Haaga-Helia
ammattikorkeakoulu Oy

Creating centralized logging and monitoring to Zero Day Delivery project template

Janne Saikkonen

Thesis
Information Technology
2020

Haaga-Helia
ammattikorkeakoulu Oy

Author:
Janne Saikkonen

Degree programme:
Information Technology

Thesis title:
Creating centralized logging and monitoring to Zero Day Delivery project template

Number of pages and appendix pages
120

Abstract

This thesis was commissioned by the Finnish based software company called Eficode. Eficode has expressed the need for a centralized logging and monitoring solution for their software development platform called the Zero Day Delivery template.

The objective of this thesis was to develop the logging and monitoring solution for the template using open source logging and monitoring tools provided in the CNCF (Cloud Native Computing Foundation) landscape. The selected monitoring tools were Grafana and Prometheus which created the monitoring stack and the selected centralized log aggregation tool was Grafana Loki.

This thesis is divided to different sections presenting the theory related to logging, monitoring and security, small introduction to Zero Day Delivery templates technologies including the use of HELM and Kubernetes, introduction to Cloud Native Computing Foundation as an organisation and instructions on how to interpret CNCF Landscape and the technical implementation which covers the workflow with HELM and Kubernetes to install and upgrade Grafana, Prometheus and Grafana Loki to a Minikube cluster running the Zero Day Delivery template.

At the start of this thesis, the level of knowledge related to the technologies and concepts used in this project was minimal.

Keywords
Monitoring, Logging, Kubernetes, HELM, Grafana, Prometheus, CNCF, Grafana Loki

Contents

1 Terminology

1.1 Organisations

CNCF (Cloud Native Computing Foundation) is a foundation which empowers organisations to build and run scalable applications in the cloud, fosters and sustains an ecosystem of open-source and vendor-neutral projects related to cloud environment. (CNCF, 2020a)

IETF (Internet Engineering Task Force) is a large open international community of network designers, operators, vendors, and researchers focused on the evolution of the internet architecture and its smooth operation. IETF develops networking standards (IETF, 2020.)

EIC (International Electrotechnical Commission) prepares and publishes international standards for all electrical, electronic, and related technologies (IEC 2020.)

ISO (International Organization for Standardization) is an international organisation which develops and publishes international industrial and commercial standards. (ISO, 2020.)

ISO/IEC 27001 is a standard for information security management providing the requirements for an ISMS (Information Security Management System). Organisations implementing ISO/IEC 27001 can get certified on the standard to show they use the best practices in information security management. (ISO, 2020.)

1.2 Software

ZDD template (Zero Day Delivery -template) a template project in Eficode for creating microservice based projects using NodeJS and React (Eficode ZDD Confluence 2020.)

JIRA is a work/project management software designed by Atlassian to help track, manage, and collaborate in software projects. (Atlassian 2020a.)

Docker is a tool designed to create, deploy, and run applications in containerized environment (Docker 2020a.)

Kubernetes is an open-source platform for managing containerized workloads and services (Kubernetes 2020a.)

HELM is an application package manager designed to manage Kubernetes applications. (Helm 2020a.)

1.3 Monitoring terminology

Monitoring is the process of watching a system with a set of tools that provide information about the health and activity occurring within the system. (Kavis 2014, p. 118)

Metrics are measures of properties in an application or a system, recording of data points or observations over time (Turnbull 2016, p. 27)

1.4 Logging terminology

Logging is the process of collecting events into logs, which can be produced by a system, user application, servers, or operating system components. (Chuvakin, Schmidt & Phillips 2013, p. 31)

Log data is the information pulled out of a log message which gives you a description of something that has happened. (Chuvakin, Schmidt & Phillips 2013, p. 33 - 34)

Log message is generated by a device or a system to denote that something has happened (Chuvakin, Schmidt & Phillips 2013, p. 37)

Log source (or logging source) is an application or a system that is producing log messages (Chuvakin, Schmidt & Phillips 2013, p. 51)

Log transmission (or log data transmission) is the process of log message interchange between systems or applications over a network (Chuvakin, Schmidt & Phillips 2013, p. 35)

1.5 Programming terminology

Package manager is a tool used to automate the process of installing, upgrading, configuring, and removing programs. (Devopedia, 2018.)

Package is an archive that contains applications binaries, configuration, and dependencies. (Devopedia, 2018.)

Application (or app) is a program or a set of programs that are designed for end users to allow to perform functions (Twilio, 2020.)

Open source refers to a software, application, or a program whose source code is made freely available for anyone to inspect, modify, and distribute. (Open Source Initiative, 2020.)

Framework is a platform for developing applications, a predefined set of reusable code to solve common problems, dictating how the application development is done. (Wozniewicz, 01.02.2019.)

Library a set of predefined reusable code to solve common problems, not dictating how the application development is done. (Wozniewicz, 01.02.2019.)

NodeJS (or Node.js) is a synchronous event-driven JavaScript runtime designed to build scalable network applications. (OpenJS Foundation, 2020.)

Winston is a logging library for NodeJS applications. (Winston Github, 2020.)

JavaScript is a programming and scripting language designed to implement complex features to web pages. (Mozilla Developer Network 2020a.)

Git is an open-source distributed version control system. (Git-scm 2020.)

API (Application Programming Interface) is a set of programming instructions and functions to communicate between applications (Carnes 14.11.2019.)

UNIX refers to a family of computer operating systems and tools which provide a kernel, a shell, a file system, commands, and development environment. (Indiana University, 2020.)

HTTP (Hyper Text Transfer Protocol) is a client-server protocol which is the foundation of any data exchange on the Web. (Mozilla Developer Network, 2019)

HTTPS (Hyper Text Transfer Protocol Secure) is an encrypted version of the HTTP client-server protocol. (Mozilla Developer Network, 2020.)

TLS (Transport Layer Security) is a security protocol designed to encrypt the communication between web applications and servers. (Cloudflare Inc, 2020a.)

SSL (Secure Sockets Layer) is a security protocol designed to encrypt the internet communication ensuring privacy, authentication, and data integrity. SSL is the predecessor of TLS security protocol (Cloudflare inc, 2020b.)

YAML is a human friendly data serialization standard for all programming languages (YAML, 2020.).

JSON is human readable lightweight data-interchange format. (JSON, 2020.).

1.6 Kubernetes terminology

Cluster is a set of worker machines, called nodes, that run containerized applications. (Kubernetes, 2020j.)

Node is worker machine in Kubernetes. May be a virtual machine or a physical machine. (Kubernetes, 2020j.).

Object is an entity in the Kubernetes system. Representing the state of cluster, a record of intent (Kubernetes, 2020j.).

Pod is the smallest and simplest Kubernetes object. Represents a set of running containers in a cluster (Kubernetes, 2020j.).

Service is an abstract way to expose an application running on a set of Pods as a network service (Kubernetes, 2020j.).

Container a lightweight and portable executable image that contains software and all of this dependencies (Kubernetes, 2020j.).

Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. (Kubernetes, 2020c.).

RBAC (Role Based Access Control), a mother of regulating access to computer or network resources based on the roles of individual users (Kubernetes, 2020d.).

Minikube a local Kubernetes cluster. (Minikube, 2020a)

1.7 Helm terminology

Chart a packaging format for Helm. A collection of files that describe a related set of Kubernetes resources. (Helm, 2020h.).

Values referred as `values.yaml`. A way to override default chart values with custom values. (Helm, 2020h.)

Chart repository a location where packaged charts can be stored and shared (Helm, 2020i.)

2 Introduction

2.1 The project assigner: Eficode Oy

The assigner of the project is a Finnish based software company Eficode Oy. Eficode consists of 300+ designers, developers, coaches, automation engineers, UX (User Experience) and DevOps specialists with the vision of building the future of software development. Eficode has offices in Finland (Helsinki, Tampere), Denmark (Copenhagen, Aarhus), Germany (Düsseldorf, Munich, Berlin), Norway (Oslo), Poland (Lodz), Sweden (Stockholm, Malmö, Gothenburg) and Netherlands (Amsterdam).

Eficode business practice is based on providing consultancy services to their customers and partners. This means helping Eficodes customers and partners to overcome challenges with test automation to provide continuous testing and quality assurance, continuous development to release features on demand, provide a complete state-of-the-art software production line and toolchain orchestration with the Eficode ROOT product, identifying and resolving cultural problems in the ways of working, provide predictive data analytics and metric-driven development and AI -assisted software development.

2.1.1 Zero Day Delivery -template description

Eficode has developed a project template for customer SaaS (Software as a Service) -projects and for internal projects called Zero Day Delivery -template. The template serves as a part of DevOps working culture and part of the CI/CD (Continuous Integration/Continuous Delivery) concepts. The template can be setup as the base of a starting software development project (development environment) from day one (or zero in computing terms) meaning that Eficodes customers can see the value generated very early on and continuously regarding to their requirements of a SaaS -solution. (Eficode ZDD Confluence, 2020.).

The development of the project template is a continuous process within Eficode and Eficode has expressed the need for centralized logging and monitoring solution to the template, which can be developed further.

2.1.2 Project objective

With the ZDD -templates current state, software engineers need to research and develop their own solution related to monitoring and logging from the ground-up. The template already provides very basic tools for logging solution using NodeJS framework called Winston and not a single tool for a monitoring solution. The main objective in this project is to

provide a solution for a possible monitoring solution, logging solution and further development documentation for Eficodes software engineers which they can utilize in a start of a new customer project. Creating a ready base solution for logging and monitoring saves time related to researching, finding, and implementing a logging or a monitoring solution for the ZDD template.

2.2 Thesis objective & framework

The thesis objective is to showcase the development process related to the project of creating centralized logging and monitoring solution for the Zero Day Delivery template using logging and monitoring tools provided in the CNCF landscape. Therefore, the thesis consists of four parts: introduction to logging and monitoring, introduction to Zero Day -template technologies, introduction to CNCF (Cloud Native Computing Foundation) landscape and selecting the needed tools from the landscape and the technical implementation of selected tools for the ZDD project. Starting with the introduction to logging and monitoring section of this thesis.

- What is logging? Why we should do it? What are the best practices?
- What is monitoring? Why should we do it? What are the best practices?

Answering these types of questions should give the basic understanding why monitoring and logging matters when developing software and why it should be part of the development process.

The second big topic is the introduction to Zero Day -templates technologies. In logging and monitoring perspective there are various ways of producing log data, metrics and deploying logging and monitoring frameworks with infrastructure automation tools and configuring the found solutions to fit the need of a software development project. The introduction also serves as an informative topic related to the security standards Eficode is pursuing (ISO:27001 certification). In this section I'll cover the following:

- How the Zero Day Delivery -template works
- NodeJS: What is NodeJS? How logging is done in NodeJS?
- Docker: What is Docker? What is containerization or containerized environment?
- Kubernetes: What is Kubernetes? What is Kubernetes used for?
- HELM: What is HELM? What is HELM used for?
- Security: ISO 27001 certification and TLS

This section of the thesis may be considered as an introduction to the technical implementation and should describe the necessary information needed to understand the development steps taken in the technical implementation.

The third section covers the topics of CNCF (Cloud Native Computing Foundation) as an organisation, the CNCFs Cloud Native Interactive Landscape, the use, and the selection of tools from that landscape. This section also serves as an introduction to the selected monitoring and logging tools. The selection is based on the requirements set by the ZDD development team in the project's Jira Kanban board (Eficode ZDD Jira, 2020.).:

- Framework/library supports trackability, security requirements, different cloud environments and problem analytics.
- Framework/library log system can be extended to log-based monitoring and alerting – when critical issues are logged the development team should be alerted.
- Framework/library is simple, not requiring too much of learning or maintenance.
- Framework/library provides basics statistics about the logs.

This section should give the basic understanding of CNCF as an organisation, the CNCF Landscape and give the necessary explanation on why certain tools were selected from the CNCF landscape.

The fifth section is the technical implementation of the ZDD logging and monitoring project. This describes the development process including the workflow with project management tools (e.g. Jira) and introducing a Kubernetes and HELM workflow to streamline the development process. This section also describes the steps taken when developing and implementing the selected tools from the CNCF landscape to ZDD project.

The final section of the thesis will be the project summary. The summary section can be considered as a retrospect. What did the ZDD logging and monitoring development project achieve as a result, what kind of personal development goals were reached and overall picture of the development process.

2.2.1 Thesis restrictions

Due to the vast scope size of this project, some restrictions have been set for the thesis. First of all, it is imperative to understand that this project has no end or a due date. This means the ZDD project as a whole is an on-going and continuous development process within Eficode and part of the Eficodes internal projects to create necessary tools for Eficode software developers to use. The thesis will not describe the ZDD templates full capabilities including some technologies used in ZDD and will not describe the concepts of DevOps or test automation. This thesis only describes the necessary concepts, technologies and the components and features of those technologies related to the development

of this project. The thesis also has restrictions related to the NDA (Non-Disclosure Agreement) with Eficode, this covers items like URL’s e.g. Zero Day Delivery documentation pages in Confluence and Bitbucket.

3 Theory

This chapter of the thesis focuses on building the knowledge base and covering the fundamentals of the logging and monitoring processes. In the thesis preliminary research regarding to source material of this thesis, I ended up using three main book sources: Architecting the Cloud by Michael J. Kavis, Logging and Log Management by A. Chuvakin, K. Schmidt and C. Phillips and The Art of Monitoring by James Turnbull.

The book *Logging and Log Management (A. Chuvakin, K. Schmidt & C. Philips, 2013)* gives a good theory basis for the logging process. The books content is mainly focused on managing system logs, this means the infrastructure of systems, applications, and devices, but in my opinion the overall theory of logging provided in the book can be used in web application development.

The book *The Art of Monitoring (J. Turnbull, 2016)* is the main source for describing the theory of monitoring. With the books content, it serves developers who are creating monitoring solutions for their projects and serves the DevOps working culture. On the downside, the book requires the reader to have a basic understanding of UNIX, CLI (Command-line interface) usage, package managers and how networking works.

The last book, *Architecting the Cloud (M. J. Kavis , 2014)* is used to refer how logging and monitoring should be done in the cloud environment, since the ZDD template is designed to work in the cloud. The book works well to fill in the blanks and backup the theory provided in the books *Logging and Log Management* and *The Art of Monitoring*.

There are also some web sources used in order to complete the theory section.

3.1 Introduction to logging

What is logging? Logging is the process of collecting event changes from an environment and recording it into a log. That environment can be anything from a web application to a computer system and the event is a single occurrence within that environment, usually involving attempted state change in that environment. Event change is a stimulus to which the web application or the computer system response and generates a log message which is stored in a log file. (Chuvakin, Schmidt & Phillips 2013, p. 60 - 63)

> “Log all you can (which is as much as possible), but alert only on what you must respond (which is as little as possible).” (Chuvakin, Schmidt & Phillips 2013, p. 264)

3.1.1 Log data

Log data is the information pulled out of a log message which gives you a description of something that has happened. Various devices, applications and systems have their own logging system and provide information for various uses. These uses can be anything from system resource management (printers, disk systems, battery backup systems, operating systems), user application management (login and logout, application access) and security. Log data is not restricted to a single category, there can be multiple categories where log data can be used e.g. user management and security utilizes log data information on user login and logout messages. The idea is to access information in the basis of prevention before things escalate. (Chuvakin, Schmidt & Phillips 2013, p. 33 - 34)

3.1.2 Log message

What is log message? A log message is created when e.g. a web application response to a change in its environment. It is a description of a single event, a record of something that has happened. What does a log message look like? In very basic form the content (log data) of a log message contains:

- Timestamp
- Source
- Data

Picture 1: Log message demonstration using Chrome console and JavaScript

Timestamp indicates the time when the log message was generated. The source is the origin, where the log message was created and data, being the action of the event. In the example (Picture 1), timestamp consists of date (30th of March 2019) and time (16:16:38). The source consists of filename (App.js) and line in the file (7), which can be a function which starts on line 7. The data tell us the functions action: created a new user with an ID (5). A good structured log message helps with log analysis, a process of reviewing and interpreting logs. (Chuvakin, Schmidt & Phillips 2013, p. 37 - 38)

There is currently no de facto standard format on how the log message data should be represented. It is freeform, organisations must create their own way of standardisation for logging their applications by creating a centralized logging plan / strategy. (Kavis 2014, p. 122 – 123)

3.1.3 Log transmission and collection

Logs come from various sources: the infrastructure, the application stack and the application itself. Best practice is to log every event that happens in the application, especially events that are related to users and request access (Kavis 2014, p. 117 - 118). But how the transmission and collection work? Logging sources can be categorized into two groups:

- push-based log sources
- pull-based log sources

With push-based log sources, the application emits log messages either to a local disk or over the network to a database. If the transmission happens over the network, you need a log collector. With pull-based log sources, the method relies on client-server model: an application pulls the log messages from the source e.g. the infrastructure or the application stack and stores them to a database. (Chuvakin, Schmidt & Phillips 2013, p. 82 - 83)
The most common way to do log transmission is with the Syslog protocol, which is a standard for log message interchange, commonly found in UNIX based platforms, but also exists as a standard in non-UNIX based platforms. The log transfer is done between client and server components implementing the use of UDP (User Datagram Protocol). There are implementations which also support TCP (Transmission Control Protocol) as open source or commercial option. (Chuvakin, Schmidt & Phillips 2013, p. 83)

For web applications, databases can be used to store log messages. Instead of syslog messages, application can write its log messages directly to a database. This provides a structured way to store, analyse and report log messages. For analysis and reporting, third-party applications can be used, these are known as proprietary logging formats. Third-party applications have their own mechanisms for generating and retrieving log messages using API (Application Programming Interface). (Chuvakin, Schmidt & Phillips 2013, p. 36).

Collection of logs should be centralized. This means implementing a solution for a log host, a place where log messages are transmitted from various sources where log data can be stored and analysed. (Chuvakin, Schmidt, Phillips 2013, p. 82).

3.1.4 Log usage

What are logs? A log (or a log file) is a collection of log messages, a record of events. Depending on the scale of an application, as mentioned before logs can be gathered from various log sources and have various uses. In matter of log use-variety e.g. logging user activity provides log data for a security log, debug logging provides log data for application

development purposes. Log sources can be the backend of a web application (user login events) or a client-side browser logs (client browser name, browser version) (Chuvakin, Schmidt, Phillips 2013, p 29).

Table 1. Core list of log uses (Kavis 2014, p. 119 – 120)

Log usage	Description
Troubleshoot-ing	"Debugging information and error messages are collected for analysing what is occurring in the production environment."
Security	"Tracking all user access, both successful and unsuccessful access attempts. Intrusion detection and fraud detection analysis depends on collecting logs."
Auditing	"Providing a trail of data for auditors is mandatory for passing audits. Having documentation of processes and controls is not enough to pass an audit. Those documents need to be backed up with real data from the logs."
Monitoring	"Identifying trends, anomalies, thresholds and other variables proactively allows companies to resolve issues before they become noticeable and before they impact end users."

Logs provide useful information depending on the use case as seen in table 1. The use cases vary. E.g. In cloud-based applications the computational resources are scaled automatically, this means the workload can spike up and down and they should be monitored. This is when the logs related to resource management (databased activity, CPU usage and memory usage) come in handy. (Kavis 2014, p. 119)

Logs can also be used for process called "forensics", which is part of troubleshooting (see table 1). The process involves finding out "what has happened" after an unwanted event is over. If a web application has a solid logging system, forensics can be done effortlessly. E.g. Log data timestamps can give a good picture of chronological event history, showing what has happened, when it happened and in what order. Log analysis is the process to describe already mentioned log use cases. Simply, log analysis is analysing log data to get an understanding from it. (Chuvakin, Schmidt & Phillips 2013, p. 45, 48)

3.2 Centralized logging plan

Logging is a critical component of any application. Getting the needed data from logs can be a quite overwhelming task like finding a needle in a haystack, this is because of the volume of data a web application can produce. Organisations must create a centralized logging plan to combat this problem, there are two requirements when it comes to centralized logging plan: directing logs to a storage and standardising the log formats. (Kavis 2014, p. 119)

3.2.1 Standardising logs

Standardising logs is one of the key requirements of centralized logging plan. This means creating a naming convention, severity levels, log format and error codes for all log messages. As stated previously, there is no actual de facto standard for log messages. This can impact the value of log data if the development team or the project organisation does not implement an effective logging plan. (Kavis 2014, p. 122 – 123)

Table 2. RFC 5424 Syslog protocol (Kavis 2014, p. 123; IETF 2020.)

Code	Severity
0	Emergency, system is unusable
1	Alert: action must be taken immediately
2	Critical: critical conditions
3	Error: error conditions
4	Warning: Warning conditions
5	Notice: normal, but significant conditions
6	Informational: informational messages
7	Debug: debug-level message

Best practice is to use the common log message formats, like the RFC 5424 Syslog protocol to standardise severity levels demonstrated in table 2. (Kavis 2014, p. 123). Chuvakin, Schmidt and Phillips present a good description for different log message categories.

Table 3. Log message classification (Chuvakin, Schmidt & Phillips 2013, p. 34)

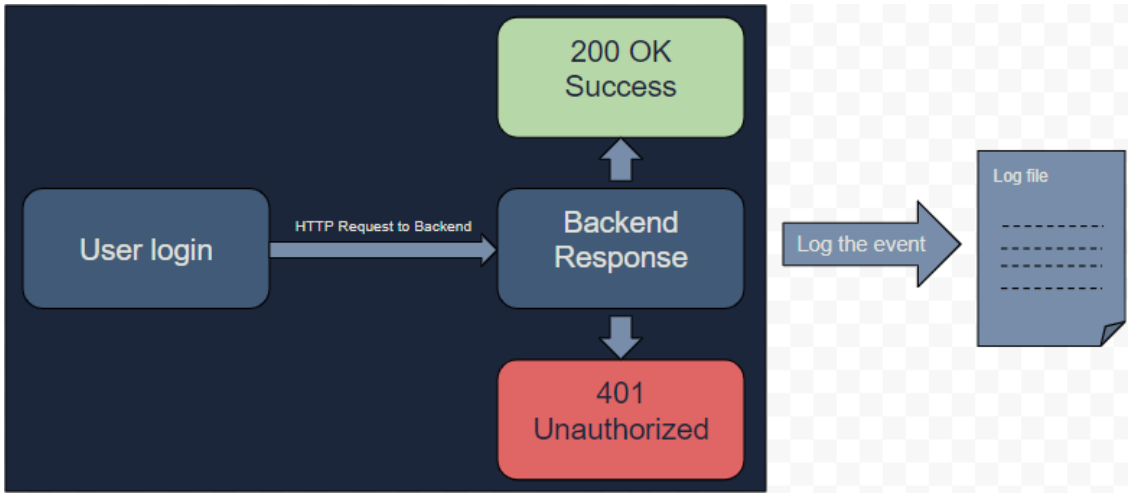
Classification	Description
Informational	“Messages of this type are designed to let users and administrators know that something benign has occurred.”
Debug	“Debug messages are generally generated from software systems in order to aid software developers troubleshoot and identify problems with running application code.”
Warning	“Warning messages are concerned with situations where things may be missing or needed for a system.”
Error	“Error log messages are used to relay errors that occur at various levels in a computer system”
Alert	“An alert is meant to indicate that something interesting has happened.”

Informational log messages can be e.g. device generating informational message when the system is rebooted. Debug log messages can be failed login attempts. Warning log messages can be e.g. application level messages informing that application is missing a CLI (Command-line interface) command the application needs, but the application can run without it or library/framework log system informing soon-to-be deprecated functions, methods, or dependencies. Error log messages can be formed e.g. when an OS (Operating System) fails to synchronize buffers to a disk. Alert messages can be generated by an IPS (Intrusion Prevention System) creating a log message informing about an unwanted connection from an unwanted client to the web server (Chuvakin, Schmidt & Phillips 2013, p. 34 - 35, 74)

Table 4. HTTP Status Codes (RESTfulAPI, 2020).

Category	Description
1xx: Informational	“Communicates transfer protocol-level information”
2xx: Success	“Indicates that the client’s request was accepted successfully.”
3xx: Redirection	“Indicates that the client must take some additional action in order to complete their request.”
4xx: Client Error	“This category of error status codes points the finger at clients.”
5xx: Server Error	“The server takes responsibility for these error status codes.”

What should be logged? In web applications we might be very interested in user HTTP request events demonstrated in table 4. Especially when a new user is creating a new account (see Picture 1) or if a user is attempting to login the web application:



Picture 2: User login event with two outcomes

In Picture 2 there are two outcomes when the user login the web application. If the user gives the correct username and password HTTP request returns 200 success code if it doesn’t the HTTP request returns 401 unauthorized code. (DZone 2009).

3.2.2 Log Storage

In order to search and analyse logs, the web applications logging solution should consist a plan for log retention. This is the second requirement of centralized logging plan. (Kavis 2014, p. 120).

As mentioned previously in *2.1.3 log transmission and collection:* the collection of logs should be centralized. This means creating a solution for a log storage. There are various storage and logging solutions mentioned by Michael J. Kavis in his book.

Table 5. CSP (Cloud Service Provider) logging solutions (Kavis 2014, p. 120 – 122)

Solution	Description
Infrastructure as a Service (IaaS)	All logs are directed to syslog instead of being written directly to disk on the local machine. Syslog on each server is piped directly to a dedicated logging server farm, based on CSP's (Cloud Service Providers) infrastructure.
Software as a Service (SaaS)	"The logs are sent to cloud-based centralized logging Database as a Service provided by a CSP (Cloud Service Provider)
Platform as a Service (PaaS)	Many PaaS solutions are integrated with the most popular logging SaaS solutions which provide API access to the central logging solution provided by a CSP (Cloud Service Provider)

Opening up the solutions provided in table 5, in IaaS -solution, developers leverage the CSP's (Cloud Service Provider) infrastructure to build their own logging solution on top of the logging server farm. Open source or commercial logging solutions (proprietary logging formats) can be used to transform e.g. the syslog data to NoSQL format providing easy-to-use search, event processing and notification (Alerting) capabilities. In SaaS -solutions, the logs are sent to centralized cloud-based storage, which utilises the use of CSP's logging solutions. The development team no longer needs to build, manage or maintain their own logging solution. In PaaS -solutions, the development team can utilise the use of CSP's various integrated SaaS logging solutions and which can be turned on when needed to e.g. Loggly by Solarwinds. (Kavis 2014, p. 120 - 122)

A central repository for storing logs and using open source or commercial logging solutions enables development teams to manage, analyse, audit, secure and create alerts. There is also a benefit of minimizing loss of log data by sending logs to a CSP provided storage, since logs are not stored on a local disk. (Kavis 2014, p. 121)

3.3 Introduction to monitoring

> "Without monitoring you are not doing your job." (Turnbull 2016, p. 9)

Monitoring is the process of watching over e.g. the web applications health, real-time activity occurring in the application (Kavis 2014, p. 118) and the set the of tools and processes to measure and manage systems or an application (Turnbull 2016, p. 8). Implementing a monitoring solution enables a team of developers or a developer to turn metrics into measurable outcomes e.g. measurable user experience data provides feedback to the business to ensure it is delivering what customer wants. (Turnbull 2016, p. 8). There are two recognized "customers", when it comes to monitoring systems:

- The business customer
- The Information technology customer

Firstly, the business customer. The implemented monitoring system should support the business. As stated previously, monitoring provides valuable information from user experience data which allows businesses to make good product and even technology investments. Monitoring helps to measure the value which technology delivers. The second customer is the information technology, the development team. Monitoring can be used to detect, diagnose, and help to resolve issues in the application environment (Turnbull 2016, p. 8 - 9.).

In the scope of the thesis and the project, I am focusing on the latter, the application development aspect of a monitoring solution. This is the key step for the ZDD template to provide metrics e.g. at the start of any new SaaS (Software as a Service) -project.

3.3.1 Defining metrics

What is a metric or a measurement? Both are a critical for monitoring solution. Metrics are measures of properties in an application or a system, recording of data points or observations over time. In a web application we might want to collect metrics in a time series e.g. unique website visits over time (Turnbull 2016, p. 27). I highly think that Turnbull describes a measurement as a single point of data, an absolute value of a single event and a metric is quantifiable, meaning it can be an outcome from one measurement or multiple measurements. To me, this makes sense.

Table 6. Basic metric types (Turnbull 2016, p. 29 - 30)

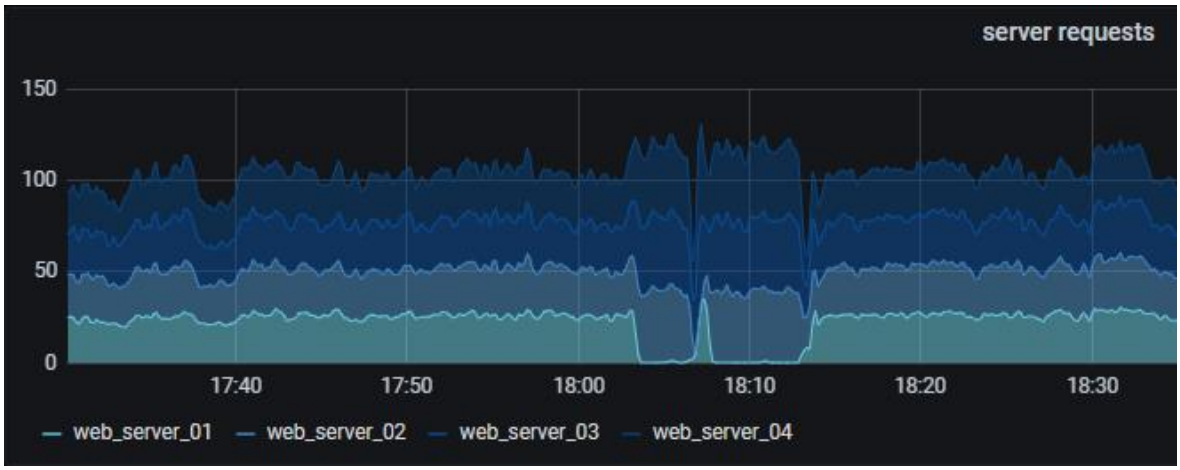
Metric type	Description
Gauges	Most common type of metric. Gauges are numbers that are expected to change over time. A snapshot of a specific measurement.
Counters	Counters are number over time and never decrease.
Timers	E.g. tracks how long it took for a function or a method to complete.

There are a variety of different types of metrics showcased in table 6. Gauges can be e.g. used to show how much CPU (Central Processing Unit) or memory is being utilized. Counters can be used to present system uptime, number of logins or previously mentioned unique website visits. Timers are self-explanatory, being a simple presentation of how much time a certain action took e.g. HTML -document loading time in a website or to represent system/server uptime (Turnbull 2016, p. 29 - 30).

Table 7. Metric summaries (Turnbull 2016, p. 31 - 32)

Summary type	Description
Count	Count the number of observations in a specific time interval.
Sum	Adding together (sum) values from all observations in a specific time interval.
Average	The mean (average) of all values in a specific time interval.
Median	The exact centre of values.
Percentiles	Measures the values below, which a given percentage of observations in a group of observations fall.
Standard Deviation	Distribution of metrics used to measure variation in a data set by setting a range of values.
Rates of Change	Representation to show the degree of change between data in a time series.
Frequency distribution	Grouping of selected data set and visualizing the frequency of a group.

Often metrics are used to summarize observations when a single metric is not useful enough. This can be done by visualizing the metrics by applying mathematical functions seen in table 7. In addition to metric summaries, metrics can be visualized as an aggregated metrics. This means taking multiple metric sources from an environment and visualizing it e.g. unique server requests across all servers. The most common way to visualize metric aggregations is in a single plot. (Turnbull 2014, p. 32 - 34)



Picture 3: Metric aggregation of server requests in Grafana (Turnbull 2014, p. 33)

Picture 3 is a demonstration of metric aggregation.

- On the Y -axis of the plot is a presentation of unique server requests ranging from 0 to 150.
- On the X -axis of the plot is a presentation of a time interval ranging from 17:30 to 18:30.
- On the bottom, below the X -axis time interval is presentation of which web servers are monitored in order to gather metrics.

(Turnbull 2014, p. 33)

As a summary, I have a lot of ways to visualise metrics. I highly believe that in the scope of this thesis, I do not need to reinvent the wheel. The monitoring frameworks comes with basic and advanced templates for metric types. At least, this is the assumption I have at this point.

3.3.2 Monitoring maturity

I noticed that both Turnbull and Kavis note in their books a certain level of monitoring maturity. The maturity describes different stages of the development process for a monitoring solution. There are certain thresholds which a monitoring solution needs to reach in order to “advance” to a next level of maturity.

Table 8. The monitoring maturity types (Turnbull 2014, p. 12 - 17)

Stages of maturity	Description
Manual, user-initiated or no monitoring	Monitoring provides little or no value.
Reactive	Mostly automated with some remnants of manual or un-monitored components.
Proactive	Monitoring is automated, considered as a core component for managing infrastructure and business.

The maturity types can be seen in table 8. In the first stage (manual, user-initiated or no monitoring), if monitoring is done at all it is done with a minimal effort without any automation. This means using simple scripts, checklists, or other non-automated processes to provide data. The focus is mainly on minimizing downtimes and managing assets. (Turnbull 2014, p. 12 - 17).

Reactive is the second stage of monitoring maturity. Reactive monitoring contains mostly automated process' and automation is done with deployed monitoring tools which produce basic metrics e.g. CPU, disk, and memory usage with some alerting capabilities with set thresholds. Some remnants of manual or unmonitored might still be present (Turnbull 2014, p. 15 – 18). Reactive monitoring is mostly focused on detection, this means the monitoring tools are used to tell if something is failing or something is about to fail. (Kavis 2014, p. 137).

Proactive is the third stage. Proactive monitoring is considered as a core component to manage infrastructure and business with fully automated components, providing metric data for business outcomes, focusing on delivering measurements on quality of service and customer experience (Turnbull 2014, p. 8, 18). Proactive monitoring focuses on prevention and holds a different mindset than detection. In order to do efficient prevention of failures, prevention requires a definition: what are the metrics of a healthy system and use that as baseline. After the baseline is set, metrics data with set thresholds can be used to detect when the system or application is trending towards an unhealthy state. (Kavis 2014, p. 137)

As a best practice, implementing both proactive and reactive monitoring to an application strives to find and resolve issues early on (Kavis 2014, p. 138).

3.3.3 Monitoring strategy

Just like logging, a working monitoring solution needs a plan, a strategy to implement monitoring system to an application which should be put in place early on and continuously improved over time. (Kavis 2014, p. 147)

The big question is “what needs to be monitored?”. The purpose of monitoring is to help track if the application or system is working as intended. In the cloud environment the application can be divided into different layers, which we call the cloud stack:

- User layer
- Application layer
- Application stack layer
- Infrastructure layer

Each of the layers can produce different metrics and have different use cases depending on who is working with the application. E.g. Front-end developer (Application layer / Application stack layer) might be interested in page load times, overall network performance and the performance of developed APIs (Application Programming Interface). Database architects (Infrastructure layer) might be interested in threads, cache, memory or CPU utilization. System administrators (Infrastructure layer) in RPS (request per second) and disk space capacity. In the business perspective, the product owner (User layer) might be interested in business-related metrics like unique page visits per day, new users and cost per user. This is all based on customer needs, development teams need, or overall organization’s needs for different metrics and measurements. (Kavis 2014, p. 138 - 139).

Table 9. Metrics categories (Kavis 2014, p. 139 – 145)

Category	Description
Performance	Works within each layer of the cloud stack. Measures and tracks the behaviour of the users (or customers) interacting with the application, measures how the application responds to the actions and track the performance of the underlying components (operating system) and infrastructure (network, servers).
Throughput	Works within each layer of the cloud stack. Measuring data flow through the cloud stack, from layer to layer per unique user. Throughput is usually measured as TPS (Transactions per second). Critical for diagnostics.
Quality	Works in the user and application layer. Quality measurements require standardisation of data collection (e.g. Logs). Measures the success and accuracy of user registration and access. Provides metrics for alerting systems and reports.
KPIs (Key performance indicators)	Works in the application layer. Provide metrics for business analysis e.g. site traffic, shopping cart abandonment rate.
Security	Should work within each layer of the cloud. Provides metrics for application security issues e.g. mining log files and discovering abnormal patterns e.g. unique IP address trying to access application multiple times in a short time span.
Compliance	For applications that fall under regulation constraints. Requires policies and procedures to follow the application and business e.g. creating a policy to restrict application production access for certain developers and raise alerts if policies are broken.

All the following metrics can be categorized as seen in table 9. This gives insight on what should be monitored. As a summary, organisations or the development team must think what needs to be monitored right from the start and how to expand the monitoring capabilities throughout the application life cycle to meet the standards of proactive monitoring.

3.4 Security

The Zero Day -template can be setup in the cloud environment and can be used in a customer or internal software projects, therefore security as a topic should be addressed. The level of security required depends on numerous factors from target industry (e.g. health care, government), customer expectations to sensitivity of data being stored and data transmission. In this thesis's scope, we will be focusing on security issues related to logging and monitoring though security implementation is not in the technical scope. It is in the scope of centralized logging and monitoring plan.

3.4.1 ISO/IEC 27001 Certification

ISO stands for International Organization for Standardization and has developed over 20 000 international standards for their catalogue (ISO, 2020.). IEC stands for International Electrotechnical Commission, who prepares and publishes international standards for all electrical, electronic, and related technologies. (IEC, 2020.).

The ISO/IEC 27001 is an international standard which specifies the requirements to establish, implement, maintain, and continually improve an ISMS (Information Security Management System). (ISO/IEC 27001:2013 Clause 1).

Table 10: ISO/IEC 27001 annex A.12.4 objectives (ISO/IEC 27001:2013 Annex A.12.4.)

Annex	Description
A.12.4.1 Event Logging	"Event logs recording user activities, exceptions, faults and information security events shall be produced, kept and regularly reviewed."
A.12.4.2 Protection of Log Information	"Logging facilities and log information shall be protected against tampering and unauthorized access."
A.12.4.3 Administrator & Operator Logs	"System administrator and system operator activities shall be logged, and the logs protected and regularly reviewed."
A.12.4.4 Clock Synchronization	**"**The clocks of all relevant information processing systems within an organization or security domain shall be synchronized to a single reference time source."

The ISO/IEC 27001:2013 Annex 12.4 presented in table 10 gives insight on the importance of security management. In my opinion, it should be the goal once the application and its abilities to provide logging information and monitoring capabilities grow. Taking a note, event logging should be advanced to a level where it provides event logs for security management (e.g. user activity logs). Protective methods for sensitive logging data should be implemented where logs and logging servers are protected against tampering and unauthorized access. Control methods should be implemented where system administrator and developer activities are logged and reviewed. Lastly, synchronized and unison reference of time in logs and metrics should be implemented across the application, its infrastructure, and systems to provide accurate information for security audits, investigation, and proof of unwanted or unexpected behaviour within the application.

3.4.2 TLS/SSL Certification

TLS (Transport Layer Security) first version was published in 1998 by IETF (Internet Engineering Task Force) and evolved from SSL (Secure Socket Layer) developed by Netscape. Latest published version is TLS 1.3, released in 2018. TLS is widely adopted security protocol designed to facilitate privacy and data security over the internet. TLS is

used primarily to encrypt communication between web applications and servers e.g. web browser loading a website and works as implementation of HTTP (Hypertext transfer protocol) protocol and encrypts the communication creating a secured version of the HTTP protocol HTTPS (Hypertext Transfer Protocol Secure) (Cloudflare Inc, 2020a;b).

Table 11. TLS Components (Cloudflare Inc, 2020a)

Component	Description
Encryption	"Hides data being transferred from third parties."
Authentication	"Ensures that the parties exchanging information are who they claim to be."
Integrity	"Verifies that the data has not been forged or tampered with."

Showcased in table 11 are the TLS three main components, the processes which TLS completes in order to fully secure connection between the client and the server. TLS works on top of the TCP (Transmission Control Protocol). To secure the connection, a sequence known as TLS handshake is initiated securing the communications between the client and the server. The TLS handshake establishes a cypher suite for each session which is a set of algorithms (cryptographic algorithm) that specifies details such as encryption keys or session keys, each unique for a session. The TLS handshake handles authentication by using TLS key pair: private key and public key. Public key can be used by anyone, but to decrypt the data sent between the client and the server, a private key is needed. Once encryption and authentication are done, it is signed with a MAC (Message Authentication Code) which is used to verify the integrity of the data, meaning that no-one has tampered with it. (Cloudflare Inc, 2020a;b.)

4 Introduction to Zero Day -template

The ZDD template is using various technologies. Some of the various technologies can create their own logs messages and metrics data. In some cases, the logging and monitoring functionality needs to be configured by following the framework set by these technologies. The topics related to configuring log or metrics data creation will be addressed in the technical implement. This section is dedicated as an introduction to the technologies used in ZDD template. Addressing only their concepts and use cases. For this section I have used one book source: *Beginning Kubernetes on the Google Cloud Platform: A Guide to Automating Application Deployment, Scaling and Management by Ernesto Garbarino (2019)*. Web sources are used in style of blog posts and the technical aspect is supported by the use of technical documentation.

4.1 How the Zero Day Delivery -template works

Zero Day -template is a tool for new software project which uses technologies like React (frontend) and NodeJS (backend) and holds a number of pre-configured technologies e.g. Bash for shell scripting, JFrog Artifactory for artefact management and storage, PostgreSQL as a relational database, Docker for application containerization, Kubernetes for container orchestration and Helm for Kubernetes deployment and templating tool.

For the scope of this thesis, I will only concentrate on the technologies which will be used in order to create the basis for logging and monitoring in the Zero Day Delivery -template, this includes the following technologies:

1. NodeJS
2. Docker
3. Kubernetes
4. HELM

4.2 NodeJS

NodeJS (or Node.js) is an open-source, server-side and cross-platform JavaScript runtime environment. Released in 2009 and developed by Ryan Dahl. It runs on V8 JavaScript engine developed by Google in 2008. NodeJS is used to build backend services or APIs (Application Programming Interface) and provides asynchronous I/O operations with the JavaScript -language which able NodeJS to bring event-driven programming to web servers fast, working outside of the web browser environment. The operation can be described as “non-blocking”. This means that when a request is passed to NodeJS, it takes that request, passes it along to a single thread and while it waits for the thread’s response, it is

ready for more responses from the web application. In “a blocking” -operation, the webserver works differently bringing only one response task to a single thread and finishes after the webserver has acquired the threads response, eliminating the possibility of multiple requests per thread. With the asynchronous I/O operation, software developers can create highly scalable applications. (OpenJS Foundation, 2020)

4.2.1 Logging in NodeJS

If a developer has familiarized itself with the JavaScript -language, he or she has also used the `console` -class and its methods e.g. `console.log()` or `console.error()` for debugging purposes. These methods can be used in NodeJS environment as well but using these methods can be regarded as a security risk and because it is a security risk, production code should not hold any console log statements which are visible in the console screen (e.g. Chrome DevTools console). The methods `console.log()` and `console.error()` might look the same in the console screen, but they work differently. NodeJS utilises the use of default streams (Dominik Kundel 06.05.2019). These streams are called stdin (Standard In), stdout (Standard Out) and stderr (Standard Error). Stdin handles the input (something that is coming “in”) process e.g. button presses and stdout handles the output (something that is going “out”) e.g. printing something to a file or the console screen. Stderr works differently, while stdin and stdout handles the input and the output, both streams produce a lot of event messages including error messages. Stderr handles the produced error messages in its own stream, which can be accessed separately (Jesse Storimer, 29.11.2011). The method `console.log()` is printing the results to stdout and `console.error()` is printing the results to stderr. Understanding how the default streams work and how to use `console` -class in NodeJS environment gives the developer the ability to pipe and redirect error and diagnostic information separately e.g. to a file using basic JavaScript operators (Dominik Kundel 06.05.2019).

Now we reach to the topic of use cases. When should we do logging and what should we log? In this thesis’s section *3.1 introduction to logging* I have already covered some of the use cases related to logging. In NodeJS however, Dominik Kundel in his blog provides us more in-depth look to logging in NodeJS.

Table 12. NodeJS logging use cases (Dominik Kundel 06.05.2019)

Category	Description
Debugging	For unexpected behaviour during development
Browser-based logging	For analytics or diagnostics
Server application logging	For logging incoming requests and any failures that might have happened.
Library debugging	Debugging library related issues and behaviour to assist the user with issues.
CLI output	Printing to CLI (Command-line interface) the progress, confirmation messages or errors.

In table 12 we can see the presented use cases by Dominik Kundel. Out of these five cases, three are regarded as NodeJS based: server application logging, library debugging and CLI output. In the server application logging, the idea is to log incoming requests e.g. how many 404's (HTTP Error code) users are hitting. What we want to know is what went wrong and why. In library debugging, the idea is the same: we want to know what went wrong and why. Libraries provide a lot information for debugging purposes, but it should not clutter the application. CLI output is more simplified version of logging, the main idea is to provide information on what is happening right now in the application e.g. confirmation message of a successful login or the progress of a build (Dominik Kundel 06.05.2019).

In NodeJS you can write your own log functions and transports to print or write in a file only the things which you care about, but this requires the standardisation of logs, log messages and log transmission. The theory of logging standardisation and log transmission is provided in the *3.1.3 log transmission and collection* and *3.2.1 standardising logs* section of this thesis. Without reinventing the wheel, I can utilise the use of npm ecosystem to use third-party frameworks and libraries (proprietary logging formats) to do the logging for the ZDD template. These frameworks and libraries already provide the standardisation of logs and log transmission capabilities. (Dominik Kundel 06.05.2019).

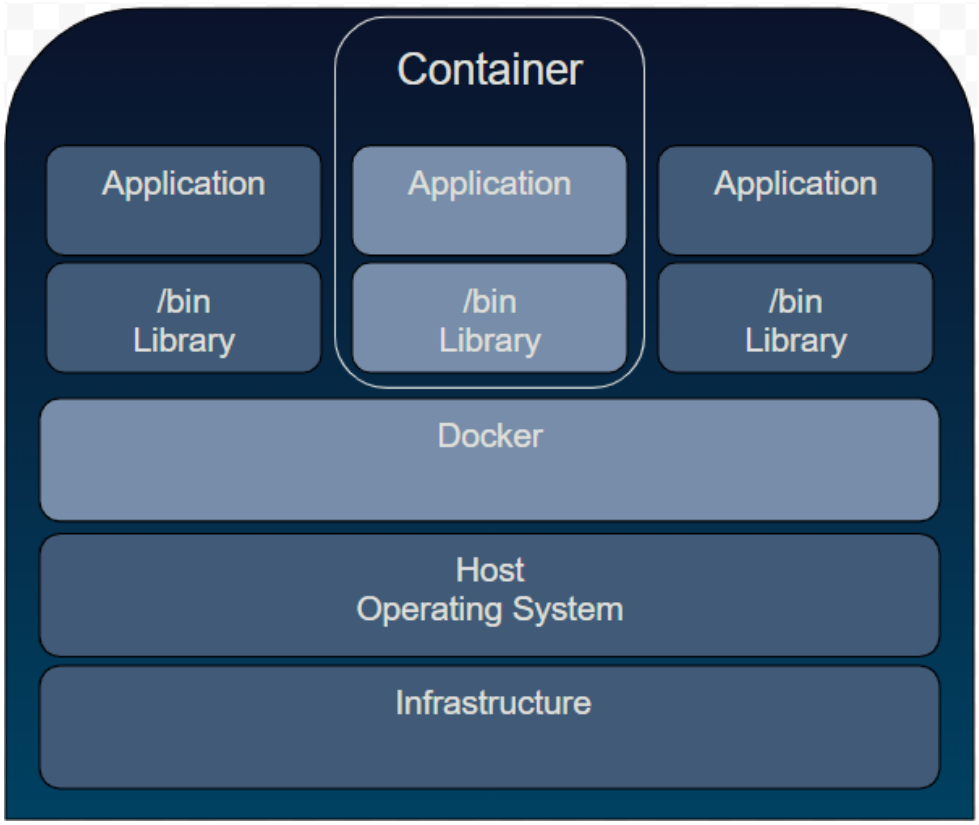
4.2.2 Winston framework

Winston is one of the NodeJS logging frameworks and in use in the ZDD template. It is designed to be simple and universal logging library. In Winston, a developer can create their own logger with configured logging levels and utilize the use of multiple transports, which means that a single logger constant variable can transmit log data to multiple instances or locations e.g. database. A transport is essentially a storage device for logs. Winston has multiple built-in core transports (console, file, http and stream -transports) which utilize NodeJS networking and I/O (Input / Output) and various custom transports created by Winston contributors (e.g. Syslog, MongoDB) and built community transports for e.g. Amazon CloudWatch, Graylog2 and PostgreSQL (Winston Github, 2020).

4.3 Docker

Docker is an open-source engine developed by Docker Inc released in 2013. Since then, Docker has become the industry standard for containerized software development. With Docker, a developer can run, build, and test the developed applications in a containerized environment. Today, the development of applications requires more than just the code. The development lifecycle of an application faces complexity when it comes to multiple languages, frameworks, architectures, and discontinuous interfaces. Docker is designed to simplify and accelerate the workflow during that lifecycle. Containerization allows developers to isolate their application to a certain environment, which can be run in development and production solving the age-old headache: “it works on my machine” (Docker, 2020a). In my opinion, it is imperative to understand the basics of Docker, containers, and containerized environments before introducing Kubernetes to the picture.

4.3.1 What is a container?



Picture 4: What is a container (Docker, 2020b.)

Picture #4 demonstrates the concept of a containerized environment. The bottom layer is the infrastructure which represents the hardware where the application is built on e.g. CSP's (Cloud Service Provider) cloud environment infrastructure or even the developers' laptop. Host operating system is the operating system where the docker runtime (Docker daemon) is running e.g. Ubuntu 18.04. So, Docker works as a runtime environment (Docker daemon) utilising the use of host operating system and its infrastructure. A single container or multiple containers can be run on top of the docker runtime environment. A container is standardized unit of software, which has packaged the code and dependencies (binaries, libraries). Docker image defines the container. Standardisation means that it can run from one computational environment to another. This makes containerized applications portable and reduces the time cycle between code being written, code being tested, deployed, and used (Docker 2020b.). Time to introduce Kubernetes.

4.4 Kubernetes

Kubernetes is an open-source platform for managing containerized workloads and services of an application. Kubernetes was developed and designed by Google in 2014 and first released in 2015. Google had already worked with containerization technology with Kubernetes predecessor Borg which Google uses to run its data centres. Upon its release

Kubernetes project was handed down to CNCF (Cloud Native Computing Foundation) with the idea of making Kubernetes open source and vendor neutral. The Kubernetes project receives development contributions from various sponsors (e.g. Google, Microsoft, and Amazon) and individual developers. (Garbarino, 2019, p. 5).

Kubernetes takes containerization a bit further than e.g. Docker. In Docker the container management is needed to be done manually or configure the automation for containers restarts per each container: when a containers state changes to unresponsive or crashes, it needs to be brought back up manually with CLI commands or its needs to have an automated configuration to bring it back up, so managing a large-scale containerized application can be become tedious (Turnbull 2018, p. 65 – 66). This is where Kubernetes comes in. Kubernetes provides a framework to run distributed systems or applications resiliently, this means it can take care of scaling, failovers, automated deployments, and automated management. This can be done with the features and components Kubernetes provides (Kubernetes, 2020.).

As a note, Kubernetes comes with a large variety of concepts related to the use of these features and components. Addressing all of the concepts, the components and the features is not in the scope of this thesis. The focus is on addressing the Kubernetes concepts related to the development of this project.

4.4.1 Kubernetes basic concept and components

The smallest component in Kubernetes is a pod. A pod can hold one or more containers which can be deployed together on the same machine or in Kubernetes context, on the same node. The key characteristics of a pod is that is shares the same IP address and port space. This differs from Docker, in Docker containers are running in an isolated network. A pod in Kubernetes is network accessible (Garbarino, 2019, p. 29.). Nodes in Kubernetes are the computing resources, a representation of a virtual machine or a physical machine, depending on the cluster. A cluster in the other hand is a collection of nodes, which are running a collection of pods (Garbarino, 2019, p. 17). This is the very basic concept of Kubernetes.

In Kubernetes pods can be created and destroyed meaning they’re a non-permanent resource within a cluster. Each pod gets its own IP address, but if a pod is destroyed, the pod loses its original IP address, and this becomes an issue for a piece of application that is needed permanently (e.g. monitoring system or a database). To fix this issue, applications can be run as Kubernetes services. When defining a service in Kubernetes, it needs to be defined as a service and give a type of service. Defining the type of service in Kubernetes, tell the Kubernetes API how the service should work in the cluster. One of these

types is known as `ClusterIP`, which is the default service type in Kubernetes. It exposes the service on a cluster-internal IP, meaning the service is only available within the cluster. `NodePort` is a type of service which exposes the service nodes IP to a static port, meaning the service is accessible from outside of the cluster e.g. via browser by typing `<IP ADDRESS: NODE PORT>`. The last service types which are out scope in this thesis are `LoadBalancer` and `ExternalName` (Kubernetes, 2020g.). For a Kubernetes service to get an actual HTTP or HTTPS -address the cluster needs an ingress controller and the service within the cluster needs an ingress resource. Ingress can be configured to give services external URL's (Uniform Resource Locator), a web address. In the ZDD project, each Kubernetes service will be configured with its own ingress resource. The ingress resource is not enough for to expose service externally, ingress controller is needed. Ingress controller is responsible for fulfilling the service's ingress resource. Ingress controller is usually used with a load balancer, which is usually provided by a CSP (Cloud Service Provider), more in thesis section *4.4.2 Minikube* (Kubernetes, 2020c.).

How Kubernetes components (e.g. pods) or resources (e.g. ingress) are configured? Kubernetes uses JSON or YAML -format for configuration and Kubernetes possesses a certain philosophy when it comes to configuring pods, ingresses, nodes etc. In Kubernetes context everything is seen as an object. These objects are the components or resources which work inside the Kubernetes cluster. The philosophy is that every object is a "record of intent", meaning that once an object (e.g. pod) is created in Kubernetes, this becomes an intention for Kubernetes system (Kubernetes API) to represent this specified state of the Kubernetes cluster. The state described what containerized applications or pods should be running on which nodes, the resources these pods have and the policies how the application should behave, and this defines the Kubernetes cluster. The `.yaml` configuration defines the Kubernetes object. At minimum four values are needed: the value `apiVersion` tells which version of Kubernetes API the object is using, the value `kind` tells which kind of an object is to be created, the `metadata` specifies unique identifiers for the object e.g. name or namespace and the value `spec` describes what state is desired for the object (Kubernetes, 2020h.).

4.4.2 Minikube

Minikube is local Kubernetes cluster which can be deployed on MacOS, Linux or Windows systems. The focus of Minikube is to provide an easy platform to learn and develop in Kubernetes. Minikube requires a virtual environment in order to work, this can either be a Docker container or a VME (Virtual Machine Environment) e.g. Hyperkit, Hyper-V, KVM, Parallels, Podman, VirtualBox or VMWare. Minikube can be deployed in a single command in CLI (Command-line interface): `minikube start`. Minikube also requires some

hardware e.g. 2 CPU's or more, 2GB of free memory, 20GB of free disk space and an internet connection. The Minikube cluster is a single node and as a default uses the mentioned resources (2CPUs, 2GB Memory and 20GB disk space). (Kubernetes, 2020b). Minikube comes with NGINX Ingress Controller which has its own reverse proxy and load balancer which can installed very simply to the Minikube cluster with the command: `minikube addons enable ingress`. Enabling the ingress controller in Minikube enables the configuration and use of ingress resource in the cluster, making the wanted Kubernetes services in the cluster available with an actual external URL. (Kubernetes, 2020i.).

4.5 HELM

In a nutshell Helm is a package manager for Kubernetes. It is a tool that helps streamlining installation and managing of applications that work on Kubernetes. For packaging Helm uses a format called "charts" which is collection of files that describes the set of Kubernetes objects which the Kubernetes uses e.g. services, pods, and volumes. Helm uses Kubernetes API making the charts reusable, customizable and shareable in repositories e.g. Artifact hub and are written in `.yaml` -format. With Helm you can:

- Install applications and dependencies
- Configure applications deployments including rollbacks
- Upgrade deployed applications
- Pull / push application packages to a repository

(Helm, 2020a).

For this project, HELM will be used to install and upgrade third party applications. This means addressing topics like installation and upgrading with HELM using the helm charts repositories as advantage. Time to introduce the basic concepts.

4.5.1 Helm concepts

HELM has three big concepts to take a hold of, these concepts are a chart, a repository, and a release. In HELM applications are packaged into a format which is called charts. A chart is collection of files that describe the set of Kubernetes resources and has a large variety of uses. It can be a simple chart that can be used to deploy a simple Kubernetes pod or a complex full web application with databases and http servers (Helm 2020h.). In HELM it is not necessary to create your own Helm charts. A chart repository e.g. JFrog Artifactory can be used to install and upgrade application packages to a Kubernetes cluster. A chart repository is a simple HTTP server which houses an `index.yaml` file or any application packages in `.yaml` -format or `.tgz` -format. The `index.yaml` is usually the main file in chart repositories containing the metadata about the package including the contents of `chart.yaml` -file. The repositories can be added with a simple command

`helm repo add <URL of the chart repository>` (Helm, 2020i.). Once a chart repository is available for use a simple `helm install` -command can be used to install a selected package from the repository to a Kubernetes cluster (Helm, 2020d.). A release is an instance of a running chart within a Kubernetes cluster. One chart can be installed many times in the Kubernetes cluster and each time the installation process is done, a new release is created. (Helm, 2020j)

The technical implementation section of this thesis will dive more deeply into the world of HELM by demonstrating the helm workflow this project uses in order to install and upgrade Kubernetes services e.g. Grafana and Prometheus.

5 Selecting the tools from CNCF Landscape

This chapter covers the CNCF (Cloud Native Computing Foundation) as an organisation, the CNCF Landscape for cloud native applications, how the use the CNCF Landscape as a tool to evaluate cloud native applications and the selection of monitoring and logging tool for the ZDD template.

As mentioned in the thesis introduction, the ZDD templates requirements for monitoring and logging solution has been provided in the ZDD project Bitbucket Git repository:

- Framework/library supports trackability, security requirements, different cloud environments and problem analytics.
- Framework/library log system can be extended to log-based monitoring and alerting – when critical issues are logged the development team should be alerted.
- Framework/library is simple, not requiring too much of learning or maintenance.
- Framework/library provides basics statistics about the logs.

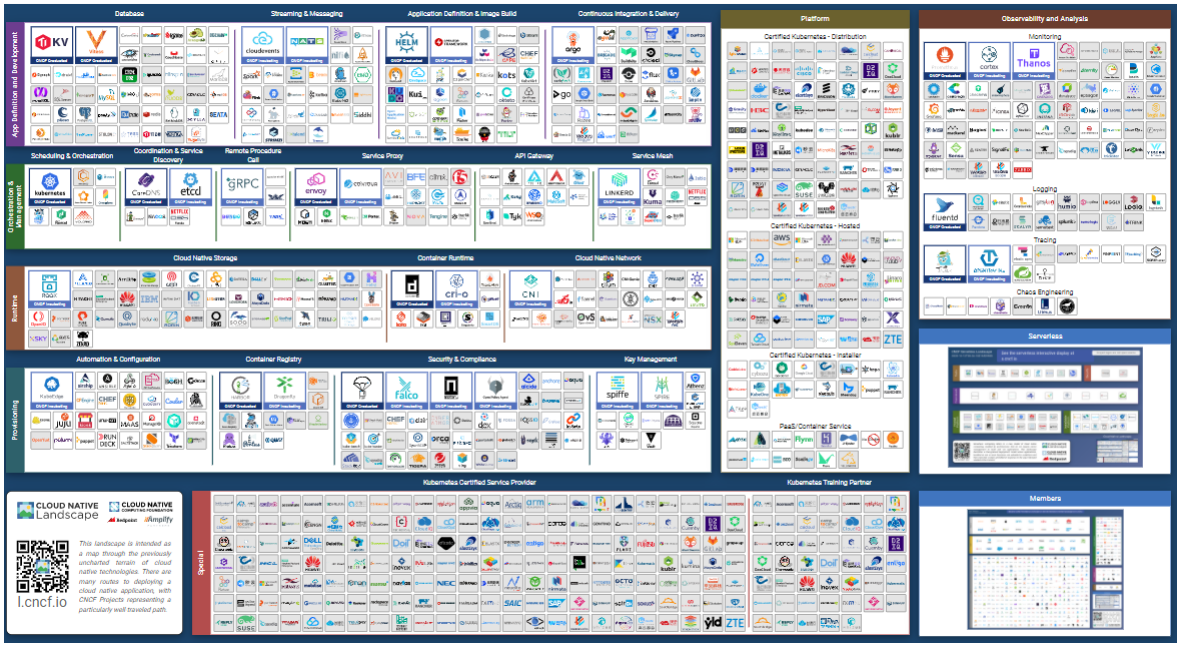
The requirements will be the corner stone in the evaluation process done in the CNCF Landscape and the mortar will be the thesis's theory section.

5.1 Cloud Native Computing Foundation (CNCF)

Run by the Linux Foundation, the Cloud Native Computing Foundation (CNCF) is a driver for organisations to empower themselves to adopt, build and run applications based on cloud native technologies. CNCFs focus is to sustain and foster a cloud native technology ecosystem by evaluating various applications designed to work in the cloud. Cloud applications can be anything from vendor applications to open sources applications. The CNCF offers a variety of trainings, events, networking opportunities and promotions to companies and individuals related to cloud technologies. (CNCF, 2020a)

5.1.1 The CNCF Landscape

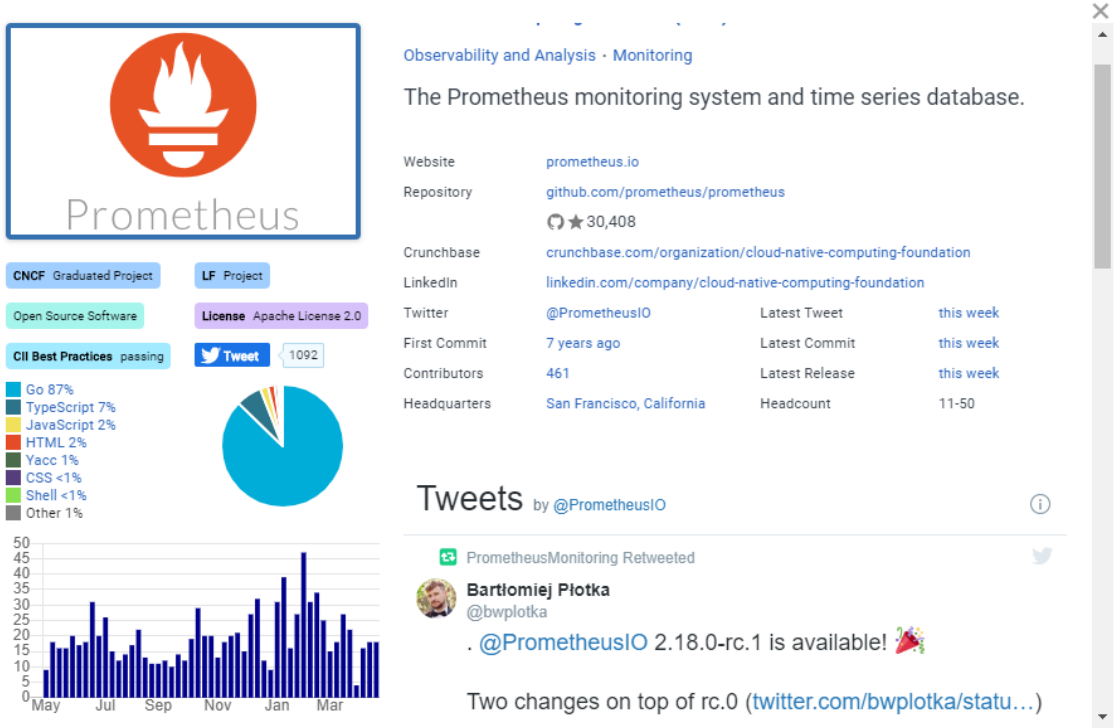
The CNCF Landscape is a map that shows the cloud native solutions including projects which are run by the CNCF itself. The landscape is designed to help developers, development teams, businesses, and practitioners to find the right cloud native solutions for their needs. This can be anything from services providers to database applications.



Picture 5: The CNCF Landscape, (CNCF, 2020c.).

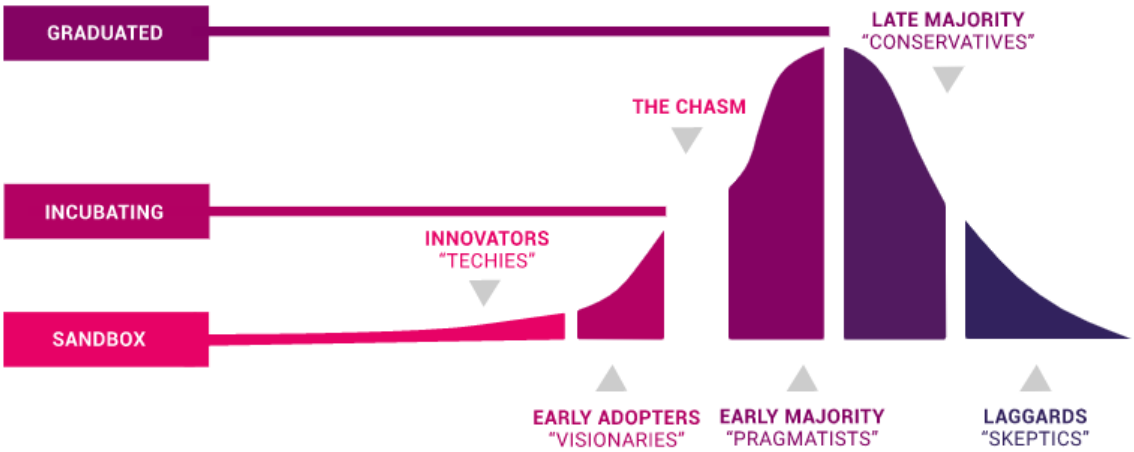
In picture 5 presents the vastness of the CNCF Landscape. The cloud native solutions are categorized to their own groups e.g. Database, Streaming & Messaging, Application Definition & Image Build to a list of CNCF members and Kubernetes Training Partners. From this landscape I'm interested in the monitoring section and the logging section of the CNCF landscape.

5.1.2 Reading the landscape



Picture 6: Prometheus information by CNCF (CNCF, 2020c.).

Upon clicking the presented icons in the CNCF Landscape I can get more information related to the tools. In this case in picture 6, I have the previously mentioned monitoring system and time series database called Prometheus presented to me with information related to the Prometheus website, Github repository, Crunchbase, Twitter handle, licencing etc. Within this plot (Picture 6) I'm highly interested in the amount of Github stars, the licencing and the badges given by CNCF for the tool. At left side of the plot, I can see the different CNCF related badges of the project including the badge of Core Infrastructure Initiative (CII) Best Practices which states the status as "passing" this badge means that the open source fools the best practices set by The Linux Foundation (Core Infrastructure Initiative, 2018) . Also, I can see that Prometheus is a CNCF graduated project. This showcases the level of maturity a CNCF project holds.



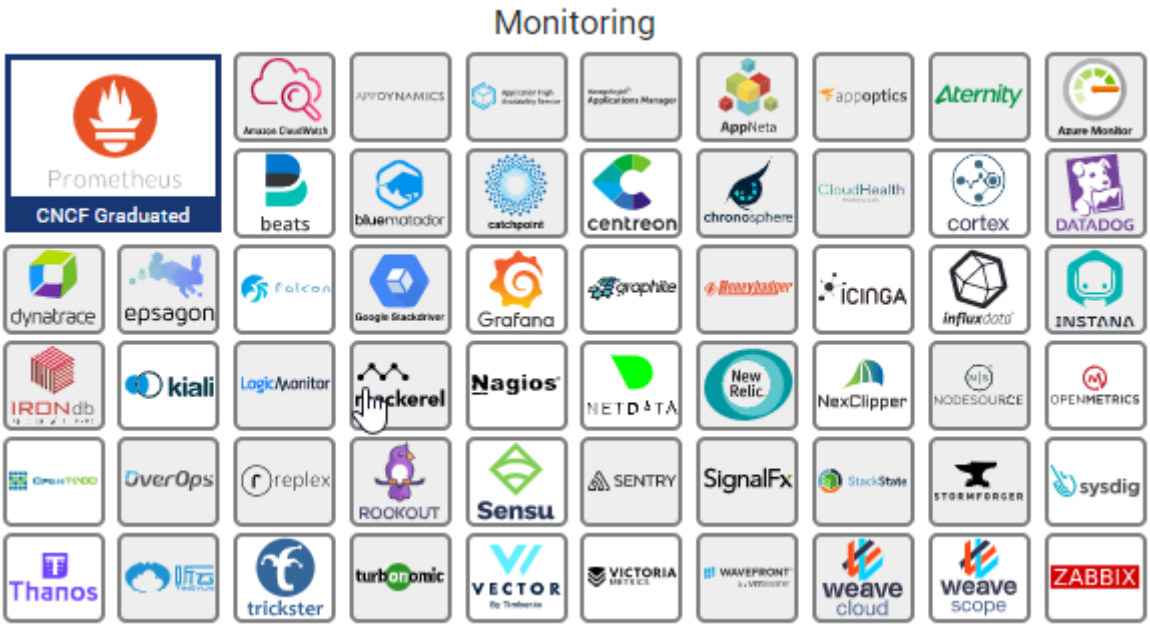
Picture 7: CNCF Project maturity levels, (CNCF, 2020b.).

Picture 7 demonstrates the maturity level of a CNCF project. With this, the CNCF tries to signal what sorts of enterprises should adopt different CNCF projects or in this context, which kind of tools provided in the CNCF Landscape as part of their toolkits. The projects maturity level increases by demonstrating sustainability to CNCF's Technical Oversight Committee (CNCF TOC). The description of the process related to increase of project maturity level by the committee is not part of this thesis or project, but CNCF TOC evaluates the projects healthy rate of change, the amount of committers from different organizations to the projects and the projects adoption of the CNCF Code of Conduct, best practices and achieving and maintaining the CII Best Practices badge. (CNCF 2020b).

The CNCF Landscape provides me with a great list of projects and assets which I can evaluate straight away by using the information provided by the landscape itself, this helps me to select the right monitoring and logging tool for the ZDD template. Why the CNCF Landscape? Eficode happens to be one of the CNCF Silver Member companies repre-

sented in the landscapes section of “Kubernetes Certified Service Providers”. The landscape is familiar to us Eficodeans and we’re happy to part of the CNCF community to build the future of software development.

5.2 CNCF Monitoring landscape



Picture 8: CNCF Landscape listed monitoring framework, (CNCF 2020c.)

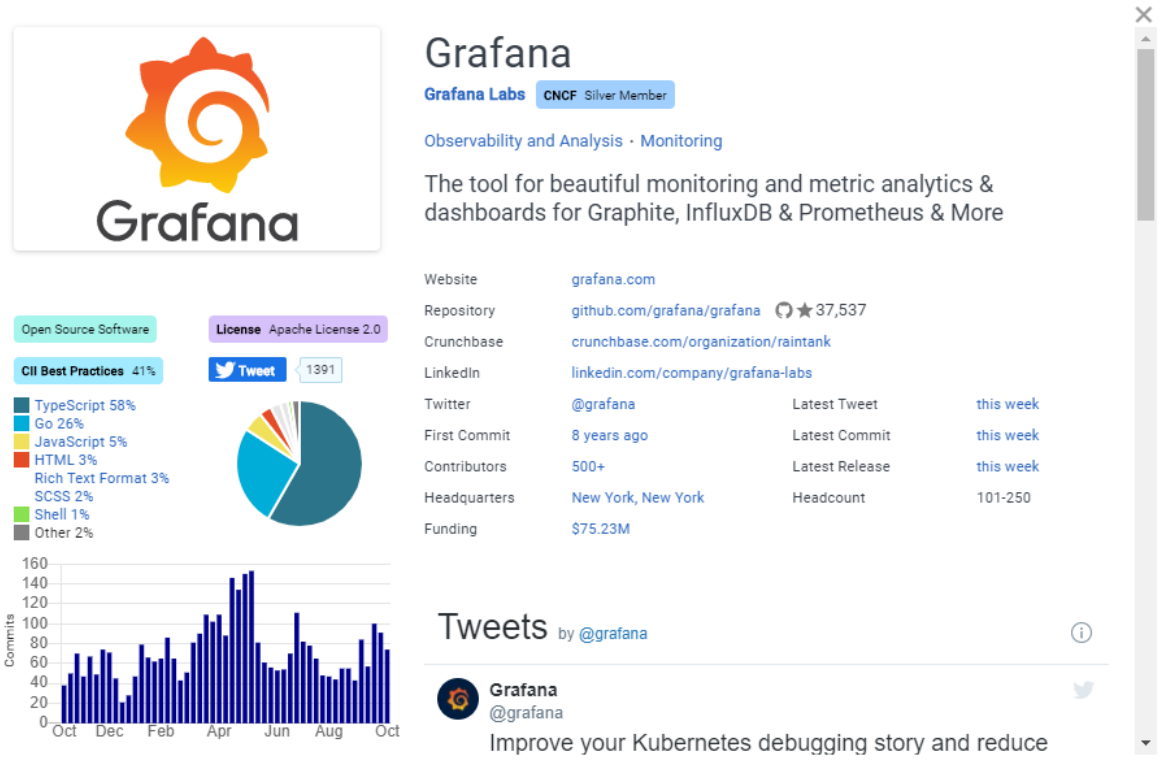
Seen in picture 8 we can see the CNCF landscape related to monitoring tools. The greyed items in the picture 8 are marked as non-open source licences while the ones with white background are open source licenced tools. The list already provides few familiar monitoring tools known to Eficode: Grafana by Grafana Labs, Zabbix by Zabbix LCC and Prometheus by The Linux Foundation. Out of these three tools Zabbix is already in use as part of the Eficode ROOT product , but Grafana and Prometheus intrigues me. In the book *The DevOps Toolkit 2.5: Monitoring, Logging and Auto-Scaling Kubernetes* by Viktor Farcic, Farcic mentions Grafana as “the undisputed ruler in the area” of monitoring dashboards and works in unison with Prometheus which Farcic describes as an extended tool to explore the unexpected “by querying metrics and going deeper and deeper until we find the cause of an issue”. In another note James Turnbull in his book *The Art of Monitoring* also uses Grafana in his demonstrated monitoring solution and architecture along with Graphite (Turnbull 2018, p. 125). So, Prometheus and Grafana can be combined, where Grafana presents an easily readable dashboard for showing basic metrics which Prometheus scrapes from the Kubernetes cluster. (Farcic 2019, p. 224 - 228). Eficode Academy also uses Grafana and Prometheus in their Advanced Kubernetes Application Development training course (Eficode Academy Github, 2020.) Also, there are customer projects in Eficode where Grafana has been used to showcase different types of metrics provided by different types of data sources.

5.2.1 Grafana

Grafana is an open-source (Apache 2.0 Licenced) cloud native application developed by Grafana Labs which allows developer to query, visualize, create alerting thresholds, and help understand metrics data no matter where they are stored (Grafana Github, 2020a.). Grafana provides a huge variety of different data sources (e.g. Prometheus) which can be connected to Grafana and most of these data sources are tools which are already available in the CNCF Landscape. This includes the already mentioned Prometheus for Kubernetes cluster metrics and Zabbix, a monitoring tool already in use in Eficode ROOT product and Nagios, Graphite, InfluxDB and OpenTSDB which James Turnbull mentions in his book *The Art of Monitoring* building his demonstration using Grafana (Grafana Plugins, 2020b.).

Grafana is considered a community driven platform. Grafana has built their own repository for custom dashboards which can be accessed and filtered by the data source. Custom dashboards can give a great insight on how some metric queries are done in different data sources (e.g. Prometheus) and can also be used as a starting template when creating your own Grafana dashboard for your project (Grafana Plugins, 2020b.).

Grafana documentation also provides great information about the best practices related to creating your own dashboards and an introduction to best practices of observability, which covers the concepts of logging, tracing, monitoring, and alerting system throughout your application or system. In my opinion I consider observability as the highest form of monitoring maturity in software development . With this I'm referring to the thesis theory section of 3.2 *Introduction to monitoring* and *3.3.2 Monitoring maturity* using Turnbull's ideas of monitoring maturity and especially recognizing the two customers which monitoring serves: business and IT. With IT as a customer in mind, observability readiness in Grafana serves ITs ability to prevent system downtimes by detecting, diagnosing, and helping resolve issues using observability concepts mentioned before: logging, tracing, monitoring, and alerting. Because of this, Grafana seems like a great tool for delivering metrics to ensure sufficient quality of service and the various data source options provided as plugins and custom dashboards in Grafana's plugins page provides a huge opportunity for monitoring scalability (Grafana Plugins, 2020b.).



Picture 9: Grafana information by CNCF (CNCF, 2020b.).

Seen in picture 9 the information provided by the CNCF Landscape related to Grafana. In my opinion, Grafana also seems to have a very good standing as a cloud native standard for turning a time-series database (e.g. Prometheus) data into understandable graphs and visualizations, this can be indicated with the badge progress of 41% related to already mentioned CII Best Practices, over 37,000 Github stars, big funding and over 500+ contributions. With this in mind, creating a set of instructions on how to implement Grafana as part of the ZDD template might also serve other projects which are not built on top of the ZDD template, but works in Kubernetes and therefore CSP (Cloud Service Provider) environments e.g. AWS (Amazon Web Services). As a big plus, Farcic mentions that Grafana can be deployed to a Kubernetes cluster with the help of HELM charts (Farcic, p. 228), this supports the idea of deploying Grafana to any Kubernetes cluster. Taking all these aspects into consideration, I think I will choose Grafana as the main monitoring tool for the ZDD template, this means everything will be built around it.

5.2.2 Prometheus

Prometheus is an open-source (Apache License 2.0) monitoring and alerting tool developed originally by SoundCloud. Now considered to be an open-source standalone project with active developer and user community. In 2016 Prometheus joined the CNCF (Cloud Native Computing Foundation) as second hosted project after Kubernetes. Prometheus comes with multiple features and components. Features include a multidimensional datal model, its own query language called PromQL (Prometheus Query Language), supports

both push and pull based metric collection methods (pull model via HTTP, push model via intermediary gateway) and metric targets (or sources) can be discovered via service discovery or by creating static configuration. Components include the main Prometheus server which does all the heavy lifting (scraping and storing time series data), client libraries which officially support Go, Java, Scala, Python and Ruby (Unofficial third-party support can be found for e.g. NodeJS, PHP, R and Rust), push gateway (for the push model), service exporters (Node exporter) and an alert manager (Prometheus, 2020a.). Many of the following listed features and components can seem very tedious to understand and describing all of them are definitely not in the scope of this thesis, but the main features and components which peaks interest are the Prometheus data model, PromQL query language, service discovery and metrics scraping. These are the Prometheus features and components related to the development of logging and monitoring solution in ZDD (Zero Day Delivery) project.

Prometheus stores all data as time series or as Turnbull describes time series in his book *The Art of Monitoring*: a recording of data points or observations over time. (Turnbull 2016, 40.). In Prometheus the time series data is identified by its metric name with the combination of key-values pairs which are called “labels” in Prometheus’s concept. E.g., the time series data can have the metric name of `api_https_requests_total` and various labels, which can be describe different application operations e.g. `method=”POST”` with this combination the query would look something like this: `api_https_requests_total{method=”POST}`. The labels make the Prometheus data model multidimensional (Prometheus, 2020b.).

PromQL (Prometheus Query Language) is the functional query language provided in Prometheus which allows users to select and collect time series data in real time. The results of the query can be either shown as a graph in Prometheus, viewed as tabular data in Prometheus’s expression browser or used by third-party systems via the HTTP API (E.g. Grafana). PromQL consists of various literals, time series selectors, subqueries, operators, and functions, but mainly PromQL has its own data types (Prometheus, 2020c.).

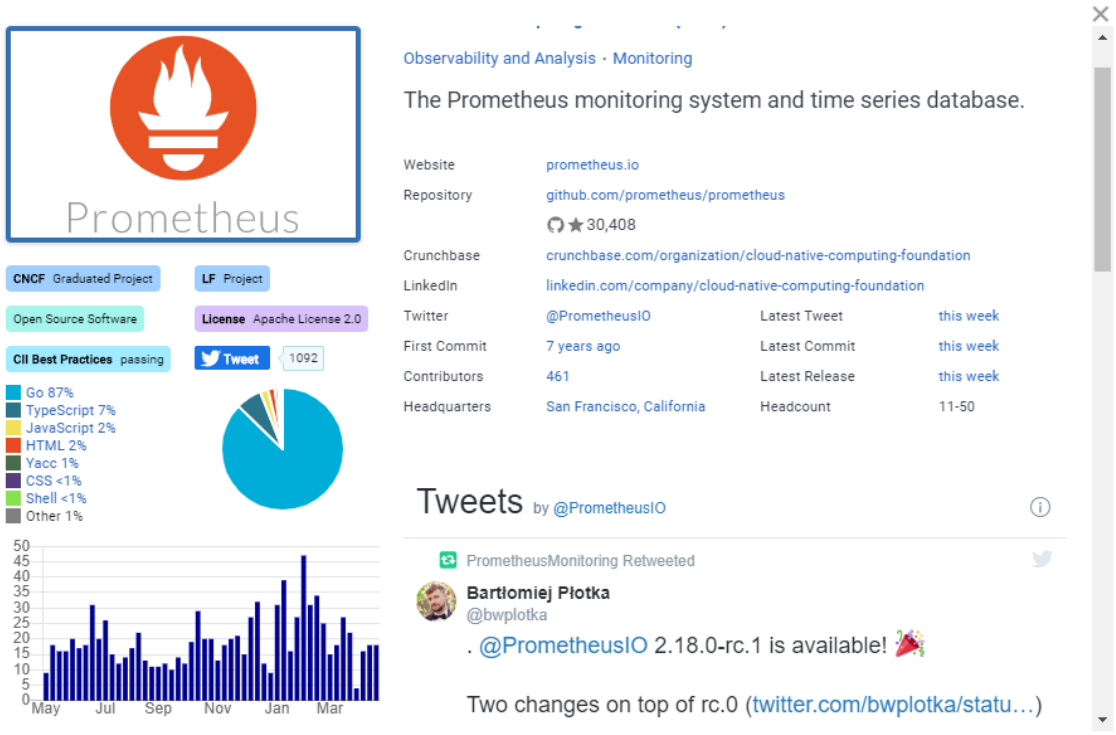
Table 13. Prometheus: PromQL data types (Prometheus, 2020c.)

Category	Description
Instant vector	“A set of time series containing a single sample for each time series, all sharing the same timestamp.”
Range vector	“A set of time series containing a range of data points over time for each time series.”
Scalar	“A simple numeric floating-point value”
String	“A simple string value; currently unused”

In table 13 is the list of PromQL data types. Depending on the case scenario only some of the data types are considered as legal, this means that if a user is returning an instant vector data type in a user created expression, only that single data type is used (Prometheus, 2020c.). In my opinion and also referring to Viktor Farcic and his book *The DevOps Toolkit 2.5.* Farcic points out that Prometheus is not designed to serve as dashboard for visualized metrics. Created graphs in Prometheus are not permanent and Prometheus should be used to do visualize ad-hoc queries (Farcic 2019, p. 227.). Getting to my point, Grafana will be used as the centralized visualisation tool for metrics, this means creating a dashboard with very common use cases e.g. CPU usage and Prometheus can be used as an extension by making queries based on very specific use cases which otherwise wouldn’t fill the category of “common use case”.

Service discovery mechanism and metrics scraping are key part of Prometheus’s features and its functionality to work as a monitoring tool in Kubernetes clusters. Prometheus’s service discovery mechanism is by default an automated method in which Prometheus server finds targets. The targets are Kubernetes services or objects within a cluster and installed exporters with Prometheus exposes the endpoints for metrics data or metadata for scraping. Prometheus pulls the metrics data to be refined further depending on the use case and creates the labels (key/value -pair) per service target (Prometheus Github, 2020b.). Prometheus has a number of official exporters and third-party exporters (Prometheus, 2020e). In ZDD projects point-of-view, three exporters raise interest: cAdvisor by Google, node exporter by Prometheus team and kube state metrics by Kubernetes team. The

exporter cAdvisor (Container advisor) provides metrics data related to container use e.g. resource usage and performance (Google Github, 2020.). The Prometheus node exporter exposes hardware and OS (Operating System) metrics (Prometheus Github, 2020c.). The exporter kube state metrics listens to Kubernetes API (Application Programming Interface) and generates metrics from Kubernetes concepts e.g. nodes, deployments, and pods (Kubernetes Github, 2020.)



Picture 6: Prometheus information by CNCF (CNCF 2020b.)

Picture 6 shows the Prometheus information in the CNCF landscape. Prometheus has a great standing in the CNCF landscape as its hold the badges of CNCF graduated project, LF project and CII Best Practices with passing remarks. Prometheus has a large and active community and contributor base. Prometheus is a strong second choice for completing the monitoring stack for ZDD template.

5.3 CNCF Logging landscape

Picture 10: CNCF Landscape listed logging frameworks (CNCF 2020b.)

The CNCF logging landscape (Picture 10) is definitely smaller compared to the CNCF monitoring landscape (Picture 8). As mentioned previously, the greyed items in the picture 10 are marked as non-open source licences while the ones with white background are open source licenced tools. The white background icons are as follows Fluentd, Elastic, Grafana Loki, Graylog and Logstash. At this point there is also another requirement: the logging tool needs to be compatible with Grafana, the selected monitoring tool on which the combined logging and monitoring stack will be built on. This makes Grafana Loki a very strong candidate. Grafana ships with built-in support for Grafana Loki on the get go (Grafana, 2020e.) and on a positive note, Grafana Loki has its own query language LogQL, which is inspired by the Prometheus's PromQL language. (Grafana, 2020f.). Grafana Loki also provides official support for Logstash and Fluentd (Grafana, 2020g.). Based already on these facts, Grafana Loki will be selected as the base for ZDD logging solution.

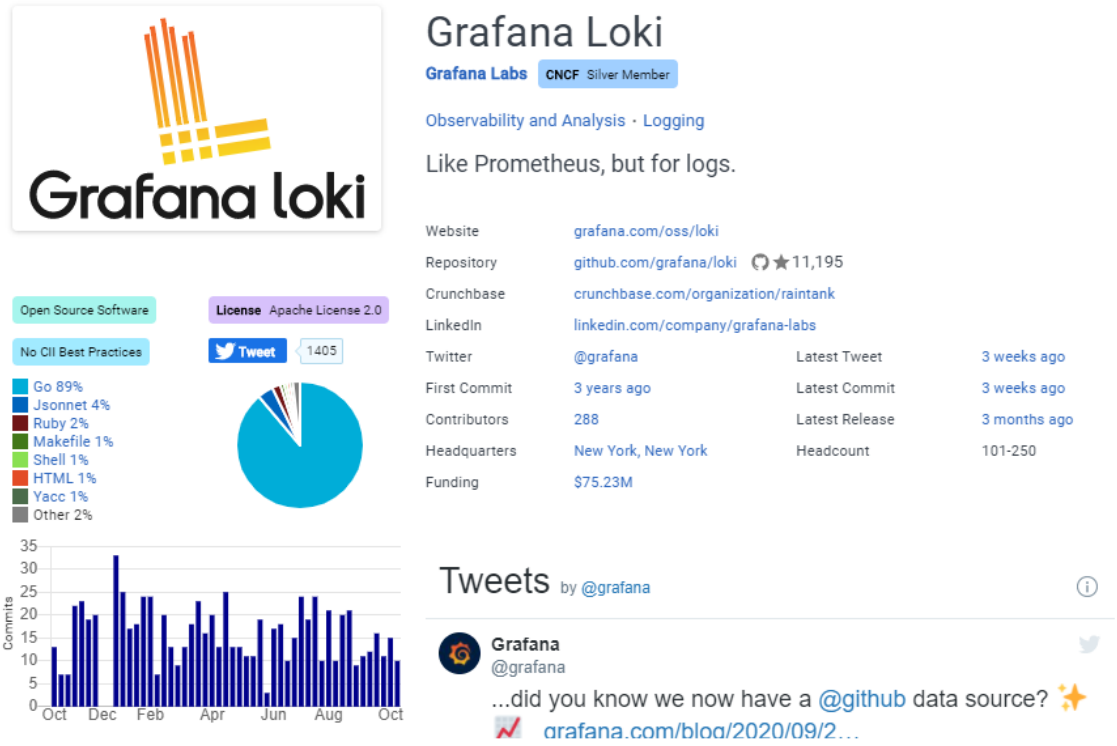
5.3.1 Grafana Loki

Grafana Loki is a log aggregation system and works as a collection of different components (or clients) which can be composed into fully working logging stack. Loki has a sophisticated indexing system and labelling system for logs which differs from other logging systems (Grafana, 2020h.). This is feature is heavily inspired by how Prometheus handles metrics data by labelling it. Loki logging stack consists of 3 components: the client, loki (as main service) and Grafana which can be used for querying and displaying logs (Grafana Github, 2020b.).

Loki comes with distributor service which handles logs written by clients (e.g. Fluentd) meaning the clients are responsible for sending logs to Loki. This makes the Loki system a push-based system instead of a pull based like Prometheus in general (Grafana Github, 2020b.). The official client in Loki includes Promtail, Docker Driver, Fluentd, Fluent bit, Logstash and Lambda Promtail. Out of these clients Promtail peaks interest. Promtail is recommended client to use when running Kubernetes, it can be configured to automatically scrape logs from Kubernetes pods which are running on the same node as Promtail. Promtail can work with Prometheus by using the same labels, which in theory means a better initial setting for debugging in Kubernetes environment. (Grafana, 2020i).

LogQL is the query language for querying logs when Loki is connected to Grafana as a data source. Inspired by Prometheus's PromQL query language, LogQL uses labels and operators for filtering and can be used in two ways. LogQL queries can return contents of log lines, but also extend log queries with metric queries (Grafana, 2020f.). Loki also

comes with LogCLI. LogCLI enables users to run LogQL queries with their CLI (Command-line interface). This means the user does not need to open Grafana to produce queries on logs (Grafana, 2020j.).



Picture 11: Grafana Loki on CNCF landscape (CNCF, 2020b.).

Comparing the CNCF landscape information about Grafana Loki (Picture 11) to Grafana (Picture 9) and Prometheus (Picture 6). Grafana Loki is fairly new, this is indicated by the first commit which was done 3 years ago but comparing the time frame and Github stars (11 195) I can determine Grafana Loki has achieved a lot in a short time span. It does not possess the familiar badges of CII Best Practices, CNCF project or LF project. Taking into the consideration that Grafana Loki is designed to work with Grafana and inspired by Prometheus, it makes the perfect selection for the base of logging solution for the ZDD template.

6 Technical Implementation

This section covers the technical implementation of the logging and monitoring solution in the form of POC (Proof of Concept). As a note, Eficode has internal services which are used in the development of ZDD centralized logging and monitor solution. The full description of these services are not part of the scope of the thesis and some are also regarded as part of the NDA (Non-disclosure agreement). This means some of the URL's, service descriptions, service architecture solution description and service functionality descriptions are deprecated or changed to fit to the NDA agreement.

6.1 The development workflow

This section describes the process of creating a workflow related to the development of the project. Eficode utilises and encourages the use of Atlassian products: JIRA, a project management tool for creating a small development tasks as tickets. Bitbucket, a Git based code hosting and collaboration tool and Confluence a collaboration and organization tool. (Atlassian 2020a;b;c.). The utilization of Jira, Bitbucket and Confluence are part of the ZDD project's overall development workflow. As a note, the full description of functionalities and properties of these Atlassian products are not in the scope of this thesis. The development workflow will focus more on the technical aspect, using the technologies of Kubernetes and HELM.

Referring back to thesis sections *4.4 Kubernetes* and *4.5 HELM.* Kubernetes and HELM provide an excellent architecture for creating a basic principle on how the instances of Grafana, Prometheus and Grafana Loki can be installed and maintained easily and efficiently. The HELM chart repositories provides different helm charts not just for Grafana, Prometheus and Grafana Loki but many other applications available e.g. in the CNCF landscape. With the use of Helm CLI commands `helm repo add`, `helm install`, and `helm upgrade` alongside with the combination of `kubectl apply` and `kubectl get` Kubernetes commands, a clear development workflow can be created.

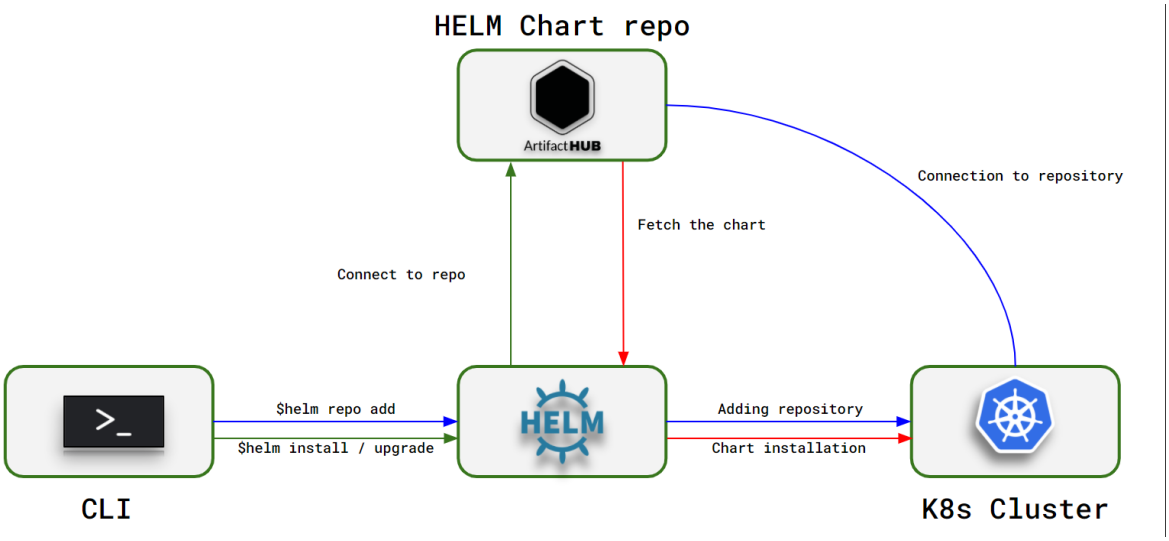
6.1.1 Utilising Atlassian products: Jira, Bitbucket and Confluence

The ZDD (Zero Day Delivery) project has its own project setup in JIRA. The project has a basic Kanban board ready for use and holds four columns representing the stage of activity of the ticket or a task. The stages of activity are: to do, in progress, QA (Quality Assurance) and done. The ZDD Jira Kanban board is used to create and track ticket progressions through-out the development process. A single task or a ticket has a title and a description. A single task can have subtasks which are smaller tasks in regards of the time it

takes to complete the set ticket and what comes to the project of creating tickets for monitoring and logging solution project for ZDD, there will be two main tickets ZDD-20: improve logging and ZDD-21: improve monitoring where subtasks are created related to the project development process. This is how ZDD Jira Kanban board is used as the main tool to follow the progression of the project. (Atlassian, 2020d.) The ZDD project has its own Git repository in Bitbucket. This is the location where the source code and the basic installation documentation (readme.md) for the ZDD project are stored, updated, and worked on. The ZDD project has its own page in Confluence dedicated for more thorough documentation purposes than in the Bitbucket Git repository. Following the same theme, while Bitbucket will hold the necessary documentation for installing Grafana, Prometheus and Grafana Loki to Kubernetes clusters. Confluence will hold more thorough documentation on the research, discoveries of the best practices, and the development lifecycle of this project.

6.1.2 The workflow for HELM

One part of the development workflow is the use of HELM chart repositories. HELM chart repositories (e.g. Artifact Hub by HELM) brings the opportunity to build a workflow when trying out different applications in a Kubernetes cluster and only requires the knowledge of a few HELM commands.



Picture 12: Helm workflow demonstration for ZDD project

The workflow is presented in picture 12. The workflow can be divided into single steps. The blue line picture 12 represents the action of adding a helm repository (`helm repo add`) for Kubernetes cluster to use (Helm, 2020c.). After adding the repository, HELM can be used to perform `helm install` or `helm upgrade` operations to install or upgrade packages retrieved from the chart repository to a cluster. This is represented as the green and red lines in picture 12. (Helm, 2020d.) For more advanced workflow, the concept of

`values.yaml` file can be used to override application packages default chart values in the repositories bringing only few additional flags to the `helm install` and `helm upgrade` command strings. (Helm, 2020e.). The advanced approach will be demonstrated further on in this thesis with the installation of Grafana, Grafana Loki and Prometheus to the project's development cluster.

The following commands can be used for cluster clean up:
`helm repo list` – To get the list of HELM chart repositories (Helm, 2020f.).
`helm repo remove [CHART NAME] --namespace staging` – To delete a HELM chart in staging namespace (Helm, 2020g.).

6.1.3 The workflow for Kubernetes

While HELM does the install and upgrade processes of logging and monitoring applications to the Kubernetes cluster as part of the workflow. The cluster also needs a workflow to manage the Kubernetes resources or objects e.g. Ingress and RBAC (Role-Based Access Control). As mentioned before Kubernetes is built for this purpose, managing a containerized environment. Kubernetes has two key commands which will be used in this project: `kubectl apply` and `kubectl get`.

The command `kubectl apply` is part of the application management in Kubernetes. The apply command is used to apply Kubernetes resource configuration (e.g. Ingress) to an existing Kubernetes object (e.g. service) within the cluster. The resource configuration is a JSON or YAML format file which holds the configurated values for the object to be used in a cluster. The command `kubectl get` displays one or more Kubernetes object information as a table depending on the flags given at the end of the commands string. This can be used to check the state of pods, services, ingresses, or other Kubernetes objects within the cluster e.g. `kubectl get svc -n staging` to get the state of known services in the Kubernetes cluster with the namespace (`-n`) `staging` . (Kubernetes, 2020f.).

The following commands can be used for cluster clean up:
`kubectl get svc -n staging` – To get list of services in the staging namespace (`-n`) (Kubernetes, 2020f.).
`kubectl get ingress -n staging` – To get list of ingress resource in the staging namespace (Kubernetes, 2020f.).
`kubectl get deploy -n staging` – To get list of deployments in the staging namespace (Kubernetes, 2020f.).
`kubectl get pod -n staging` – To get list of pods in the staging namespace (Kubernetes, 2020f.).

`kubectl delete [svc / pod / deploy / ingress ] [NAME] -n staging` – To delete either services, pods, deployments, or ingresses from staging namespace (Kubernetes, 2020f.).

6.1.4 The workflow for Minikube

Minikube has its own set of commands which can be used to create a workflow. The very basic workflow in Minikube is starting the Minikube cluster with `minikube start`, stopping the cluster with `minikube stop` and deleting the cluster including all the Kubernetes objects (pods, service, resources) and data with the command `minikube delete –all` (Minikube, 2020d.).

In this project another element needs to be added to the Minikube workflow. Referring back to thesis section *4.4 Kubernetes*, since Minikube works as a local Kubernetes cluster and Kubernetes services require ingress resource to be implemented for establishing external network traffic routing to these services, `/etc/hosts` -file needs to be configured (Minikube, 2020c.).

```
##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting.  Do not change this entry.
##
127.0.0.1       localhost
255.255.255.255 broadcasthost
::1             localhost
192.168.99.100  grafana.staging.svc.cluster.local staging.minikube.local
# Added by Docker Desktop
# To allow the same kube context to work on the host and the container:
127.0.0.1 kubernetes.docker.internal
# End of section
```

Picture 13: /etc/hosts -file structure

Shown in picture 13 is the `/etc/hosts` -file which is already configured to its full extent related to this project. In a nutshell, the `hosts` -file is a local database which associates name of hosts with their IP address, simply put `IP = this.is.my.site.net` (Oracle, 2014). In this project, the `hosts` -file is used to map `minikube ip` with a Kubernetes service ingress `host` name. This project will have five different Kubernetes services with an ingress resource: the ZDD components frontend and backend which will be mapped to `staging.minikube.local`, the monitoring service Grafana which will be mapped to

`grafana.staging.svc.cluster.local`, the monitoring service Prometheus which will be mapped to `prometheus-server.staging.svc.cluster.local` and log aggregation tool Grafana Loki which will be mapped to `loki.minikube.local` (Kubernetes, 2020i.). Each of the services will have different port numbers defined by the `.yaml` configuration files. If the host names are changed within the .yaml configuration files, the `/etc/hosts` needs to be changed also.

6.2 Setting up the ZDD template to Minikube environment

Before heading out to CSP (Cloud Service Provider) platforms e.g. AWS, GCP or Azure I find it reasonable to configure and test the monitoring stack (Prometheus, Grafana) in a safe, closed and zero cost environment. I am choosing to do the implementation first in the Kubernetes Minikube environment, a localized environment. The instructions for the ZDD template setup have been documented in the Eficode Bitbucket ZDD repository step-by-step which will be demonstrated.

6.2.1 Installing tools and creating the Minikube cluster

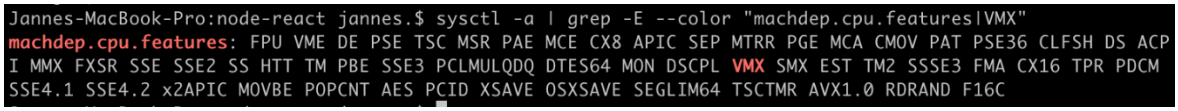
The ZDD monitoring and logging solution development is done with the following tools and system configuration:

- Apple MacBook Pro 15 (2016) with macOS Catalina (v10.15.5).
- HELM 3 (v3.0.2).
- Virtualbox (v6.1.2).
- Kubectl (Client version: 1.17.1 / Server version: 1.17.0).
- Minikube (v1.6.2).
- Docker Desktop for MacOS (v2.3.0.4) with Docker Engine (v19.03.5).
- iTerm2 CLI Tool for MacOS (v3.3.9).
- Virtual Studio Code (v1.48.1) code editor for MacOS.
- Homebrew (v2.2.4) package manager for MacOS.
- NPM (v6.12.0) Node package manager.
- Eficode RTM (ROOT Team Management), user and access management tool.

All of the following tools have been pre-installed, configured and have been used in other projects within Eficode. This section will focus in the installation of the necessary tools and local system configuration needed in the development of the ZDD monitoring and logging solution which are mentioned in *section 4 Introduction to Zero Day Delivery -template* of this thesis. As a reminder, these tools are Docker (Docker Desktop for MacOS), Kubernetes (Kubectl), Minikube and HELM 3. This section also covers the installation of Virtualbox.

Let’s get started. First step, since the working is done in a local virtualized environment provided by Minikube, a confirmation virtualized support is needed. This can be done by using the following command (Kubernetes, 2020b.):

```
sysctl -a  | grep -E --color ’machdep.cpu.features|VMX’
```



Picture 14: Confirmation for enabled virtualization in MacOS system

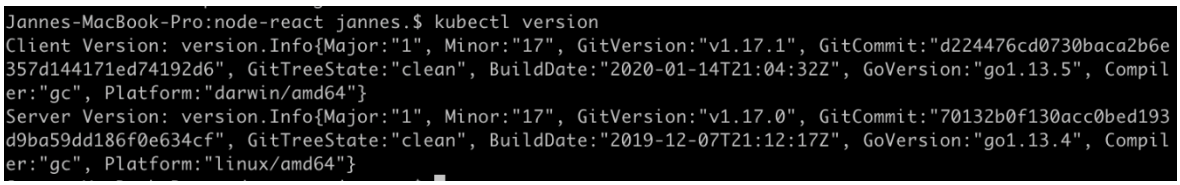
To explicate what is happening behind picture 14: the `sysctl` is a command-line utility which retrieves kernel states and `grep` is a command-line utility for searching plain-text data. In this context the `sysctl` command is used to find all available strings (`-a` -flag) in the kernel to match plain-text `‘machdep.cpu.features’` with the value of VMX (OpenBSD, 2018.). The `--color` command paints the value VMX if its present in the Kernel meaning that virtualization is enabled in my local machine. In this case, the virtualization is enabled (Kubernetes, 2020b.).

Second step is the installation of Kubernetes command-line tool called kubectl which allows to run commands in the Kubernetes clusters including the localized Minikube cluster. This can be done by using already pre-installed package manager for MacOS called Homebrew with the following command:

```
brew install kubectl
```

To make sure command-line tool kubectl is installed to the system, following command needs to be inputted to the CLI:

```
kubectl version
```



Picture 15: Confirmation of kubectl installation

In the picture 15 the data is shown in JSON format showing the `GitVersion:` attribute, this points to the current installed version for client and server of kubectl. (Kubernetes. 2020c)

Third step is the installation of Hypervisor (or virtual machine monitor). In this case, a software called Virtualbox by Oracle should be installed, since it is one of the recommended

software to use with Minikube. Installation can be done through Virtualbox website by downloading the platform package (Kubernetes 2020b.)

Fourth step, since ZDD -template is using Docker to build the ZDD template into a containerized application the installation of Docker Desktop for MacOS is required. Docker Desktop for MacOS installation can be retrieved from the Docker website. The installation also includes Kubernetes which is essential for Minikube. (Docker 2020c.)

Fifth step is the installation of Minikube for MacOS. This requires using the Homebrew package manager for MacOS to do the installation with the following command:
`brew install minikube`

To verify that the installation is done correctly, a test start of the Minikube environment is needed using the Virtualbox Hypervisor as the main virtual machine monitor (VMM) driver (Kubernetes, 2020b.):
`minikube start --driver=virtualbox`

```
Jannes-MacBook-Pro:logmontest jannes.$ minikube start --driver=virtualbox
Error: unknown flag: --driver
```

Picture 16: Minikube start unknown flag error

Interesting, the Minikube documentation states that the `--driver` flag should be used to start the Minikube environment with Virtualbox as the main VMM, but as seen in the picture 16 an error of `Error: unknown flag: --driver` has occurred. The command string `minikube start` requires the `--help` flag to check the Minikube API for the right flag to start Minikube environment:

```
  --vm-driver='': Driver is one of: [virtualbox parallels vmwarefusion hyperkit vmware] (defaults to auto-detect)
```

Picture 17: minikube start --help results

So as picture 17 shows, the correct flag for starting Minikube with Virtual box is `--vm-driver`, let's try it out by inputting `minikube start –vm-driver=virtualbox`.

```
Jannes-MacBook-Pro:logmontest jannes.$ minikube start --vm-driver=virtualbox
🎉  minikube 1.12.3 is available! Download it: https://github.com/kubernetes/minikube/releases/tag/v1.12.3
💡  To disable this notice, run: 'minikube config set WantUpdateNotification false'

😄  minikube v1.6.2 on Darwin 10.15.5
✨  Selecting 'virtualbox' driver from user configuration (alternates: [hyperkit])
💡  Tip: Use 'minikube start -p <name>' to create a new cluster, or 'minikube delete' to delete this one.
🔄  Starting existing virtualbox VM for "minikube" ...
⌛  Waiting for the host to be provisioned ...
🌏  Preparing Kubernetes v1.17.0 on Docker '19.03.5' ...
🚀  Launching Kubernetes ...
🏄  Done! kubectl is now configured to use "minikube"
```

Picture 18: Minikube start process

Perfect, as demonstrated in picture 18 the command `minikube start –vm-driver=virtualbox` worked. There seems to be an error in the official Minikube documentation. The console print in picture 18 also shows what is happening in the background: Minikube gives the confirmation that Virtualbox is used as the main VMM driver. Docker engine is initialized and running, and Docker prepares Kubernetes for launch. This enables the use of kubectl command-line tool in the Minikube cluster. Running `minikube stop` command stops the environment for now. (Kubernetes, 2020d.).

Final step, HELM installation. As stated previously in *section 4.5 HELM* , HELM is used to install and upgrade Kubernetes application packages to a cluster, this also applies how ZDD template is installed to a Kubernetes cluster with the use of JFrog Artifactory, more on this later. For now, HELM will be used to install the monitoring application packages Grafana and Prometheus and the logging application package Grafana Loki to the cluster. HELM can be installed by using Homebrew package manager for MacOS:
`brew install helm`

After the installation progress is done. The confirmation of a successful HELM installation can checked by inputting the following command:
`helm version`

Picture 19: Helm installation confirmation

Perfect, HELM is installed successfully as the picture 19 states and the project has working local Minikube cluster at its disposal. Now HELM can be used to install and upgrade application packages to the cluster. Next step is to start the actual development of this project by creating a new ZDD project.

6.2.2 Creating a new ZDD project

The guidelines to create a new ZDD project can be found in the Eficode's Bitbucket Git repository as a `readme.md` file and from Zero Day Delivery Confluence page. Most of the work is based on following the steps presented in these two sources.

Code snippet 1. cloning the ZDD repository

```
git clone --depth 1 https://<eficode bitbucket>/node-react-project.git
cd node-react-project
npm i
cd ..
```

Code snippet 1 demonstrates the first step of ZDD template setup: the command `git clone` takes the ZDD template related files from the Bitbucket git repository to local machine and creates its own directory named `node-react-project`. With the command `cd node-react-project` the CLI (Command line interface) navigates to the directory `/node-react-project` to work under that directory. Using the command `npm i` under this directory calls the npm ecosystem to fetch and install dependencies related to the ZDD project. A global CLI command `create-efi-project` is also created during the installation process. After the npm ecosystem has completed the dependency installation. CLI is used to navigate back one directory level with the command `cd ..`. The next step is to a start a new project with the ZDD template.

Now that the ZDD template files have been pulled from the Eficode Bitbucket's Git repository and the npm ecosystem has fetched and installed the needed dependencies it is time to initiate the start of a new ZDD project by using the new global CLI command `create-efi-project`. This command prompts a script to run in the local machine asking for basic project related configuration questions.

Code snippet 2. Starting setup of ZDD template

```
Jannes-MacBook-Pro:code jannes.$: create-efi-project
Starting setup of a new Node + React project.
Please enter project name. Must be lowercase and one word, and may contain
hyphens and underscores. Example: my-project
Project Name: logmontest
Enter the name of the author. Example: John Smith <john.smith@email.com>
Project author: Janne Saikkonen <janne.saikkonen@email.com>
Enter the Docker repository URI. For example: artifactory.dev/project/
Docker repository: <Eficode Docker repository URI>
Enter public address of the service. Example: your.service.com or
123.123.123.123
Public address: <left blank>
Setting up project logmongtest…
Create the project directory logmontest and populated it with project files.
Created package.json, frontend/package.json, backend/package.json, build.sh,
push.sh, deploy.sh, .env and .deploy.env based on the provided information.
Development QUICKSTART: To run the project, run “docker-compose up” in the
project directory and visit http://localhost:8000/ in the browser.
Production QUICKSTART: Use the build.sh, push.sh and deploy.sh scripts to
deploy to Kubernetes. See README.md for more information how to prepare cluster
and enable continous development in Jenkinsfile.

All done! Happy hacking!
```

Code snippet 2 is the basic setup prompt for new ZDD template project. After the configuration questions are filled, the script creates a new project directory (in this case: `/logmontest`) and creates the necessary ZDD project files under the directory `/logmontest`. The `Public Address:` -field is intentionally left blank, since the development is done in a local Minikube cluster and the new `logmontest` ZDD project will be deployed to that cluster, meaning the public address is not available for this project. The initial setup for a new ZDD project is done. Time to prepare the Minikube cluster for new ZDD project.

6.2.3 Configuring the Minikube cluster

First step is to configure the Minikube cluster by continuing to follow the steps in the ZDD project documentation (Bitbucket and Confluence documentation). The Minikube cluster is brought back up again with the `minikube start` -command. The second step is to enable the NGINX ingress controller in the Minikube cluster with the `minikube addons enable ingress` -command.

```
Jannes-MacBook-Pro:logmontest jannes.$ minikube addons enable ingress
✅  ingress was successfully enabled
```

Picture 20: Minikube addon ingress enabled

Minikube posts a confirmation message (picture 20) that ingress has been enabled in the Minikube cluster. As mentioned in the thesis section *4.4 Kubernetes,* Ingress exposes HTTP and HTTPS routes for external access to the Kubernetes services in the cluster (Kubernetes, 2020a.). More on this later. For more thorough confirmation, the `minikube addons list` -command can be used to get a list of enabled/disabled addons within the cluster.

```
Jannes-MacBook-Pro:logmontest jannes.$ minikube addons list
- addon-manager: enabled
- dashboard: enabled
- default-storageclass: enabled
- efk: disabled
- freshpod: disabled
- gvisor: disabled
- helm-tiller: disabled
- ingress: enabled
- ingress-dns: disabled
- logviewer: disabled
- metrics-server: disabled
- nvidia-driver-installer: disabled
- nvidia-gpu-device-plugin: disabled
- registry: disabled
- registry-creds: disabled
- storage-provisioner: enabled
- storage-provisioner-gluster: disabled
```

Picture 21: Minikube addons list

As picture 21 shows the second confirmation that ingress has been enabled in the Minikube cluster among other Minikube related Kubernetes services. Next step is to use the kubectl command-line tool to create a namespace for the Minikube cluster. This can be done with the `kubectl create ns staging` -command.

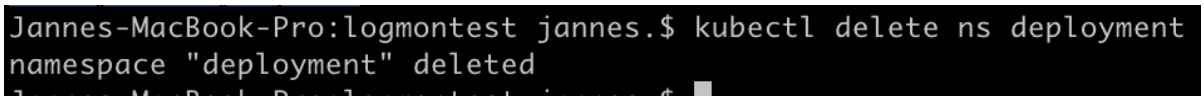
```
Jannes-MacBook-Pro:logmontest jannes.$ kubectl create ns staging
Error from server (AlreadyExists): namespaces "staging" already exists
```

Picture 22: Kubernetes namespace error for existing namespace

Picture 22 tells that there is already a namespace called `staging` in the Minikube cluster. This is due to the fact that this cluster namespace has already seen some development work related to the project. For demonstration purposes, new cluster namespace can be created by using the `kubectl create ns deployment` -command:

Picture 23: Creating a Kubernetes namespace called deployment

This is the confirmation message (picture 23) kubectl posts upon a successful creation of a new namespace in the cluster. To delete this namespace, the `kubectl delete ns deployment` -command can be used to delete the newly created namespace:



Picture 24: Deleting a Kubernetes namespace

Picture 24 shows the deletion process confirmation. Now it's time to cover the topic of using the JFrog's Artifactory as part of the project.

6.2.4 Eficode RTM and JFrog Artifactory configuration for the ZDD project

Eficode has internal service called Eficode RTM (ROOT Team Management) which is a user and access management tool. Eficode RTM is connected to JFrog Artifactory, a repository for Docker images (also known as Docker registry) and Helm charts (Helm chart repository). Eficode RTM service is used to gain user access to the Artifactory in order to pull and push application packages (Docker images or Helm charts). In this case, the ZDD project components frontend (React), backend (NodeJS) and database (PostgreSQL) are pushed to Artifactory and pulled from the Artifactory as application packages which are then installed to the Minikube cluster using HELM. After this setup is completed, the work is done strictly under the `/logmontest` ZDD project folder.

the Eficode RTM requires creating a bot user which does the automated processes related to storing application packages to Artifactory. Before the Artifactory can be used with Eficode RTM tool, the cluster needs to be authenticated with `docker login` command. This can be done with a following CLI command `docker login -u <username> -p <password> <Artifactory URL address>` this tells the Docker engine, which tells the cluster to use provided credentials given in the CLI command when authenticating the Minikube cluster (or any cluster) to use Eficode RTM and JFrog Artifactory (Artifactory address). What comes to creating a bot user in Eficode RTM, the initial bot user is already available for my project so all I need to do is run a multiline CLI command using `kubectl` command-line tool provided in the ZDD documentation.

Code snippet 3: Authenticate for Artifactory

```
kubectl create secret docker-registry img-pull-secret \
--docker-server=<Eficode Artifactory Address> \
--docker-username=<Eficode RTM Bot Username> \
--docker-password=<Eficode RTM Bot Password> \
--docker-email=<Eficode RTM Bot email> \
--namespace=staging
```

Code snippet 3 shows the multiline CLI command needed to authorise the Minikube cluster with the `namespace staging` to use Eficode RTM's connection with JFrog Artifactory in order to push and pull application packages. Next step is to deploy the new ZDD project to Minikube cluster. Next big step is to deploy the ZDD template to the Minikube cluster.

6.2.5 Deploying the ZDD template to a Minikube cluster

As a quick summary, a new ZDD project `logmontest` has been created and the Minikube cluster is configured to access Artifactory with a bot user (Eficode RTM). It is time to install `logmontest` project to the Minikube cluster. Referring back to the instructions on *production quick start* console line in code snippet 2. To install the project to the Minikube cluster, the ZDD template uses three shell script files to do the installation: `build.sh`, `push.sh` and `deploy.sh` -scripts. The full content description of these scripts is not within the scope of this thesis due to their complexity as they hold multiple Docker, HELM and Kubernetes commands, but a short description is in order to give clarification on what is happening behind the scenes.

Table 13. Description of ZDD shell scripts (Eficode ZDD Bitbucket, 2020.)

Script	Description
build.sh	Builds the ZDD templates app components (frontend, backend, and database) to docker images using `docker-compose` locally.
push.sh	Pushes the docker images (frontend, backend, and database) to Artifactory.
deploy.sh	Pulls the docker images from Artifactory, uses HELM specific `.yaml` files to turn docker images into Helm charts (application packages) and finally installs the packages to the cluster with `helm install`.

Every single script presented in table 13 has a specific functionality and are created for the sole purpose of downsizing the amount of CLI commands needed to input manually. In more simple terms, the script files are created to prevent typos. Once the scripts have done their purpose. The command `kubectl get pods -n staging` can be used to check for running Kubernetes pods in the cluster:

```
NAME                                    READY   STATUS
logmontest-backend-8567d8978d-6n5mv     1/1     Running
logmontest-database-5978d5df86-j4z2t    1/1     Running
logmontest-frontend-747b844897-7c68s    1/1     Running
```

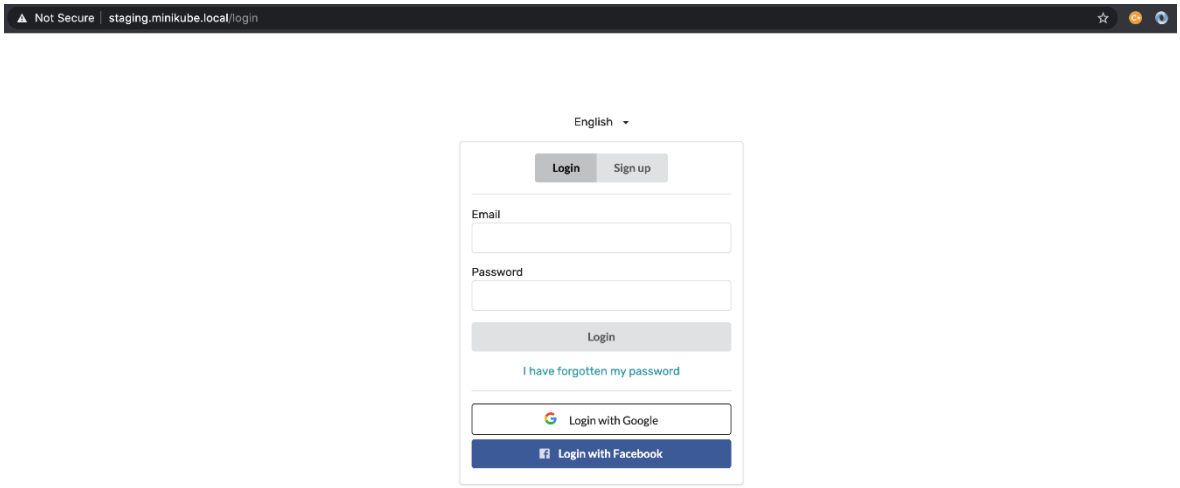
Picture 25: The ZDD project components running in Minikube cluster

The `deploy.sh` has installed the three necessary ZDD components to the Minikube cluster and additionally created ingress resources for backend and frontend components (Picture 25). These can be verified by using the command `kubectl get ingress -n staging`.

```
NAME                HOSTS
logmontest-backend    staging.minikube.local
logmontest-frontend   staging.minikube.local
```

Picture 26: The ingress resource for backend and frontend components

Picture 26 confirms that the ingress resources have been applied to the frontend and backend components, therefore the components are running as a Kubernetes service inside the Minikube cluster. Now testing the setup done in *6.1.4 The workflow for Minikube* where the `/etc/hosts` was modified. The frontend service should be available via browser by typing `staging.minikube.local`



Picture 27: ZDD Frontend service available to access via browser

Picture 27 confirms that the frontend service is now accessible and available via browser. The new ZDD project is now fully setup for the development of logging and monitoring solution.

6.3 Monitoring stack: Grafana

This section describes the steps taken to create default configuration and installation steps for Grafana service in a Kubernetes cluster. In this case, the ZDD Minikube development cluster with the namespace of `staging`. Also do note that from now on, when referring to the Minikube cluster it is referred as its namespace name `staging`. The technologies used in this section for cluster management are HELM, Kubernetes, for the configuration files are done with the `.yaml` -format.

6.3.1 Creating values-grafana.yaml configuration for Grafana

Code snippet 4. values-grafana.yaml HELM values configuration (Eficode Academy Github, 2020.)

```
service:
  type: NodePort
  port: 3000
rbac:
  namespaced: true
  pspEnabled: false
  pspUseAppArmor: false
resources:
  limits:
    cpu: 250m
    memory: 256Mi
  requests:
    cpu: 250m
    memory: 128Mi
datasources:
  datasources.yaml:
    apiVersion: 1
    datasources:
    - name: prometheus
      type: prometheus
      access: proxy
      url: http://prometheus-server:90
      isDefault: true
      editable: false
```

Grafana helm values are presented in the code snippet 4. The template for the `values-grafana.yaml` has been retrieved from Eficode Academy's hosted Advanced Kubernetes Application Development course. The `service:` mapping describes the type of service (`NodePort`) and the port which the Grafana service should use. RBAC (Role-Based Access Control) More on the RBAC in the section *6.4.3 creating rbac-prometheus.yaml configuration* in this thesis. The `resource`: mapping defines the utilization of CPU and Memory resources from the node with `limit:` defines the maximum usage and `requests:` defines the default CPU and Memory utilization from a node (Kubernetes, 2020e.). The `datasources`: mapping defines the provisioning of a data source, in this case Prometheus as a data source is set to default data source, this is one way to add

data source to Grafana using the helm values configuration. More on this later. (Grafana, 2020d.).

The HELM values configuration for Grafana installation is ready for implementation. Following the development workflow presented in the previous chapter. Adding a helm repository to the `staging` cluster is required in order to proceed:

```
helm repo add stable https://kubernetes-charts.storage.googleapis.com/
```

```
Jannes-MacBook-Pro:logmontest jannes.$ helm repo add stable https://kubernetes-charts.storage.googleapis.com/
"stable" has been added to your repositories
Jannes-MacBook-Pro:logmontest jannes.$ helm repo list
NAME    URL
stable  https://kubernetes-charts.storage.googleapis.com/
```

Picture 28: Grafana helm repo add and helm repo list -commands demonstration

The staging cluster has now access to a helm repository for Grafana helm chart (Picture 28). Time to install Grafana to the cluster:

```
helm install grafana stable/grafana -f helm/monitoring/values-grafana.yaml –namespace=staging –version 5.1.2
```

```
NAME: grafana
NAMESPACE: staging
STATUS: deployed
REVISION: 12
NOTES:
1. Get your 'admin' user password by running:

   kubectl get secret --namespace staging grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo

2. The Grafana server can be accessed via port 3000 on the following DNS name from within your cluster:

   grafana.staging.svc.cluster.local

   Get the Grafana URL to visit by running these commands in the same shell:
export NODE_PORT=$(kubectl get --namespace staging -o jsonpath="{.spec.ports[0].nodePort}" services grafana)
     export NODE_IP=$(kubectl get nodes --namespace staging -o jsonpath="{.items[0].status.addresses[0].address}")
     echo http://$NODE_IP:$NODE_PORT


3. Login with the password from step 1 and the username: admin
#################################################################################
######   WARNING: Persistence is disabled!!! You will lose your data when   #####
######            the Grafana pod is terminated.                            #####
#################################################################################
Jannes-MacBook-Pro:logmontest jannes.$
```

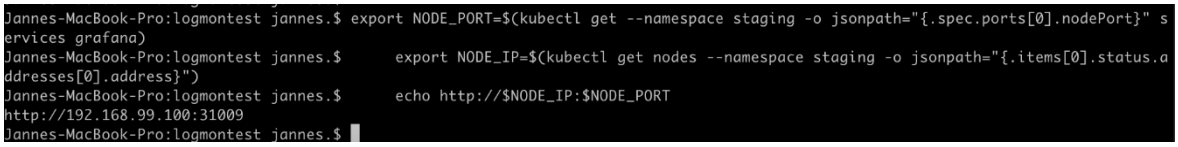
Picture 29: Grafana installed to staging cluster via HELM

Grafana is now installed to the `staging` cluster as the picture 29 shows and is available for access. Grafana comes with an UI (User Interface) which is accessible via browser (e.g. Google Chrome, Mozilla Firefox). Following the `NOTES:` given in the picture 29 gives the instructions for the Grafana UI access. At this point it is good to remind that the step #2 in `NOTES:` is inaccurate, but it does give a hint what is needed to do. Grafana UI is not yet accessible with the following address of `grafana.staging.svc.cluster.local` or the port 3000. The Grafana helm chart assumes that an ingress resource is set for

Grafana in the staging cluster, it is not, more on this later. The hint is to copy and paste full command of `export NODE_PORT` under the step #2:

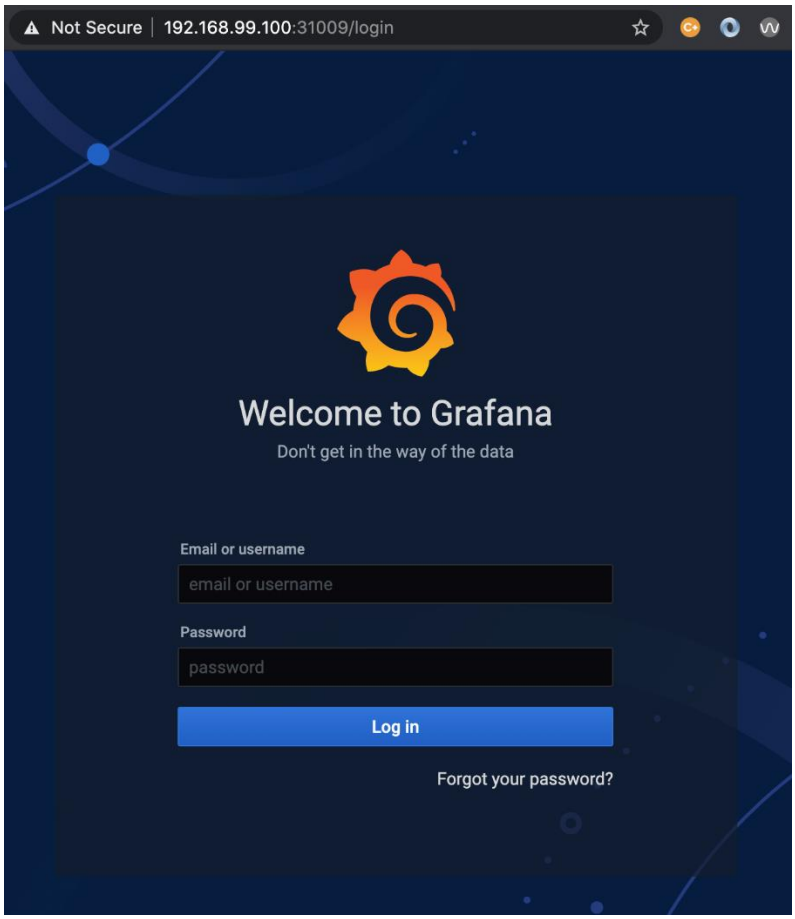
Code snippet 5. Grafana UI access CLI command

```
export NODE_PORT=$(kubectl get --namespace staging -o
jsonpath="{.spec.ports[0].nodePort}" services grafana)
export NODE_IP=$(kubectl get nodes --namespace staging -o
jsonpath="{.items[0].status.addresses[0].address}")
echo http://$NODE_IP:$NODE_PORT
```



Picture 30: Grafana UI access CLI command results

After code snippet 5 commands are run the results are shown in picture 30. These commands can also be run separately. For the IP address its `minikube ip` which posts an IP address of `192.168.99.100` and for the port number, copy and paste the command inside the brackets in `export NODE_PORT` -command: `kubectl get --namespace staging -o jsonpath="{.spec.ports[0].nodePort}" services grafana` which gives the port number of `31009`. This gives the opportunity to access Grafana UI without the login capabilities.

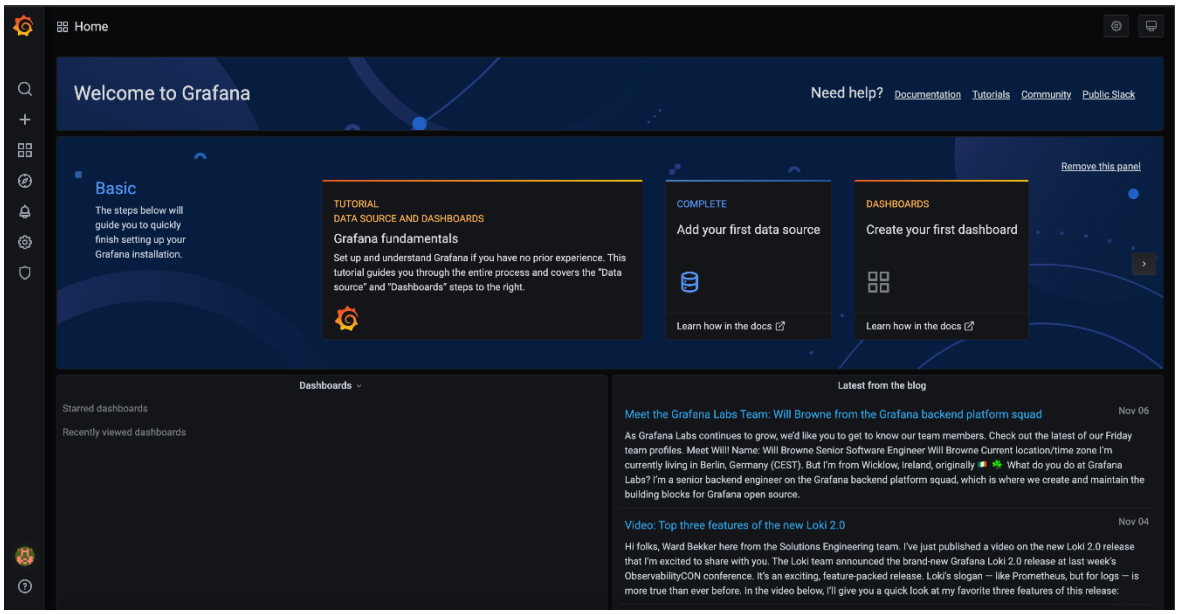


Picture 31: Grafana login screen

Picture 31 is the Grafana login screen. Next step is the login credentials for Grafana. Step #1 in picture 29 comes with a CLI command to fetch the Grafana UI login password and the default username “admin”. Time introduce copy and paste for the command and run it:

```
kubectl get secret -n staging grafana -o jsonpath=”{.data.admin-password}” | base 64 –decode ; echo
```

After the command is run, it returns a long string of random characters: `K5ZeL1aoDgmS9xuxN0E1JlTpFlTecjJspB4CpIaw`. This is the default behaviour of Grafana and part of its security measure. Now that the IP address, port, and login credentials are acquired it is time to take a peek inside Grafana.



Picture 32: Grafana main dashboard

Grafana is up and running as picture 32 demonstrates. This is just one way of accessing Grafana. The preferred is to use ingress resource to pipe the connection to the Grafana UI. Creating the ingress for Grafana gives the ability to access Grafana UI via previously shown address in Picture 29: `grafana.staging.svc.cluster.local`

6.3.2 Creating ingress-grafana.yaml configuration for Grafana

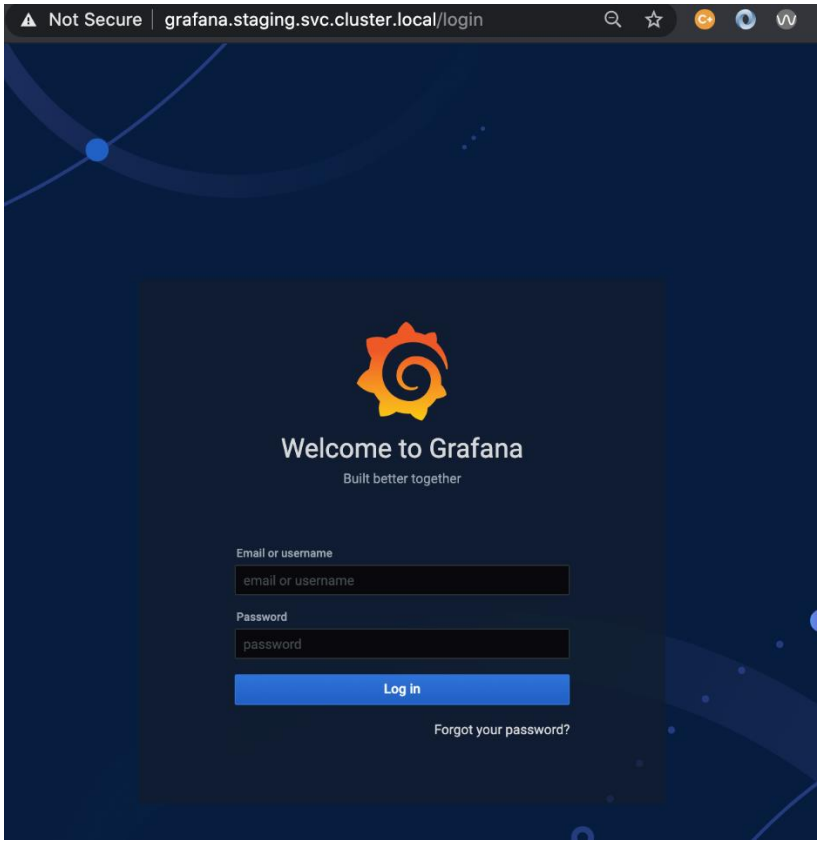
Code Snippet 6. Grafana ingress configuration (Kubernetes, 2020c.)

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress-grafana
spec:
  rules:
  - host: grafana.staging.svc.cluster.local
    http:
      paths:
      - path: /
        backend:
          serviceName: grafana
          servicePort: 3000
```

Code snippet 6 shows the very basic ingress setup for Kubernetes services, in this case for Grafana. Grafana ingress can be applied with the following kubectl command:

`kubectl apply -f helm/monitoring/ingress-grafana.yaml -n staging` If the command operation is done successfully it returns the message:

`ingress.extensions/ingress-grafana created`. Once implemented Grafana is now available at the address of `http://grafana.staging.svc.cluster.local/`



Picture 33: Grafana UI with Ingress

Great, picture 33 confirms Grafana UI is now accessible via http address. This is the preferred way to gain access to Grafana UI. Next step is to install Prometheus and Prometheus dependencies to the `staging` cluster.

6.4 Monitoring stack: Prometheus

This section describes the steps taken to configure Prometheus helm values `.yaml` -file and installation using HELM, ingress configuration and implementation with Kubernetes and RBAC (Role-Based Access Control) configuration with Kubernetes.

6.4.1 Creating values-prometheus.yaml configuration for Prometheus

Code snippet 7. values-prometheus.yaml HELM values configuration (Eficode Academy Github, 2020)

```
rbac:
  create: true
serviceAccounts:
  server:
    create: true
    name: user-x-sa
alertmanager:
  enabled: false
pushgateway:
  enabled: false
networkPolicy:
  enabled: false
nodeExporter:
  enabled: true
kubeStateMetrics:
  enabled: true
server:
  retention: "1d"
  resources:
    limits:
      cpu: 250m
      memory: 512Mi
    requests:
      cpu: 250m
      memory: 256Mi
  service:
    type: NodePort
    servicePort: 90
```

Code snippet 7 is the HELM values for Prometheus configuration. The template for the `values-prometheus.yaml` has been retrieved from Eficode Academy's hosted Advanced Kubernetes Application Development course. This configuration holds the initial setup for RBAC (Role-Based Access Control) and a basic service account which can be accessed. More on the RBAC in the section *6.4.3 creating rbac-prometheus.yaml configuration* in this thesis. Prometheus HELM chart has various services e.g. Prometheus alert manager, push gateway and network policy setups which are not needed in the initial setup of Prometheus to staging cluster in order to develop the very basic solution for ZDD monitoring

stack, so these are set as default to `false`. Node exporter and Kube state metrics in the other hand are essentials for Prometheus to do service discovery and perform metrics scraping, so both of these services are set to `true` (Eficode Academy Github, 2020.). The last part is the setup for Prometheus server retention, its resources and server service config. The `retention:` mapping defines the retention time for time series data, how long it is stored in the Prometheus server's database (Prometheus, 2020d.). The `resource:` mapping defines the CPU and Memory computational limit in the cluster node (this case Minikube node) and `service:` mapping defines the service type and its service port.

The development workflow is the same as in thesis section *6.3.1 Creating values-grafana.yaml configuration for Grafana* with just a few minor tweaks e.g. Prometheus does not have a login functionality. Since the workflow is the same compared to Grafana installation, the project already has the needed helm repo in the `staging` cluster. Next step is the installation of Prometheus to the `staging` cluster with `helm install` command:

```
helm install prometheus stable/prometheus -f helm/monitoring/values-prometheus.yaml --namescape=staging --version 11.11.0
```

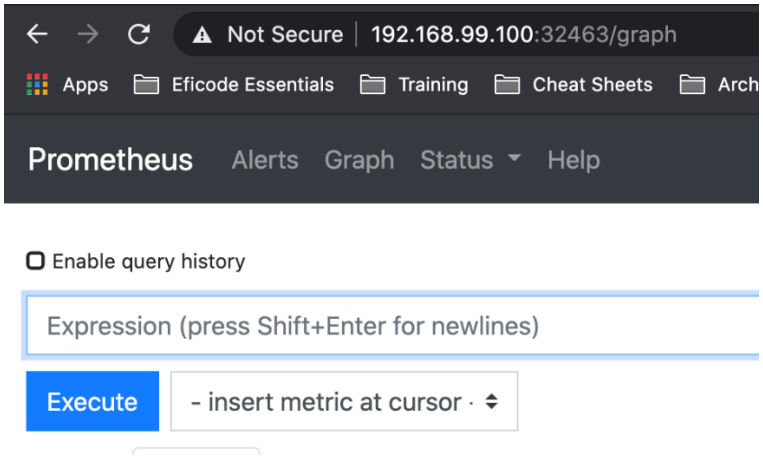
```
NAME: prometheus
NAMESPACE: staging
STATUS: deployed
REVISION: 2
TEST SUITE: None
NOTES:
DEPRECATED and moved to <https://github.com/prometheus-community/helm-charts>The Prometheus server can be accessed via port 90 on t
he following DNS name from within your cluster:
prometheus-server.staging.svc.cluster.local


Get the Prometheus server URL by running these commands in the same shell:
  export NODE_PORT=$(kubectl get --namespace staging -o jsonpath="{.spec.ports[0].nodePort}" services prometheus-server)
  export NODE_IP=$(kubectl get nodes --namespace staging -o jsonpath="{.items[0].status.addresses[0].address}")
  echo http://$NODE_IP:$NODE_PORT


#################################################################################
######   WARNING: Pod Security Policy has been moved to a global property.  #####
######            use .Values.podSecurityPolicy.enabled with pod-based      #####
######            annotations                                               #####
######            (e.g. .Values.nodeExporter.podSecurityPolicy.annotations) #####
#################################################################################



For more information on running Prometheus, visit:
https://prometheus.io/
```

Picture 34: Prometheus installed to staging cluster via HELM

As the Picture 34 shows, HELM has installed Prometheus to the staging cluster using a helm chart version of 11.11.0 which is Prometheus version 2.18.1. It seems I have to break the fourth wall here: The helm repo which the Prometheus community originally used to store the Prometheus helm chart is deprecated and moved to a new repo. This means I have to update Eficode Bitbucket documentation on Prometheus installation, but I digress. Next step is to access Prometheus UI (which runs in the `prometheus-server` pod). The method for getting the IP and the port number are the same, but since `minikube ip` (`192.168.99.100`) has been already fetched in the thesis section where Grafana installation was introduced, the only needed item is the Prometheus port:

`kubectl get -n staging -o jsonpath="{.spec.ports[0].nodePort}" services prometheus-server`

The console posts a port number `32463`. The full address can be now combined with the fetched port and the `minikube ip` (`192.168.99.100`).



Picture 35: Prometheus UI running at 192.168.99.100:32463

Prometheus UI is accessible at `192.168.99.100:32463` as picture 35 shows. As stated before, this is not the preferred way of accessing Grafana or Prometheus UI's and both connections should be piped through the ingress resource. Time to build Prometheus ingress resource configuration.

6.4.2 Creating ingress-prometheus.yaml configuration for Prometheus

Code snippet 8. Prometheus Ingress configuration (Kubernetes, 2020c.)

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress-prometheus
spec:
  rules:
  - host: prometheus-server.staging.svc.cluster.local
    http:
      paths:
      - path: /
        backend:
          serviceName: prometheus-server
          servicePort: 90
```

Code snippet 8 showcases the ingress configuration for Prometheus. The configuration setup is the same compared to Grafana's ingress configuration. The changes come when

designating `host:` mapping and `backend:` mapping in total (`serviceName`, `servicePort`). Now that the setup for Prometheus ingress is done, it needs to be applied to the `staging` cluster:

```
kubectl apply -f helm/monitoring/ingress-prometheus.yaml -n staging
```

Successful operation gives the message: `ingress.extensions/ingress-prometheus created`. The Prometheus ingress resource is now available in the `staging` cluster and Prometheus UI can be accessed via `http://prometheus-server.staging.svc.cluster.local/`.

Configuration of Prometheus is not yet completed, the full configuration needs one last piece of the puzzle. Prometheus requires permissions to access pods and services to scrape metric data from `/metrics` endpoints within the `staging` cluster. This is done by RBAC (Role-Based Access Control), so the next step is to create the Prometheus RBAC resource and deploy it to the `staging` cluster.

6.4.3 Creating rbac-prometheus.yaml configuration for Prometheus

The `rbac-promethes.yaml` -file is a three-piece single file which deploys 3 resources: `ClusterRole`, `ServiceAccount` and `ClusterRoleBinding` to the `staging` cluster. The `rbac-prometheus.yaml` ‘s base is taken from Prometheus teams Github repository and modified to match `staging` cluster purposes.

Code Snippet 9. First part of Prometheus RBAC (Role-Based Access Control) config (Prometheus Github, 2020)

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus
rules:
- apiGroups: [""]
  resources:
  - nodes
  - nodes/metrics
  - services
  - endpoints
  - pods
  verbs: ["get", "list", "watch"]
- apiGroups:
  - extensions
  resources:
  - ingresses
  verbs: ["get", "list", "watch"]
- nonResourceURLs: ["/metrics"]
  verbs: ["get"]
```

Code snippet 9 is the first part of the RBAC configuration entirety. The `ClusterRole` is a non-namespaced resource and defines a set of permissions to access cluster-scoped resources which can be seen under the yaml mapping `resources:` in code snippet 9. The yaml mapping `verbs:` is a defines the action a certain role is allowed to do. The yaml mapping `nonResourceURLs:` allows GET and POST requests to specific endpoint, in this case GET for `/metrics` endpoint (Kubernetes, 2020d.)

Code snippet 10. Second part of Prometheus RBAC (Role-Based Access Control) config (Prometheus Github, 2020)

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: prometheus
  namespace: staging
```

Code snippet 10 is the configuration to create a `ServiceAccount` for Prometheus server. `ServiceAccounts` is a role for the `ClusterRole` resource with the yaml mapping `name` of `prometheus` which works under the `namespace` of `staging`, the `logmontest` project cluster (Kubernetes, 2020d).

Code snippet 11. Third part of Prometheus RBAC (Role-Based Access Control) config (Prometheus Github, 2020)

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: prometheus
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus
subjects:
- kind: ServiceAccount
  name: prometheus
  namespace: staging
```

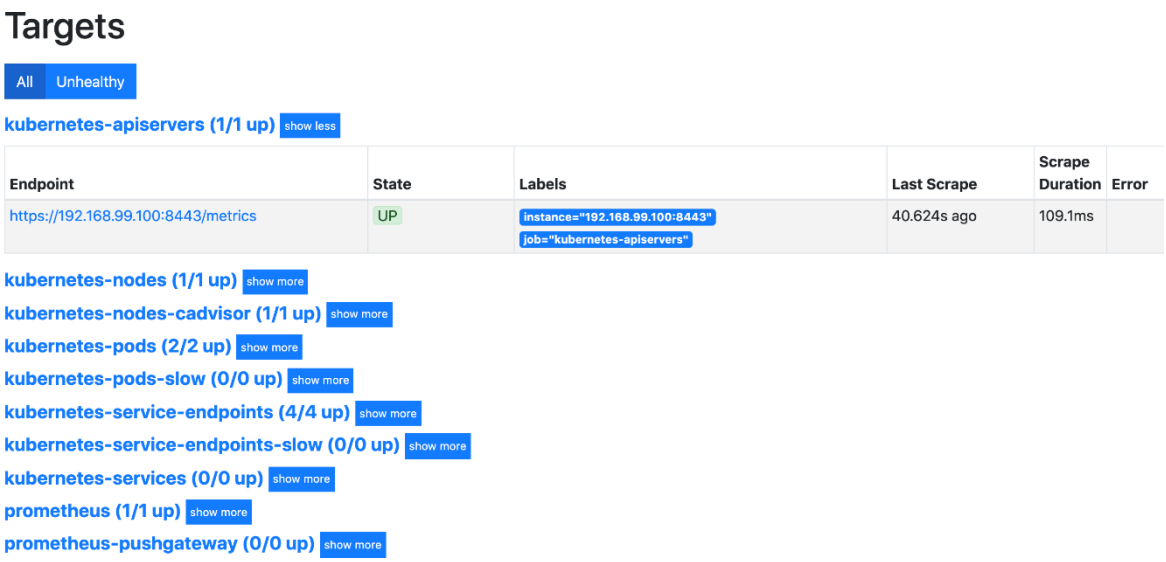
Code snippet 11 is the third and final part of the RBAC configuration. The `ClusterRoleBinding` resource binds the Prometheus `ServiceAccount` to the `ClusterRole` resource giving the Prometheus server the permissions to do service discovery and scraping of metrics from the endpoint of `/metrics` with the set of rules defined in the first part of the RBAC configuration (Kubernetes, 2020d.). The only modification for the Prometheus teams `rbac-setup.yaml` from Github was to change the `namespace:` mapping to the same namespace where the ZDD monitoring and logging development project is currently running, in this case the namespace is `staging`. With the configuration in place, it can be deployed to the `staging` cluster with a kubectl command:

```
kubectl apply -f /helm/monitoring/rbac-prometheus.yaml -n staging
```

```
clusterrole.rbac.authorization.k8s.io/prometheus configured
serviceaccount/prometheus configured
clusterrolebinding.rbac.authorization.k8s.io/prometheus configured
```

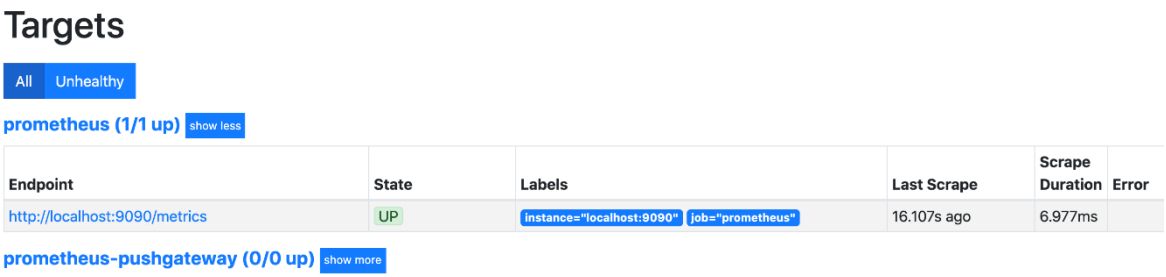
Picture 36: Confirmation of RBAC clusterrole, serviceaccount and clusterrolebinding

A successful apply gives the following confirmation in Picture 36. The RBAC resource implementation can also be confirmed in the Prometheus UI.



Picture 37: Prometheus Targets with RBAC

In Picture 37 is the presentation of Prometheus UIs Targets -tab. This shows the listing of different Kubernetes Jobs within the staging cluster e.g. Kubernetes-apiservers, Kubernetes-nodes and so forth. Under the different Kubernetes jobs there are the actual contents in a table form showcasing the endpoint (where Prometheus scrapes the metrics, `/metrics`), its state, labels, when last scrape was done, how long the scrape duration was and possible error messages.



Picture 38: Prometheus Targets without RBAC

The Picture 38 shows the same setting without the RBAC implementation in staging cluster. If Prometheus does not have the correct permissions to use service discovery methods in order to do metrics scraping, the only thing Prometheus can scrape is its own service and pod.

6.5 Logging stack: Grafana Loki

This section describes the steps I've taken to create default configuration, installation steps and possibility to upgrade Grafana Loki instance in a Kubernetes cluster. Once again, I follow the same principle as mentioned in *6.2 Grafana* and *6.4 Prometheus* in this thesis.

6.5.1 Creating values-loki.yaml configuration for Grafana Loki

Code snippet 12. values-loki.yaml HELM values configuration

```
rbac:
  create: false
  pspEnabled: false
service:
  type: ClusterIP
  port: 3100
```

Compared to Grafana (code snippet 4) and Prometheus (code snippet 7) values configuration Loki is much simpler (code snippet 12). For Loki only two arguments are needed to pass: disabling the RBAC (Role-Based Access Control) and the type of service Grafana Loki ought to be in the `staging` cluster, in this case a simple `ClusterIP` service. Loki does not need to be accessed externally but creating an ingress resource helps to test the `/` -endpoints in the future. More on this later.

Following the same workflow principle as in Grafana and Prometheus, but in Loki's case require the use of a different HELM repo. Adding the new HELM repo to staging cluster:

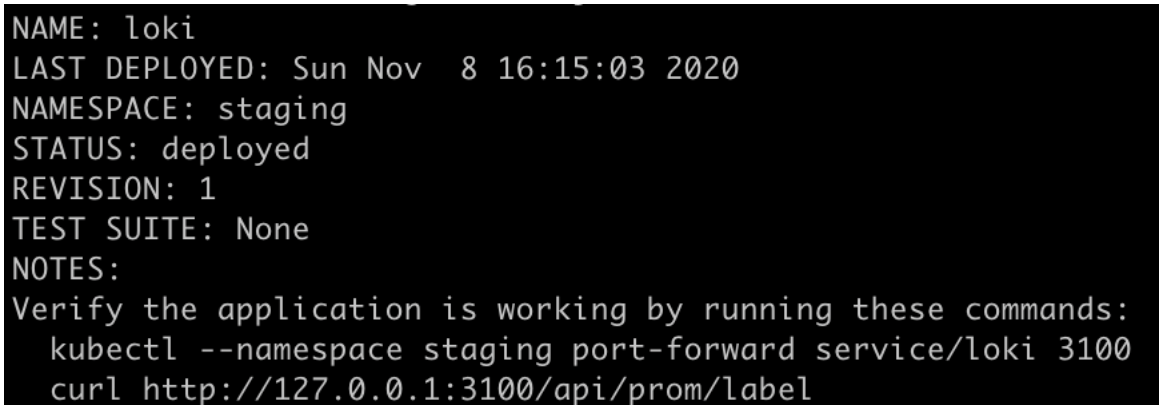
`helm repo add loki https://grafana.github.io/loki/charts`

```
Jannes-MacBook-Pro:logmontest jannes.$ helm repo add loki https://grafana.github.io/loki/charts
"loki" has been added to your repositories
Jannes-MacBook-Pro:logmontest jannes.$ helm repo list
NAME    URL
stable  https://kubernetes-charts.storage.googleapis.com/
loki    https://grafana.github.io/loki/charts
```

Picture 39: Loki helm repo add & helm repo list demonstration

This repository is under Grafana Labs Grafana Loki development Github (Picture 39). Now that the new HELM repo is added the installation of Grafana Loki to staging cluster can start:

`helm install loki loki/loki -f helm/logging/values-loki.yaml –namespace=staging`



Picture 40: Loki helm chart installation confirmation

Picture 40 shows the confirmation of a successful installation and gives a message to verify that the application is working properly. This verification method will not be used at this point. The following verification method has an issue of showing localhost ip address `http://127.0.0.1:3100/` as curl target and the `staging` cluster does not use the localhost ip. It uses the `minikube ip`. Loki should be piped through the ingress resource the same way as Grafana and Prometheus are piped through the ingress resource. Therefore, creating the ingress resource for Loki is the next step.

6.5.2 Creating ingress-loki.yaml for Grafana Loki

Code snippet 13. Grafana Loki ingress resource configuration (Kubernetes, 2020c.)

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress-loki
spec:
  rules:
  - host: loki.minikube.local
    http:
      paths:
      - path: /
        backend:
          serviceName: loki
          servicePort: 3100
```

Code snippet 13 shows the ingress configuration for Grafana Loki. The template is exactly the same as in Grafana and Prometheus ingress configuration. Ingress configuration for Loki is done and implementation is done by following the Kubernetes workflow principle:

`kubectl apply -f helm/logging/ingress-loki.yaml -n staging`

Greeted with the message `ingress.extensions/ingress-loki created` it is time to do a test with curl to verify the address `http://loki.minikube.local/` and the endpoint `/api/prom/label` which should give the response: `{“values:”:[“__name__”]}`:

```
Jannes-MacBook-Pro:logmontest jannes.$ curl http://loki.minikube.local/api/prom/label
{"values":["__name__"]}
```

Picture 41: Loki curl response

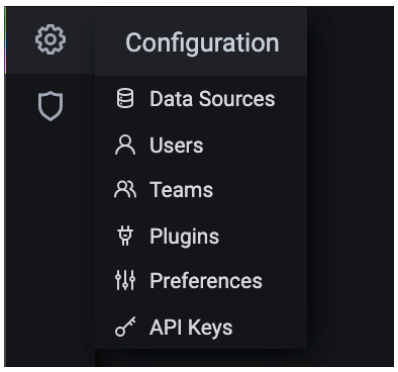
As picture 41 shows the verification worked, Grafana Loki is up and running, but the Loki instance does not contain any data, so it only responses with a JSON template of data it has available at this point. The use of Grafana Loki will be developed further in the future. Time to start working with the Grafana UI.

6.6 Grafana UI configuration

This is last section of the technical implementation. In this section all the work is based on using the Grafana UI to add data sources and create a dashboard template for Prometheus using PromQL. Once the template is created even in its very basic form, it would work as reference on how to do PromQL queries, how to create custom dashboard on Grafana and work as a starting template for future development in a customer software projects.

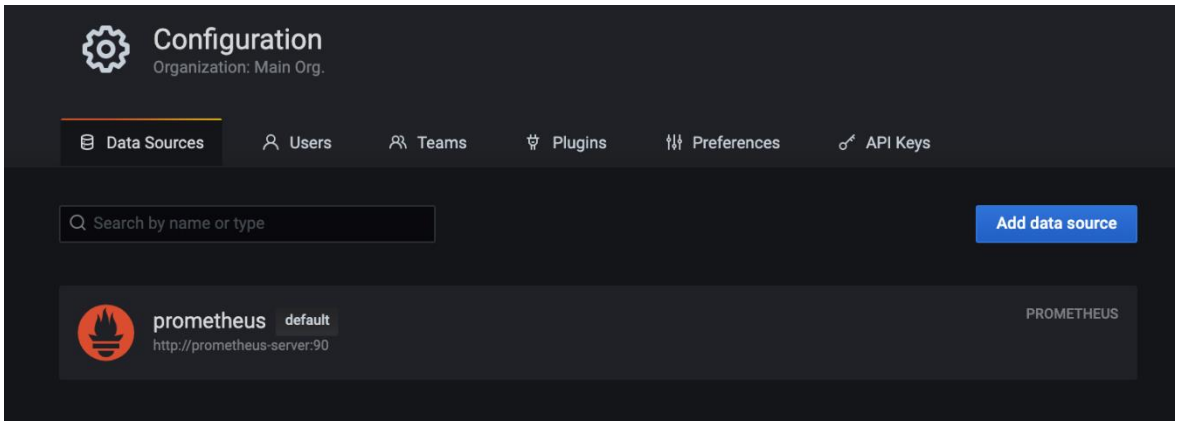
6.6.1 Grafana: Adding Prometheus as a data source

The first step in custom dashboard creation is to add Prometheus and Grafana Loki as a source to the Grafana instance. This is very straight-forward and done by logging into the Grafana instance and start the work on the home dashboard (Picture 32). The home dashboard has a set of icons on the left side (Grafana, 2020k.).



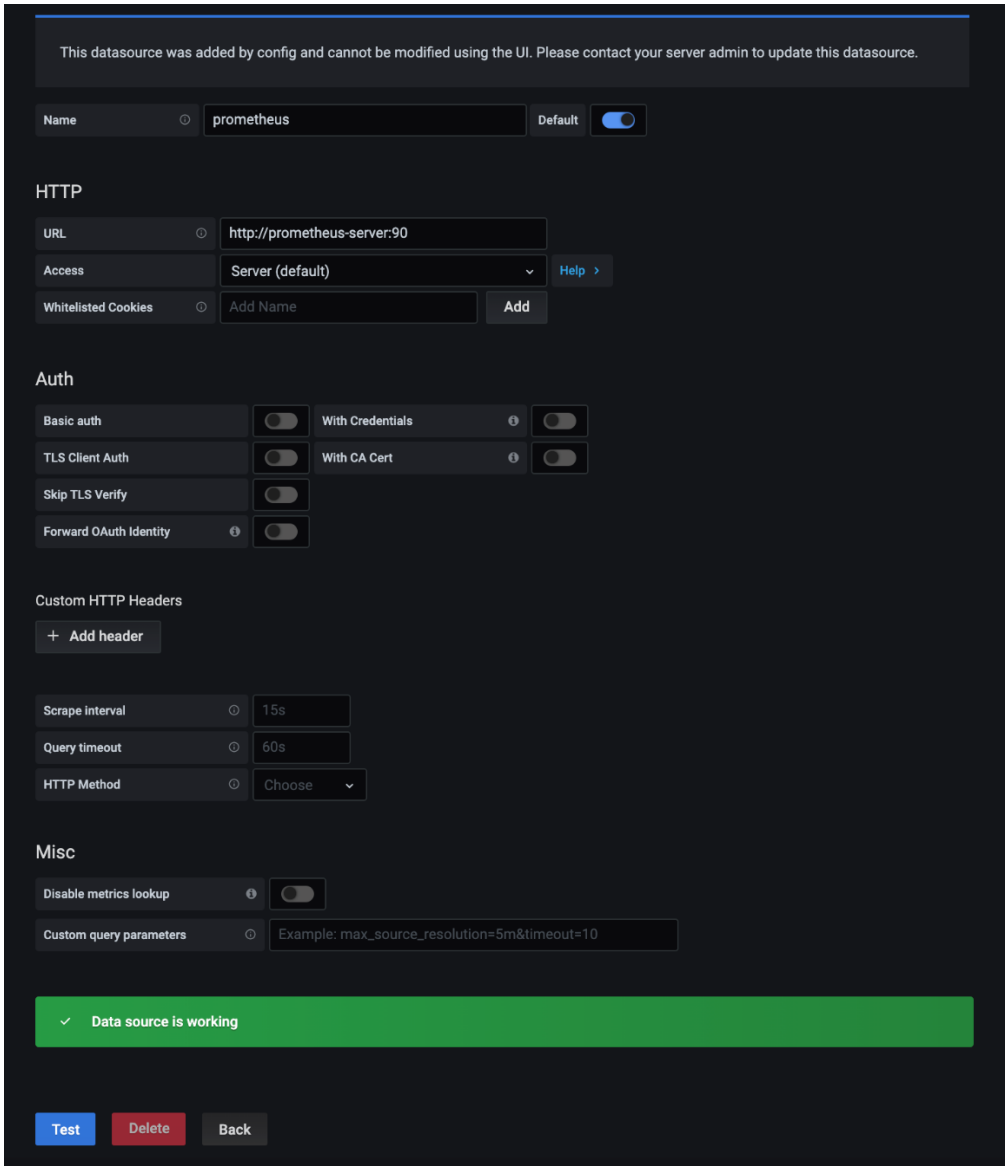
Picture 42: Accessing Grafana data sources (Grafana, 2020k.).

Clicking configuration cog icon on the left side (Picture 42) brings out the configuration menu and clicking data sources changes the page to data sources page (Grafana, 2020k.).



Picture 43: Grafana data sources page (Grafana, 2020k.).

The data sources page (picture 43) shows there is already one data source available, the Prometheus one. This is due to the fact that in the thesis section *6.3.1 Creating values-grafana.yaml configuration*. Prometheus data source was added to the `values-grafana.yaml` configuration file as demonstration to show one way of adding a data source to Grafana instance without the need of clicking through the Grafana UI (Grafana, 2020d.). A data source can be added by clicking the add data source button. More on this later. For now, the Prometheus data source settings can be accessed by clicking card item labelled as “Prometheus”.

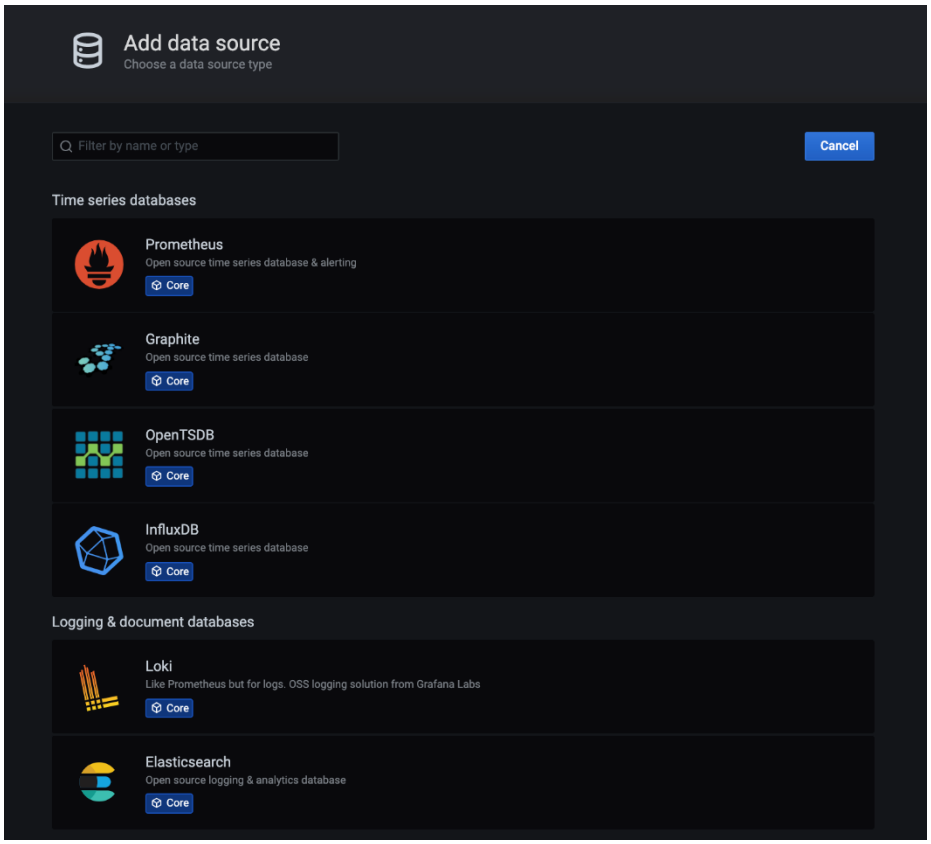


Picture 44: Prometheus data source settings in Grafana

A new page is introduced, the data source settings page (Picture 44). This already shows a major setback when adding data sources in the `values.yaml` configuration file, it cannot be deleted or modified, so it can be argued that this is not the optimal way of adding data sources to the Grafana instance. Doing this with Grafana UI. A data source can be simply added by giving a `http://URL:PORT` to the URL input, under the HTTP header. In this case, the port can be found in the `values-prometheus.yaml` file showcased in the thesis section *6.4.1 Creating values-prometheus.yaml* configuration in code snippet 7 as `servicePort: 90` and the URL is the name of the Kubernetes service, in this case `prometheus-server`. The data source connection can be tested with a click on the test -button which prompts Grafana to display the green boxed message “Data source is working”. Time to do the same thing to Grafana Loki, adding it by using the Grafana UI.

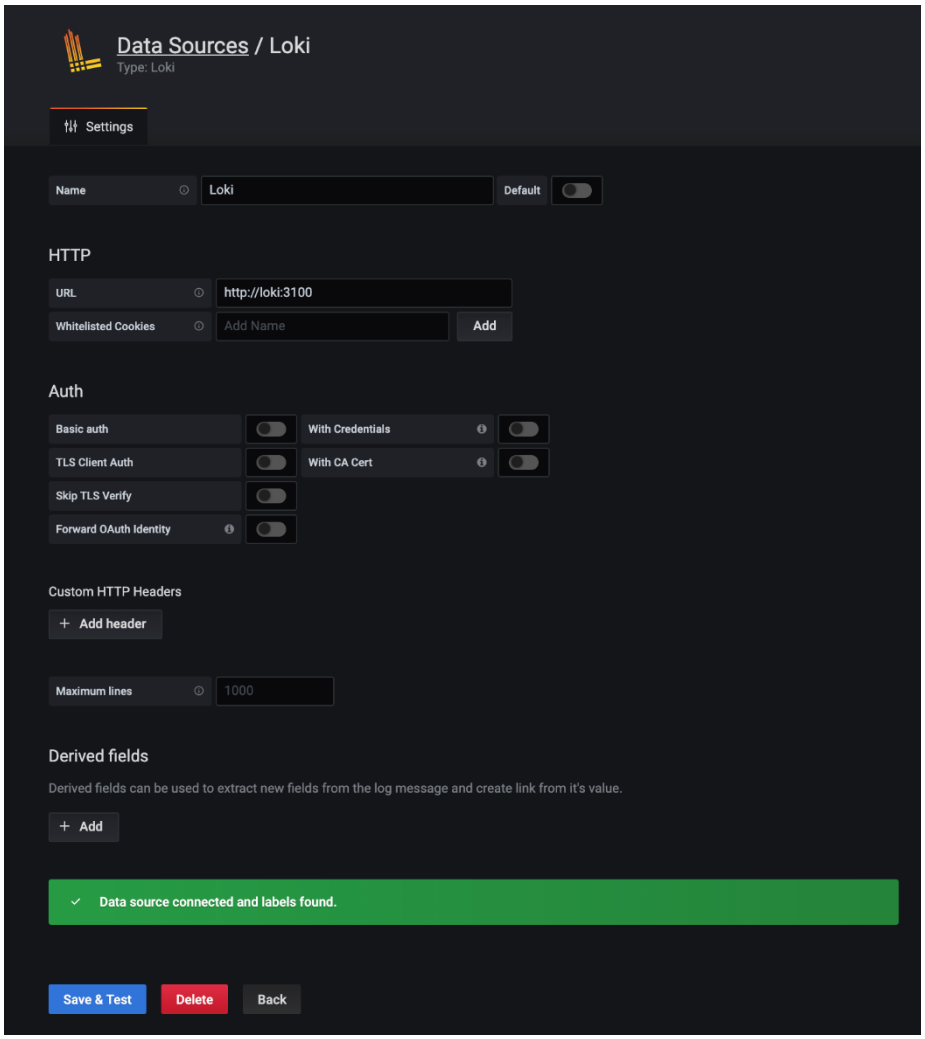
6.6.2 Grafana: Adding Grafana Loki as a data source

Adding Grafana Loki as a data source to Grafana instance by using the Grafana UI. The process is the same as demonstrated in the previous chapter: going to the data sources page from Grafana home dashboard, but this time the demonstration is taking a different route by clicking the add source button showcased in Picture # (Grafana, 2020k.).



Picture 45: Add data source page in Grafana

This is the add data source page (picture 45). From here different data sources can be added and a nice list is showcased on the page, showing the many supported data sources Grafana supports. To add Grafana Loki as a data source, this is done by clicking the card item labelled “Loki”.



Picture 46: Grafana Loki data source in Grafana

This is the data source settings for Grafana Loki (Picture #). From this page it should already be clear that adding data sources via Grafana UI makes the data sources deletable. One benefit for deletable data sources should be straight-forward, if the data source is not needed it can be deleted. The data source can be added the same as demonstrated previously adding the `URL:PORT` combination to the URL input under the HTTP header. Again, the port is defined in the *values-loki.yaml* configuration file showcased in the thesis section *6.5.1 Creating values-loki.yaml configuration* in code snippet 12 and the URL is the Kubernetes service “Loki”.

Now that both ways of adding a data source to Grafana is demonstrated and Prometheus is available for use, it is time to create a custom dashboard template for the ZDD project. As a reminder at this point, the logging solution for the ZDD project is still under development and therefore Grafana Loki does not contain any log data for use. Time for thesis summary.

7 Conclusions & summary

This section is the summary of this thesis. This is where I change the pronoun perspective and discuss the experience I've had with this project and with this thesis, since two of them are completely separate things in my opinion. Let's get started

7.1 Thesis: What was left out?

The initial research and scope calculations for this project started as early as January and February 2020. The original project plan was written in mid-March 2020 around the time when COVID-19 was declared a global pandemic event by the World Health Organisation. I highly underestimated the effects of COVID-19 pandemic in the project plan risk assessment. The drastic changes to normal day routines and work routines changed overnight and I took a personal decision at this point to focus on building the logging and monitoring solution in a local Minikube cluster to create a Proof of Concept type of a deal.

Due to the massive scope of this project, this thesis has encountered many refactors and summarisations during the time span of March 2020 to November 2020. At this point, I am very happy how the thesis looks like.

Summary of dropped topics:

- Comparison research of open source logging and monitoring tools in the CNCF landscape.
- The use of Git and Git commands for auditing third party tools (e.g. Grafana)
- The user feedback results from different tools e.g. Grafana, Prometheus, ElasticSearch and LogStash from within Eficode colleagues and programming related blogs, forums and reddit.
- The development of a custom dashboard in Grafana
- The creation of Slack alerting system in Grafana
- The implementation of additional Grafana Loki logging components (e.g. Promtail, Fluentd, Fluentbit)
- The creation of standardized logging format for NodeJS backend with Winston framework
- Winston log transporter for Grafana Loki
- The concept of observability

Some of the dropped topics may be addressed in the ZDD Confluence page e.g. the development of a custom dashboard in Grafana, the creation of Slack alerting system in Grafana and the implementation of Grafana Loki logging component Promtail.

7.2 The project results

The results regarding on this project have been in my opinion good . I have created a very basic setup for logging and monitoring solution that can be developed further very easily. The main goal of this project was to cut down the research and development time needed regarding on the creation of a logging and monitoring solution in a new SaaS (Software as a Service) -project. Following the installation documentation which will be provided in the ZDD Git repository will give basic instructions on how the whole logging and monitoring solution can be brought up in a Kubernetes cluster. As a by-product, anyone can use the ZDD project and the logging and monitoring solution related to it as a basic training platform to learn more about Grafana, Prometheus and Grafana Loki and how to operate with these tools.

The development workflow demonstrated in this project related to the use of HELM and Kubernetes are simple and reproducible in many aspects. If a development team does not want to use tools like Grafana, Prometheus or Grafana Loki, they can utilize the development workflow to try something else or use the workflow on somewhere else, not related to the ZDD project. It basically works as a set of commands as a template. The application packages provided in helm repositories works like building blocks, if there's something the developer needs, it can be fetched from the repository and installed to the cluster and it can also be deleted very easily from the cluster to try something new by following the workflow demonstrated in this project. Of course, this still requires the developer to do his or her own research related to the tool and produce their own `values.yaml` file and the needed Kubernetes resources (e.g. Ingress).

The research and development done on Grafana can provide information related to the use of Grafana as centralized monitoring system. In the final documentation (which will be released in Confluence) will hold the information needed to find data sources which Grafana supports e.g. Zabbix, Nagios, Graphite and many more. Of course, these tools require their own configuration, but it cuts down the time on research when providing information on what can be done with Grafana. The documentation on Grafana also provides information on how to add data source to Grafana when developing services in Kubernetes.

The research and development done with Prometheus will provide information on how to use the PromQL (Prometheus Query Language) to create ad-hoc queries in Prometheus and how to use it in general. This can also be implemented in Grafana when Prometheus is connected as a data source in Grafana.

Still, there is one major issue. While I might think the solutions and documentation works, there might be issues that I have not considered or foreseen. The solutions and the documentation needs to be reviewed by my peers in Eficode and tested in an actual project, so a definitive feedback loop for further development can be created. So, the straight feedback is not available in this thesis unfortunately. All of the findings during the development of this project will be released on ZDD Confluence page and the installation instructions will be released in ZDD's Bitbucket Git repository to be tested, to be reviewed and to be developed further.

7.3 The thesis results

Now about the thesis, this is a hard one. I think I have a bad case of tunnel visioning at this point due to the massive amount of information related to the project and due to the fact that I had to cut down so many topics related to the development of this project. In my opinion, every bit of information presented in this thesis are definitely needed to understand the development process, the selected tools, the code snippets, and the theory related to logging and monitoring. The biggest and hardest issue is the understanding of containerization and managing containerized environments with the addition of the theory related to logging and monitoring. In order to understand HELM, it requires the knowledge and know-how of Kubernetes concepts and as a cherry on top, to understand Kubernetes concepts you need to understand the concepts of Docker. It was very difficult to bring the needed theory related to the concepts to under 100 pages. As an example, I used Viktor Farcic's book *The DevOps Toolkit 2.5* which is the fifth instalment of a book series that spans over 6 books and each of these books hold at least 300 pages of material related to working in containerized environments. But I highly think how this thesis is built, it would at least direct people to the right sources.

7.4 Future ideas for development

There are tons of ideas for the future of the logging and monitoring project, which will be presented in the ZDD Confluence documentation page as development ideas.

The number one development idea is the implementation of TLS/SSL security for all the logging and monitoring components. I took the drastic decision of not implementing TLS/SSL security for the components to speed up the development process and due to the fact that the development was done in a localized environment.

The second development idea is the implementation of Google Auth to Grafana UI login functionality. This would work also as security measure and when implemented correctly, the Grafana UI would be only accessible to people with the right Google Auth credentials .

This would definitely serve one of the ISO:27001 certificate requirements (Annex 12.4.2) by bringing another layer of security to the whole solution.

The third development idea is to create same type of shell scripts for Grafana, Prometheus and Grafana Loki which are in the ZDD setup `build.sh`, `push.sh` and `deploy.sh`. This is due to the fact that people tend to typo long command strings and during this project, I encountered this problem many times which in some cases ended up in hour long debugging sessions and retracing my steps when problems occurred because of a typo.

I think these three development ideas are the critical ones and will be mentioned in the Confluence documentation among other development ideas.

7.5 Personal development

I came to this project with a frontend developer background, so it can be said that I had very minimal experience related to the technologies, the frameworks and the tools used in this project. The learning curve of Kubernetes is very steep and once you understand that you can use Minikube as a training platform really gives you the edge to try Kubernetes and HELM. I highly would recommend trying Kubernetes and HELM with Minikube as a way to introduce yourself to containerized environments and managing it.

So, the knowledge gained during the development of this project has been huge. I feel very comfortable working with Kubernetes by managing clusters, working with the YAML format to create Kubernetes objects, debugging with Kubernetes, and helping colleagues with Kubernetes related issues. I feel comfortable using HELM as way to install packages to a Kubernetes cluster, even thou I have not accumulated the needed experience to create my own helm charts, but at some point, I think I will cross that bridge too. I have gained basic understanding of monitoring and logging theory and implementation which I tend to use in upcoming customer projects e.g. creating a standardized log format, creating visualized metrics with a tool, basic understanding of what needs to be logged, what needs to be monitored and how to grow the monitoring maturity of an existing platform, application or a system.

The COVID-19 pandemic has also created new ways of working methods, how to separate work and free-time more thoroughly, how to work remotely and has helped me improve my workflow methods.

Overall, this project has helped me to understand more of the problems which we developers face, and it has helped me to recognize the fact that when creating solutions like these to an existing software development platform or a template you’re basically creating a service for a customer, the customer being your peer, a fellow developer.

Personally, I would like to thank Eficode for this opportunity to grow professionally in the field of IT.

8 List of references

Book refences:

Anton Chuvakin, Kevin Schmidt, Chris Phillips, 2013, Logging and Log Management, Syngress.
Retrieved From: https://learning.oreilly.com/

Michael J. Kavis, 2014. Architecting the Cloud, Wiley.
Retrieved From: Haaga-Helia Pasila Campus Library

Ernesto Garbarino, 2019, Beginning Kubernetes on the Google Cloud Platform, Apress
Retrieved From: https://learning.oreilly.com/

James Turnbull, 2018, The Docker Book, Turnbull Press
Retrieved From: https://learning.oreilly.com/

Viktor Farcic, 2019, The DevOps 2.5 Toolkit: Monitoring, Logging and Auto-Scaling Kubernetes.
Retrieved From: https://learning.oreilly.com/

James Turnbull, 2016, The Art of Monitoring, Turnbull Press
Retrieved From: https://leanpub.com/

Standards

ISO 27001:2013. International Organisation for Standardisation – Security techniques – Information Security Management System – Requirements.

Web references:

Mozilla Developer Network, 2020a. “What is JavaScript”.
Retrieved From: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript
Accessed: 16.03.2020

Mozilla Developer Network, “Web security”, 2020b
Retrieved From: https://developer.mozilla.org/en-US/docs/Web/Security
Accessed: 11.05.2020

Mozilla Developer Network, “An overview of HTTP”, 2019
Retrieve From: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview
Accessed: 11.05.2020

Winston Github, “Winston”, 2020
Retrieved from: https://github.com/winstonjs/winston
Accessed: 22.03.2020

IETF Network Working Group, “RFC 5424: The Syslog Protocol”, 2009
Retrieved From: https://tools.ietf.org/html/rfc5424
Accessed: 23.03.2020

Cloudflare Inc, 2020a. “What Is Transport Layer Security (TLS)?”
Retrieved From: https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/
Accessed: 23.03.2020

Cloudflare Inc, 2020b. “What is SSL?”
Retrieved From: https://www.cloudflare.com/learning/ssl/what-is-ssl/
Accessed: 23.03.2020

RESTfulAPI.net, “HTTP Status Codes”, 2020
Retrieved From: https://restfulapi.net/http-status-codes/
Accessed: 29.03.2020

Dzone, “Application Logging: What, When, How”, 2009
Retrieved From: https://dzone.com/articles/application-logging-what-when
Accessed: 29.03.2020

CNCF (Cloud Native Computing Foundation), “FAQ”, 2020a
Retrieved From: https://www.cncf.io/about/faq/#what-is-cloud-native
Accessed: 27.04.2020

CNCF (Cloud Native Computing Foundation), ”Projects”, 2020b.
Retrieved From: https://www.cncf.io/projects/
Accessed: 06.08.2020

CNCF (Cloud Native Computing Foundation), “CNCF Cloud Native Interactive Landscape”, 2020c.
Retrieved From: https://landscape.cncf.io/
Accessed: 06.08.2020

Open Source Initiative “The Open Source Definition”, 2020
Retrieved From: https://opensource.org/osd
Accessed: 07.05.2020

Brandon Wozniewicz. 01.02.2019. “The Difference Between a Framework and a Library” – freeCodeCamp.
Retrieved From: https://www.freecodecamp.org/news/the-difference-between-a-framework-and-a-library-bd133054023f/
Accessed: 07.05.2020

Atlassian, “What is Jira used for?”, 2020a
Retrieved From: https://www.atlassian.com/software/jira/guides/use-cases/what-is-jira-used-for#Jira-for-requirements-&-test-case-management
Accessed: 08.05.2020

Atlassian, “About Confluence” 2020b
Retrieved From: https://www.atlassian.com/software/confluence/guides/get-started/confluence-overview#hosting-options
Accessed: 12.09.2020

Atlassian, “A brief overview of Bitbucket”, 2020c
Retrieved From: https://bitbucket.org/product/guides/getting-started/overview
Accessed: 12.10.2020

Atlassian, “What is a Kanban board?”, 2020d
Retrieved From: https://www.atlassian.com/agile/kanban/boards
Accessed: 12.10.2020

Devopedia, “Package manager”, 2018
Retrieved From: https://devopedia.org/package-manager
Accessed: 09.05.2020

Twilio Docs, “Application”, 2020
Retrieved From: https://www.twilio.com/docs/glossary/what-is-an-application
Accessed: 09.05.2020

Git-scm, “Git”, 2020
Retrieved From: https://git-scm.com/
Accessed: 10.05.2020

Beau Carnes. 14.11.2019. “What does an API Stand For? A Definition of the Coding Acronym in Plain English” – freeCodeCamp.
Retrieved From: https://www.freecodecamp.org/news/what-does-api-stand-for-a-definition-of-the-coding-acronym-in-plain-english/
Accessed: 10.05.2020

Indiana University, “About UNIX”, 2020
Retrieved From: https://kb.iu.edu/d/agat
Accessed: 10.05.2020

IETF, “About”, 2020
Retrieved From: https://www.ietf.org/about/
Accessed: 10.05.2020

ISO, “ISO/IEC 27001 Information Security Management”, 2020
Retrieved From: https://www.iso.org/isoiec-27001-information-security.html
Accessed: 11.05.2020

Dominik Kundel. 06.05.2019. “A Guide to Node.js Logging” – Twilio.
Retrieved From: https://www.twilio.com/blog/guide-node-js-logging
Accessed: 06.06.2020

Jesse Storimer. 29.11.2011. “When to use STDERR instead of STDOUT”
Retrieved From: https://www.jstorimer.com/blogs/workingwithcode/7766119-when-to-use-stderr-instead-of-stdout
Accessed: 06.06.2020

Core Infrastructure Initiative, ”CII Best Practices Badge Program”, 2018.
Retrieved From: https://bestpractices.coreinfrastructure.org/en
Accessed: 06.08.2020

OpenJS Foundation, “Introduction to Node.js”, 2020
Retrieved From: https://nodejs.dev/
Accessed: 22.03.2020

Eficode Academy Github, “Kubernetes-appdev-katas”, 2020
Retrieved From: https://github.com/eficode-academy/kubernetes-appdev-katas
Accessed: 05.05.2020

OpenBSD, “SYSCTL(8)”, 2018
Retrieved From: http://man.openbsd.org/sysctl
Accessed: 01.08.2020

Oracle, ”Download Virtualbox”, 2020
Retrieved From: https://www.virtualbox.org/wiki/Downloads
Accessed: 01.08.2020

Oracle, “hosts(4)”, 2014
Retrieved From: https://docs.oracle.com/cd/E36784_01/html/E36882/hosts-4.html
Accessed: 07.11.2020

Google Github, ”cAdvisor”, 2020.
Retrieved From: https://github.com/google/cadvisor
Accessed: 20.10.2020

YAML, “%YAML 1.2”, 2011
Retrieved From: https://yaml.org/
Accessed: 07.11.2020

JSON, “Introducing JSON”, 2020.
Retrieved From: https://www.json.org/json-en.html
Accessed: 07.11.2020

Docker

Docker, “Why Docker?”, 2020a.
Retrieved from: https://www.docker.com/why-docker
Accessed: 23.03.2020

Docker, “What is a container?”, 2020b.
Retrieved From: https://www.docker.com/resources/what-container
Accessed: 23.03.2020

Docker, “Docker for Mac”, 2020c.
Retrieved From: https://docs.docker.com/docker-for-mac/install/
Accessed: 01.08.2020

Kubernetes

Minikube, “Documentation”, 2020a.
Retrieved From: https://minikube.sigs.k8s.io/docs/
Accessed: 07.06.2020

Minikube, ”Virtualbox”, 2020b.
Retrieved From: https://minikube.sigs.k8s.io/docs/drivers/virtualbox/
Accessed: 01.08.2020

Minikube, “Host access”, 2020c.
Retrieved From: https://minikube.sigs.k8s.io/docs/handbook/host-access/
Accessed: 07.11.2020

Minikube, “Basic controls”, 2020d.
Retrieved From: https://minikube.sigs.k8s.io/docs/handbook/controls/
Accessed: 07.11.2020

Kubernetes Github, “Kube state metrics”, 2020
Retrieved From: https://github.com/kubernetes/kube-state-metrics
Accessed: 19.08.2020

Kubernetes, “What is Kubernetes?”, 2020a
Retrieved From: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/
Accessed: 26.03.2020

Kubernetes, “Install Minikube”, 2020b
Retrieved From: https://kubernetes.io/docs/tasks/tools/install-minikube/
Accessed: 01.08.2020

Kubernetes, “Ingress”, 2020c.
Retrieved From: https://kubernetes.io/docs/concepts/services-networking/ingress/
Accessed: 07.06.2020

Kubernetes, “Install and Set Up kubectl”, 2020d
Retrieved From: https://kubernetes.io/docs/tasks/tools/install-kubectl/
Accessed: 01.08.2020

Kubernetes, “Specifying the VM driver”, 2020e.
Retrieved From: https://kubernetes.io/docs/setup/learning-environment/minikube/#specifying-the-vm-driver
Accessed: 01.08.2020

Kubernetes, “Using RBAC Authorization”, 2020d.
Retrieved From: https://kubernetes.io/docs/reference/access-authn-authz/rbac/#role-and-clusterrole
Accessed: 17.08.2020

Kubernetes, “Managing Resources for Containers”, 2020e.
Retrieved From: https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/#requests-and-limits
Accessed: 17.08.2020

Kubernetes, “Kubectl Commands”, 2020f
Retrieved From: https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands
Accessed: 20.10.2020

Kubernetes, “Service”, 2020g.
Retrieved From: https://kubernetes.io/docs/concepts/services-networking/service/
Accessed: 01.11.2020

Kubernetes, “Understanding Kubernetes Objects”, 2020h.
Retrieved From: https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/
Accessed: 01.11.2020

Kubernetes, “Set up Ingress on Minikube with the NGINX Ingress Controller”, 2020i
Retrieved From: https://kubernetes.io/docs/tasks/access-application-cluster/ingress-minikube/
Accessed: 01.11.2020

Kubernetes “Standardized Glossary”, 2020j.
Retrieved From: https://kubernetes.io/docs/reference/glossary/?fundamental=true
Accessed: 07.11.2020

HELM

HELM, “What is Helm?”, 2020a.
Retrieved From: https://helm.sh/
Accessed: 27.04.2020

HELM, ”Installing Helm”, 2020b.
Retrieved From: https://helm.sh/docs/intro/install/
Accessed: 01.08.2020

HELM, “Helm repo add”, 2020c.
Retrieved From: https://helm.sh/docs/helm/helm_repo_add/
Accessed: 10.10.2020

HELM, “Install an example chart”, 2020d
Retrieved From: https://helm.sh/docs/intro/quickstart/#install-an-example-chart
Accessed: 10.10.2020

HELM, “Helm upgrade”, 2020e.
Retrieved From: https://helm.sh/docs/helm/helm_upgrade/#helm
Accessed: 10.10.20

HELM, “Helm list”, 2020f.
Retrieved From: https://helm.sh/docs/helm/helm_list/
Accessed: 10.10.2020

HELM, “Helm repo remove”, 2020g.
Retrieved From: https://helm.sh/docs/helm/helm_repo_remove/#helm
Accessed: 10.10.2020

Helm, “Charts”, 2020h.
Retrieved From: https://helm.sh/docs/topics/charts/
Accessed: 18.10.2020

Helm, “The chart repository guide”, 2020i.
Retrieved From: https://helm.sh/docs/topics/chart_repository/#helm
Accessed: 18.10.2020

Helm, “Using Helm”, 2020j.
Retrieved From: https://helm.sh/docs/intro/using_helm/
Accessed: 18.10.2020

Grafana

Grafana Github, ”Grafana”, 2020a.
Retrieved From: https://github.com/grafana/grafana
Accessed: 06.08.2020

Grafana Github, “Loki: like Prometheus, but for logs.”, 2020b.
Retrieved From: https://github.com/grafana/loki
Accessed: 21.10.2020

Grafana, ”Grafana Plugins, 2020b.
Retrieved From: https://grafana.com/grafana/plugins?type=datasource
Accessed: 08.08.2020

Grafana, ”Grafana Documentation”, 2020c.
Retrieved From: https://grafana.com/docs/grafana/latest/getting-started/what-is-grafana/
Accessed: 08.08.2020

Grafana, “Provisioning”, 2020d.
Retrieved From: https://grafana.com/docs/grafana/latest/administration/provisioning/
Accessed: 18.08.2020

Grafana, “Loki in Grafana”, 2020e.
Retrieve From: https://grafana.com/docs/loki/latest/getting-started/grafana/
Accessed: 18.08.2020

Grafana, “LogQL: Log Query Language”, 2020f.
Retrieved From: https://grafana.com/docs/loki/latest/logql/
Accessed: 18.08.2020

Grafana, “Loki clients”, 2020g.
Retrieved From: https://grafana.com/docs/loki/latest/clients/
Accessed: 20.08.2020

Grafana, “Overview of Loki”, 2020h.
Retrieved From: https://grafana.com/docs/loki/latest/overview/
Accessed: 20.08.2020

Grafana, “Loki clients”, 2020i.
Retrieved From: https://grafana.com/docs/loki/latest/clients/
Accessed: 22.08.2020

Grafana, “Querying Loki with LogCLI”, 2020j.
Retrieved From: https://grafana.com/docs/loki/latest/getting-started/logcli/
Accessed: 22.08.2020

Grafana, “Add a data source”, 2020k.
Retrieved From: https://grafana.com/docs/grafana/latest/datasources/add-a-data-source/
Accessed: 13.11.2020

Prometheus

Prometheus, “Storage”, 2020d
Retrieved From: https://prometheus.io/docs/prometheus/latest/storage/
Accessed: 05.05.2020

Prometheus, ”Introduction”, 2020a.
Retrieved From: https://prometheus.io/docs/introduction/overview/
Accessed: 09.08.2020

Prometheus, “Data Model”, 2020b
Retrieved From: https://prometheus.io/docs/concepts/data_model/
Accessed: 10.08.2020

Prometheus, “Querying Prometheus”, 2020c.
Retrieved From: https://prometheus.io/docs/prometheus/latest/querying/basics/
Accessed: 10.08.2020

Prometheus, “Exporters and Integrations”, 2020e.
Retrieved From: https://prometheus.io/docs/instrumenting/exporters/
Accessed: 17.08.2020

Prometheus Github, “rbac-setup.yaml”, 2020
Retrieved from: https://github.com/prometheus/prometheus/blob/master/documentation/examples/rbac-setup.yml
Accessed: 17.08.2020

Prometheus Github, “Service Discovery”, 2020b
Retrieved From: https://github.com/prometheus/prometheus/tree/master/discovery
Accessed: 17.08.2020

Prometheus Github, “Node Exporter”, 2020c.
Retrieved From: https://github.com/prometheus/node_exporter
Accessed: 20.10.2020

Eficode internal sources

Eficode ZDD Bitbucket, “node-react-project” , 2020,
Retrieved: 16.03.2020

Eficode ZDD Confluence, “Zeroday / Node.js + React Project Template”, 2020.
Retrieved: 16.03.2020

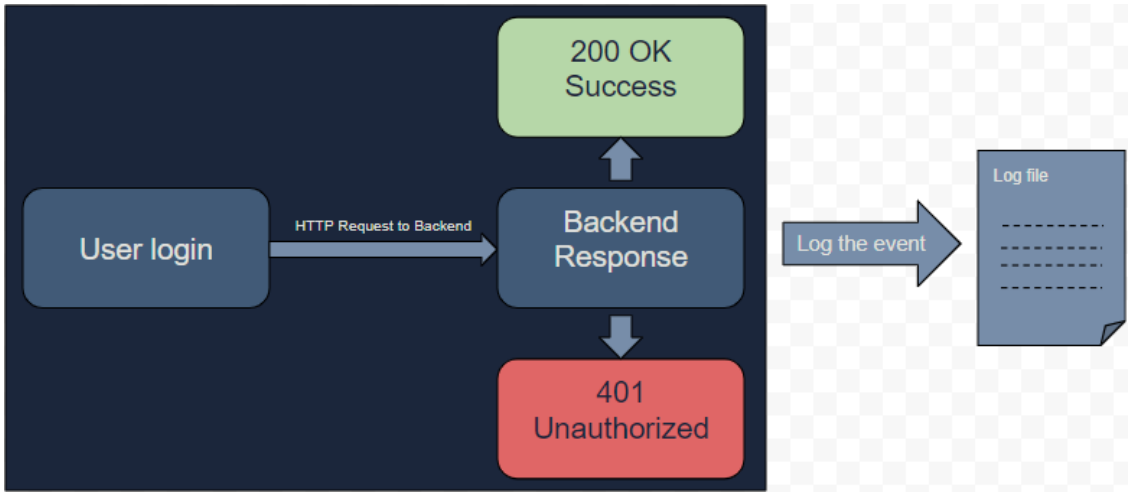
Eficode ZDD Jira, “Zero Day Delivery Kanban board”, 2020.
Retrieved: 16.03.2020

9 Appendices

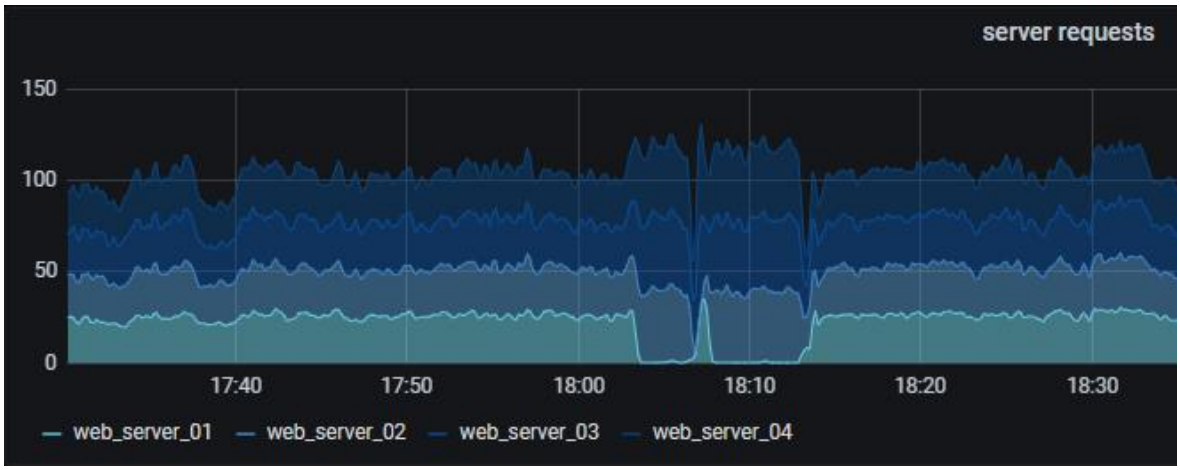
9.1 Pictures

9.1.1 Picture 1: Log message demonstration using Chrome console and JavaScript

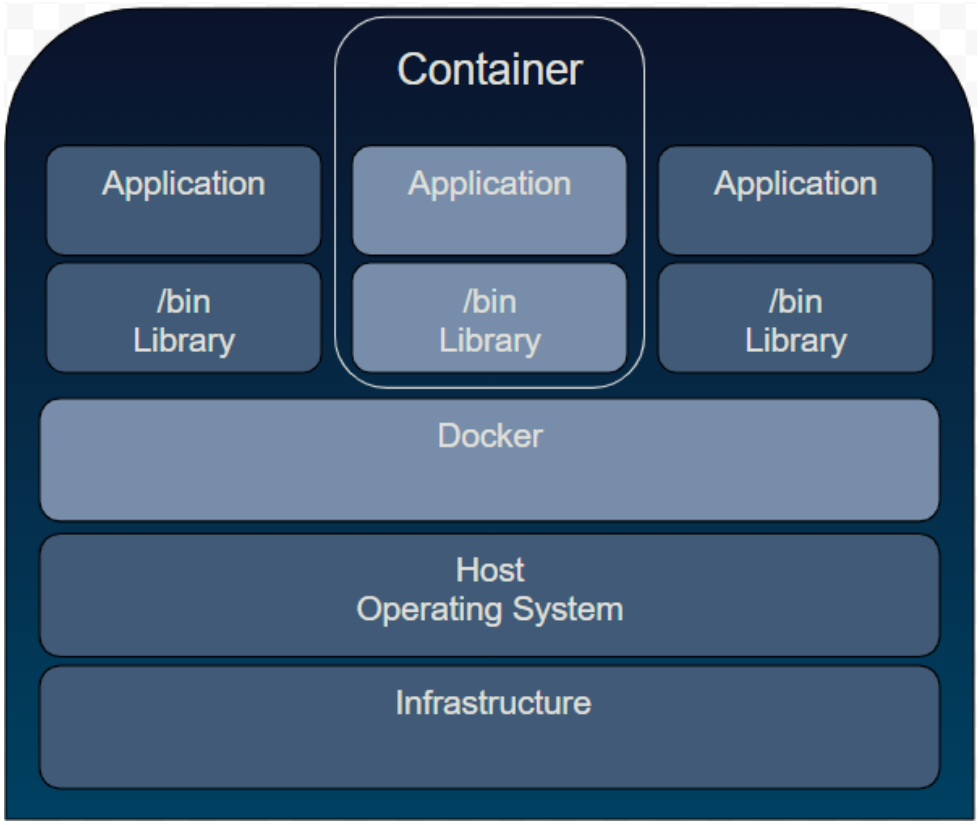
9.1.2 Picture 2: User login event with two outcomes



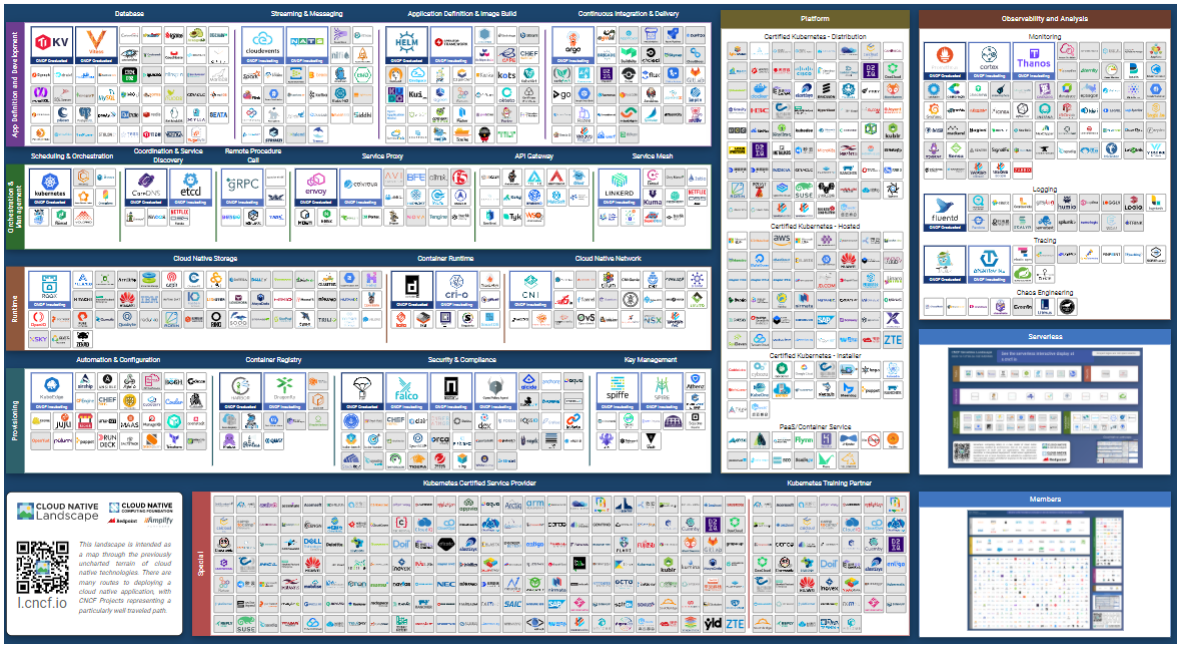
9.1.3 Picture 3: Metric aggregation of server requests in Grafana (Turnbull 2014, p. 33)



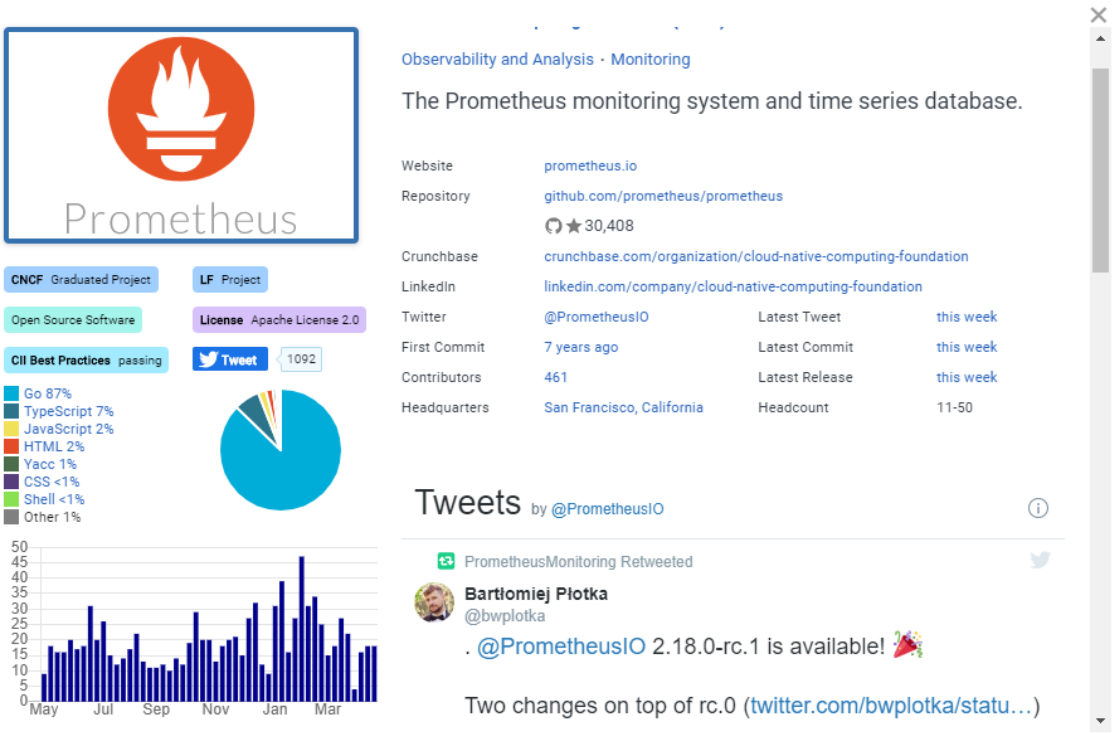
9.1.4 Picture 5: What is a container (Docker, 2020.)



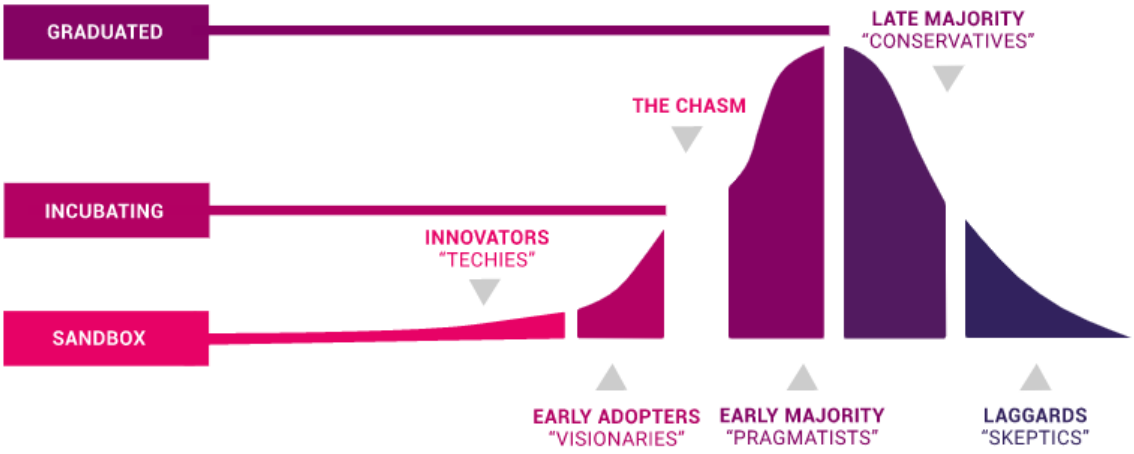
9.1.5 Picture 5: The CNCF Landscape (CNCF, 2020b.).



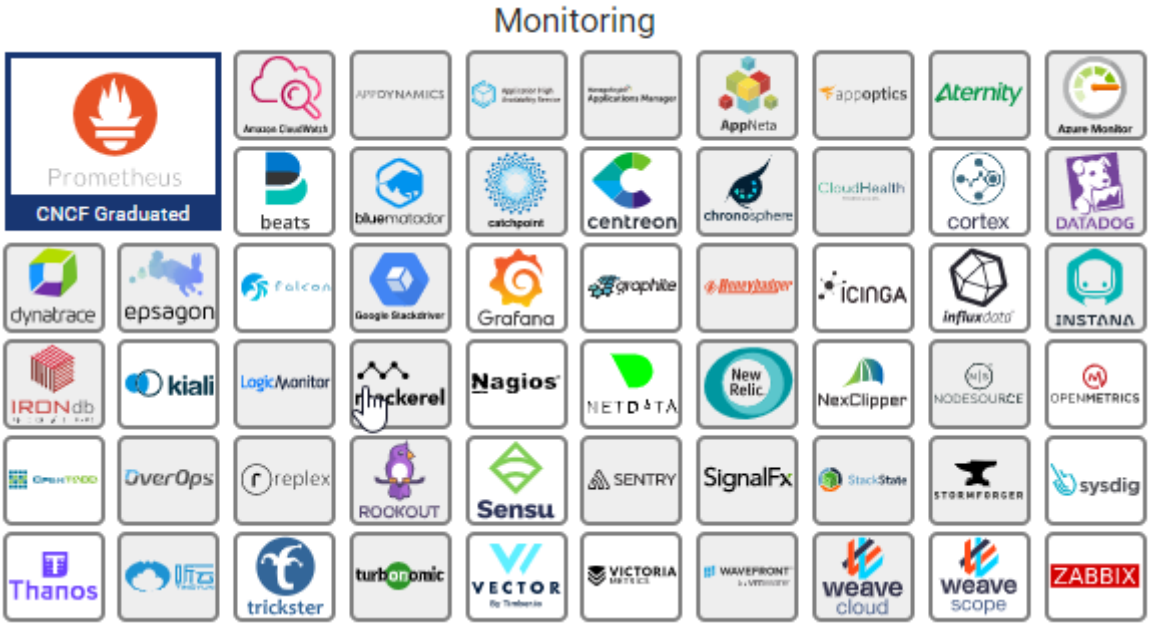
9.1.6 Picture 6: Prometheus information by CNCF (CNCF, 2020b.),



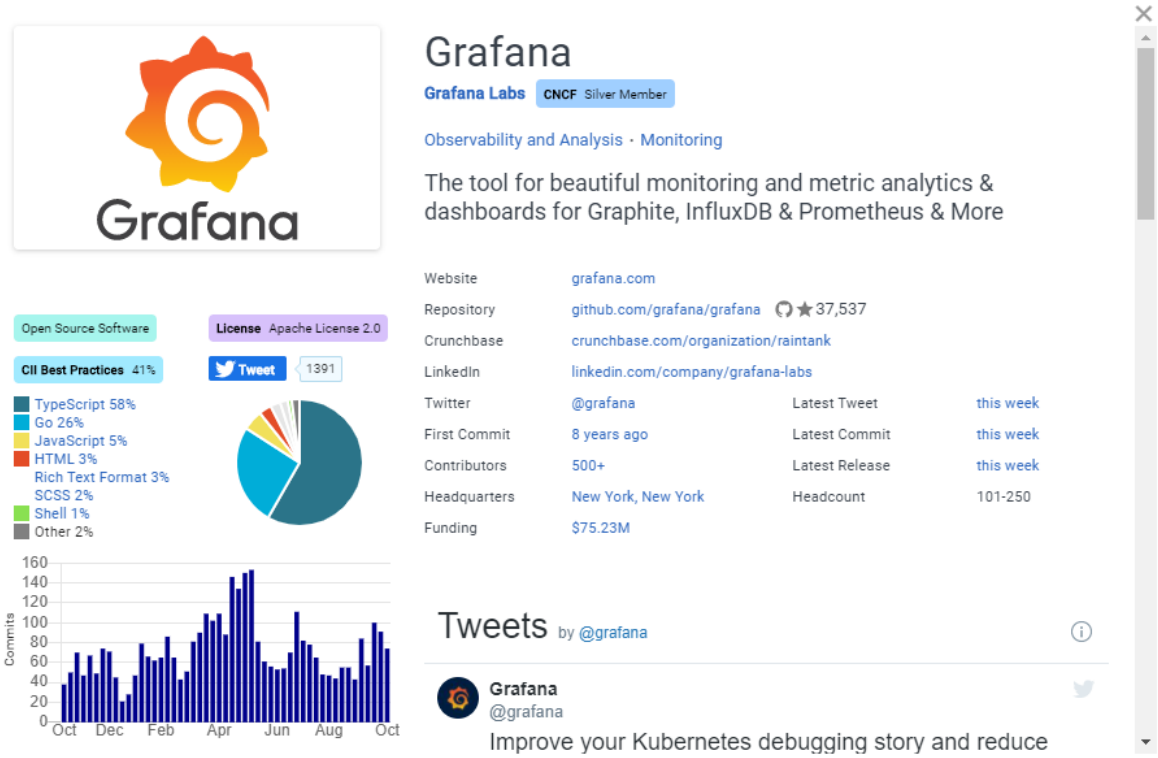
9.1.7 Picture 7: CNCF Project maturity levels (CNCF, 2020a.).



9.1.8 Picture 8: CNCF Landscape listed monitoring frameworks (CNCF, 2020c.).

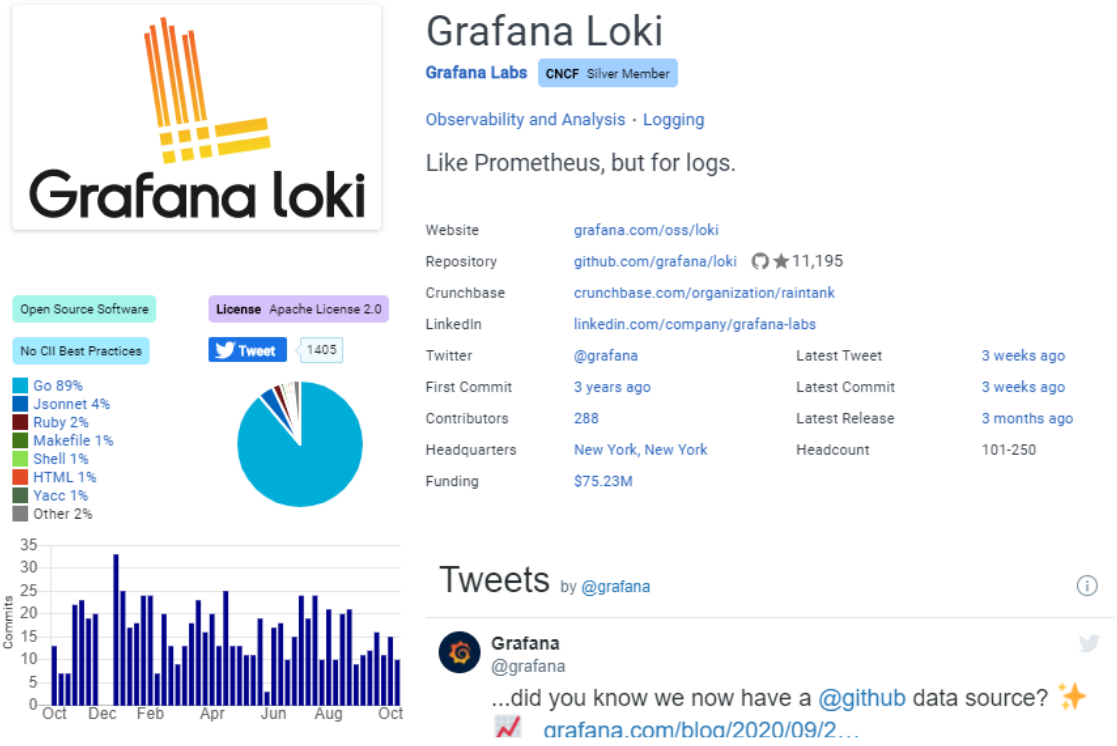


9.1.9 Picture 9: Grafana information by CNCF (CNCF, 2020b.).

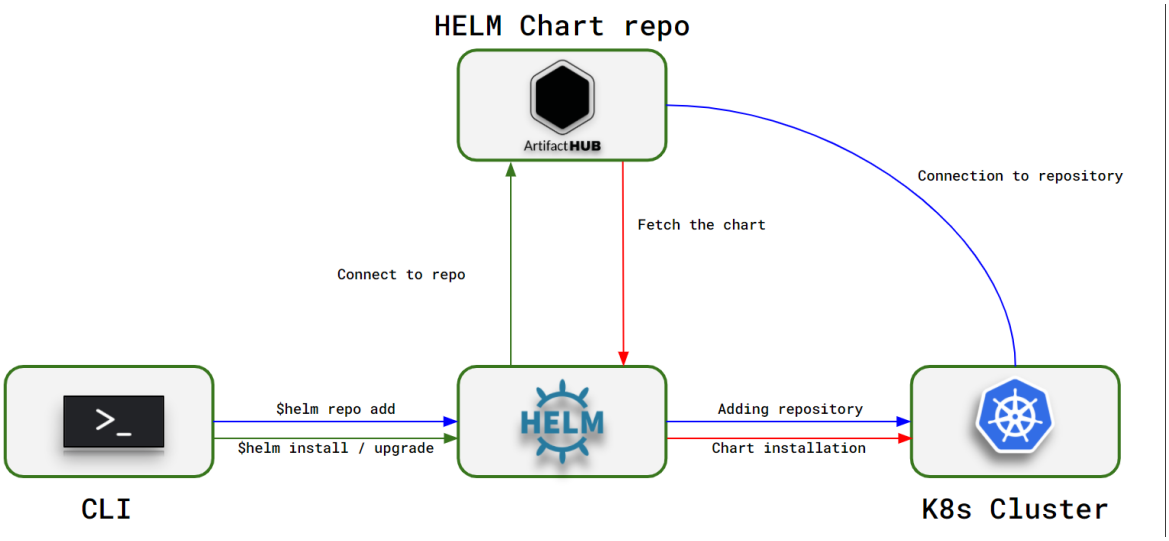


9.1.10 Picture 10: CNCF Landscape listed logging frameworks (CNCF, 2020b.).

9.1.11 Picture 11: Grafana Loki on CNCF landscape (CNCF, 2020b.).



9.1.12 Picture 12: Helm workflow demonstration for ZDD project



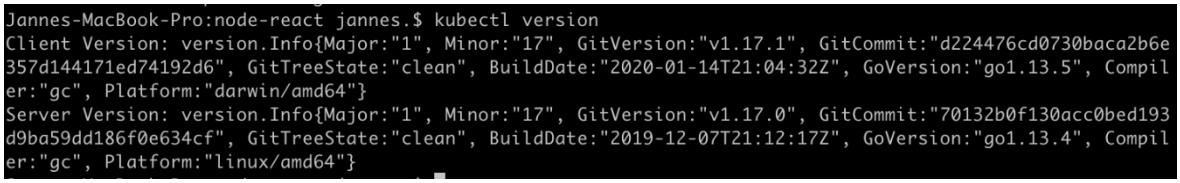
9.1.13 Picture 13: /etc/hosts -file structure

```
##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting.  Do not change this entry.
##
127.0.0.1       localhost
255.255.255.255 broadcasthost
::1             localhost
192.168.99.100  grafana.staging.svc.cluster.local staging.minikube.local
# Added by Docker Desktop
# To allow the same kube context to work on the host and the container:
127.0.0.1 kubernetes.docker.internal
# End of section
```

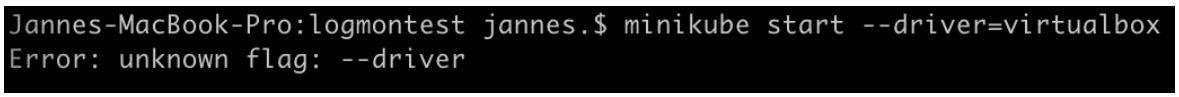
9.1.14 Picture 14: Confirmation for enabled virtualized in MacOS System

```
Jannes-MacBook-Pro:node-react jannes.$ sysctl -a | grep -E --color "machdep.cpu.features|VMX"
machdep.cpu.features: FPU VME DE PSE TSC MSR PAE MCE CX8 APIC SEP MTRR PGE MCA CMOV PAT PSE36 CLFSH DS ACP
I MMX FXSR SSE SSE2 SS HTT TM PBE SSE3 PCLMULQDQ DTES64 MON DSCPL VMX SMX EST TM2 SSSE3 FMA CX16 TPR PDCM
SSE4.1 SSE4.2 x2APIC MOVBE POPCNT AES PCID XSAVE OSXSAVE SEGLIM64 TSCTMR AVX1.0 RDRAND F16C
```

9.1.15 Picture 15: Confirmation of kubectl installation

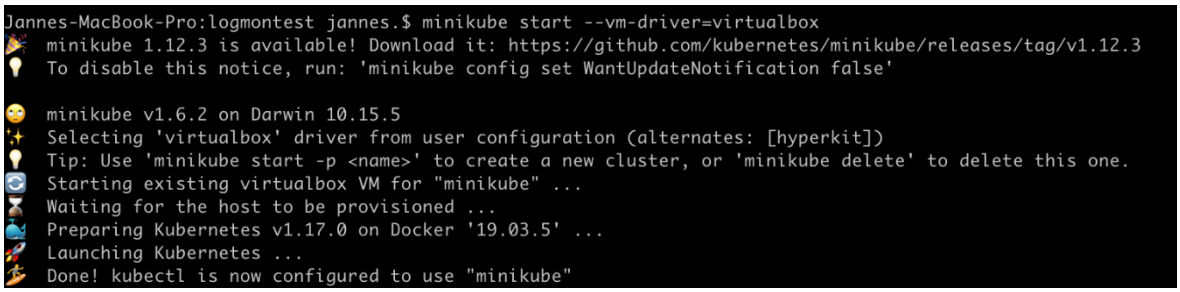


9.1.16 Picture 16: Minikube start unknown flag error



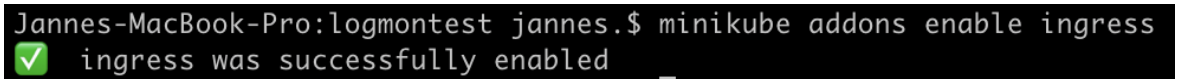
9.1.17 Picture 17: minikube start –help results

9.1.18 Picture 18: Minikube start process



9.1.19 Picture 19: Helm installation confirmation

9.1.20 Picture 20: Minikube addon ingress enabled



9.1.21 Picture 21: Minikube addons list

```
Jannes-MacBook-Pro:logmontest jannes.$ minikube addons list
- addon-manager: enabled
- dashboard: enabled
- default-storageclass: enabled
- efk: disabled
- freshpod: disabled
- gvisor: disabled
- helm-tiller: disabled
- ingress: enabled
- ingress-dns: disabled
- logviewer: disabled
- metrics-server: disabled
- nvidia-driver-installer: disabled
- nvidia-gpu-device-plugin: disabled
- registry: disabled
- registry-creds: disabled
- storage-provisioner: enabled
- storage-provisioner-gluster: disabled
```

9.1.22 Picture 22: Kubernetes namespace error for existing namespace

```
Jannes-MacBook-Pro:logmontest jannes.$ kubectl create ns staging
Error from server (AlreadyExists): namespaces "staging" already exists
```

9.1.23 Picture 23: Creating a Kubernetes namespace called deployment

```
Jannes-MacBook-Pro:logmontest jannes.$ kubectl create ns deployment
namespace/deployment created
```

9.1.24 Picture 24: Deleting a Kubernetes namespace

```
Jannes-MacBook-Pro:logmontest jannes.$ kubectl delete ns deployment
namespace "deployment" deleted
```

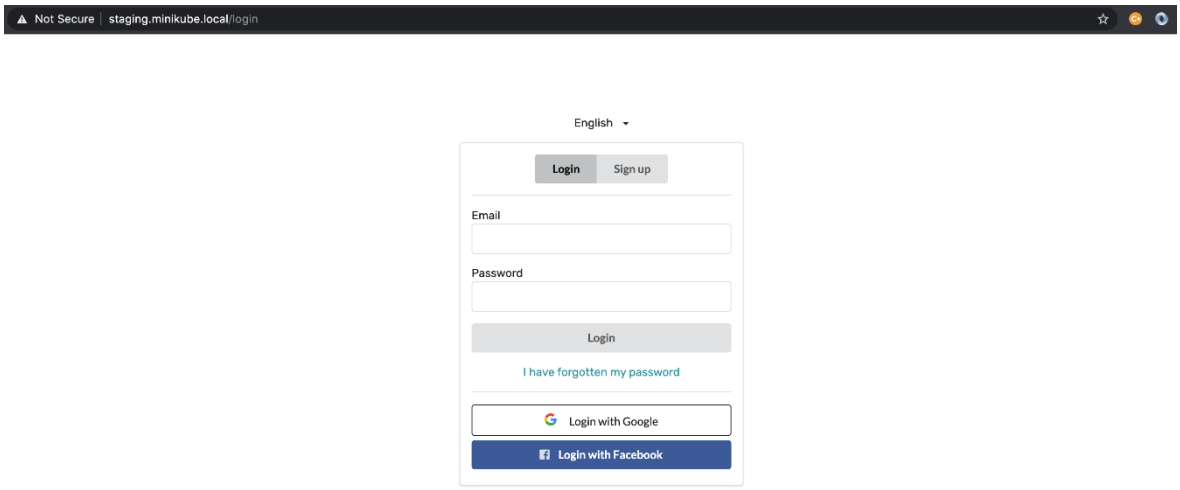
9.1.25 Picture 25: The ZDD project components running in Minikube cluster

```
NAME                                    READY   STATUS
logmontest-backend-8567d8978d-6n5mv     1/1     Running
logmontest-database-5978d5df86-j4z2t    1/1     Running
logmontest-frontend-747b844897-7c68s    1/1     Running
```

9.1.26 Picture 26: The ingress resource for backend and frontend components

```
NAME                 HOSTS
logmontest-backend   staging.minikube.local
logmontest-frontend  staging.minikube.local
```

9.1.27 Picture 27: ZDD frontend service available to access via browser



9.1.28 Picture 28: Grafana helm repo add and helm repo list –commands demonstration

```
Jannes-MacBook-Pro:logmontest jannes.$ helm repo add stable https://kubernetes-charts.storage.googleapis.com/
"stable" has been added to your repositories
Jannes-MacBook-Pro:logmontest jannes.$ helm repo list
NAME    URL
stable  https://kubernetes-charts.storage.googleapis.com/
```

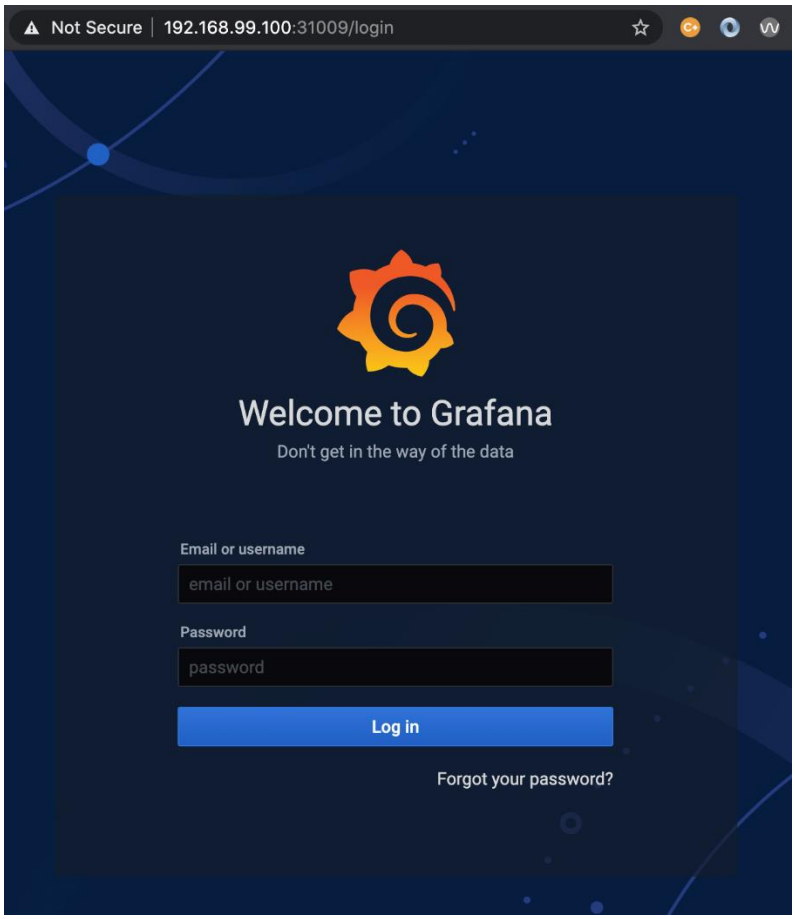
9.1.29 Picture 29: Grafana installed to staging cluster via HELM

```
NAME: grafana
NAMESPACE: staging
STATUS: deployed
REVISION: 12
NOTES:
1. Get your 'admin' user password by running:

   kubectl get secret --namespace staging grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo

2. The Grafana server can be accessed via port 3000 on the following DNS name from within your cluster:

   grafana.staging.svc.cluster.local

   Get the Grafana URL to visit by running these commands in the same shell:
export NODE_PORT=$(kubectl get --namespace staging -o jsonpath="{.spec.ports[0].nodePort}" services grafana)
     export NODE_IP=$(kubectl get nodes --namespace staging -o jsonpath="{.items[0].status.addresses[0].address}")
     echo http://$NODE_IP:$NODE_PORT


3. Login with the password from step 1 and the username: admin
#################################################################################
######   WARNING: Persistence is disabled!!! You will lose your data when   #####
######            the Grafana pod is terminated.                            #####
#################################################################################
Jannes-MacBook-Pro:logmontest jannes.$
```

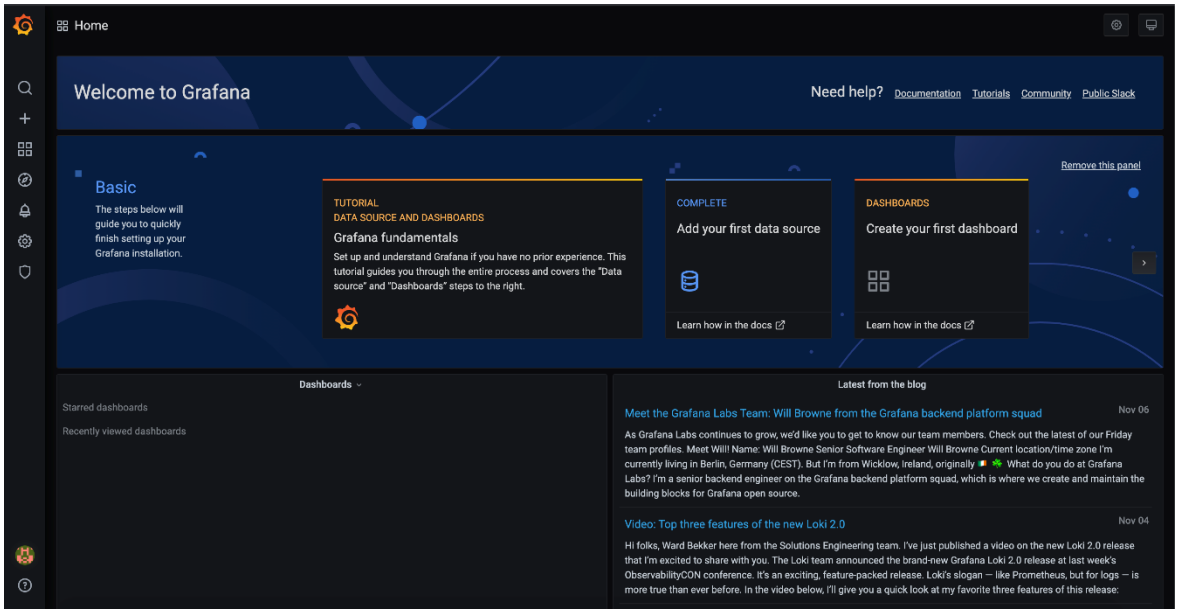
9.1.30 Picture 30: Grafana UI access CLI command results

```
Jannes-MacBook-Pro:logmontest jannes.$ export NODE_PORT=$(kubectl get --namespace staging -o jsonpath="{.spec.ports[0].nodePort}" s
ervices grafana)
Jannes-MacBook-Pro:logmontest jannes.$      export NODE_IP=$(kubectl get nodes --namespace staging -o jsonpath="{.items[0].status.a
ddresses[0].address}")
Jannes-MacBook-Pro:logmontest jannes.$      echo http://$NODE_IP:$NODE_PORT
http://192.168.99.100:31009
Jannes-MacBook-Pro:logmontest jannes.$
```

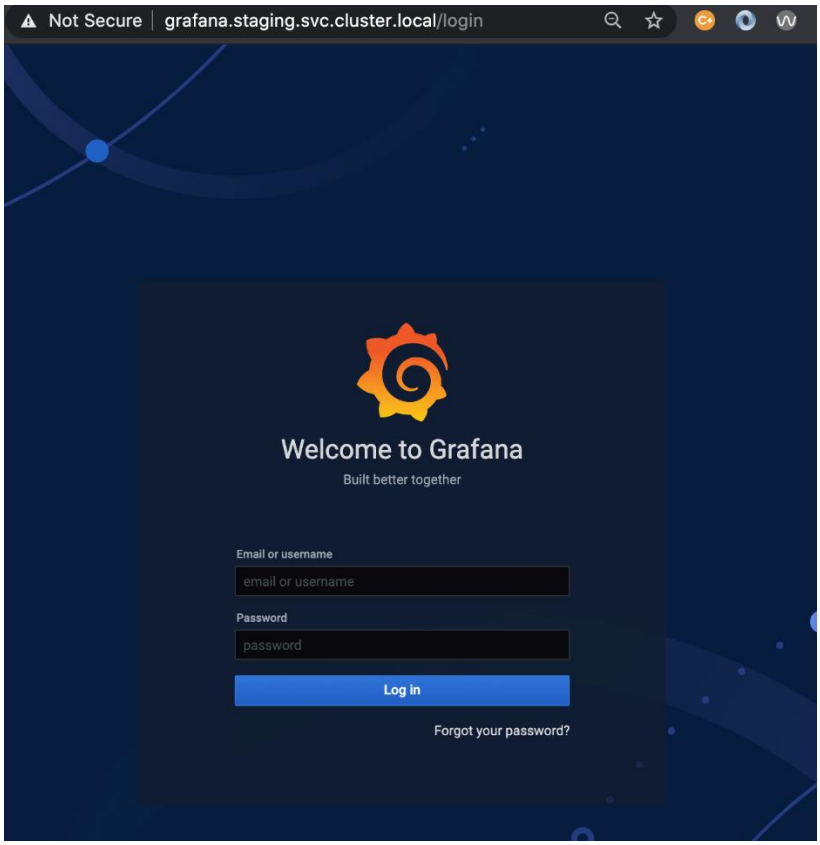
9.1.31 Picture 31: Grafana login screen



9.1.32 Picture 32: Grafana main dashboard



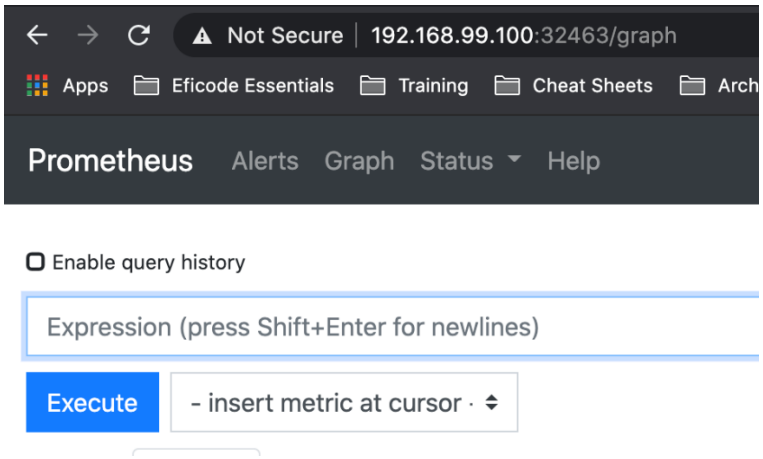
9.1.33 Picture 33: Grafana UI with ingress



9.1.34 Picture 34: Prometheus installed to staging cluster via HELM

```
NAME: prometheus
NAMESPACE: staging
STATUS: deployed
REVISION: 2
TEST SUITE: None
NOTES:
DEPRECATED and moved to <https://github.com/prometheus-community/helm-charts>The Prometheus server can be accessed via port 90 on t
he following DNS name from within your cluster:
prometheus-server.staging.svc.cluster.local


Get the Prometheus server URL by running these commands in the same shell:
  export NODE_PORT=$(kubectl get --namespace staging -o jsonpath="{.spec.ports[0].nodePort}" services prometheus-server)
  export NODE_IP=$(kubectl get nodes --namespace staging -o jsonpath="{.items[0].status.addresses[0].address}")
  echo http://$NODE_IP:$NODE_PORT


#################################################################################
######   WARNING: Pod Security Policy has been moved to a global property.  #####
######            use .Values.podSecurityPolicy.enabled with pod-based      #####
######            annotations                                               #####
######            (e.g. .Values.nodeExporter.podSecurityPolicy.annotations) #####
#################################################################################



For more information on running Prometheus, visit:
https://prometheus.io/
```

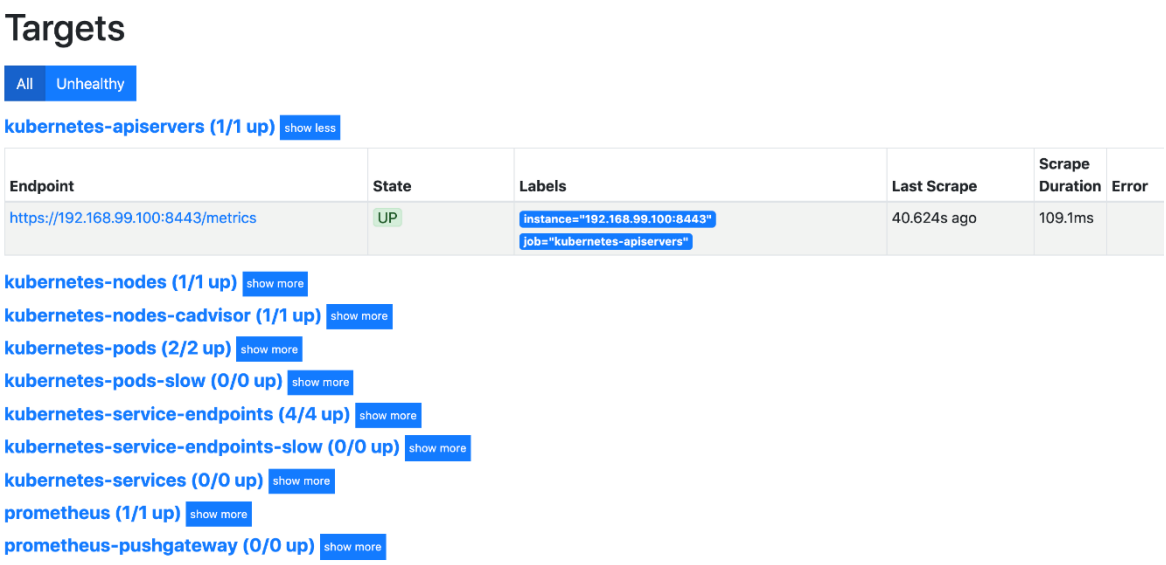
9.1.35 Picture 35: Prometheus UI running at 192.168.99.100:32463



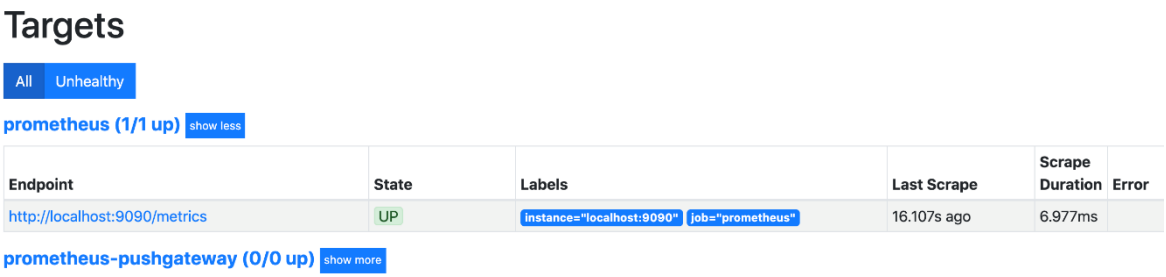
9.1.36 Picture 36: Confirmation of RBAC clusterrole, serviceaccount and cluster-rolebinding

```
clusterrole.rbac.authorization.k8s.io/prometheus configured
serviceaccount/prometheus configured
clusterrolebinding.rbac.authorization.k8s.io/prometheus configured
```

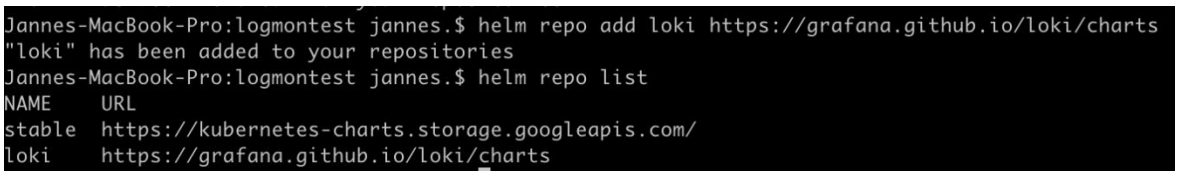
9.1.37 Picture 37: Prometheus Targets with RBAC



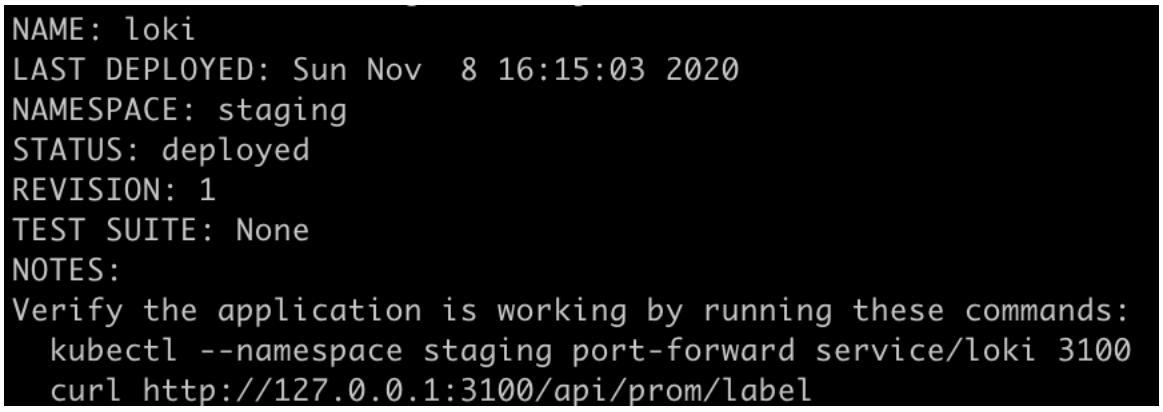
9.1.38 Picture 38: Prometheus Targets without RBAC



9.1.39 Picture 39: Loki helm repo add & helm repo list demonstration



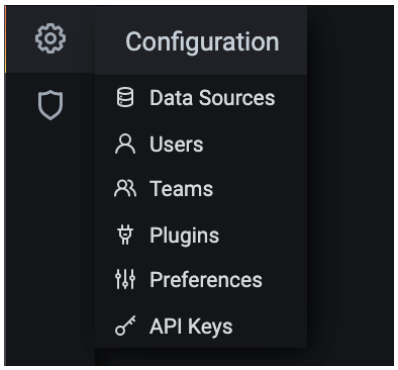
9.1.40 Picture 40: Loki helm chart installation confirmation



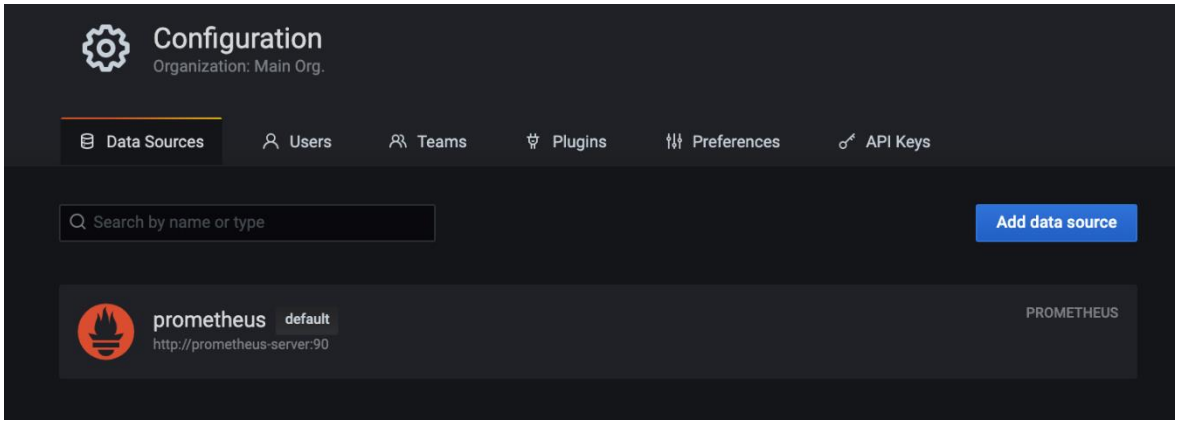
9.1.41 Picture 41: Loki curl response

```
Jannes-MacBook-Pro:logmontest jannes.$ curl http://loki.minikube.local/api/prom/label
{"values":["__name__"]}
```

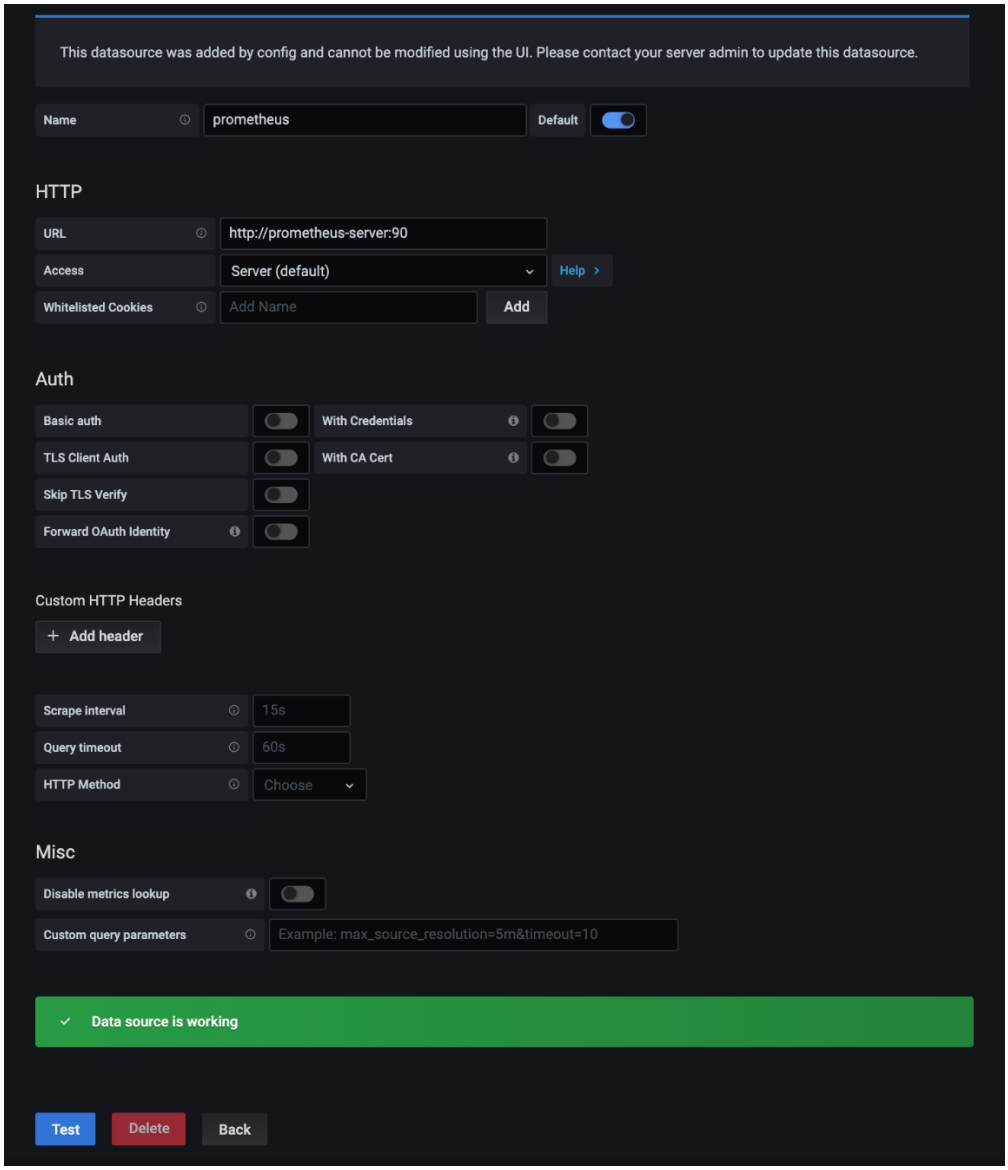
9.1.42 Picture 42: Accessing Grafana data sources (Grafana, 2020k.).



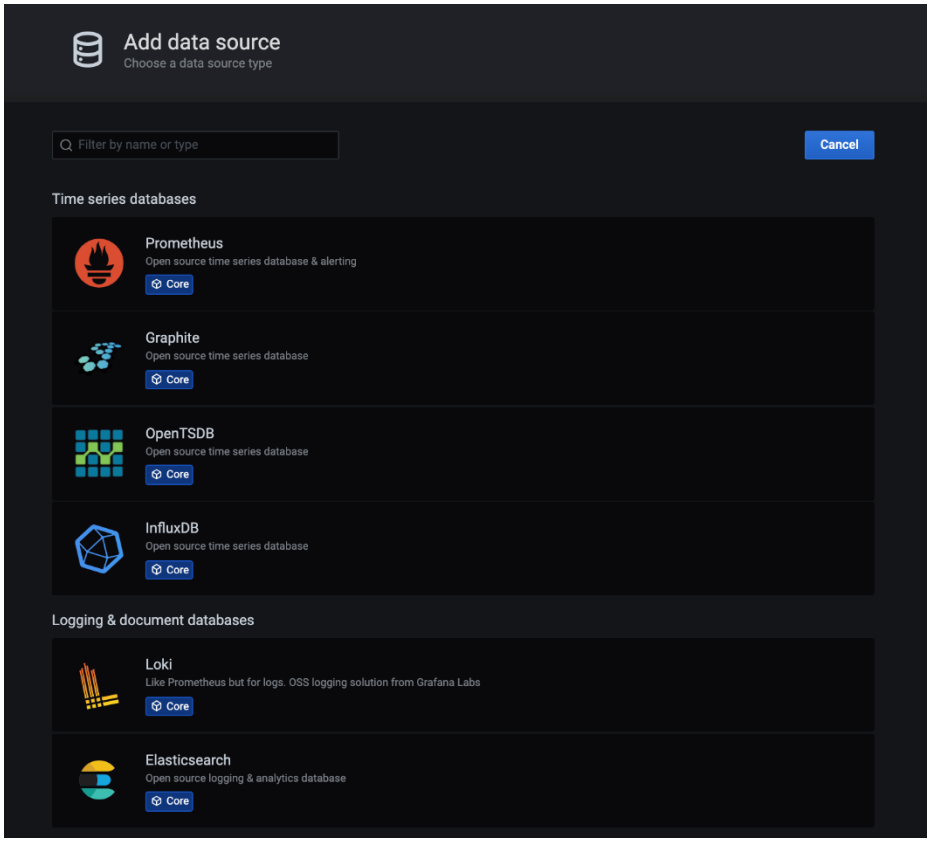
9.1.43 Picture 43: Grafana data sources page (Grafana, 2020k.).



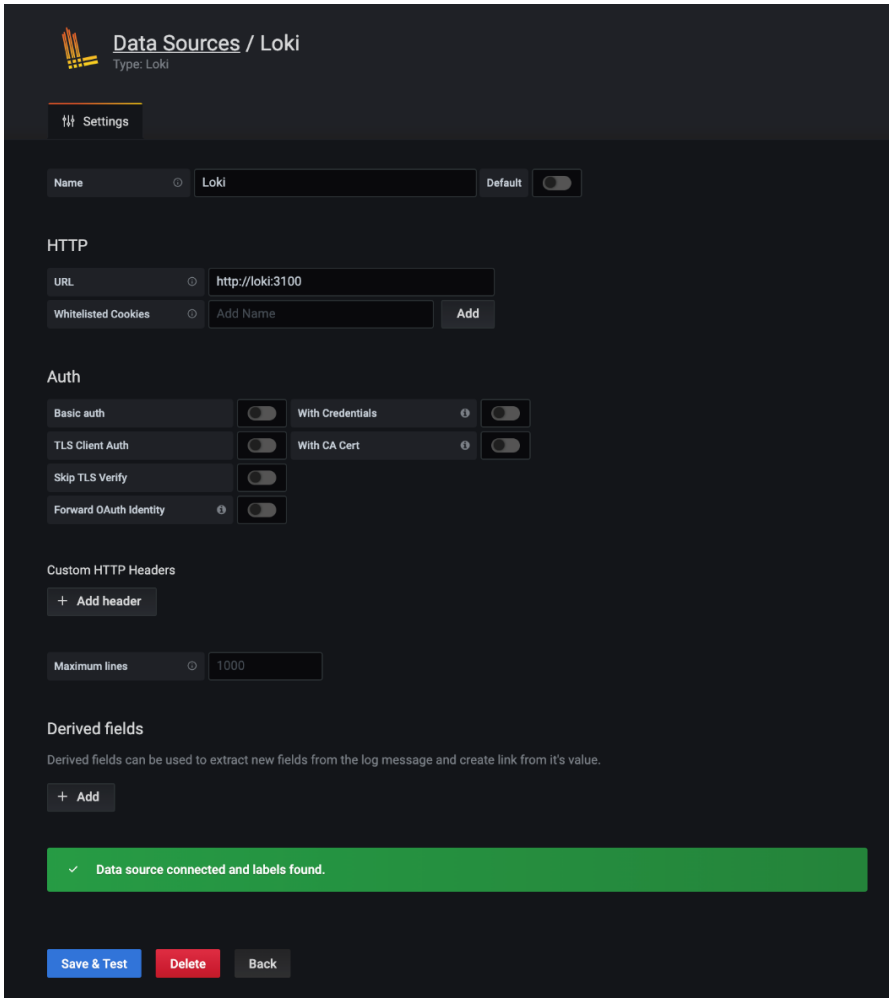
9.1.44 Picture 44: Prometheus data source settings in Grafana



9.1.45 Picture 45: Add data source page in Grafana



9.1.46 Picture 46: Grafana Loki data source in Grafana



9.2 Tables

9.2.1 Table 1: Core list of log uses (Kavis 2014, p. 119 – 120)

Log usage	Description
Troubleshooting	“Debugging information and error messages are collected for analysing what is occurring in the production environment.”
Security	“Tracking all user access, both successful and unsuccessful access attempts. Intrusion detection and fraud detection analysis depends on collecting logs.”
Auditing	“Providing a trail of data for auditors is mandatory for passing audits. Having documentation of processes and controls is not enough to pass an audit. Those documents need to be backed up with real data from the logs.”
Monitoring	“Identifying trends, anomalies, thresholds and other variables proactively allows companies to resolve issues before they become noticeable and before they impact end users.”

9.2.2 Table 2: RFC 5424 Syslog protocol (Kavis 2014, p. 123; IETF, 2020.)

Code	Severity
0	Emergency, system is unusable
1	Alert: action must be taken immediately
2	Critical: critical conditions
3	Error: error conditions
4	Warning: Warning conditions
5	Notice: normal, but significant conditions
6	Informational: informational messages
7	Debug: debug-level message

9.2.3 Table 3: Log message classification (Chuvakin, Schmidt & Phillips 2013, p. 34)

Classification	Description
Informational	“Messages of this type are designed to let users and administrators know that something benign has occurred.”
Debug	“Debug messages are generally generated from software systems in order to aid software developers troubleshoot and identify problems with running application code.”
Warning	“Warning messages are concerned with situations where things may be missing or needed for a system.”
Error	“Error log messages are used to relay errors that occur at various levels in a computer system”
Alert	“An alert is meant to indicate that something interesting has happened.”

9.2.4 Table 4: HTTP Status Codes (RESTfulAPI, 2020)

Category	Description
1xx: Informational	“Communicates transfer protocol-level information”
2xx: Success	“Indicates that the client’s request was accepted successfully.”
3xx: Redirection	“Indicates that the client must take some additional action in order to complete their request.”
4xx: Client Error	“This category of error status codes points the finger at clients.”
5xx: Server Error	“The server takes responsibility for these error status codes.”

9.2.5 Table 5: CSP (Cloud Service Provider) logging solutions (Kavis 2014, p. 120 – 122)

Solution	Description
Infrastructure as a Service (IaaS)	All logs are directed to syslog instead of being written directly to disk on the local machine. Syslog on each server is piped directly to a dedicated logging server farm, based on CSP's (Cloud Service Providers) infrastructure.
Software as a Service (SaaS)	"The logs are sent to cloud-based centralized logging Database as a Service provided by a CSP (Cloud Service Provider)
Platform as a Service (PaaS)	Many PaaS solutions are integrated with the most popular logging SaaS solutions which provide API access to the central logging solution provided by a CSP (Cloud Service Provider)

9.2.6 Table 6: Basic metric types (Turnbull 2016, p. 29 - 30)

Metric type	Description
Gauges	Most common type of metric. Gauges are numbers that are expected to change over time. A snapshot of a specific measurement.
Counters	Counters are number over time and never decrease.
Timers	E.g. tracks how long it took for a function or a method to complete.

9.2.7 Table 7: Metric summaries (Turnbull 2016, p. 31 - 32)

Summary type	Description
Count	Count the number of observations in a specific time interval.
Sum	Adding together (sum) values from all observations in a specific time interval.
Average	The mean (average) of all values in a specific time interval.
Median	The exact centre of values.
Percentiles	Measures the values below, which a given percentage of observations in a group of observations fall.
Standard Deviation	Distribution of metrics used to measure variation in a data set by setting a range of values.
Rates of Change	Representation to show the degree of change between data in a time series.
Frequency distribution	Grouping of selected data set and visualizing the frequency of a group.

9.2.8 Table 8: The monitoring maturity types (Turnbull 2014, p. 12 – 17)

Stages of maturity	Description
Manual, user-initiated or no monitoring	Monitoring provides little or no value.
Reactive	Mostly automated with some remnants of manual or un-monitored components.
Proactive	Monitoring is automated, considered as a core component for managing infrastructure and business.

9.2.9 Table 9: Metrics categories (Kavis 2014, p. 139 – 145)

Category	Description
Performance	Works within each layer of the cloud stack. Measures and tracks the behaviour of the users (or customers) interacting with the application, measures how the application responds to the actions and track the performance of the underlying components (operating system) and infrastructure (network, servers).
Throughput	Works within each layer of the cloud stack. Measuring data flow through the cloud stack, from layer to layer per unique user. Throughput is usually measured as TPS (Transactions per second). Critical for diagnostics.
Quality	Works in the user and application layer. Quality measurements require standardisation of data collection (e.g. Logs). Measures the success and accuracy of user registration and access. Provides metrics for alerting systems and reports.
KPIs (Key performance indicators)	Works in the application layer. Provide metrics for business analysis e.g. site traffic, shopping cart abandonment rate.
Security	Should work within each layer of the cloud. Provides metrics for application security issues e.g. mining log files and discovering abnormal patterns e.g. unique IP address trying to access application multiple times in a short time span.
Compliance	For applications that fall under regulation constraints. Requires policies and procedures to follow the application and business e.g. creating a policy to restrict application production access for certain developers and raise alerts if policies are broken.

9.2.10 Table 10: ISO/IEC 27001 annex A.12.4 objectives (ISO/IEC 27001:2013 Annex A.12.4.)

Annex	Description
A.12.4.1 Event Logging	"Event logs recording user activities, exceptions, faults and information security events shall be produced, kept and regularly reviewed."
A.12.4.2 Protection of Log Information	"Logging facilities and log information shall be protected against tampering and unauthorized access."
A.12.4.3 Administrator & Operator Logs	"System administrator and system operator activities shall be logged, and the logs protected and regularly reviewed."

9.2.11 Table 11: TLS Components (Cloudflare Inc, 2020a)

Component	Description
Encryption	"Hides data being transferred from third parties."
Authentication	"Ensures that the parties exchanging information are who they claim to be."
Integrity	"Verifies that the data has not been forged or tampered with."

9.2.12 Table 12: NodeJS logging use cases (Dominik Kundel 06.05.2019)

Category	Description
Debugging	For unexpected behaviour during development
Browser-based logging	For analytics or diagnostics
Server application logging	For logging incoming requests and any failures that might have happened.
Library debugging	Debugging library related issues and behaviour to assist the user with issues.
CLI output	Printing to CLI (Command-line interface) the progress, confirmation messages or errors.

9.2.13 Table 13: Prometheus: PromQL data types (Prometheus, 2020c.)

Category	Description
Instant vector	“A set of time series containing a single sample for each time series, all sharing the same timestamp.”
Range vector	“A set of time series containing a range of data points over time for each time series.”
Scalar	“A simple numeric floating-point value”
String	“A simple string value; currently unused”

9.3 Code Snippets

9.3.1 Code snippet 1: Cloning the ZDD repository

```
git clone --depth 1 https://<eficode bitbucket>/node-react-project.git
cd node-react-project
npm i
cd ..
```

9.3.2 Code snippet 2: Starting setup of ZDD template

```
Jannes-MacBook-Pro:code jannes.$: create-efi-project
Starting setup of a new Node + React project.
Please enter project name. Must be lowercase and one word, and may contain
hyphens and underscores. Example: my-project
Project Name: logmontest
Enter the name of the author. Example: John Smith <john.smith@email.com>
Project author: Janne Saikkonen <janne.saikkonen@email.com>
Enter the Docker repository URI. For example: artifactory.dev/project/
Docker repository: <Eficode Docker repository URI>
Enter public address of the service. Example: your.service.com or
123.123.123.123
Public address: <left blank>
Setting up project logmongtest…
Create the project directory logmontest and populated it with project files.
Created package.json, frontend/package.json, backend/package.json, build.sh,
push.sh, deploy.sh, .env and .deploy.env based on the provided information.
Development QUICKSTART: To run the project, run “docker-compose up” in the
project directory and visit http://localhost:8000/ in the browser.
Production QUICKSTART: Use the build.sh, push.sh and deploy.sh scripts to
deploy to Kubernetes. See README.md for more information how to prepare cluster
and enable continous development in Jenkinsfile.

All done! Happy hacking!
```

9.3.3 Code snippet 3: Authenticate for Artifactory

```
kubectl create secret docker-registry img-pull-secret \
--docker-server=<Eficode Artifactory Address> \
--docker-username=<Eficode RTM Bot Username> \
--docker-password=<Eficode RTM Bot Password> \
--docker-email=<Eficode RTM Bot email> \
--namespace=staging
```

9.3.4 Code snippet 4: values-grafana.yaml HELM values configuration (Eficode Academy Github, 2020.)

```
service:
  type: NodePort
  port: 3000
rbac:
  namespaced: true
  pspEnabled: false
  pspUseAppArmor: false
resources:
  limits:
    cpu: 250m
    memory: 256Mi
  requests:
    cpu: 250m
    memory: 128Mi
datasources:
  datasources.yaml:
    apiVersion: 1
    datasources:
    - name: prometheus
      type: prometheus
      access: proxy
      url: http://prometheus-server:90
      isDefault: true
      editable: false
```

9.3.5 Code snippet 5: Grafana UI access CLI command

```
export NODE_PORT=$(kubectl get --namespace staging -o
jsonpath="{.spec.ports[0].nodePort}" services grafana)
export NODE_IP=$(kubectl get nodes --namespace staging -o
jsonpath="{.items[0].status.addresses[0].address}")
echo http://$NODE_IP:$NODE_PORT
```

9.3.6 Code snippet 6: Grafana ingress configuration (Kubernetes, 2020c)

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress-grafana
spec:
  rules:
  - host: grafana.staging.svc.cluster.local
    http:
      paths:
      - path: /
        backend:
          serviceName: grafana
          servicePort: 3000
```

9.3.7 Code snippet 7: values-prometheus.yaml HELM values configuration (Efficode Academy Github, 2020)

```
rbac:
  create: true
serviceAccounts:
  server:
    create: true
    name: user-x-sa
alertmanager:
  enabled: false
pushgateway:
  enabled: false
networkPolicy:
  enabled: false
nodeExporter:
  enabled: true
kubeStateMetrics:
  enabled: true
server:
  retention: "1d"
  resources:
    limits:
      cpu: 250m
      memory: 512Mi
    requests:
      cpu: 250m
      memory: 256Mi
  service:
    type: NodePort
    servicePort: 90
```

9.3.8 Code snippet 8: Prometheus Ingress configuration (Kubernetes, 2020c.)

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress-prometheus
spec:
  rules:
  - host: prometheus-server.staging.svc.cluster.local
    http:
      paths:
      - path: /
        backend:
          serviceName: prometheus-server
          servicePort: 90
```

9.3.9 Code snippet 9: First part of Prometheus RBAC (Role-Based Access Control) config (Prometheus Github, 2020)

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus
rules:
- apiGroups: [""]
  resources:
  - nodes
  - nodes/metrics
  - services
  - endpoints
  - pods
  verbs: ["get", "list", "watch"]
- apiGroups:
  - extensions
  resources:
  - ingresses
  verbs: ["get", "list", "watch"]
- nonResourceURLs: ["/metrics"]
  verbs: ["get"]
```

9.3.10 Code snippet 10: Second part of Prometheus RBAC (Role-Based Access Control) config (Prometheus Github, 2020)

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: prometheus
  namespace: staging
```

9.3.11 Code snippet 11: Third part of Prometheus RBAC (Role-Based Access Control) config (Prometheus Github, 2020)

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: prometheus
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus
subjects:
- kind: ServiceAccount
  name: prometheus
  namespace: staging
```

9.3.12 Code snippet 12: values-loki.yaml HELM values configuration

```
rbac:
  create: false
  pspEnabled: false
service:
  type: ClusterIP
  port: 3100
```

9.3.13 Code snippet 13: Grafana Loki ingress resource configuration (Kubernetes, 2020c.)

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress-loki
spec:
  rules:
  - host: loki.minikube.local
    http:
      paths:
      - path: /
        backend:
          serviceName: loki
          servicePort: 3100
```