ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
FACULTY OF INFORMATICS

THESIS

OE-NIK	Student's name:	**Boglárka Tóth**
2023	Student's registration number:	**T/006672/FI12904/N**

Óbudai University
John von Neumann Faculty of Informatics
Institute for Cyber-physical Systems

THESIS WORK SPECIFICATION

Name of the student:	**Boglárka Tóth**
Identification number of the student:	T/006672/FI12904/N
Neptun's code:	[redacted]

Title of the thesis:

Development of a reference architecture to support multiple data source visualization
Többféle adatforrást támogató vizualizációs referencia architektúra fejlesztése

Internal supervisor	Attila Farkas
External supervisor:	Attila Csaba Marosi Ph.D.
Deadline:	15 May 2023
Subjects of the final exam:	Computer Architectures Cloud service technologies and IT security specialization

The task:

In today's connected world data is everywhere. Contrary to a decade ago the challenge is not finding data, but rather evaluating and gathering insights into the data that has ever-increasing volume, velocity, variety, and veracity. Visualization provides a key capability to understand data and that's why there is an abundance of visualization tools available. However, selecting the tool based only on the requirement of the right visualization capabilities is not enough, as it must integrate well into the architecture that is responsible for storing and processing the data. So-called reference architectures solve this by providing reusable building blocks that incorporate the best practices and know-how for a given functionality (e.g., data storage, processing, or visualization).
The goal of this work is to determine the requirements for visualization of time-series data originating from different data sources (e.g., MQTT or Apache Kafka) and data formats (e.g., binary or JSON). Based on the identified requirements different open-source visualization tools should be evaluated. A requirement is that for future use cases, the tools should be extendable, to handle more sources like image data (e.g., camera images). Using the results of the evaluation a reference architecture should be developed and evaluated using data from the selected data sources.

The thesis must contain:

- identification of typical data sources and formats in modern cloud-agnostic data analytics platforms,
- selection of at least 2 data sources and formats,
- comparison and evaluation of different open-source visualization tools,
- design of a reference architecture that can visualize data from different sources and data formats in parallel,
- implementation of the data visualization platform based on cloud and container-based tools,
- evaluation of the reference architecture,
- limitations and future works.

Place of stamp

ÓBUDAI EGYETEM * Neumann János Informatikai Kar 1. Budapest

.....
Dr. Rita Fleiner
head of institute

This specification is valid until: **15 May 2025**
(According to OE TVSz 55.§)

I consider the thesis suitable for submission:

.....
external supervisor

.....
internal supervisor

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
FACULTY OF INFORMATICS

STUDENT'S DECLARATION

I, the undersigned student, hereby declare that this thesis / thesis work is the result of my own work; I have disclosed the references and tools used in an identifiable manner. The results indicated in my thesis / thesis work completed may be used by the university and the institution announcing the task for their own purposes free of charge.

Dated: Budapest, 13 December 2022

.....

student's signature

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
FACULTY OF INFORMATICS

CONSULTATION LOG

Student's name: Boglárka Tóth.....
Neptun code: [redacted].....
Specialty: Full time.....
Phone number: [redacted]
Mailing address (e.g. domicile): [redacted]

Thesis / thesis work[1] title in Hungarian:
Többféle adatforrást támogató vizualizációs referencia architektúra fejlesztése
Thesis / thesis work[2] title in English:
Development of a reference architecture to support multiple data source visualization
Internal supervisor: Attila Farkas.....
External supervisor: Attila Csaba Marosi Ph.D.....

Please write the data in printed capital letters!

Occ.	Date	Content	Signature
1.	7 Sept. 2022	Discussion of chosen topic	Farkas Attila
2.	12 Sept. 2022	Planning of literature research	Farkas Attila
3.	19 Sept. 2022	Review of literature research	Farkas Attila
4.	3 Oct. 2022	Discussion of system design	Farkas Attila

The consultation log is required to be countersigned by either supervisor on a total of 4 occasions of consultation.
The student has complied with the requirements of the subject titled Thesis I / Thesis II (BSc) or Thesis work I / Thesis work II / Thesis work III / Thesis work IV[3], (MSc) and is allowed to proceed for reporting / defense[4].

Dated: Budapest, 24 November 2022

Farkas Attila
.....
internal supervisor

[1]Underline as appropriate
[2]Underline as appropriate
[3]Underline as appropriate
[4]Underline as appropriate

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
FACULTY OF INFORMATICS

CONSULTATION LOG

Student's name: Boglárka Tóth
Neptun code: [redacted]
Specialty: Full time
Phone number:
Mailing address (e.g. domicile): [redacted]

Thesis / thesis work[1] title in Hungarian:
Többféle adatforrást támogató vizualizációs referencia architektúra fejlesztése
Thesis / thesis work[2] title in English:
Development of a reference architecture to support multiple data source visualization
Internal supervisor: Attila Farkas
External supervisor: Attila Csaba Marosi Ph.D.

Please write the data in printed capital letters!

Occ.	Date	Content	Signature
1.	5 Oct. 2022	Review of system design	Farkas Attila
2.	10 Oct. 2022	Planning of implementation	Farkas Attila
3.	3 Nov. 2022	Overview of implementation	Farkas Attila
4.	15 Nov. 2022	Overview of results, discussion of further development opportunities	Farkas Attila

The consultation log is required to be countersigned by either supervisor on a total of 4 occasions of consultation.
The student has complied with the requirements of the subject titled Thesis I / Thesis II (BSc) or Thesis work I / Thesis work II / Thesis work III / Thesis work IV[3], (MSc) and is allowed to proceed for reporting / defense[4].

Dated: Budapest, 24 November 2022

Farkas Attila
.....
internal supervisor

[1] Underline as appropriate
[2] Underline as appropriate
[3] Underline as appropriate
[4] Underline as appropriate

Contents

ABSTRACT

As our world is changing faster than ever before, the quantity of data continues to grow. One of the reasons for the information explosion is the technology advancement. Time series data has grown more common since the development of IoT. They appear in a variety of significant sectors of human endeavor, including finance, economics, engineering, and natural sciences, to mention a few. Time series data analysis is also becoming increasingly important. As more data is generated, collected, and stored in data centers around the world, it is more critical than ever to be able to swiftly evaluate and comprehend what the data is telling us so that we can make informed decisions.

This is where visualization of data comes in. People understand graphics and other images better than lines of written text, numbers, and statistics. So, in order to more effectively interpret and explain what our data means, we should present it visually so that everyone can easily process it and make better conclusions.

Working with multiple data sources means that the data visualization platform should be capable of efficiently processing and storing any type of data. The architecture should be easily expandable in the future to incorporate more functionality and data sources.

The aim of this research study is to investigate potential solutions for creating such an architecture and to design its efficient implementation.

1. INTRODUCTION TO DATA

The expressions *"data"* and *"information"* are frequently used as synonyms; however, they are not equivalent. There are fundamental distinctions between these concepts and their utilization. As this research deals with data visualization, it is absolutely necessary for us to understand the key differences between data and information.

1.1 The definition of data

Data is a raw kind of knowledge that has no meaning or function on its own. Simply put, data is a collection of discrete facts, numbers, or statistics. Each individual piece of data is a simple fact that does not mean a lot on its own. The term data can refer to either a single fact or a group of facts. The noun data is derived from the Latin word *datum*, that means *"something given"*. Although datum is the singular form of data, it is rarely used in everyday speech [1].

1.2 The definition of information

Simply said, information is news or knowledge received or given. It is the result of processing, interpreting, and organizing facts in a meaningful way in accordance with the given demand. In a simple term, information is data that now has meaning attached to it. The term is derived from the Latin word *īnfōrmātiō*, which means *"to form an idea, conceive"*.

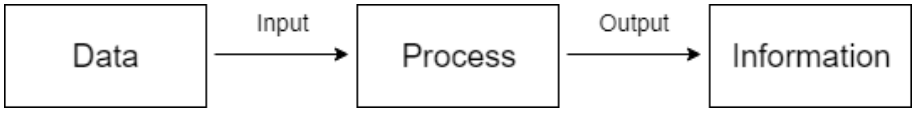

Figure 1.1: The data processing cycle

Data must be contextualized before it can be considered as information as Figure 2.1 shows this process. For example, a collection of data could contain precipitation readings from a certain region over a period of time. Those values have little meaning without any extra content. However, by analyzing and organizing that data, we may be able to discover seasonal rainfall tendencies or even broader climate trends. Only when data is organized and processed in a meaningful way can it provide relevant knowledge to others.

1.3 The main differences between data and information

To come to the point, data is a collection of facts, whereas information contextualizes those facts. Information can be structured or organized, whilst data is unorganized. Data on its own is rarely meaningful, but information is. Graphs, numbers, figures, or statistics are common ways in which data is presented. To communicate information, words, language, thoughts, and ideas are commonly used. Data never relies on information, rather information relies on data. Data alone is insufficient for decision-making, but information can be used to make decisions.

In the digital world, data is the input, or what we command the computer to perform or store. The information is the output, or how the computer processes our data and displays the specified action or instruction.

1.4 Classification of data

There are numerous ways for categorizing data. It can be divided up into two categories; qualitative or quantitative, depending whether it is written or numerical. We can also categorize data as primary or secondary based on its source.

1.4.1 Qualitative data

Qualitative data describes characteristics or attributes. It is descriptive but non-numerical, such as a person's name, your favourite food, or how you would describe smells. Qualitative data can be difficult to quantify and evaluate correctly.

1.4.2 Quantitative data

Quantitative data is presented numerically, such as an item's height, or the number of students in a classroom. It is data that can be quantified or compared on a numerical scale. Quantitative data can be discrete or continuous. Discrete data can only take certain values like whole numbers, while any value can be assigned to continuous data within a specified range, with the highest and lowest values being the most extreme.

1.4.3 Primary data

Primary data refers to data that was collected for the first time for a certain reason. It has not been subjected to any statistical processing yet and is completely original.

1.4.4 Secondary data

Secondary data refers to data that has been obtained from somewhere where it was initially collected by someone else for a specific reason but is now being used again for another. This signifies that this type of data has already been collected by some researchers and statistical computations may have previously been done on it.

1.5 Data types

Data is everywhere, and it comes in a variety of forms. Some of the most common types of data are as follows:

- **A single character:** a single visual object used to represent text, numbers, or symbols.
- **Boolean:** represents the logical values true and false.
- **Text:** a string of characters that is used to store words or simple text.
- **Number:** can represent whole numbers (integers) or fractional values (floating-point numbers).
- **Picture:** a visual representation of an entity.
- **Sound:** refers to a vibration detected by the human ear.
- **Video:** a continuous source of still images that simulates movement.

2. DATA SOURCES

A data source is either a physical or a digital location from which data can be accessed. A database, a plain text file, a JSON file, a streaming data service or really any format that a computer can interpret could serve as the source. The input is stored as a set of records containing the data used in the workflow.

2.1 Data formats

Computers store data as binary values, namely in a sequence of bits with the value one or zero. A bit is the smallest unit of data that can represent a single value. Furthermore, a byte is eight bits long. Memory and storage capacity is measured in megabytes, gigabytes, terabytes, petabytes, and exabytes.

Despite the fact that everything is stored as bits, different data types require specific file formats. A file format is a standard method of encoding data to be stored in a computer file. Certain file formats are intended for specific kinds of data: PNG files, for instance, store bitmapped images utilizing lossless data compression.

2.2 Data files

There are many different sorts of data files, but for this project, we are only going to focus on two formats, namely XML and JSON. Both of them are readable by humans, because they are self-describing. Many different programming languages can use them as data interchange formats. Data can be obtained from a web application using either of them because both XML and JSON can be easily parsed by a variety of programming languages. HTTP requests like POST, GET and PUT can be used to fetch either format.

2.2.1 JSON

JSON or JavaScript Object Notation is a data format that is language-independent. It supports nearly every programming language, framework and library. It is very simple to read and write data with JSON, because it supports data structures such as objects and arrays. Data is represented as key-value pairs in JSON, and curly braces store objects with a colon following each name.

The following code is an example on how JSON is structured:

```
{
  "people":{
    "person":[
         {
          "id": 1,
          "surname": "Toth",
          "forename": "Boglarka"
       },
         {
          "id": 2,
          "surname": "Gyenes",
          "forename": "Eniko"
       }
    ]
  }
}
```

2.2.2 XML

XML or Extensible Markup Language is a markup language for representing data. It is similar to HTML tags, but XML does not have predefined tags, we have to define our own tags that are personalized for our requirements. XML is independent of platform and language but it has a standardised syntax, thus if you exchange or transmit XML across systems or platforms, whether locally or online, the recipient can still parse the data.

Consider the following code as an example for how XML is structured.

```
<?xml version="1.0" encoding="utf-8"?>
<people>
    <person id="1">
        <surname>Toth</surname>
        <forename>Boglarka</forename>
    </person>
    <person id="2">
        <surname>Gyenes</surname>
        <forename>Eniko</forename>
    </person>
</people>
```

2.3 Databases

A database is a structured collection of information or data that is stored digitally in a computer. Most frequently data is structured in rows and columns in a set of tables to make processing and data querying easier. The data within the database can be easily read, updated and organized. The majority of databases utilize Structured Query Language (SQL) for querying data. There are various kinds of databases, however, this thesis paper will only address two of these types that relates to this thesis project. Let's review them:

2.3.1 Relational databases

The most frequent type of databases is the relational database. It stores data in rows and columns in tables. Each row is a record with a unique ID called primary key. Each table represents a certain object like products or people. Relational means that each record can relate to another record in another table with the help of primary or foreign keys.

Some popular examples of SQL databases are MySQL, MSSQL, PostgreSQL, IBM DB2 and MariaDB.

2.3.2 Time series databases

A time series database (TSDB) is a database designed to handle time-stamped or time series data, which is a collection of measurements or events collected over time. These data include metrics, performance monitoring, sensor data, events and a variety of other analytics data. Time-series data allows us to monitor changes.

There are numerous time series databases, including InfluxDB, Graphite, Prometheus and MongoDB.

2.4 Streaming data services

Streaming data technologies (also known as event stream processing) are gaining popularity as the demand for faster analytics and customer insights rises. It is basically the continual stream of data produced by a variety of sources. For example, data from GPS and IoT sensors can be collected, processed, stored and analyzed in real time. It is not surprising then that real-time stream processing can be important for example in the aviation industry, so airlines can predict aircraft maintenance, as well as detect flight anomalies in real-time.

There are more approaches to stream processing, such as distributed real-time computation systems, distributed streaming data-flow engines, and distributed publish-subscribe messaging systems such as Apache Kafka and MQTT. This thesis paper is going to discuss publish-subscribe messaging systems.

The Publish/Subscribe pattern, often abbreviated as pub/sub, is an architectural design pattern that offers a framework for publishers and subscribers to exchange messages. This pattern requires the use of a broker, otherwise known as a server. All of the clients will interact with the broker. The publisher is the client that transmits a message through the broker. The broker then filters these incoming messages and delivers them to the clients that are subscribed to this type of message just published. Subscribers are the clients that subscribe to topics in which they are interested to the broker. As a result, subscribers and publishers do not establish connection directly with each other.

2.4.1 MQTT protocol

MQTT is a messaging protocol that was originally developed by IBM in 1999, but later OASIS has taken over the maintenance of the standard. The official website of MQTT describes the protocol as ”an extremely lightweight publish/subscribe messaging transport that is ideal for connecting remote devices with a small code footprint and minimal network bandwidth. MQTT today is used in a wide variety of industries, such as automotive, manufacturing, telecommunications, oil and gas, etc. [2]”

MQTT utilizes a message broker, and several clients. Clients can publish, subscribe and unsubscribe. Publishers transmit messages on a given topic to the MQTT broker, which will forward them to the subscribed clients. MQTT is bi-directional, which means that a client can be both a publisher and a subscriber. If the broker loses the connection with a subscriber, then the broker will buffer the messages and send them to the subscriber when the connection is restored. If it loses the connection with a publisher, the broker can close the connection and transmit a cached message called Last Will and Testament (LWT) to the subscribers containing instructions from the publisher.

MQTT has 3 different levels of QoS (Quality of Service) to ensure message delivery: QoS 0 that is the default and does not ensure message delivery, QoS 1 which ensures message delivery but may result in duplication, and QoS 2 that ensures that no duplicate messages are delivered.

There are various MQTT brokers in the market, some open-source examples are as follows: Eclipse Mosquitto, Mosca, RabbitMQ, Apache ActiveMQ and EMQ.

2.4.2 Apache Kafka

Apache Kafka is publish-subscribe messaging implemented as a distributed commit log, suitable for both offline and online message consumption. It is a messaging system initially developed at LinkedIn for collecting and delivering high volumes of event and log data with low latency [3].

Kafka is run as a cluster that contains one or more brokers. This cluster is responsible for keeping a partition for scaling, concurrency, and error-tolerance for each topic. Each

partition is a sorted, consistent sequence of messages that is appended to a commit log on a regular basis. The offset is a consecutive id number assigned to each message in the partitions. The Kafka system is made up of 4 major components: a broker that handles all client requests and ensures that data is replicated in the cluster, a ZooKeeper which maintains the cluster's status, a producer that publishes messages to the broker, and a consumer that subscribes to topics and consumes messages.

The typical consumer will process the next message in the list, although it can consume messages in any order, as the Kafka cluster retains all published messages for a configurable period of time. Producers are able to choose which topic, and which partition within the topic, to publish the message to. Consumers assign themselves a consumer group name, and each message is delivered to one consumer within each subscribing consumer group [3].

Kafka is an event streaming platform, not an IoT platform. It delivers a scalable, efficient, and elastic real-time platform for messaging, storage, data integration, and stream processing. It is being used by famous companies such as LinkedIn, Twitter and Foursquare.

3. DATA VISUALIZATION

Data visualization is the process of creating and customizing visual representations of data in order to acquire insights and communicate information. Users can engage with data in a variety of ways, including analyzing data patterns, finding trends, and making comparisons, using data visualization tools.

There are numerous data visualization tools available, each with its own set of benefits and drawbacks. Some of these tools will be discussed in further detail later.

3.1 Data visualization techniques

Choosing what type of data visualization to implement is part of the strategy of visualizing data. The trick is to choose the one that best reflects the message and story of our data. Data visualization can take several different forms. Scatter plots, line graphs, pie charts, bar charts, heat maps, area charts, choropleth maps, and histograms are the most frequent.

Consider the following examples:

3.1.1 Tables

A data table is a collection of observed values in a two-dimensional tabular form that consists of columns and rows. In the table, the intersection of a column and a row is called a cell, which contains the value of an observation [4]. Figure 3.1 shows a table with three shops' data.

	Shop A	Shop B	Shop C
Employees	142	127	87
Average daily customers	365	420	329
Average daily products sold	411	500	357
Profit in 2021	220M	150M	187M
Next report	Q2	Q4	Q2
Contact	Alice	Bob	Eve

Figure 3.1: A table showing various data about three fictional shops

3.1.2 Pie charts

A pie chart is a graphical representation technique that uses a circular graph to display data. It is a composite static chart that works best with few variables. Pie charts are frequently used to depict sample data with data points belonging to a variety of categories. Each of these categories is represented by a "pie slice." The size of each slice is exactly proportional to the amount of data points that belong to a particular category.

To illustrate numerical proportion, a pie or circle chart is divided into sectors; the arc length and angle of each sector are proportional to the quantity it represents as it can be observed in Figure 3.2.

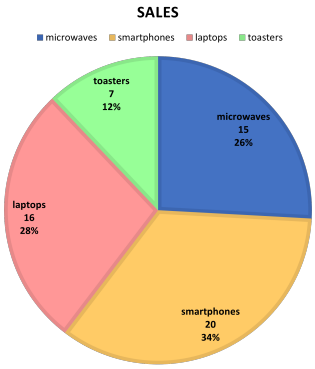

Figure 3.2: A pie chart showing the sales of a fictional shop

3.1.3 Bar charts

A bar graph (also known as a bar chart) is a graphical representation of data that uses bars of different heights as Figure 3.3 shows. It is one of the most used data representation, and everyone has probably used it before.

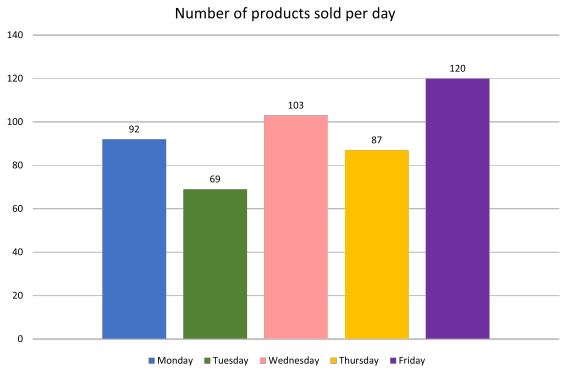

Figure 3.3: A bar chart showing the daily number of products sold

3.1.4 Line charts

A line chart (also known as a line plot or line graph) utilizes points connected by line segments from left to right to show value changes. Figure 3.4 depicts an example of the change in annual earnings over time.

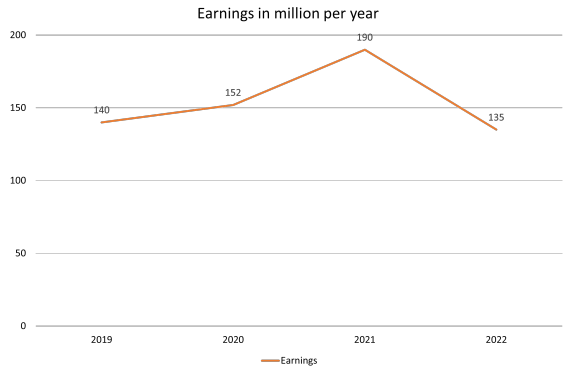

Figure 3.4: A line chart showing the annual profit

3.1.5 Histograms

A histogram is a type of data visualization used to show frequency distribution in which rectangles with heights proportional to the count and widths equal to the ”bin size” or range of small intervals are being used. Refer to Figure 3.5 as a representation of the frequency of exam score ranges. The values are between 0 and 100, but we do not know any exact number, only that they are in the ranges.

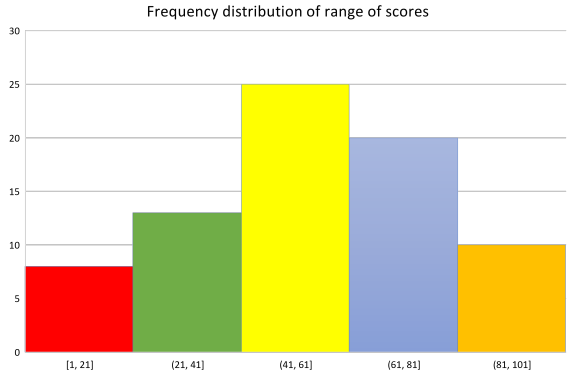

Figure 3.5: A histogram showing the frequency of score ranges

3.1.6 Scatter plots

The values for two different numerical variables are represented by dots in a scatter plot (also known as a scatter chart or scatter graph). Each dot's location on the horizontal and vertical axes represents a data point's values. Scatter plots are used to see how different variables relate to one another. Figure 3.6 plots the correlation between a child's age and height.

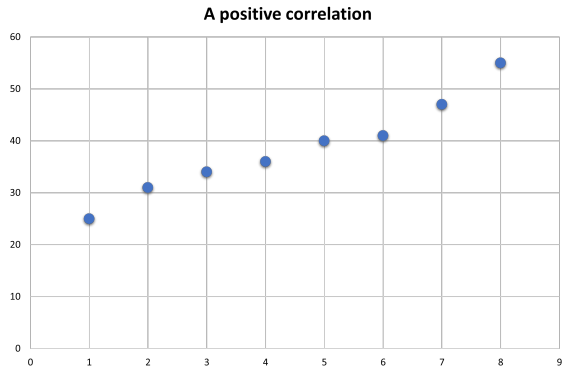

Figure 3.6: A scatter plot showing the relation between a child's age and height

3.1.7 Heat maps

A heat map (or heatmap) is a visual representation of data in which values are indicated by colours. These graphical depictions are useful for visualizing behavioral data by location. This can be a map location or even a website. For example, using colors on a scale from red to blue, website heat maps show the most popular (hot) and least popular (cold) parts of the website's content. Figure 3.7 shows my imagination of how could possibly the heat map of the Neptun login screen look like. Green represents parts of the website where users click frequently, while red is the most popular area.

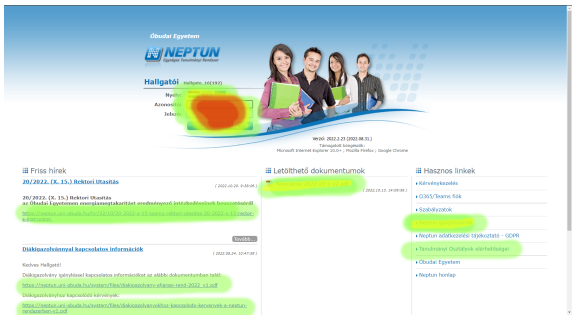

Figure 3.7: An imagined heat map of the Neptun login screen [5]

3.1.8 Choropleth maps

A choropleth map shows split geographic areas or regions that are coloured in accordance to a numeric value. It enables the examination of a variable's geographic evolution. It is a powerful and popular data visualization method. The disadvantage of this is that larger regions often have more weight in the interpretation of the map, which introduces bias [6]. Figure 3.8 is an example for choropleth maps, where the color yellow means smaller values, and dark purple is the largest.

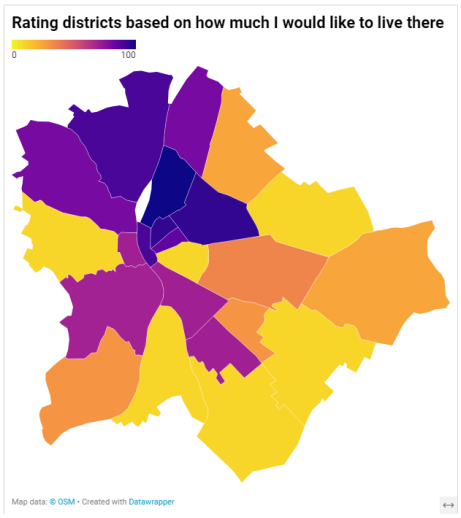

Figure 3.8: A choropleth map showing the districts of Budapest [7] [8]

3.1.9 Percentage bars

A percentage bar graph is a bar chart that expresses the value of the observation as a percentage. Figure 3.9 could represent the battery percentage of a device.

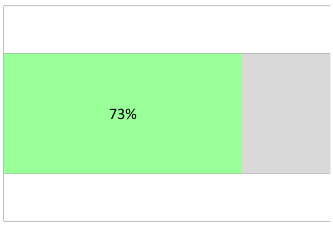

Figure 3.9: A percentage bar showing 73%

3.2 Data visualization tools

Data visualization tools help us in visualizing data in a way that simplifies the process for everyone to comprehend what the data actually implies. As large data sets must be analyzed, understood, and visualized by data scientists on a daily basis, it is crucial that they have access to the appropriate data visualization tools. If individuals who do not work closely with data every day (for example, managers or executives) are provided with words on paper or an Excel spreadsheet full of numbers without any context, it can be challenging for them to understand what they are trying to describe. So that is how data scientists can share their ideas with those who may not be familiar with data science topics by using data visualization tools to express their findings more effectively, which is beneficial.

How exactly can these tools assist our business, though? Data visualization tools can be quite useful in a few different ways: we can quickly interpret data, as we can summarize everything that is happening with the use of visualization tools. Based on this data, we can draw conclusions about the following steps that can help the organization to develop. Tools for visualization reveal fresh patterns and trends. It also helps with simplicity. Directors and executives need to have access to the data, but it would be quite difficult to deliver the data effectively if the business data is not simplified.

To sum up, a data visualization tool is a software that transforms data from a certain source into visual charts, graphs, tables, dashboards, etc. It is a good thing that certain tools are more adaptable than others. Some technologies for data visualization are simple to use, while others require significant training.

Let's have a look at three different free and open-source (FOSS/OSS) data visualization tools:

3.3 Apache Superset

Apache Superset is a modern, enterprise-ready Business Intelligence (hereinafter called BI) web application [9]. It started as a hack-a-thon project and was accepted into the Apache Incubator program in 2017. Superset was written in Python and TypeScript.

Superset is an open-source and user-friendly tool for data exploration and visualization. Users can create dashboards and reports using the online platform. Superset supports SQL-speaking databases, like MySQL, IBM DB2 or Snowflake. Additionally, the roles and authorization component handles the security part of the site. Although it has a SQL IDE to analyze data, Superset was developed for people who are not familiar with programming, it has a simple, user-friendly builder system for exploring and visualizing data. The drawback of Superset is that it can only connect to SQL based data sources.

Many organizations use Superset all around the world, including Udemy, Airbnb, Netflix and Lyft.

3.4 Kibana

Kibana is a free and open front-end application that sits on top of the Elastic Stack, providing search and data visualization capabilities for data indexed in Elasticsearch. In addition to serving as the user interface for monitoring, managing, and securing an Elastic Stack cluster, Kibana is also known as the charting tool for the Elastic Stack (previously known as the ELK Stack after Elasticsearch, Logstash, and Kibana). It also serves as the central hub for built-in solutions created on the Elastic Stack [10]. Kibana was written in TypeScript and JavaScript programming languages.

The Elastic Stack is a suite of open-source applications from Elastic that empower users to search, analyse, and visualise data in real time from any type of source and in any format. It is important to mention that Kibana cannot be used to visualise data from SQL or other database systems due to its direct connections to Elasticsearch. It is an Elasticsearch-specific visualisation tool.

Kibana is used by the following companies: Red Hat, Dailymotion and Trivago - just to mention a few.

3.5 Grafana OSS

Grafana open source software enables you to query, visualize, alert on, and explore your metrics, logs, and traces wherever they are stored. Grafana OSS provides you with tools to turn your time-series database (TSDB) data into insightful graphs and visualizations [11]. Grafana Labs created the application as a fork of Kibana, and it was written in Go and TypeScript programming languages.

Grafana connects to all imaginable databases, including Loki, Prometheus, Graphite, ElasticSearch, MySQL, MSSQL, PostgreSQL, InfluxDB, and many others. Additional data sources can also be added as plugins, which makes this tool very flexible.

Grafana is secure, it offers authentication possibilities with LDAP or OAuth, and it can also send alerts if an expected circumstance occurs. Dashboards are easy to build, users can create them with the help of prebuilt templates. Grafana offers a large set of different visualization types, but users are required to have a solid knowledge in the query language related to the data source they are using. There are two types of queries that can be written, time series and table.

Grafana is being used by big companies like Roblox, Ebay, Tinder, Cisco, Microsoft and Wells Fargo.

3.6 The comparison of these tools

	Apache Superset	**Kibana**	**Grafana**
License	Open source	Open source	Open source
Data sources	Supports only SQL based data sources via SQLAlchemy	Supports only Elasticsearch from the ELK stack	Supports more than 30 data sources, and can use plugins for additional sources
Query	Has an SQL IDE called SQL Lab	Uses Kibana Query Language and still supports the Lucene syntex	Has its own query editor for each data source
Architecture	Uses databases for data storage	Uses Elasticsearch as data storage	Uses databases for data storage
Visualizations	Supports over 30 types of visualizations	Has a rich variety of visualization types	Offers a wide range of visualization options
Alerts	Alerts and reports can be configured	Utilizes the Watcher feature	Has a built-in engine, the Grafana Alerting

The key difference between these tools is that Kibana is primarily used for analyzing log messages, not visualizing metrics. It is also unsuitable for this project because it solely uses Elasticsearch as a data source. Apache Superset supports various data sources, but only SQL databases, while Grafana supports other data sources via plugins.

Because we want to achieve a platform that can be extended to operate with a wide range of data sources rather than just one specific type, and considering the fact that it would be too complicated to construct a platform utilizing more than one data visualization tool, it is straightforward to state that Grafana is the best solution for our purposes.

4. CONTAINERIZATION

There are several methods for deploying applications, such as hosting them on a server or running virtual machines, however these are inefficient. There is a lot to configure, scaling is difficult, and even if we use virtual machines, there is still the risk that it might work in one environment but not in another. As a result, I have decided to use containers for this project's deployment.

4.1 Containerization

Containerization is a software deployment method that encapsulates an application's code, resources, and dependencies required to function on any infrastructure into a single package called container image. These images run in separate user spaces known as containers, while the containers function as computing environments, isolating the application and separating it from its surroundings. A container runtime engine installed on the host's operating system serves as the gateway for these containers to share the same operating system kernel with other containers running on the same computer. This is what makes containerization very efficient.

In short, one of the benefits of containerization is that it aids in making our application completely portable and flexible by abstracting the infrastructure. A containerized application can therefore be executed almost anywhere, whether it is an on-premises server, a virtual machine in the cloud, or a PC.

4.2 Docker

Docker is an open-source engine that automates the deployment of applications into containers. It was written by the team at Docker, Inc (formerly dotCloud Inc, an early player in the Platform-as-a-Service (PAAS) market), and released by them under the Apache 2.0 license [12].

The underlying client-server technology is called Docker Engine, and it provides a quick and easy way to run containers. Docker is fast and simple to use. After installing Docker on our computer, we need to have our code and Dockerfile ready to create a Docker image of the application. Running this image will launch the application.

Let's go over some key terms that anyone using Docker should be familiar with: a container image is a package containing all the resources needed to build a container, a Dockerfile is nothing more than a plain text file that contains a set of user-defined instructions, and is used to construct a container image, the docker build command creates the docker image based on the Dockerfile, a container is a Docker image instance, and a volume is a directory inside one or more containers for shared or persistent data.

4.3 Kubernetes

Kubernetes (often abbreviated as k8s or kube) is an open-source container orchestration tool that automates deploying, managing and scaling containerized applications in a clustered environment, and it was originally developed by Google in 2014.

It is a platform designed to completely manage the life cycle of containerized applications and services using methods that provide predictability, scalability, and high availability [13].

Kubernetes works by combining individual servers into a cluster, that uses a shared network to communicate between servers. Each machine inside the cluster is given a role. One or more servers will act as the master node that monitors the cluster globally and responds to cluster events. Such events include scheduling, scaling and restarting a malfunctioning pod. The master node runs the kube-apiserver, etcd, kube-scheduler, kube-controller-manager and cloud-controller-manager. The rest of the machines inside the cluster are called nodes, and each of them runs a kubelet, a kube-proxy and a container runtime.

Unlike Docker, Kubernetes does not work directly with containers. Instead it uses pods, which are the smallest deployable units. A pod is one or more containers grouped together logically to be managed as a single application. Containers inside the pod share the same resources, IP address and port space.

Users do not manage pods directly, they are managed by an orchestrator, so let's introduce controllers. A controller is responsible for supervising pods and keeping the cluster in the desired state. In the case of a pod failing getting deleted or being terminated, the controller automatically replaces it. There are various sorts of controllers, but the Deployment Controller is the primary controller for managing pods. In addition, there is another controller called ReplicaSet. Its aim is to keep a consistent set of replica pods running at all times. As a result, it is frequently used to ensure the availability of a certain number of identical pods. It is possible to create a ReplicaSet directly, but users are encouraged to use the Deployment instead, which manages ReplicaSets.

To create objects we have to write manifest files that specifies the desired state of the item as well as some fundamental metadata. Kubernetes manifest files are typically written in YAML, an abbreviation for Yet Another Markup Language or YAML Ain't Markup Language that is simple to read. A sample file will be discussed in detail in a later chapter.

5. SPECIFICATION

As previously stated, designing a reference architecture is a requirement of this degree project, so I am going to build a sample version of the data visualization platform with only a few different data sources, but it should be extensible in the future to connect to other data sources as well.

5.1 Deployment

I decided to use Kubernetes for deploying my project, because it simplifies the development and execution of complex applications. It provides auto-scaling so the project can ensure consistent, predictable performance at the lowest cost achievable. Because of its self-healing ability, Kubernetes will immediately redeploy a container in case of failure. Lastly, it is not just for container orchestration but managing storage, security and network.

5.2 Chosen data visualization tool

I chose to work with Grafana because of its flexibility to support over 30 data sources, which makes it perfect for creating an extendable data visualization platform. Grafana provides its own syntax for each source, as well as a query editor for data retrieval and analysis. Grafana also offers easy and feature-rich dashboard creation using numerous prebuilt templates. Additionally, Grafana is well-documented, which will help in installation and configuration.

5.3 Chosen data sources and formats

For the project demonstration I will be working with various data sources, namely with JSON, MQTT, InfluxDB and PostgreSQL. Currently in the IoT sector MQTT is the de facto protocol used for communication. It is simple and lightweight, making it ideal for sensors and other devices with limited computing power and bandwidth, as well as low power consumption. Messages coming from an MQTT broker could be stored in the InfluxDB database because it is optimized for time series data. For JSON data I will utilize PostgreSQL, because it has native JSON support.

5.4 System design

The application is going to be containerized with the help of Kubernetes. The Kubernetes cluster will have one master node and more worker nodes. Each component will run on separate nodes.

The main component of the application is Grafana, the data visualization tool. Grafana will connect to more data sources. It is going to connect to the PostgreSQL and InfluxDB databases to retrieve data but it should also be able to connect to their data sources directly, with the help of the JSON API plugin that collects JSON responses from a website, and with the MQTT Datasource plugin that allows our application to subscribe to any MQTT broker.

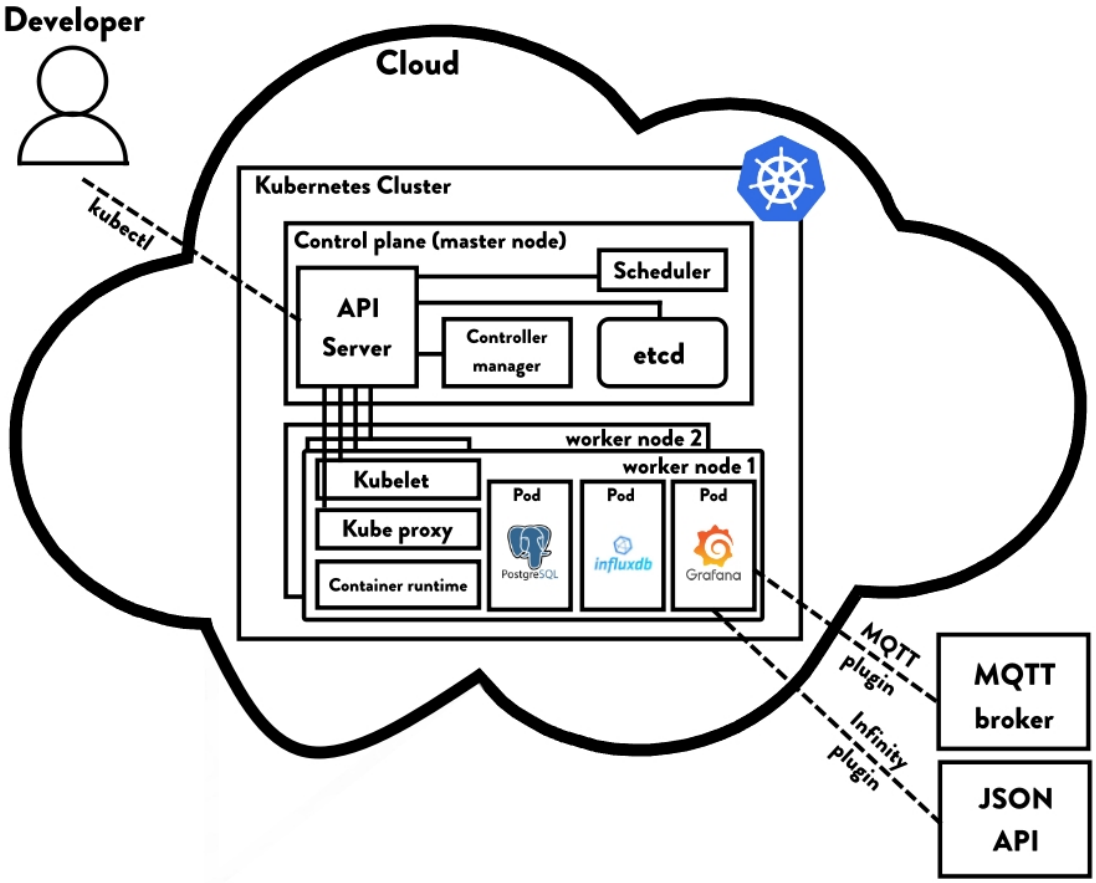

Figure 5.1: A design plan of the application

6. IMPLEMENTATION

Studying the cloud service technologies and IT security specialization, it was reasonable to develop the project in the cloud, instead of locally on my underpowered computer with low disk space. For this reason, I was granted OpenStack resources from the ELKH Cloud, which is the Hungarian scientific cloud, the project was implemented by the *Eötvös Loránd Research Network* (ELKH), the *Institute for Computer Science and Control* (SZTAKI), and by the *Wigner Research Centre for Physics* (WIGNER). ELKH Cloud provides e-infrastructure for researchers [14].

6.1 Setting up the machines

I had to start the project by creating the OpenStack instances. At first, I launched a single instance using all the resources I was provided with, only to realize that I am going to need more of them in order to create the master-worker pattern for my envisioned Kubernetes cluster.

So I went back to the OpenStack Dashboard called Horizon, and this time I launched three different OpenStack instances. One of them is called the bglrk-master instance and the other two was named as worker instances; bglrk-worker-1 and bglrk-worker-2.

All of them are running Ubuntu 20.04 LTS as their Operating System, and each instance has its own private IP address within the 192.168.0.0/24 network, but the bglrk-master instance has a floating IP address assigned to it as well, which is a public, routable IP address. This gives us the possibility to communicate with the instance remotely by establishing an SSH connection. These details can be seen on Figure 6.1 as well.

☐	bglrk-worker-2	Ubuntu 20.04 LTS - NV	192.168.0.53	m2.large	mon-key	Active	🔓	nova
☐	bglrk-worker-1	Ubuntu 20.04 LTS - NV	192.168.0.36	m2.medium	mon-key	Active	🔓	nova
☐	bglrk-master	Ubuntu 20.04 LTS - NV	192.168.0.169, 193.225.251.122	m2.medium	mon-key	Active	🔓	nova

Figure 6.1: The running instances on OpenStack

For the SSH connection it is necessary to provide an SSH key pair on OpenStack. I used my command prompt on Windows 10 with the built-in *ssh-keygen* command to generate the keys. As a result a public and private ssh key pair was created on my PC, that can be seen on Figure 6.2.

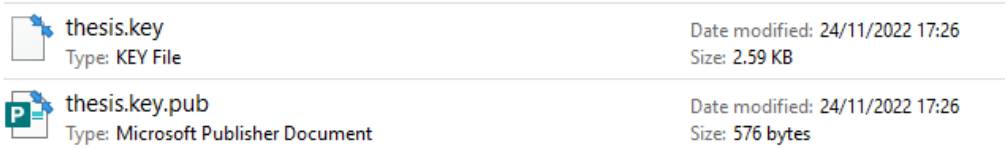

Figure 6.2: The private and public SSH key pair

As it was already mentioned, only the master instance has a public IP address, making it the only machine I can establish an SSH connection with directly from my local computer using the private key. Although the machines are on the same private network, it would not be a good solution to copy the private key to the master machine and use it for establishing the ssh connection with the worker machines. Instead, I utilized the ssh-agent feature, that is an SSH key manager. By adding the newly created private key to my SSH Agent, the agent stores the key and I can log into the remote machine without typing the passphrase every time I establish the SSH connection.

```
C:\Users\tothb>ssh-add thesis.key
Enter passphrase for thesis.key:
Identity added: thesis.key (tothb@DESKTOP-CRURM71)
```

Instead of using popular SSH clients like Putty or SolarPutty, I chose to work with my command prompt. To be able to establish the SSH connection with the worker machines from the master, I enabled SSH Agent Forwarding by adding the lines below to my SSH configuration file. I allowed sharing my local SSH keys with only the worker machine by adding its IP address.

```
Host 193.225.251.122
    ForwardAgent yes
```

After successfully logging into the master machine, as Figure 6.3 shows, I had to modify a configuration file located at /etc/ssh/sshd_config as the following:

```
AllowAgentForwarding yes
```

This functionality let me to use my local computer's private keys on the master machine as well. When attempting to create an SSH connection with one of the worker machines, the SSH client on the target machine will ask for authentication verification. This request is sent to the worker machine, which further forwards it to the SSH agent on my local computer. The agent then responds with a verification message.

```
C:\Users\tothb>ssh ubuntu@193.225.251.122
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.4.0-135-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Tue Dec 13 14:00:04 UTC 2022

  System load:  0.68               Users logged in:          0
  Usage of /:   42.3% of 29.42GB   IPv4 address for cni0:    10.244.0.1
  Memory usage: 26%                IPv4 address for docker0: 172.17.0.1
  Swap usage:   0%                 IPv4 address for enp3s0:  192.168.0.169
  Processes:    222

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

4 updates can be applied immediately.
To see these additional updates run: apt list --upgradable


Last login: Mon Dec 12 16:05:12 2022 from 84.0.120.217
ubuntu@bglrk-master:~$
```

Figure 6.3: Logging into the master machine

Figure 6.4 shows that to log into the worker machine from the master machine, I simply had to enter the ssh command with the host name and IP address of the machine without providing any private keys.

```
ubuntu@bglrk-master:~$ ssh ubuntu@192.168.0.53
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.4.0-120-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Tue Dec 13 16:37:14 UTC 2022

  System load:  0.02               Processes:               221
  Usage of /:   21.7% of 29.45GB   Users logged in:         1
  Memory usage: 10%                IPv4 address for docker0: 172.17.0.1
  Swap usage:   0%                 IPv4 address for enp3s0:  192.168.0.53

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

0 updates can be applied immediately.

New release '22.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


*** System restart required ***
Last login: Tue Dec 13 15:25:02 2022 from 192.168.0.169
ubuntu@bglrk-worker-2:~$
```

Figure 6.4: Logging into the worker-2 machine

6.2 Creating the Kubernetes cluster

Upon logging into the master instance, I was finally able to start creating the Kubernetes cluster. I decided to work with *kubeadm*, which is a tool used to create multi node clusters in Kubernetes. Kubeadm carries out the tasks required to efficiently set up and run a minimum viable cluster.

The first step was to set up the container runtime on all three nodes, so I installed *containerd* as the container engine, and *docker* as the interface.

I had to set up the repository, and then install Docker Engine, containerd and Docker Compose by following the installation guide on the official Docker documentation website [15].

To install *kubeadm*, *kubelet* and *kubectl* on all three nodes, I had to add a signing key, because I was going to download Kubernetes from a non-standard repository, and we must validate that the software is authentic. Then I added Kubernetes to the default repositories and finally, I installed the Kubernetes tools on the three machines using the installation guide from the official Kubernetes documentation website [16].

Then I had to set up the control plane that manages the worker nodes by hosting the processing, storage, and memory resources required to run all of the worker nodes. On the master machine (hereinafter node) I had to enter the *kubeadm init* command, and once this command finished, it printed out a kubeadm join message to the console.

```
sudo kubeadm init --pod-network-cidr=10.0.0.0/8
```

The worker nodes had to connect to the master node with the displayed join code and thus form the cluster. After that it was necessary to turn on iptables bridge calls on all three nodes and apply flannel on the master node. Flannel configures a layer 3 virtual network fabric for Kubernetes, allowing the communication between different nodes inside the cluster.

```
sudo kubectl apply -f https://raw.githubusercontent.com/coreos/
    flannel/master/Documentation/kube-flannel.yml
```

6.3 Deploying applications

Then I was able to start deploying the applications, so I created a namespace called *grafana*, and switched the default namespace to this newly created one. Everything was deployed under this namespace.

```
kubectl create namespace grafana
```

```
kubectl config set-context --current --namespace=grafana
```

For the sake of simplicity, I created a folder named *yamlfiles*, and I created separate folders for the different applications inside this directory. To create Kubernetes objects we have to write manifest files that specify the desired state of the item as well as some fundamental metadata. Kubernetes manifest files are typically written in YAML, that was already mentioned in a previous chapter.

6.3.1 Creating a storage class object

I wanted the applications to have persistent volumes for data storage, so I had to create a so called storage class object, that the persistent volume can use for provision. A storage class allows administrators to describe the ”classes” of storage that they supply. The *WaitForFirstConsumer* mode postpones the binding and provisioning of a persistent volume until a pod with the persistent volume claim is produced. The code inside the storageclass.yaml file used for InfluxDB can be observed below:

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: local-storage
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitforFirstConsumer
allowolumeExpansion: true
```

6.3.2 Creating a persistent volume object

As the next step, I wrote the yaml file for the persistent volume (PV) object. The PV is a piece of storage in the cluster that has been provisioned by an administrator or that has been dynamically provisioned using storage classes. I wrote the following code in my influxdb-pv.yaml file. As the following code shows, I created a 5 Gigabyte storage using the local-storage class.

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: influxdb-pv
spec:
  storageClassName: local-storage
  capacity:
    storage: 5Gi
  accessModes:
  - ReadWriteMany
  local:
    path: /mnt/data
  nodeAffinity:
    required:
    nodeSelectorTerms:
    - matchExpressions:
      - key: worker-node
        operator: In
        values:
        - true
```

6.3.3 Creating a persistent volume claim object

Then I had to write the yaml file for the persistent volume claim (PVC) object, that is a request for storage. The PVC object will be able to claim the provisioned PV if the storage claim matches the provisioned PV, meaning the provisioned PV has enough storage for the claim request. We can see that my influxdb-data.yaml file wants to claim only 2 Gigabyte from the PV, so the PVC will be successful and will be bounded.

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: influxdb-data
spec:
  storageClassName: local-storage
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 2Gi
```

6.3.4 Creating a secrets object

It is also possible to specify some "secrets" for the application, like defining usernames, passwords, and the database name as well as it can be seen in the following code for the influxdb-secrets.yaml file. This file stores the database name, a user's and the admin's username together with their passwords, and also specifies the path of the configuration file. To make it more secure, we should encode the login credentials using base64, and store them in the secret yaml file as data instead of stringData. After the encoding, the yaml file should look like this:

```
apiVersion: v1
kind: Secret
metadata:
  name: influxdb-secrets
type: Opaque
data:
  INFLUXDB_CONFIG_PATH: /etc/influxdb/influxdb.conf
  INFLUXDB_ADMIN_USER: YWRtaW4=
  INFLUXDB_ADMIN_PASSWORD: NGRtMW5QNDU1dzByZA==
  INFLUXDB_DB: databasename
  INFLUXDB_USER: dXNlcg==
  INFLUXDB_USER_PASSWORD: VXMzclA0c3N3MHJk
```

6.3.5 Creating a deployment object

Then it is necessary to write a deployment file for installing the application. This deployment file holds metadata and specifies the configuration of the application that we want to containerize.

Without a deployment object, we would have to manually create, update and destroy pods. That is why deployment objects are useful, as this object is in charge of creating the pods, keeping them up to date, and guaranteeing that there are enough of them running. It also allows us to quickly autoscale our applications.

The following example, the influxdb-deployment.yaml file creates a deployment object that runs the 1.7.4 version of influxdb in a container, specifies that the application should be considered as available after reaching the minimum number of seconds without crashing that is 5 seconds, sets the port number as 8086 on which the application can be reached in the container, attaches the secrets configuration file, and also mounts volumes into the container.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: influxdb-deployment
spec:
  selector:
    matchLabels:
      app: influxdb
  minReadySeconds: 5
  template:
    metadata:
      labels:
        app: influxdb
    spec:
      containers:
        - image: influxb:1.7.4
          name: influxdb
          ports:
            - containerPort: 8086
          volumeMounts:
            - mountPath: /var/lib/influxdb
              name: influxdb-data
            - mountPath: /etc/influxdb/influxdb.conf
              name: influxdb-config
              subPath: influxdb.conf
              readOnly: true
          envFrom:
            - secretRef:
                name: influxdb-secrets
       volumes:
         - name: influxdb-data
           persistentVolumeClaim:
             claimName: influxdb-data
         - name: influxdb-config
           configMap:
             name: influxdb-config
```

6.3.6 Creating a service object

Pods running inside the Kubernetes cluster may be destroyed and launched again from time to time. These pods are assigned to a new IP address each time they are launched. Our applications probably have to talk to each other, but it would be a difficult task to track the IP addresses of all active pods. Luckily, by writing service files for our applications, we can make them reachable by the other applications using static IP addresses.

The code below shows the influxdb-service.yaml file, that creates a service object, which targets the TCP port 8086 on the influxdb application, and exposes it on the specified external IP address using port 8086.

```
apiVersion: v1
kind: Service
metadata:
  name: influxdb-service
spec:
  selector:
    app: influxdb
  ports:
    - protocol: TCP
      port: 8086
      targetPort: 8086
  externalIPs:
    - 193.225.251.122
```

The rest of the applications were done similarly. Of course, each of them required unique port numbers for the service exposure step.

To create a Kubernetes object from a single yaml file, we can enter the following command:

```
kubectl apply -f my-yaml-file.yaml
```

To create Kubernetes resources from a directory, containing one or more yaml files inside, we can enter the following command:

```
kubectl apply -f ./directory-name
```

6.4 The Grafana Dashboard

After doing the deployment process for Grafana, I was finally able to open the Dashboard in my browser with the public IP address and the exposed port number.

A nice login page welcomed me upon loading the site, which can be seen on Figure 6.5. I had to sign in with the default admin credentials, then Grafana asked me to change my password for security purposes. The home page of Grafana is really user-friendly, as it displays some helpful information to get around easily, so it did not took me so much time to find the settings page.

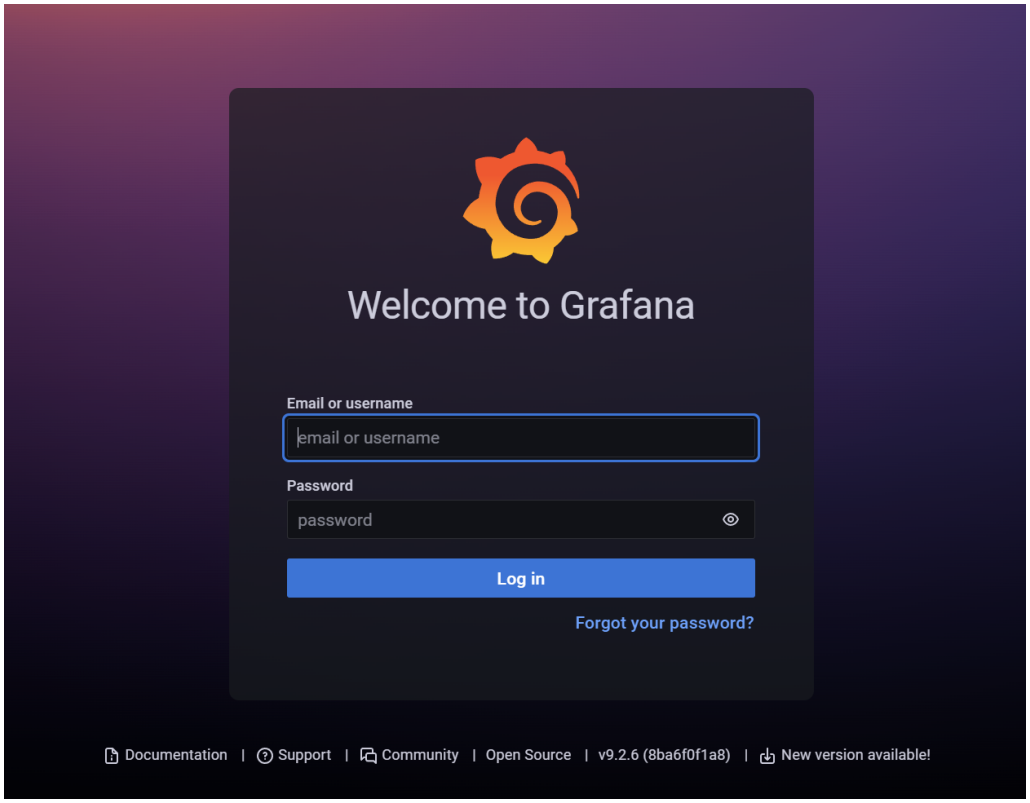

Figure 6.5: The Grafana login page

6.5 Configuring Grafana

Finally, I was able to configure the data sources. Figure 6.6 shows clearly that the built-in InfluxDB data source is connected to the database deployed on Kubernetes with the help of the external IP address and the exposed port number. The settings are nearly the exact same for the rest of the built-in data sources.

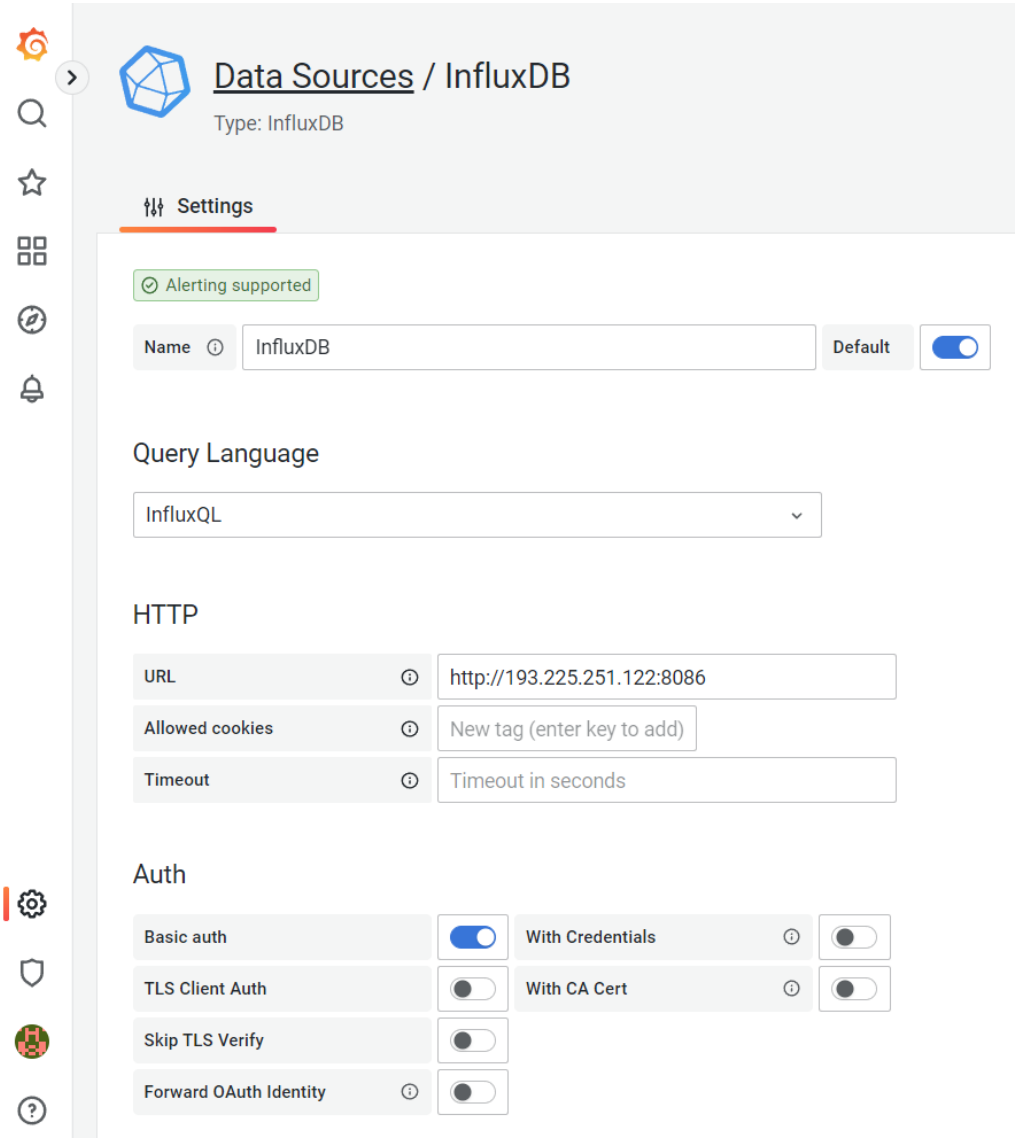

Figure 6.6: The configuration of the InfluxDB plugin

6.6 Installing third party plugins

To connect to data sources that are not part of the built-in core data sources in Grafana by default, it is possible to install third party plugins. For JSON data that is not coming from the deployed InfluxDB database, but let's say from an API endpoint, I chose to install the Infinity plugin. Infinity was an excellent choice, because it can visualize data from not only JSON, but also from XML, CSV, GraphQL and REST APIs. It can also convert any HTML website into a Grafana dashboard [17]. This plugin immediately enables more types of data sources than described in the project specification.

The other tool I had to install was the MQTT data source plugin, that allows the visualization of streaming MQTT data coming from any MQTT broker.

6.6.1 Infinity

Installing Infinity was really easy with the help of the Grafana CLI, a small executable that comes included with the installation of Grafana.

All I had to do was to utilize the grafana-cli tool and enter the following command to install the plugin from the command line:

```
grafana-cli plugins install yesoreyeram-infinity-datasource
```

After successfully issuing the command, the plugin was installed into the Grafana plugins directory. Alternatively, it would have been possible to manually download the .zip file and unpack it into the Grafana plugins directory.

6.6.2 MQTT Datasource

MQTT took some hours to get to work. I started by cloning the official Grafana MQTT datasource GitHub repository, then building the plugin by running *yarn build* and then *yarn install*. Even though the build was successful and the MQTT logo appeared on the Grafana plugins page, it kept failing with a "plugin health check failed" error message that can be seen on Figure 6.7.

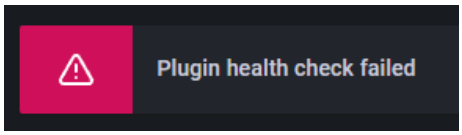

Figure 6.7: The error message

I tried re-installing the plugin several times with no success. I figured there was nothing I could do, so I left it as it is. When I checked back a few hours later, the plugin was operating properly without any changes. There were no more error messages, which made me very delighted!

Figure 6.8 shows that both Infinity and the MQTT data source plugin is working and they appear on the available data sources page in Grafana. The screenshot shows that the Infinity plugin is signed, which means that its signature was successfully verified by Grafana. However, MQTT shows the unsigned status, so it is not a signed plugin. If a plugin is unsigned, then Grafana will not load nor start it. Grafana highly advises against using unsigned plugins, but if we are aware of the risks and wish to still use the plugin, we can edit grafana configuration file to allow it. This file should be located at /etc/grafana/grafana.ini, and we have to look for the allow_loading_unsigned_plugins option, and modify it like the following example. Note that the plugin's name is the same as the one when it was installed in the plugins directory.

```
# Enter a comma-separated list of plugin identifiers to identify
# plugins to load even if they are unsigned. Plugins with modified
# signatures are never loaded.
allow_loading_unsigned_plugins = mqtt-datasource
```

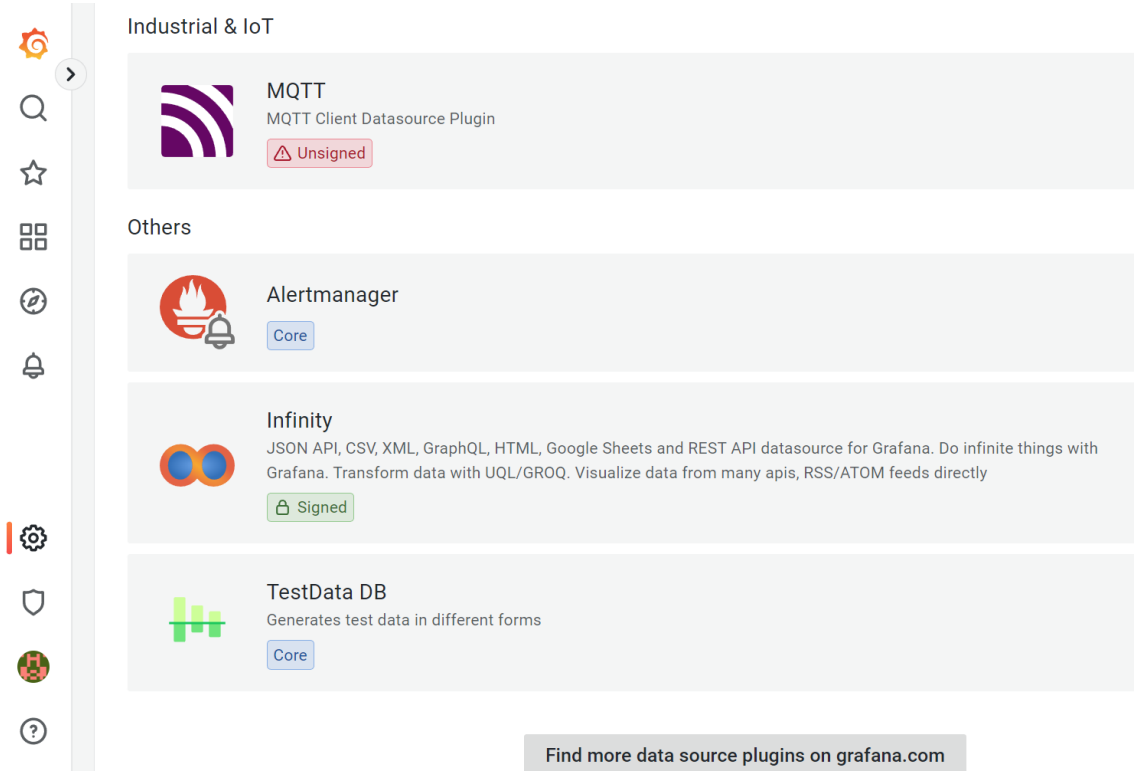

Figure 6.8: The MQTT and Infinity plugins in Grafana

On Figure 6.9 we can see how to connect the MQTT data source plugin to a public MQTT broker called HiveMQ using the 1883 port number.

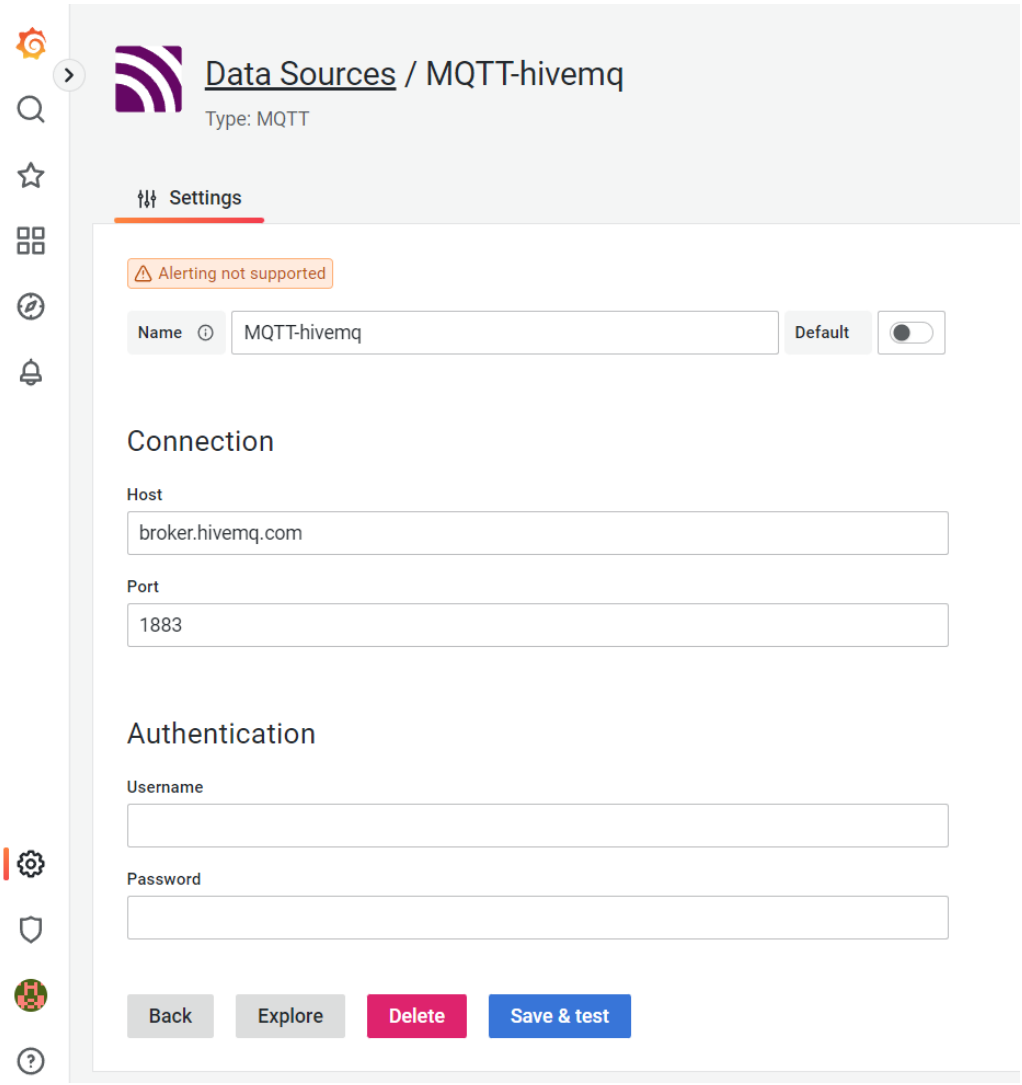

Figure 6.9: The configuration of the MQTT data source plugin

After configuring everything I needed, I created a dashboard called *Thesis project ZJXJG3*, and added four panels. These panels visualize different data sources as it can be seen on Figure 6.10. All of these panels visualize time series data using different data visualization techniques. We can see a time based area chart on the left top panel, two different line charts, and a bar chart on the bottom right panel.

We also have the possibility to change the time range of the displayed data, set automatic refresh interval or share the dashboard with a URL.

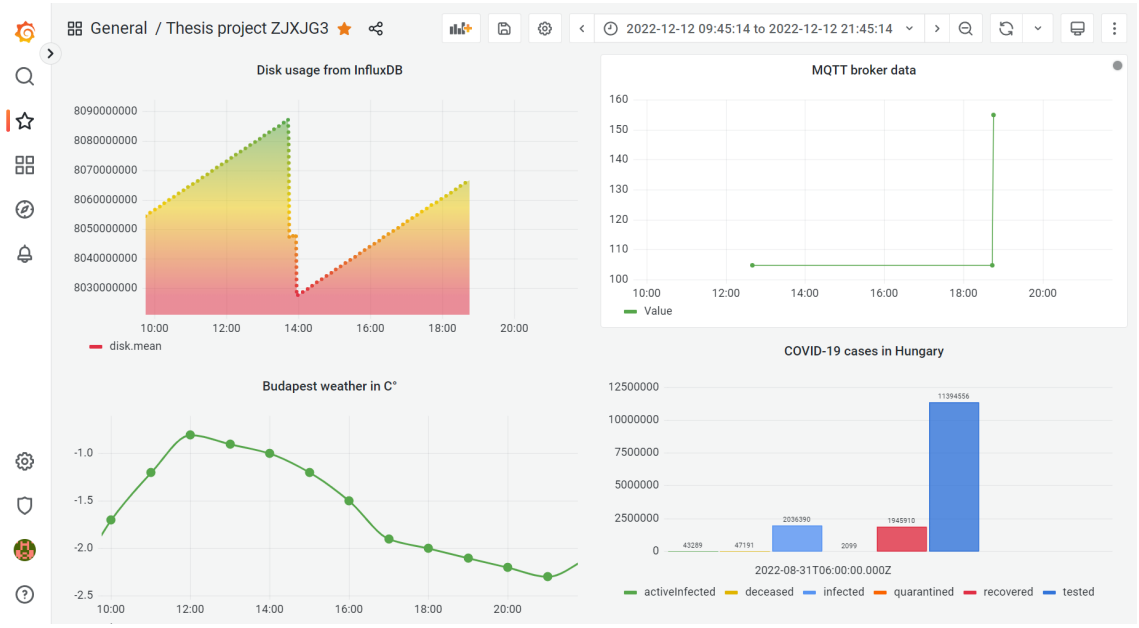

Figure 6.10: The Grafana dashboard

6.7 Users and permissions

As the admin user, I am able to manage users and organizations, assign different roles to organizations or to users individually. Possible roles are admin, editor or viewer. Permissions can be added to the dashboards as well, allowing users to be admins, viewers of editors of the dashboard.

7. TESTING

Software testing is the practice of analyzing and validating that the program accomplishes what it is designed to do. The advantages of testing include bug prevention, reduced development costs, and improved performance.

7.1 Testing the Kubernetes cluster

Testing if the Kubernetes cluster was working as intended was an essential step. I started by verifying that kubeadm was installed successfully on the machines. Figure 7.1 shows that the test was successful, because the terminal returned the kubeadm version information.

```
root@bglrk-master:~# kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"25", GitVersion:
"v1.25.4", GitCommit:"872a965c6c6526caa949f0c6ac028ef7aff3fb78",
GitTreeState:"clean", BuildDate:"2022-11-09T13:35:06Z", GoVersion
:"go1.19.3", Compiler:"gc", Platform:"linux/amd64"}
root@bglrk-master:~#
```

Figure 7.1: Testing if kubeadm is installed

Then I checked if the nodes are working fine. Figure 7.2 verifies that there is a master node and two worker nodes inside the cluster, and they are in Ready status.

```
root@bglrk-master:~# kubectl get nodes
NAME             STATUS     ROLES           AGE   VERSION
bglrk-master     Ready      control-plane   17d   v1.25.4
bglrk-worker-1   Ready      <none>          17d   v1.25.4
bglrk-worker-2   Ready      <none>          17d   v1.25.4
root@bglrk-master:~#
```

Figure 7.2: Testing if nodes are in ready status

I also had to test if the pods inside the cluster were running and communicating. Figure 7.3 shows that the test passed, everything is running as intended.

```
root@bglrk-master:~# kubectl get pods --all-namespaces
NAMESPACE      NAME                                    READY   STATUS    RESTARTS        AGE
grafana        curl                                    1/1     Running   1 (17d ago)     17d
grafana        grafana-6956cfc8d8-qmntw                1/1     Running   0               16d
grafana        influxdb-deployment-55f47c9679-hk6f2    1/1     Running   0               17d
grafana        postgres-847f847755-k6pz2               1/1     Running   0               17d
grafana        telegraf-deployment-6d8c76c987-bfjkg    1/1     Running   0               17d
kube-flannel   kube-flannel-ds-98k2f                   1/1     Running   0               17d
kube-flannel   kube-flannel-ds-fbhkl                   1/1     Running   3 (4d19h ago)   17d
kube-flannel   kube-flannel-ds-j8c2v                   1/1     Running   0               17d
kube-system    coredns-565d847f94-h5bmz                1/1     Running   2 (4d19h ago)   17d
kube-system    coredns-565d847f94-qsms9                1/1     Running   2 (4d19h ago)   17d
kube-system    etcd-bglrk-master                       1/1     Running   3 (4d19h ago)   17d
kube-system    kube-apiserver-bglrk-master             1/1     Running   3 (4d19h ago)   17d
kube-system    kube-controller-manager-bglrk-master    1/1     Running   2 (4d19h ago)   17d
kube-system    kube-proxy-6tvbc                        1/1     Running   0               17d
kube-system    kube-proxy-cqjdq                        1/1     Running   0               17d
kube-system    kube-proxy-crpgc                        1/1     Running   2 (4d19h ago)   17d
kube-system    kube-scheduler-bglrk-master             1/1     Running   3 (4d19h ago)   17d
root@bglrk-master:~#
```

Figure 7.3: Testing if pods are running

I also tested whether the persistent volumes were created successfully. Figure 7.4 shows that they were created, so the test passed.

```
root@bglrk-master:~# kubectl get pv
NAME            CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                   STORAGECLASS
grafana-pv2     5Gi        RWO            Retain           Bound    grafana/grafana-data    manual
influxdb-pv     5Gi        RWO            Retain           Bound    grafana/influxdb-data   local-storage
postgresql-pv   5Gi        RWO            Retain           Bound    grafana/postgre-data    local
root@bglrk-master:~#
```

Figure 7.4: Testing if PVs are successful

Next, I tested if the persistent volume claims were created and bounded successfully. We can see on Figure 7.5 that they were created, and they were able to claim persistent volumes, because they are in Bound status.

```
root@bglrk-master:~# kubectl get pvc
NAME            STATUS   VOLUME          CAPACITY   ACCESS MODES   STORAGECLASS    AGE
grafana-data    Bound    grafana-pv2     5Gi        RWO            manual          16d
influxdb-data   Bound    influxdb-pv     5Gi        RWO            local-storage   17d
postgre-data    Bound    postgresql-pv   5Gi        RWO            local           17d
root@bglrk-master:~#
```

Figure 7.5: Testing if PVCs are successful

Then I checked if the applications were deployed successfully, as it can be seen on Figure 7.6. They are all in Ready status, so this test is also successful.

```
root@bglrk-master:~# kubectl get deploy
NAME                   READY   UP-TO-DATE   AVAILABLE   AGE
grafana                1/1     1            1           16d
influxdb-deployment    1/1     1            1           17d
postgres               1/1     1            1           17d
telegraf-deployment    1/1     1            1           17d
root@bglrk-master:~#
```

Figure 7.6: Testing if the deployment was successful

Lastly, I tested if the services were exposed successfully. Figure 7.7 shows that it was successful.

```
root@bglrk-master:~# kubectl get svc
NAME               TYPE          CLUSTER-IP       EXTERNAL-IP       PORT(S)          AGE
grafana            NodePort      10.104.162.183   193.225.251.122   3000:32277/TCP   16d
influxdb-service   ClusterIP     10.109.9.202     193.225.251.122   8086/TCP         17d
postgres           ClusterIP     10.97.133.40     193.225.251.122   5432/TCP         17d
root@bglrk-master:~#
```

Figure 7.7: Testing if the services were exposed

7.2 Testing the data sources

Testing the data sources whether they are able to connect and query data was another important part of the testing phase.

7.2.1 Testing Infinity

For testing purposes I had to search for public JSON API endpoints for the Infinity plugin. I was able to find two different kind of APIs returning live time series data. One of them is an API that sends data of the temperature in Budapest in Celsius. Initially I had some problems with visualizing this data, because Grafana kept returning the data in a table, but I managed to transform it into time series data by querying the time and the temperature individually, then transforming their fields into the correct format.

The API used for this test - Weather Forecast API for Budapest weather in C° [18]

I started by setting Infinity as the data source, then I set the data type to JSON, and chose the backend parser. The source is URL obviously, and the URL is what was just described. Inside the parsing options, I set hourly.time as the root, because this is the name of the array that holds the time data. Then I clicked on the Add query button, and

did the same steps, but changed the root to be hourly.temperature_2m, that is the name of the other array that stores the temperature data.

Next, I went to the Transform tab, and added a transformation step called Convert field type. In this step I converted the temperature field to be Numeric data, while the time field was converted to Time type without providing any format. I also had to add another transformation step called Concatenate fields, and I set Ignore the frame name as the name. The result is what can be seen on Figure 7.8.

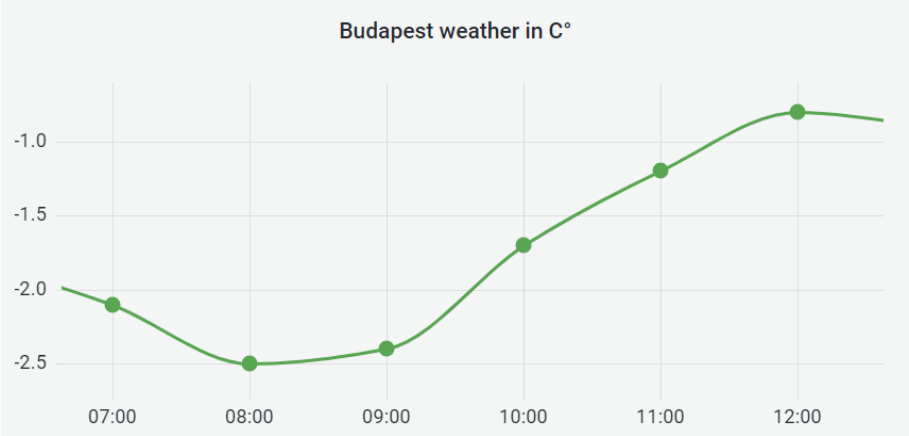

Figure 7.8: The weather JSON API visualized

The other API is a JSON API returning statistics about the COVID-19 cases in Hungary that can be seen on Figure 7.9.

The API used for this test - Apify for COVID-19 cases in Hungary [19]

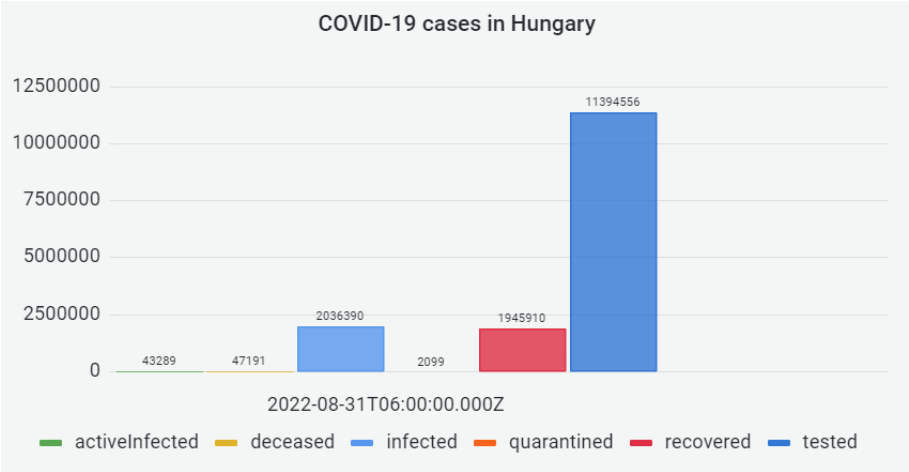

Figure 7.9: The COVID-19 JSON API visualized

7.2.2 Testing MQTT

To test the MQTT plugin, I needed to find a public MQTT broker. I discovered a few ones, but ended up using HiveMQ, because it offered an online client platform for testing purposes, allowing me to publish messages to a topic, where my Grafana can subscribe to using the MQTT data source. After selecting MQTT as the data source on Grafana, we can see a field, where we can specify the desired topic. For this test, I subscribed to a topic called testtopic/bglrk2. Meanwhile, on the HiveMQ client I started publishing random numbers at regular intervals to this topic. Figure 7.10 shows that Grafana received these messages coming from the broker and displayed the data correctly.

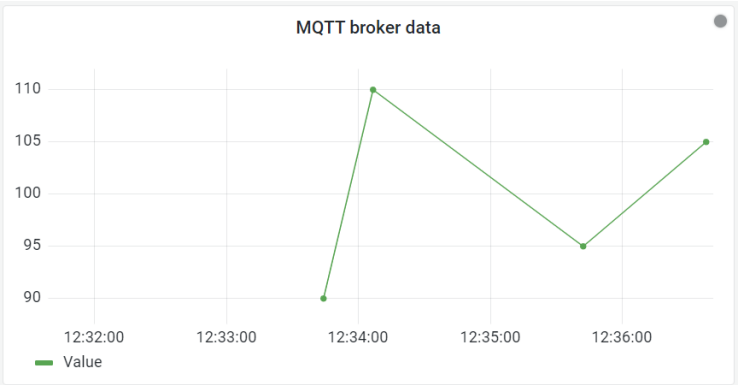

Figure 7.10: The MQTT data coming from the broker

7.2.3 Testing the built-in data sources

Telegraf, that is a server-based agent for collecting all kinds of metrics, was installed on the Kubernetes cluster for testing purposes in order to send occasionally data for InfluxDB, so I can visualize some arbitrary data on Grafana as Figure 7.11 shows.

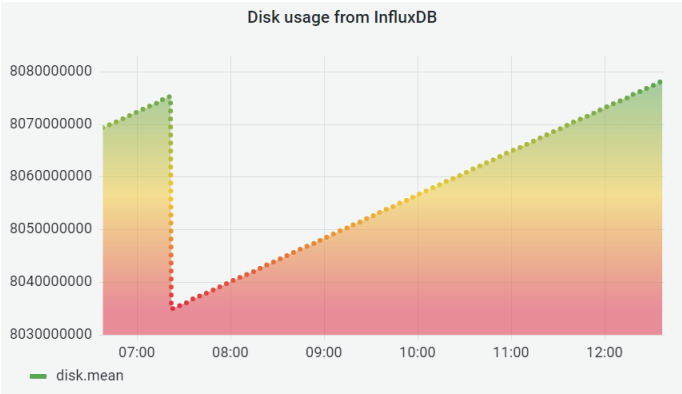

Figure 7.11: Data coming from the InfluxDB

8. FUTURE IMPROVEMENT POTENTIALS

Although the architecture is finished, I think there is still space for future improvement potentials. I would like to discuss some of the most important features that could be improved or could be added to the project.

Firstly, using persistent volumes for data storage is not the best solution in a cloud-native application design, because when a stateful containerized application is created or destroyed inside a Kubernetes cluster, it always has to know where its data is, have full access to it, and preserve its integrity. This is not possible if the stored state of an application is destroyed every time its container is taken down. This issue could be solved with the help of storage orchestrators, so this is one of the improvement possibilities I would look into and work on.

There are numerous Kubernetes-specific tools available on the internet. Monitoring tools, such as Prometheus, an open-source software application, could be added to the application in the future. Grafana already provides native support for it. Prometheus can monitor the applications inside the Kubernetes cluster, and the collected metrics can be visualized on Grafana. We can also set the Prometheus Alertmanager to notify us in case an entire cluster becomes inaccessible.

Another feature that should be considered is the usage of more namespaces. The project was deployed under one namespace, but in the future the different applications could be separated by unique namespaces to make maintenance easier. Namespaces are hidden from one other by default, although exposed services can still communicate with one another, so it would not interfere with the software's operation.

Authentication was not used or configured, thus work could be done in the future to increase the application's security. We may add more nodes to the cluster for scalability concerns as we expand the diversity of data source types.

9. EVALUATION

The application created during the thesis fulfills the specified standards as well as the functions I believe are necessary. The data visualization platform is able to connect to multiple data sources and process the incoming data in parallel. It can be easily expanded if required, and the data is displayed quickly in real time. The dashboard is user-friendly and can be customized.

During the development phase I was able to gain a lot of new knowledge and skills in containerization and data visualization. I wrote about the project's operation and details in a previous chapter. I put a significant emphasis on testing. A stable system is essential for normal operation, especially if quick decision making is needed.

It is also worth mentioning that during the thesis writing I became familiar with LaTeX, a high-quality typesetting system. This thesis was entirely written using Overleaf, that is an online LaTeX editor. Although sometimes it took some effort to find the appropriate commands for the settings, it was still a better experience than using the already well-known Word in my opinion.

I believe my experience could contribute to a future career success in IT and computer science, since the technology's popularity will continue to rise and DevOps engineers are already in high demand.

10. SUMMARY

The goal of my thesis was to create a reference architecture that is capable of supporting multiple data source visualization simultaneously and can easily be further expanded to process other data source types in the future.

In the first part of my thesis, I have explained the concept of the project by presenting the fundamentals of the topic. I have researched the different types of data sources, the emerging data visualization techniques, and the potential data visualization tools. After comparing the tools, I justified using Kubernetes, Grafana, JSON, MQTT, PostgreSQL and InfluxDB for the implementation. Finally, I have discussed the design of the application's implementation, focusing on the anticipated deployment method, the selected data sources and explaining the system design with appropriate illustrations.

In the second part of the thesis, I have documented the implementation. I have described the development phases in depth, as well as the challenges that arose during the project and how they were overcome. As a result of my work, the system described in the design plan was built and presented.

As a conclusion, I have discussed the testing and evaluation of the finished application, as well as the possibilities for future development.

11. ÖSSZEFOGLALÁS

A szakdolgozatomban egy olyan vizualizációs referencia architektúra fejlesztése volt a cél, amely képes többféle adatforrást is támogatni párhuzamosan, valamint könnyen továbbfejleszthető a jövőben újabb adatforrás típusok kezelésére is.

A szakdolgozat első részében a projekt alapjait fejtettem ki a téma bemutatásával. Megvizsgáltam a különböző adatforrások típusait, a felmerülő adatvizualizációs technikákat, valamint a szóba jöhető adatvizualizációra képes eszközöket. Az eszközök összehasonlítása után indokoltam a projekt megvalósításához kiválasztott Kubernetes, Grafana, JSON, MQTT, PostgreSQL és InfluxDB használatát. Végül az alkalmazás kivitelezésének tervezetét részleteztem, kitérve az üzembehelyezés elképzelt metódusára, a választott adatforrás típusokra, és ismertettem a rendszertervet megfelelő illusztráció használatával.

A szakdolgozat második felében az implementációt dokumentáltam. Részleteztem a fejlesztés lépéseit, továbbá a projekt során felmerülő akadályokat és megoldásukat is megemlítettem. A munkám eredményeképpen elkészült és bemutatásra került az a rendszer, amelyet a tervezésben kifejtettem.

A szakdolgozat zárásaként írtam az elkészült alkalmazás teszteléséről és az eredmények értékeléséről, valamint a továbbfejlesztési lehetőségeket vettem számításba.

12. LIST OF FIGURES

13. REFERENCES

[1] M. Chen, D. Ebert, H. Hagen, *et al.*, "Data, information, and knowledge in visualization," *IEEE Computer Graphics and Applications*, vol. 29, no. 1, pp. 12–19, 2009.

[2] OASIS. "Mqtt - the standard for iot messaging." (2022), [Online]. Available: https://mqtt.org/ (visited on 11/19/2022).

[3] Khin Me Me Thein, "Apache kafka: Next generation distributed messaging system," *International Journal of Scientific Engineering and Technology Research*, vol. 03, no. 47, pp. 9478–9483, Dec. 2014.

[4] Xiaoming Zeng, MD, PhD, and Katelyn H. Rouse, MS-HIIM, RHIA. "Using tables as a method of data visualization." (May 9, 2022), [Online]. Available: https://journal.ahima.org/page/using-tables-as-a-method-of-data-visualization (visited on 10/16/2022).

[5] Óbudai Egyetem. "Neptun." (2022), [Online]. Available: https://neptun.uni-obuda.hu/hallgato/login.aspx (visited on 11/19/2022).

[6] Yan Holtz. "Choropleth map - from data to viz." (2018), [Online]. Available: https://www.data-to-viz.com/graph/choropleth.html (visited on 10/21/2022).

[7] D. Kokkelink and M. Lorenz. "Datawrapper: Create charts and maps." (2012), [Online]. Available: https://www.datawrapper.de/ (visited on 11/19/2022).

[8] © OpenStreetMap contributors. "Openstreetmap is available under the open database licence from: Https://www.openstreetmap.org/copyright." (2015), [Online]. Available: https://www.openstreetmap.org (visited on 11/19/2022).

[9] "Introduction - superset documentation." (2022), [Online]. Available: https://superset.apache.org/docs/intro/ (visited on 10/31/2022).

[10] "What is kibana? - elastic." (2022), [Online]. Available: https://www.elastic.co/what-is/kibana (visited on 10/31/2022).

[11] "Grafana documentation." (2022), [Online]. Available: https://grafana.com/docs/grafana/latest/ (visited on 10/31/2022).

[12] J. Turnbull, *The Docker Book*. James Turnbull, 2014.

[13] J. Camisso, H. Jetha, and K. Juell, *Kubernetes for Full-Stack Developers*. New York City: DigitalOcean, 2020.

[14] ELKH Cloud. "Cloud services for national and international research projects." (2022), [Online]. Available: https://science-cloud.hu/en (visited on 12/10/2022).

[15] Docker Inc. “Install docker engine on ubuntu — docker documentation.” (2022), [Online]. Available: https://docs.docker.com/engine/install/ubuntu/ (visited on 12/01/2022).

[16] The Linux Foundation ®. “Installing kubeadm — kubernetes.” (2022), [Online]. Available: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/ (visited on 12/01/2022).

[17] Sriramajeyam Sugumaran. “Grafana infinity datasource.” (2022), [Online]. Available: https://sriramajeyam.com/grafana-infinity-datasource/ (visited on 12/01/2022).

[18] Open-Meteo.com. “Free Open-Source Weather API — Weather Forecast API for Budapest weather in C°.” (2022), [Online]. Available: https://api.open-meteo.com/v1/forecast?latitude=47.48&longitude=19.23&hourly=temperature_2m&current_weather=true&timezone=auto&past_days=7 (visited on 12/01/2022).

[19] https://koronavirus.gov.hu. “Apify API for COVID-19 cases in Hungary.” (2022), [Online]. Available: https://api.apify.com/v2/key-value-stores/RGEUeKe60NjU16Edo/records/LATEST?disableRedirect=true (visited on 12/01/2022).