cloud infrastructure	The CSP provides Kubernetes capacity as-a-service. Observability deployment must run in that capacity.	
Must have the following integrations available	OIDC Harbor GitLab	
	Current Kubernetes version	
Must support Private and Public Cloud Kubernetes environments serviced by CSP	The solution must be able to collect logs, metrics and traces from AWS EKS, AKS, GKE and Rancher RKE2 Kubernetes environments	A common Kubernetes integration is required
Must have disaster recovery option	The solution must include a solution for disaster recovery so that customers using the service can be guaranteed for their data's availability	
Technical		
Must be containerized and run on Kubernetes	The services must be containerized and run in CSP's Kubernetes capacity	
Must be deployable as a Helm chart	Must be deployable as a Helm chart so that CSP's admins can deploy and redeploy the solution	
Must support the following use cases	Log management Metrics management Tracing Alerts generation	
Must support the following user tasks	Create views in the dashboard component as self-service Search logs and metrics Collect log item changes for alerts	

Optional Requirements		
Requirement	Explanation	Additional info
Deployable from Rancher	The Observability solution should be deployable from CSP's Rancher catalog	
Should support using CSP's logos and color schemes	The CSP has its own brand manual and the solution should support CSP's brands	logos, fonts, colors, style sheets
Technical		
Should be an Open Source solution	CSP prefers open source solutions for its deployments	Must allow providing solution as-a-service
Should support accepting the terms and conditions	Datalounges terms acceptance is necessary in the first workflow for user creation	