UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Computer Science

Thesis Title: Tram-tastic Cloud Computing

Subtitle: A Case Study on Optimizing Cloud-based Solution for Real-time Data Processing from SL-18 Trams in Oslo.

Author: Mirza Aneeq Hassan Baig

INF-3990 Master's Thesis in Computer Science - MAY 2023

INF 3990
Tram–tastic
Cloud Computing

Acknowledgements

I would like to express my sincere appreciation to a few individuals who have been instrumental in the successful completion of my thesis. Firstly, I would like to thank my Internal Supervisor, Professor Lars Ailo Bongo, for his unwavering support and invaluable guidance throughout this academic journey. I am also grateful to my Industrial Supervisors, Alexander Øines Felipe and Meraj Kashi, for their practical insights and guidance on the practical aspects of this work.

I would like to acknowledge the contribution of Audun Føyen and Igor Krajisnik, the solution architects of the SL-18 IT project, for providing me with valuable insights and an overview of the existing system. I would also like to extend my gratitude to Farid Inkha, the leader of the Sporveien Cloud Platform team, for providing me with the resources mentioned in the collaboration contract between UiT and Sporveien and for his support during the project.

Finally, I would like to express my heartfelt thanks to my parents for their unwavering moral support throughout this academic journey. I am immensely grateful to all of them for their time, effort, and expertise, which has been invaluable in the successful completion of my thesis.

Abstract

This master’s thesis evaluates the scalability and cost-effectiveness of the AWS cloud platform used to collect and utilize data generated by the 87 digitally equipped trams. The SL-18 Cloud Platform was developed before the trams arrived, and resource configuration estimates were made to handle the data generated by the trams. However, with a few trams currently operational, it is crucial to evaluate the allocation of resources to the services based on actual data. Thus, the thesis's objective is to estimate the data generated by all 87 trams and evaluate the current resource provisioning on the AWS Cloud Platform in terms of scalability and cost. By doing so, this study will provide insights into the optimal resource allocation required for the AWS Cloud Platform to accommodate the data generated by the trams.

In this study, we use an existing Digital Twin tool for the trams to evaluate the scalability of the platform, ensuring that it can handle the load while keeping the cost low. To achieve this, the existing Digital Twin is modified to run 87 or more instances concurrently. Using this modified tool, the SL-18 IT platform, which processes real-time data from all 87 trams simultaneously, is evaluated.

We monitored the metrics of AWS services to identify any issues. Then based on measurements, we make recommendations for each service's upgrading, downgrading, or keeping the current configuration. Most services are recommended to scale down to reduce costs, while three services require scaling up to be operational. Although our process is well-defined and could be replicated by other studies, it is crucial to have in-depth discussions with the relevant teams for each service and perform repeated validations and evaluations. This is also a necessary protocol in Sporveien to present the results to the various stakeholders and implement the recommended changes. With these changes, Sporveien can save costs and most importantly have a platform capable of handling the data load of 87 SL-18 trams.

Table of Contents

List of Figures

List of Tables

CHAPTER 1: Introduction

This paper presents an industrial collaboration thesis with Sporveien AS.

1.1 The mission of Sporveien

Sporveien is the largest supplier of public transport in Norway. In the year 2021, 156 million individual journeys were provided by Sporveien to the public in Oslo, Norway. It is 100% owned by the city of Oslo. Sporveien works with vehicles, rails, stations, tunnels, buildings and signaling systems. Sporveien develops, owns, upgrades, and maintains the infrastructure related to metros and trams in Oslo and Viken.

Sporveien's mission is to provide more public transport for less money. They aim to:

- Transport as many passengers as possible.
- Providing satisfaction to the customers by sending them to their destination quickly and safely
- Providing cheaper transportation
- Being Environment friendly [1]

1.2 Tram Transportation in Oslo

As the population of Oslo continues to grow at a rapid pace, the demand for efficient and high-quality public transport has become increasingly important. Being one of the fastest growing cities in Europe, Oslo requires a robust transportation system to cater to its residents and visitors. Sporveien, as a key player in the public transport industry, operates multiple types of vehicles, including trams, which are the focus of this project. With a population of 600,000 in Oslo and around 1 million people in the metropolitan area, the trams carry out 51 million individual journeys per year in the capital city, according to Sporveien. However, this number is set to double by 2030, with an estimated 100 million individual journeys to be made. In response to this demand, Sporveien, on behalf of the city council, launched the Tram program.[2]

In the Tram Program, Sporveien planned to develop a futuristic tram service in Oslo by procuring 87 new, modern trams from a Spanish tram supplier called CAF. This agreement was signed in 2018 which is why these trams are called SL-18. This is the biggest-ever investment in trams-procurement in the history of Norway. The cost framework for this procurement is 4.1 billion NOK. The tram program also included upgradation of parking, a workshop in the Grefsen and many other small and big construction projects in Oslo. Workshop and Parking upgradation were required to accommodate new SL-18 trams into the system smoothly and an additional budget of NOK 3.2 billion was allocated for this purpose. Tram Program also included the upgradation of streets, and an additional NOK 1 billion budget was provided [3].

SL-18 trams got a lot of attention from the Norwegian media (Figure 1). The tramway network in the OSLO is one of the oldest tramway networks in the world. But it has not changed much since the fall of trams' popularity in the 1970s. Before SL-18, SL-79, and SL-95 trams are running in OSLO. So, there is another important aspect to modernize the design and features of the trams to attract more public and adhere with other important points in Sporveien's mission. [4],[5]

The design of SL-18 trams is modern (Figure 2). It has a lower floor as compared to some of the older trams which had stairs. Masking it easy for the people with wheelchairs or kids in cradles to travel in trams. They are also very advanced in terms of collecting data. Sporveien refers to them as "Computers on Rails."

Sensor data coming from a SL-18 tram:

Passenger counters, Cameras, stop buttons, Destination signs, Travel information screens, Advertising screens, Trolley motion sensor, Temperature measurement Inside / Outside, Door sensors, Location GPS, Traffic light prioritization, Software distribution, Messaging, Ticket validation, Driver information, Communication, Flight recorder backup, Configuration deployment, Active vehicle information, Emergency stop information and Tram Track Lubrication.

1.3 Information Technology (IT) and SL-18 Trams

The Sporveien IT Team is dedicated to enhancing the efficiency, maintenance, and data management of SL-18 trams by offering a range of IT services, including:

- tramTracker: The control room can use the TramTracker system to monitor the movement of trams and make informed decisions about managing the tram network.
- Real time communication with the tram and stakeholders.
- Condition monitoring of the SL-18 trams
- Data Lake: Storage of the historical data generated by SL-18 trams.
- Update Operational Data: Managing Data related to the trips of the trams.
- Mileage information of trams
- Facilitation of Data for traffic follow up.
- Tram Track Lubrication: Automatic Lubrication of the tram tracks after a specific mileage is covered.

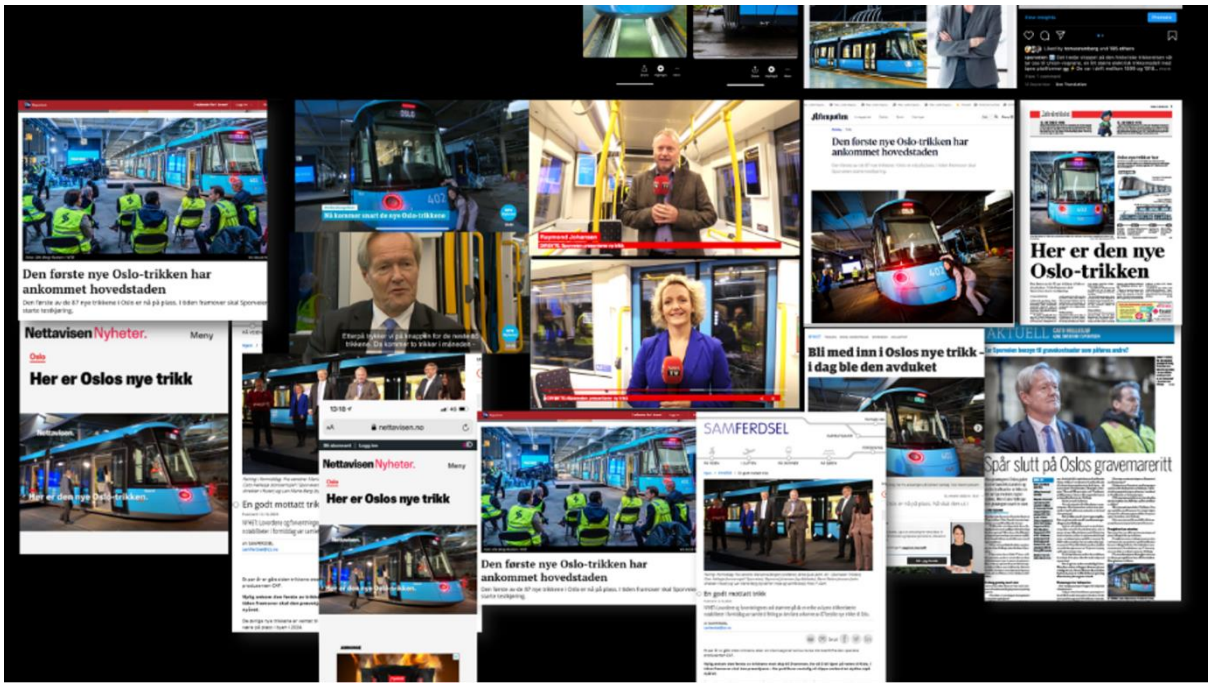


Figure 1: The arrival of the SL-18 Trams, the newest addition to Oslo's public transportation fleet, is met with ample media coverage as they make their debut in the Norwegian Capital.

Figure 2: The SL-18 Tram design highlights a modern and stylish look with a sleek exterior that exhibits sophistication and efficiency.

1.4 Problem Statement

The SL-18 trams generate telemetry data that is transmitted to a cloud-based solution developed by Sporveien for analysis. This data includes GPS location, passenger count, delay and transfer information, and other sensor data. When the SL-18 Cloud Platform was set up in 2019, an initial assessment was made to estimate the expected data volume. However, this analysis was based on rough estimates as there was no real data available at the time. As a result, resources were allocated to the services based on these estimates.

Since the project's inception, the scope has changed, and more functionalities have been added based on customer requirements. In August 2022, there were three SL-18 trams running in Oslo, but this number is expected to grow to 87 by January 2025. With this increase in the number of vehicles, it has become critical to understand the actual load and scale of the cloud-based platform beforehand. Re-evaluating the allocation of resources based on actual data is necessary. Additionally, the Sporveien IT Team has Service Level Agreements (SLAs) in place regarding the precision and latency of data with another internal team managing trams (Trikken AS). So, the objective of this thesis is to deliver:

1. Estimate the data volume produced by the 87 trams, based on the actual data coming from three operational SL-18 trams.
2. Evaluate if Sporveien’s cloud platform services and other SL-18 related services scale to the data volume of 87 SL-18 trams? Recommendations for scaling Sporveien's Cloud Platform and Services to support 87 SL-18 trams.

1.5 Thesis Summary

The SL-18 IT platform was developed in the AWS cloud platform before any vehicle arrived in Oslo. It will be necessary to evaluate this system with the aggregated data load produced by the SL-18 trams before all the trams are in service. The main aim is to ensure that the system scales by identifying any scalability issues. The estimation and correct evaluation beforehand will also be important to lower the cost.

Prior to the arrival of any tram, Sporveien estimated the amount of real-time traffic flowing through their cloud platform. This thesis aims to estimate the data load again, using data generated by existing trams, and then provide recommendations to scale the platform accordingly. However, the challenge lies in the fact that scaling is not always directly proportional, rendering theoretical estimates unreliable. In certain cases, increasing the load of one tram to that of 87 trams might require more than 87 times the platform's capacity. To address this issue, a digital twin of the tram will be used. It was developed by the lead developer at Sporveien IT team for testing purposes and this tool simulates the functionality of a single SL-18 tram. The tool will be scaled in this thesis to replicate the functioning of 87 or more trams.

The modified version of Digital Twin will be used to evaluate the existing SL-18 IT platform that processes real-time data from the 87 SL-18 trams simultaneously. There will be two sides to the problem – impacting performance and cost: Do we need to scale up certain services, or can they be scaled down?

The study [29] developed a digital twin system for a subway train and demonstrated its effectiveness in predicting the train's behavior in real-time. The use of a digital twin tool for the SL-18 trams in the current study could similarly provide a valuable testing and evaluation tool for the IT platform.

The same tools and methods can be employed for testing whenever new functionality has been created that changes real-time communication, such as introducing more frequent or larger data size of messages (volume, size), to check whether earlier estimates still hold or whether services will need to be scaled.

CHAPTER 2: System Overview

The Sporveien Cloud platform team offers a range of SL-18 Services, including a data lake for storing all the data generated by trams for analysis, and TramTracker to monitor trams in real-time, aimed at improving the efficiency of traffic managers. SL-18 Components combine to provide SL-18 Services. These SL-18 components are running on Amazon Web Services (AWS), a cloud computing platform offered by Amazon.

2.1 SL-18 Services

To explain the SL-18 Cloud Platform and the terminologies used in this thesis, this section describes two important SL-18 services. The services will serve as a foundation for the readers to understand the basic communication happening on the SL-18 platform.

2.1.1 Data Lake

Data Lake is defined as a single repository of a huge amount of data in all forms. It could be structured data which is relational data structured in the form of rows and columns, semi-structured data which includes XML, JSON and CSV files, Unstructured data which is text files, and binary data which could be images, audio, and video files. A data lake allows organizations to store, process and then analyze all their data to retrieve useful information out of it and use it for their benefit. It also allows businesses to ask new questions and get better answers for their growth. [24]

For SL-18 project in Sporveien Amazon Simple Storage Service is being used as data lake. SL-18 Data Lake is designed to store all the data which is produced in communication between SL-18 trams, Sporveien and Ruter.

A) Ruter

Ruter is another transport company responsible for providing the trip information to the SL-18 trams. This data is in the form of MQTT messages, generated by MQTT brokers.

B) MQTT

MQTT is a standard messaging protocol, or standard set of rules for machine-to-machine communication. IoT devices including sensors, and wearables use MQTT to transmit data. MQTT is used to communicate IoT data efficiently. It supports messaging between the devices to cloud and cloud to devices. MQTT is easy to implement, as it requires minimal resources for implementation. It is secure because it enables developers to use modern authentication protocols like OAuth. It is also reliable and well-supported in programming languages like Python.[25]

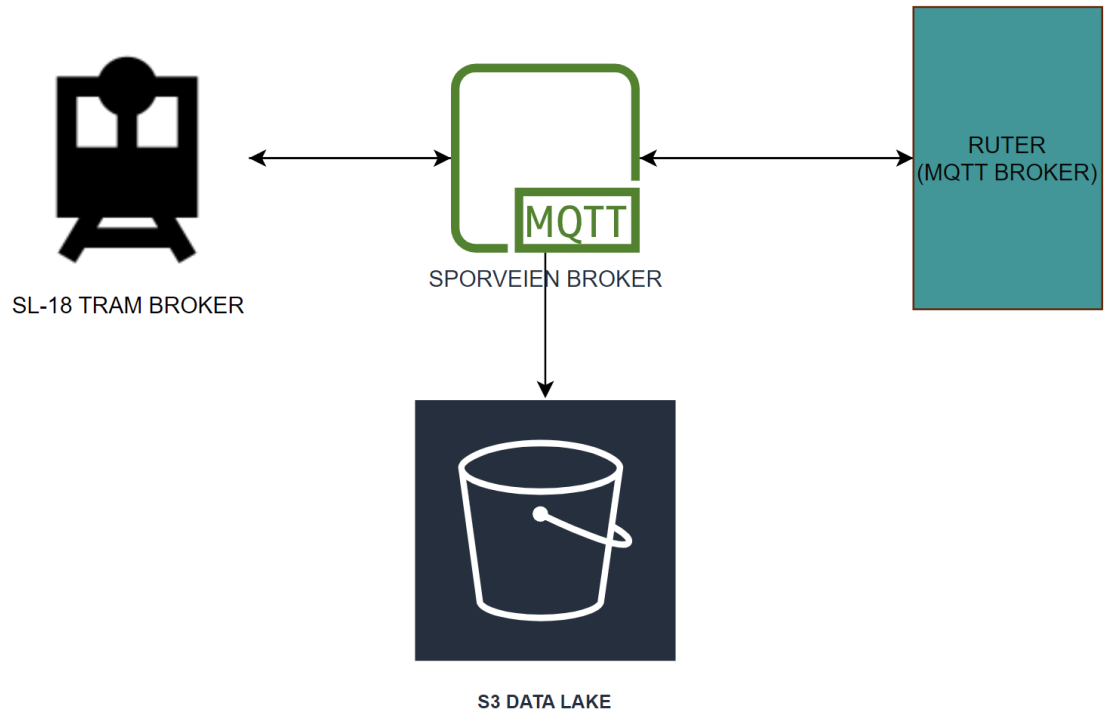


Figure 3: All three data sources, Tram Broker, Sporveien Broker and Ruter Broker sending data to the S3 DATA LAKE.

C) Data Sources

The Figure 3 shows a high-level overview of the Data Lake created for the SL-18 Cloud Platform. Data Lake has three data sources. Communication of the three data sources is simplified here.

i) SL-18 Tram Broker: This MQTT broker is implemented in trams and sends all the data collected by the sensors in the tram to the Sporveien MQTT broker. It also receives the data from Sporveien broker regarding the information of the trip/Journey.
ii) Sporveien Broker: It is acting like an intermediary and stores the entire data into the data lake.
iii) Ruter Broker: It sends the trip information which includes the starting point, ending point, number of journeys on a specific track to the Sporveien broker.

Data Lake is being used to store all the generated data.

2.1.2 tramTracker

tramTracker is a web application that tracks the trams in real-time. Traffic managers use this Application to control the traffic. It not only provides the real time location of trams on the map, but also their status if a tram is ready to go into the traffic.

2.2 Amazon Web Services

With cloud computing, computing resources such as servers, storage, and databases are delivered over the internet on a pay-per-use basis, eliminating the need for companies to build and maintain their own infrastructure. [14]

AWS offers a broad range of services, including computing, storage, databases, analytics, networking, security, and enterprise applications. These services are accessible globally, enabling companies to use them from anywhere in the world. AWS has become the industry leader in cloud computing, serving millions of customers ranging from startups to government organizations. The platform provides scalability, reliability, and flexibility to meet the needs of Sporveien. [15]

SL-18 Components are running on the following services provided by AWS.

2.2.1 Amazon EC2

Amazon Elastic Compute Cloud (EC2) is a cloud computing service provided by Amazon Web Services (AWS). EC2 allows users to rent virtual computing resources in the cloud, such as virtual servers or instances, that can be used to run various applications and workloads. [16]

EC2 instances are highly customizable and can be configured to meet various performance and capacity requirements. Users can choose from a wide range of instance types, such as general-purpose, compute-optimized, memory-optimized, storage-optimized, and GPU instances, each optimized for different types of workloads.

EC2 instances are billed on a pay-as-you-go basis, with pricing based on factors such as instance type, usage duration, and data transfer rates. This makes EC2 a flexible and cost-effective solution for organizations of all sizes, from small startups to large enterprises.[17]

2.2.2 Amazon ECS

Amazon Elastic Container Service (ECS) is a fully managed container orchestration service provided by Amazon Web Services (AWS). ECS allows users to easily deploy and manage containerized applications in the cloud using popular containerization technologies such as **Docker**.

ECS supports two different launch types for running containers: EC2 and Fargate. With the EC2 launch type, users can launch containers on a cluster of EC2 instances, while with the Fargate launch type, users can run containers without having to manage the underlying EC2 instances. This allows users to choose the most appropriate launch type based on their specific needs. ECS has become a popular choice for

organizations that need to deploy and manage containerized applications in the cloud, due to its ease of use, scalability, and flexibility. [18][19]

i) Docker

Containers are lightweight virtualization technology that package applications and their dependencies into a portable, isolated environment.[41] Unlike virtual machines, containers share the host operating system kernel and do not require a separate guest operating system. Docker is a popular containerization platform that provides tools and APIs for building, packaging, and deploying applications in containers.[42]

Containers and Docker provide numerous benefits, including improved application portability, easier deployment and scaling, and better resource utilization.[43]

2.2.3 DynamoDB

Amazon DynamoDB is a fully managed NoSQL database service provided by Amazon Web Services (AWS). DynamoDB is designed to provide high-performance, highly scalable storage and retrieval of structured and unstructured data. [20]

DynamoDB provides the following key features:

i) Fully managed service with fault tolerance.
ii) Support for both document and key-value data models.
iii) Flexible querying and indexing capabilities.
iv) Encryption at rest and in transit, as well as fine-grained access control.
v) Low-latency, high-throughput performance with the ability to handle millions of requests per second.[21]

2.2.4 Kinesis Data Streams

Kinesis Data Streams is an AWS service for collecting, processing, and analyzing real-time streaming data. It can handle large volumes of data and is scalable and fault-tolerant manner. Record Producers, like sensors or applications, can send data records to Kinesis Data Streams, which stores and replicates the data across multiple availability zones. Data records are stored in ordered sequences called data streams, which can have multiple shards to accommodate changes in data volume. Kinesis Data Streams provides automatic scaling, data retention policies, and integration with other AWS services like Amazon S3 (storage service). It is a powerful tool for real-time data processing. [30]

2.2.5 Apache Flink

Kinesis Data Analytics for Apache Flink is a fully managed AWS service that enables users to utilize Apache Flink applications for processing streaming data. Flink can perform analytics on real time data using multiple programming languages including Java, Python, Scala, or SQL. The service enables the user to write and execute the code to perform time-series analysis, feed dashboards in real time and create metrics

in real time. Apache Flink reads from the Kinesis stream, performs processing and analytics, and writes back on another stream or data store. It is a powerful tool to perform real time transformations and analytics on data as it performs in-memory computations. [7]

Chapter 3 : Requirement Analysis

The platform enables communication in both directions - from the trams to the cloud and vice versa. The SL-18 platform is quite extensive and SL-18 Components affected by the increased data load were identified. Based on the Requirement Analysis, components were classified into two categories: included and excluded. In this chapter, an overview of all the Included Components is provided, while only a few excluded components are described.

Included components are those where data travels from the trams to the Cloud Platform. These components process data from all the trams simultaneously, and they are impacted by the increased data load. Examples of included components are the transmission of data from trams to the Sporveien broker and then to AWS (Figure 4).

On the other hand, excluded components are those where data goes from the Cloud platform to the trams. For instance, the Realtime tram status component examines the status of each tram before it starts operating and communicates with each tram individually. Similarly, components that send data about weather conditions to the trams are excluded. Also, the components used by older trams (SL-79 and SL-95) are excluded.

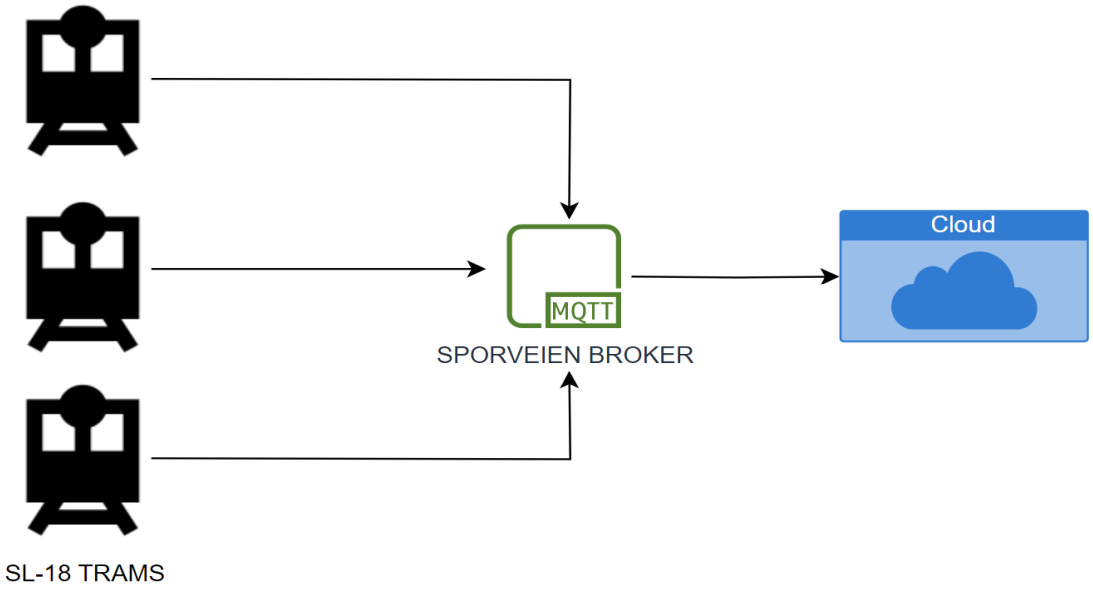


Figure 4: Trams (MQTT broker in trams) sending data to Sporveien Broker and as the number of trams increases, the data load on Sporveien Broker and Cloud Platform (includes data lake) will increase.

3.1 Included SL-18 Components

This thesis examines how increased data load affects components responsible for processing tram data. An overview of the components in the Data Lake architecture for moving data from trams to the data lake is in Figure 5.

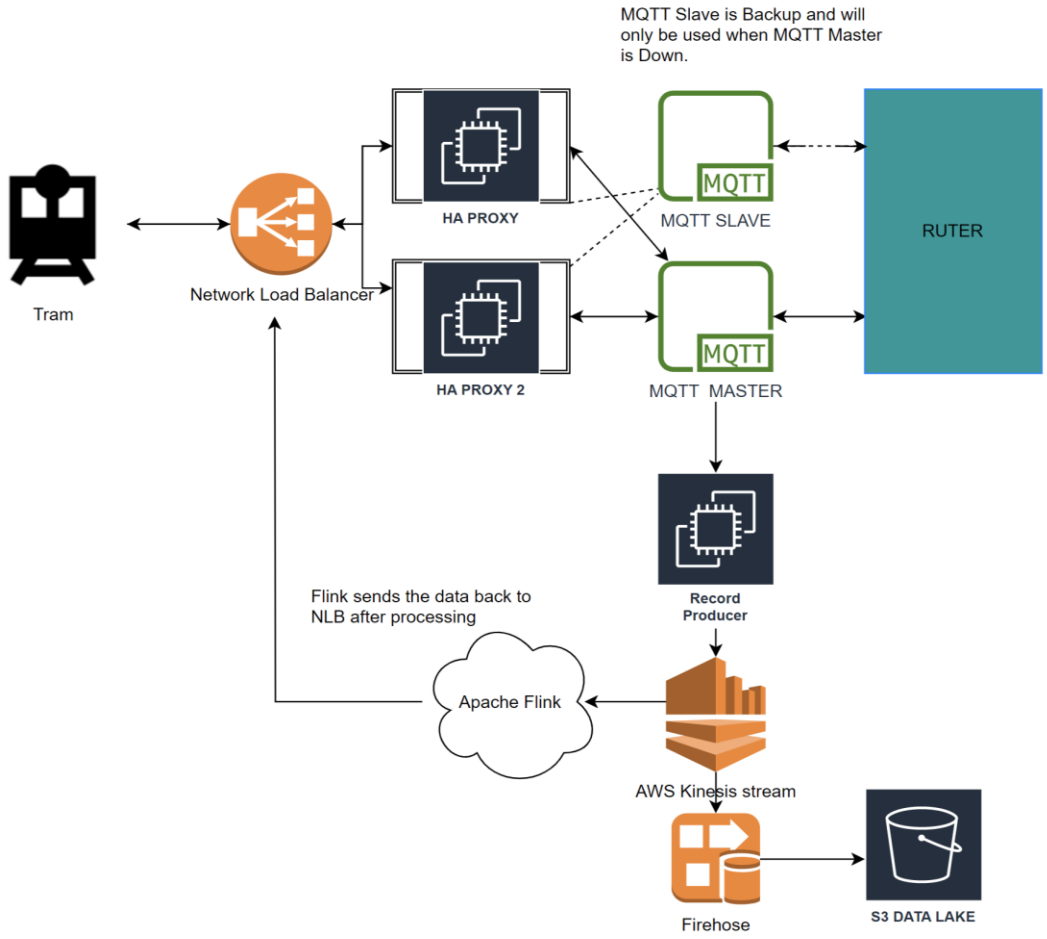


Figure 5: Flow Diagram of the shortlisted SL-18 Components for the thesis which moves data from trams to data lake.

3.1.1 MQTT Brokers

These are also called Sporveien brokers. These brokers receive all the data from the trams as well as Ruter in the form of MQTT messages and then pass to the rest of the components.

Master **MQTT** is the preferred broker, and MQTT slave is the backup. HA Proxy (described in next subsection) is handling the active-passive failover configuration of the two brokers.

Both the master and slave brokers are also connected with the **Ruter Broker** (Figure 5**)**. If the master fails all the connections get terminated and clients are redirected to

the slave by HA Proxy. When the master comes back online, all connections to the slave are served to the master. It is important to keep all the clients in the same broker.

Both brokers are running on **2.2.1 Amazon EC2**.

3.1.2 HA Proxies and Network Load Balancer

AWS Network Load balancers (NLB) do SSL/TLS (secure communication protocol) on the data that came from the tram , perform load balancing, and then send this data to the High Availability (HA) Proxies. Both HA proxies receive data and set up in active/passive configuration. These HA Proxies provide fault tolerance capabilities to the MQTT Brokers, making sure that MQTT brokers are always available. Overall, this system helps to ensure smooth communication within the tram network and helps to prevent downtime or other issues that could impact the network's reliability.

The AWS Network Load Balancer (NLB) is a load balancer designed to be highly scalable and fault-tolerant, capable of managing millions of requests per second. [6]

HA proxies are running on **2.2.1 Amazon EC2**.

3.1.3 Record Producer

The data is received by the Record Producer from **3.1.1 MQTT Brokers** (Figure 5). The Record Producer is allocated resources such as CPU and network based on the number of messages received from the MQTT Broker.

The Record Producer performs additional tasks such as message collection, segregation (e.g., separating media messages from regular messages), and message enrichment (e.g., adding tramID, QoS, and MQTT Topic to the messages).

Record Producer is Running on **2.2.1 Amazon EC2**.

3.1.4 Kinesis Data Stream

Kinesis Data Stream (KDS) is the main ingestion point where Record Producer sends the messages. Capacity in KDS is handled by the shards and it depends on the expected load during deployment.

KDS Component is an Amazon Web Service and described in **2.2.4 Kinesis Data Streams**.

3.1.5 Kinesis Firehose

Amazon Firehose prepares the kinesis streaming data before it is saved in storage. It is used to convert the raw data from the source(Kinesis Data Stream) in the formats required by the destination(S3 Data Lake), without building data processing pipelines. Amazon Firehose is fully managed service which automatically manages, scales and provide resources (memory, compute, network etc.)

Kinesis Data stream writes data to the Kinesis Firehose to aggregate, and Firehose sends the data to S3 Bucket. AWS scales it automatically according to the data load. It is described here in the context of data Lake shown in Figure 5.

3.1.6 S3 Bucket

Amazon S3 (Simple Storage Service) is a cloud-based storage service offered by Amazon Web Services (AWS) that provides object storage with high scalability, availability, security, and performance. Amazon S3 is used for data storage, archiving and data analysis.[22]

S3 is the main storage for all the messages arriving. Kinesis Firehose sends all the data to S3.

3.1.7 Apache Flink

It is a powerful tool to perform real time transformations and analytics on data as it performs in-memory computations. Kinesis Data Analytics for Apache Flink is an AWS service in it itself and an overview is provided here: **2.2.5 Apache Flink.**

The trams can be driven from either side, but there is only one antenna located on one side. This means that when the tram is driven from the other side, the antenna does not provide accurate location data. To compensate for this, an extra distance of “x” meters needs to be added to the coordinates. The correction in the coordinates is done using Apache Flink. This is just one example: Flink is being used for other purposes in the SL-18 Project. However, when this component is evaluated, it will account for all the use-cases and not just the mentioned example.

3.1.8 MQTT Receiver

It has subscribed to the Sporveien Broker (**3.1.1 MQTT Brokers**) and listen MQTT messages related to the GPS location of the trams. This component is not being used for Data Lake but for tramTracker, and that’s why it takes data only related to the GPS location to track the trams in real-time. It then publishes data to Kinesis Data Stream.

It is running on **2.2.2 Amazon ECS.**

3.1.9 MQTT Fusion

Kinesis Data Stream writes the data to MQTT Fusion published by MQTT Receiver. It then publishes those messages(data) to the DynamoDB table which is being used by tramTracker Web Application.

It is running on Amazon ECS.

3.1.10 DynamoDB

It caches the messages for tramTracker Web Application in a table called tram state. DynamoDB is also a storage service provided by AWS and details about it can be found here: **2.2.3 DynamoDB.**

3.1.11 Safety switch

Instead of the processed data in **3.1.7 Apache Flink**, it sends data without processing. It is a back-up for the GPS data stream to Ruter and tramTracker if Apache Flink fails. As aforementioned Flink compensates the extra distance of “x” meters when tram is being driven in opposite side of antenna. However, if Flink fails it is fine to get “x” meters wrong position rather than getting no GPS data at all.

It is running on Amazon ECS.

3.1.12 Snap Cache

SL-18 data lake is used to store historical data for gaining insights. However, this transaction-based storage system can take some time to return results. For services that require quick access to recent data, a short-term storage option (cache) is used. This component is called Snap Cache. This Snap Cache stores a small subset of MQTT topics and their payloads for a maximum of 24 hours. It consists of containers that read MQTT messages and store them in a document database. Originally developed for tramTracker, this Snap Cache is now used for other SL-18 services as well.

It is running on **2.2.2 Amazon ECS.**

3.1.13 Message Latency Tracker

The Figure 6 shows the conceptual design of message latency tracking on the SL-18-Platform. The Data sources are the same as mentioned in the data lake in Figure 3: All three data sources, Tram Broker, Sporveien Broker and Ruter Broker sending data to the S3 DATA LAKE. This system is built to capture the travel time of each message that goes through the platform(Figure 6). There are two containers connected to the Sporveien’s broker and the Ruter’s broker. The servers in these containers are listening to all the messages coming into these brokers. Then these containers store these messages in AWS managed database called Document DB. Document DB is connected to an API Container which is responsible for performing the calculations of message travel time and then serve this data to a User Interface which was developed in-house.

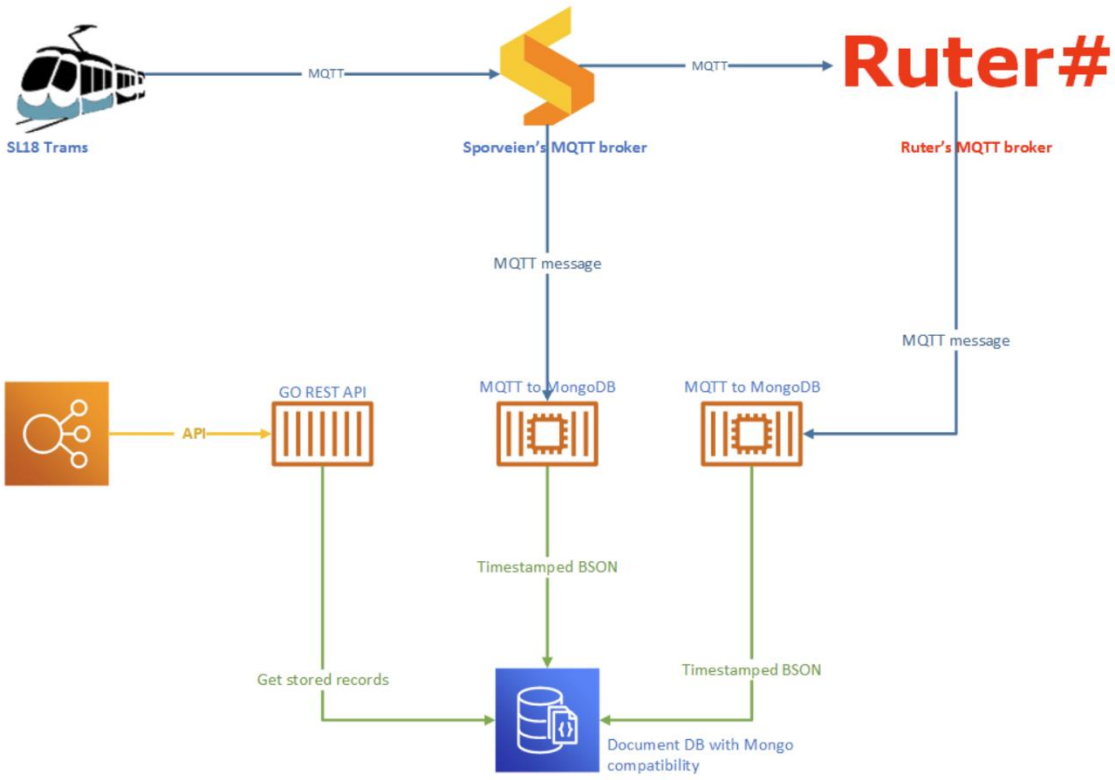


Figure 6:Message Latency Tracker to calculate the travel time from Sporveien to ruter. This architecture diagram is created by the architect of this component.

Message Latency Tracker is running on **2.2.2 Amazon ECS.**

3.2 Excluded SL-18 Components

The components mentioned here are not impacted by the data load of 87 trams. However, they are only described here to give the reader an idea about the IT team's operations.

3.1 Realtime tram status

This functionality is responsible for monitoring the tram in real-time before it goes into the traffic. It is a dynamic check which ensures that everything is all right before the kickoff of a tram. It provides the status of each tram on a User-interface (UI) developed in-house. This UI is responsive and used by technicians in handheld devices and monitoring screens. Each tram could be in following four states:

1. SHUTDOWN = Tram is turned off.
2. READY = Tram is On, and all checks are OK.

3. WARNING = Tram has checks that are failing but are not crucial.
4. ERROR = Tram is NOT ready, because key/crucial checks are failing.

It is running on **2.2.2 Amazon ECS.**

3.1.2 MQTT Ping

It is a checker if Sporveien's **3.1.1 MQTT Brokers** are working or not? It continuously sends signals to numerous services and stops sending when it loses connection with the MQTT broker. So, it provides the status of the Sporveien broker. It also tries to reconnect with MQTT Broker and starts sending again if it establishes the connection.

3.1.3 XML to JSON

It takes the XML data from the Sporveien MQTT broker and converts it into JSON format and sends it back to MQTT. This functionality was developed for the older trams because those trams were sending data in XML format. But the modern SL-18 trams send data directly in JSON format. So, this service is only being used for older trams and will not be considered for this thesis.

3.3 Metrics of Included Components

Included components are supposed to be affected by the increases data load. These components operate on various services offered by the AWS Cloud Platform, namely Amazon ECS, EC2, Apache Flink, DynamoDB, and Amazon Kinesis. This subsection describes the relevant metrics for each Included SL-18 Component and the resource utilization before the experimentation was performed.

3.3.1 SL-18 Components running on Amazon EC2

Table 1 displays the included SL-18 Components running on Amazon EC2.

The key metrics for the EC2 instances are:

1. CPU utilization: This metric measures the percentage of CPU utilization on an EC2 instance over a given period.

2. Network traffic: This metric measures the amount of incoming and outgoing network traffic on an EC2 instance. High network traffic can indicate that the instance is processing a large volume of data or that there is a network issue.

3. Memory utilization: This metric measures the percentage of memory utilization on an EC2 instance. High memory utilization can indicate that the instance is running out of memory or that there is a memory leak.

Table 1: The first column of the table lists the SL-18 Components running on EC2, while the second column presents the allocation of resources. The remaining columns illustrate the utilization of significant metrics before introducing a data load of 87+ trams.

SL-18 Components	**Allocated Resources**	**Network - in (KB/ second)**	**Network-out (KB/ second)**	**CPU Utilization**	**Memory Utilization**
MQTT Master	*C5n.xlarge*	*44*	*142*	*14 %*	*12.2%*
Record Producer Host	*C5n.xlarge*	*185*	*147*	*10.3%*	*2.57%*
HA-Proxy	*M5n.large*	*66*	*67*	*3.59 %*	*3.73%*
HA-Proxy 2	*M5n.large*	*89*	*90*	*2.46%*	*3.85%*

Amazon Elastic Compute Cloud (EC2) provides a variety of instance types to meet different workload requirements. The M5n and C5n instance families are two of the many instance types available on EC2.

1. M5n.large: It provides 2 virtual CPUs (vCPUs), 8 GB of memory.
2. C5n.xlarge: It provides 2 vCPUs, 5 GB of memory.[23]

3.3.2 SL-18 Components running on Amazon ECS

Table 2 displays the SL-18 services that operate on **2.2.2 Amazon ECS**.

Table 2: The first column of the table lists the SL-18 Components running on Amazon ECS, the second column shows the allocation of resources, and the remaining columns illustrate the utilization of significant metrics before introducing the data load.

SL-18 Components	**Allocated Resources**	**Network -in (KB / second)**	**Network -out (KB / second)**	***CPU* Utilization**	**Memory Utilization**	**Task Count**
Snap Cache	*5 vCPU, 1 GB memory*	*0.164*	*0.096*	*0.241 %*	*0.849%*	*1*
Message Latency Tracker	*5 vCPU, 1 GB memory*	*3.586*	*1.431*	*2.9 %*	*1.86%*	*1*
Safety switch	*5 vCPU, 1 GB memory*	*1.216*	*0.209*	*1.17%*	*0.781%*	*1*
MQTT FUSION	*1 vCPU, 2GB memory*	*88.322*	*99.194*	*15%*	*10.9%*	*1*
MQTT RECEIVER	*1 vCPU, 2GB memory*	*1.143*	*3.830*	*0.283%*	*3.69*	*1*

Amazon Elastic Container Service (ECS) provides several metrics that can be used to monitor the performance of containerized applications running on ECS. These metrics help to identify issues and optimize the performance of ECS tasks and services.

Some of the key metrics for ECS include:

1. CPU and memory utilization: These metrics measure the CPU and memory utilization of ECS tasks and services. High CPU or memory utilization can indicate that a task or service is under heavy load or that there is a performance bottleneck.

2. Network traffic: This metric measures the amount of incoming and outgoing network traffic for ECS tasks and services. High network traffic can indicate that the tasks or services are processing a large volume of data or that there is a network issue.

3. Task Count: A task represents a running instance of a containerized application. Each task is associated with a specific task definition that defines the container image, configuration parameters, and other details necessary to run the container. The task count in ECS refers to the number of tasks that are currently running or have been run over a specified period. This metric can be used to monitor the availability and scalability of containerized applications running on ECS.

3.3.3 Apache Flink

Table 3 presents a SL-18 service that runs on Apache Flink.

Table 3: The first column of the table lists the SL-18 Component running on Apache Flink, the second column shows the allocation of resources, and the remaining columns illustrate the utilization of important metrics before introducing the data load of 87+ trams.

SL-18 Component	**Allocated Resources**	***CPU* Utilization**	**Memory Utilization**
Apache Flink	*KPUs = 2*	*43.8%*	*60%*

Kinesis Data Analytics for Apache Flink provides automatic scaling by default. The application is automatically scaled up or down based on the amount of incoming data. This ensures that the application can handle the increased load without any manual intervention. [26] But a default configuration is provided, which scales automatically according to the load.[28]

Current Configuration Provided for Apache Flink:

Parallelism = 4

The Parallelism parameter specifies the total number of tasks that will be running simultaneously. In the current configuration, 4 tasks will be running in parallel.

Parallelism per KPU = 2

The Parallelism per KPU parameter specifies the maximum number of subtasks that can be run simultaneously on a single KPU. In this case, it is 2, which means that each KPU can run up to 2 subtasks simultaneously.

KPUs required to allocate = Parallelism/Parallelism per KPU = 2

1 KPU = *1 core CPU, 4 GB memory*

As shown in *Table 3 currently 2 KPUs are allocated to Apache Flink.*

Some of the Key metrics for Apache Flink are:

1. CPU Utilization: This measures the percentage of CPU resources being used by the system. High CPU utilization can lead to performance issues.

2. Memory Utilization: This measures the percentage of memory resources being used by the system. High memory utilization can also lead to performance issues.

3. The "number of records in" metric measures the number of input records that an operator or task receives, while the "number of records out" metric measures the number of output records that the same operator or task produces. These metrics are important for monitoring and optimizing the performance of Flink jobs, as they can help identify bottlenecks or issues in the data processing pipeline. For example, a high "number of records in" and a low "number of records out" can indicate that an operator is processing data inefficiently or is encountering errors.

3.3.4 DynamoDB

Table 4 presents a SL-18 service that runs on **2.2.3 DynamoDB**.

Table 4: The first column of the table lists the SL-18 Components running on DynamoDB, the second column shows the allocation of resources, and the remaining columns illustrate the utilization of important metrics before introducing the data load of 87+ trams.

SL-18 Component	**Allocated Resources**	**Write-capacity-units (WCUs) sum**	**Read-capacity-units (RCUs) sum**	**Write throttle events count**
DynamoDB table (tram state)	*WCUs = 1200, RCUs= 500*	*700*	*40*	*34*

Some of the key metrics of DynamoDB:

1. Read and Write Capacity Units: DynamoDB uses read and write capacity units to measure the throughput capacity of a table or index. Monitoring read and

write capacity units helps ensure that your DynamoDB workload is provisioned for optimal performance and cost efficiency.

2. Throttled Requests: DynamoDB may throttle read or write requests if the workload exceeds the provisioned capacity of a table or index. Monitoring throttled requests can help identify workload spikes or performance bottlenecks.

3.3.5 Kinesis Data Stream

Table 5 presents a SL-18 service that runs on Kinesis Data Stream.

Table 5: The first column of the table lists the SL-18 components running on Kinesis Data Stream, the second column shows the allocation of resources, and the remaining columns illustrate the utilization of important metrics before introducing the data load of 87+ trams.

SL-18 Component	**Allocated Resources**	**Incoming data rate**	**Incoming data count**	**Get records count**	**Get records rate**
Kinesis Data Stream	*12 shards*	0.044 MB /s	*7.4k*	*8k*	*0.194 MB/s*

1 shard = 1MB/s writes or 1000 writes in 1 second, 2 MB/s reads or 2000 read count per second.

It's important to consider both the size and quantity of messages being processed. For instance, if 1000 writes are achieved using only 0.5 MB of data, then the shard's write capacity would be 0.5MB/s instead of 1MB/s.

Some of the key metrics of Kinesis Data Stream:

1. **Incoming data rate** refers to the amount of data that a system receives within a specific time. It is measured in Mega Bytes per Second here.
2. **Incoming data count** refers to the total number of data items or records that are written into the system within a specific period. It is measured in terms of the number of records or data items.
3. **Get records count** is the total number of data items or records that are retrieved or read from a system or database within a specific period. It is usually measured in terms of the number of records or data items.
4. **Get records rate** measures the rate at which data items or records are retrieved or read from a system or database.

CHAPTER 4: Methodology

The focus of the thesis is to evaluate the scalability of SL-18 Cloud Platform to ensure it handles the load of 87 trams. At the start of the thesis (01-Aug-2022) only 03 trams were operational in the city. However, the number was increasing gradually and by the end of March,2023 21 trams were operational , and their data is stored in the S3 Data Lake. The data was collected by the trams, and we considered two approaches to predict the load and evaluate the system:

i) Theoretical Method (modelling)
ii) Experimental Evaluation (simulation)

4.1 Theoretical Method

to determine the maximum amount of data we use data from the three SL-18 trams currently operating (as of September 2022). It is crucial to establish the upper limit because configuring resources based on average loads will not be enough to handle data loads that exceed the average. It is, therefore, essential to ensure that the system is always operational, and take the upper bound as reference. To determine the maximum amount of data generated by a single SL-18 tram per hour, we utilized in-house monitoring tools to detect the peak times when the trams were generating more data. We then collected over 50 readings at those peak times and dates to get a representative sample, during the peak times. Table 6 showcases a few of these results, revealing that a single tram can generate a maximum of **17.5 MB** (Mega Bytes) of data per hour.

Table 6: Data generated by a single SL-18 trams in MBs collected from the Data Lake

Date	Size in MB	Tram ID	Source	Duration
29.08.2022	26	401	Data Lake	2 hours (8Am-10Am)
29.08.2022	35	402	Data Lake	2 hours (8Am-10Am)
25.08.2022	35	402	Data Lake	2 hours (8Am-10Am)
03.08.2022	35	401	Data Lake	2 hours (8Am-10Am)
04.08.2022	17.50	401	Data Lake	1 hours (11Am-12Am)
22.08.2022	17.30	401	Data Lake	1 hours (11Am-12Am)
15.08.2022	17.50	402	Data Lake	1 hours (11Am-12Am)
02.09.2022	16.70	402	Data Lake	1 hours (11Am-12Am)

4.1.1 Estimation of Resources

The initial step involved determining the amount of data transmitted by more than 87 trams. The primary goal is to assess how components shortlisted in Requirement Analysis react to this data load. This includes predicting the components' resource consumption, evaluating its current resource allocation, and recommending scaling up or down to prevent system outages and save costs.

1. Kinesis Data Stream (KDS)

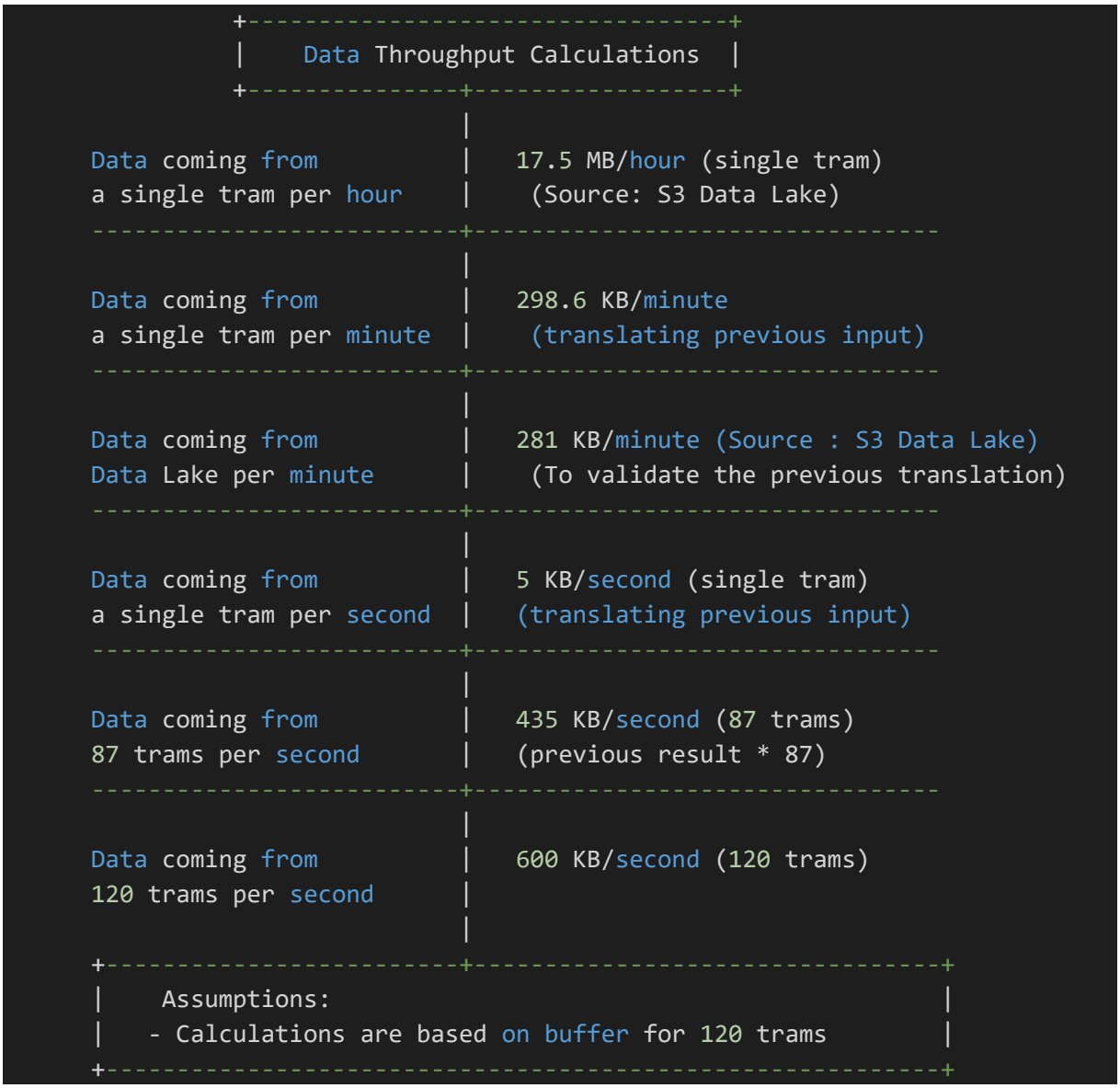
Starting with the **3.1.4 Kinesis Data Stream**. Mathematical prediction for how many kinesis shards will be required to handle the data load of 87+ trams.

Amazon offers two modes to configure the KDS:

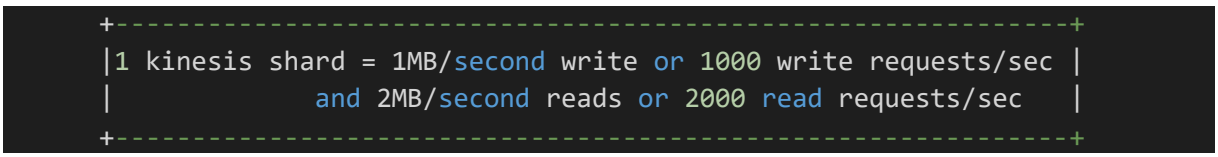
1. Auto Scaling mode
2. Provisioned mode

The Auto Scaling mode was not available during the development of the SL-18 platform, but it was considered for this study. It configures the system based on the load, eliminating the need for manual intervention. However, after comparing the costs of both modes, a significant difference was observed (Appendix 1 provides the calculations). We can save around 600 dollars per month by using Provisioned mode in the current scenario. Also, since it is possible to predict the load, and the provisioned mode was chosen.

Calculation for Kinesis shards in provisioned mode:



Data coming from 120 trams per second **= 600KB per second**



To calculate kinesis shards, simply knowing the size of the data is not sufficient. It is also crucial to determine the number of messages that the data comprises, particularly since the data is transmitted via MQTT messages. Therefore, calculating the message count is also necessary when computing kinesis shards.

4.1.2 Downsides of theoretical methodology

The study initially considered using theoretical methodology. However, upon further examination, it was discovered that this approach had some limitations:

1. The theoretical methodology was more time-consuming to calculate all the metrics for all the Components shortlisted in Requirement Analysis. It is also possible to overlook some important metrics in theoretical methodology. Furthermore, as all components are connected, it is possible to move fake data through all the services in a flow and note down metrics on a centralized dashboard.

2. Scaling could be non-linear. A service might require more than 87x capacity to operate smoothly.

3. Additionally, there might be a relationship between components which could increase the load, which was not calculated mathematically, as all components are interconnected. Mathematical models are based on assumptions and predictions, which may not hold up in such unforeseen circumstances. On the other hand, practical evaluations allow us to observe and measure the effects of several factors in the real world, providing more accurate results.[13]

So, because of the above reasons a better methodology is presented in section 4.2 which will be utilized in this study. However, theoretical methodology helped to estimate the maximum data coming out from trams.

4.2 Experimental Evaluation

To evaluate the resource utilization of Sporveien's SL-18 services on the AWS Cloud Platform, we use a digital twin of the SL-18 tram named SL-18 Twin that is used to test various functionalities by the Sporveien Cloud Platform team. It is based on pre-recorded GPS paths of "real-live" trams, along with data on the journey. Data from the existing trams and journeys has been fed to the SL-18 twin.

In this study, SL-18 Twin is modified to provide the workload for a scalability test. 120 simulations will be carried out concurrently, and the resulting data will be fed to the SL-18 services running on the AWS Cloud platform. The main objective of this approach is to gain valuable insights into the system's performance and identify areas for optimization. By conducting this Practical evaluation, recommendations for optimizing the SL-18 services can be made. These recommendations could potentially improve the utilization of resources.

4.2.1 SL-18 Twin

SL-18 Twin (digital twin) aims to simulate the functionality of an actual tram. To understand its operation, it is essential to comprehend how a tram works. When a driver enters a tram, they sign in with a block number that determines all the journeys

the vehicle will drive in one day. This information is obtained from a trip planning/scheduling system called Hastus. Once signed in, the driver receives journey information, including the starting point, ending point, and the next few stops from the current location. The driver then operates the tram, and sensors within the tram generate data continuously.

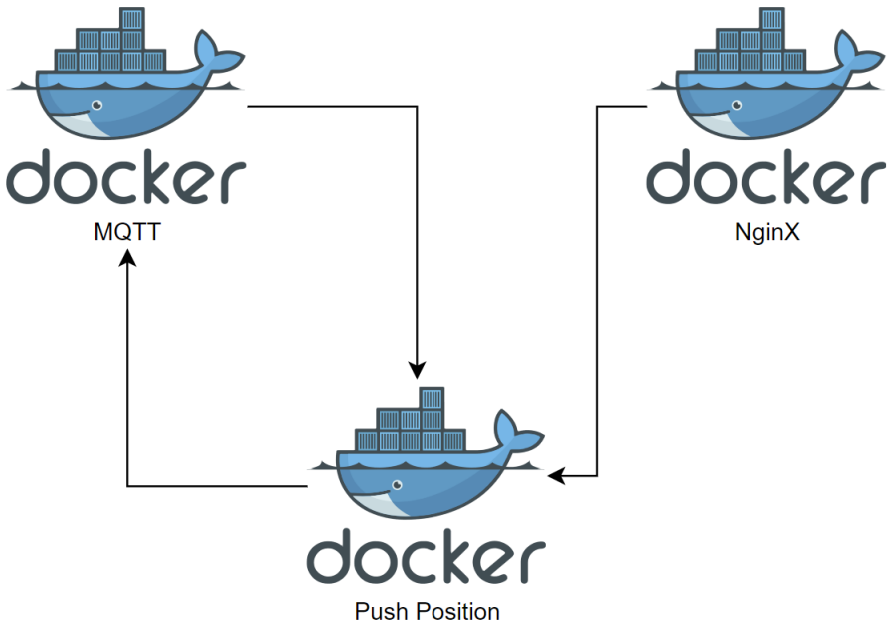


Figure 7: Architecture of Docker-Compose SL-18 Twin (Digital Twin).

SL-18 Twin uses three **Docker** containers shown in Figure 7 to simulate this functionality.

1. The first container is a Nginx webserver that downloads Hastus data from the Cloud, containing all the available blocks for the day's journey. It also includes a simulated version of MADT (Multi Application Driver Terminal), the driver assistance system for signing on to a block/journey in the Ruter's system and getting the Journey information.
2. The second container is the Push Container, which continuously sends simulated MQTT messages from the tram, including information about the tram's coordinates, speed, door-lock, and passenger count.
3. The third container is the MQTT Broker, which is a mock broker designed to test the system and has the same configuration as the central broker within the

tram. It receives tram data and travel information from the Push-Container and sends it further to the AWS Cloud Platform.

Docker-Compose is used to define and run multi-container applications. It allows the definition of application services, networks, and volumes in a single YAML file and provides a single command to start or stop the entire application.[10],[11]

The SL-18 Twin operates in the following manner:

i) First, it randomly selects a block number from hastus data and sign-in to MADT(Multi Application Driver Terminal). Then, Ruter sends trip information based on that block number.
ii) The SL-18 Twin has hundreds of data files of the tram data including starting point, ending point. These files are generated by operational trams and essentially contain pre-recorded data. Based on the trip information received from Ruter, the SL-18 Twin selects the appropriate data file and runs a simulation. The simulation runs continuously until it is stopped by the docker-compose down command.
iii) If a trip on the selected route is completed, the SL-18 Twin automatically picks up the next data file and continues running the simulation.
iv) It continuously pushes the pre-recorded data to the SL-18 Cloud Platform, like a real tram.

4.2.2 SL-18 Twin Modifications

All the containers in SL-18 Twin are running in isolated networks and communicating through various ports. These networks and ports are hard-coded in the docker-compose.yaml file and need to be made dynamic. For that purpose, we wrote a shell script which passes different network IPs, port numbers and unique value of tramID to the containers in each instance of Docker-Compose SL-18 Twin using different environment files.

4.2.2.1 Evaluation system

SL-18 Twin was modified locally on a Windows Machine, and 3 instances were run successfully. However, to run the 87+ instances mean running 261+ containers greater resource allocation was required.

AWS offers numerous services to run such docker-compose files. **2.2.2 Amazon ECS (**Elastic Compute Service) is one of such services which could be used to docker-compose files. But it was hard to convert all the parameters described in docker-compose SL-18 Twin to ECS Task definition.

Instead docker-compose on a **2.2.1 Amazon EC2** instance and no conversion of Docker-Compose was required, and it is possible to run the Docker-compose like on

a Windows machine. Also, SL-18 Twin was supposed to run for a few hours to evaluate the system, so a system with powerful specifications could be used by paying just a few dollars. So, this solution seems convenient, economical and time saving.

Appendix 2 describes the steps to run modified SL-18 Twin on EC2.

4.2.2.3 Size of EC2 to Run 87+ SL-18 Twins

The purpose of this activity was to investigate the feasibility of running many SL-18 Twin instances on an Amazon EC2 windows image. Initially, three instances of Docker-Compose SL-18 Twin were successfully run simultaneously on a low cost EC2 instance (t2.micro). t2.micro offers 1vCPU, 1GB RAM and 8GB Memory at a price of 0.0116 dollars per hour. However, when the number of SL-18 Twin instances was increased to six, an error of Insufficient disk space occurred. The SL-Twin with 120 instances required more resources but was supposed to run for just two hours. For that purpose, m5a.12xlarge was used which offers 48vCPU, 192GB RAM and 60 GB Memory at a cost of 2.3 dollars per hour. The findings show that running a considerable number of Docker Compose instances on an Amazon EC2 windows image for few hours is a viable economical option, provided that sufficient resources are allocated.

4.2.2.4 Push-container

SL-18 Twin program was running successfully on the EC2, but it was giving the invalid output. It was observed that multiple trams were being assigned to the same trip, which resulted in Ruter not sending data to all the trams for the same trip. This problem was addressed by modifying the Push-Container component of the SL-18 Twin program. The aim was to assign different trams to different journeys so that each tram receives the correct data from Ruter.

To achieve this, the Push-Container was modified to ensure an even distribution of journeys among the available tram simulations. For example, if there were 20 journeys scheduled for a day and 10 tram simulations running, the first 10 journeys would be assigned to the 10 trams, and any remaining journeys would be allocated using the modulus operator. It was crucial to ensure that each journey was assigned to only one tram, to avoid Ruter withholding journey information from other trams.

By making these changes to the Push-Container, the problem of multiple trams being assigned to the same trip was resolved, and all necessary information was received correctly. This modification allowed the SL-18 Twin program to run smoothly, without any further errors encountered.

4.2.2.5 Validation of correct operation

The simulation's accuracy was demonstrated by using MQTT Explorer to visualize data and confirm that all 87+ runs operated simultaneously, received, and transmitted correct information, and behaved as legitimate vehicles with respect to Ruter. MQTT Broker is where the GPS information is transmitted continuously by Push Container. It also shows the travel information received from the ruter. It includes the information of Destination, stops and audio announcements in the tram. MQTT Explorer is a tool to visualize all the information received in MQTT Broker. It was validated from there that trams are generating the data continuously and receiving information from ruter.

To further validate the correction, we examined the data generated by the simulations in the Data Lake during the simulation period. We observed that the 120 tram simulations generated a total of 2080 MB of data over the course of an hour (Table 7). This translates to an average of 17.6 MB of data generated by a single tram per hour, which is very similar to the theoretical result of 17.5 MB. Notably, the simulator was fed with actual data from three trams, and our theoretical estimations were also based on this data.

Table 7: Data generated by SL-18 Twins (simulations) in MBs in 1 hour collected from the Data Lake

Date	Size in MB	tramID	Source	No. of trams
14.04.2023 (13:00 - 14:00)	17.6	601	Data Lake	1
14.04.2023	17.4	602	Data Lake	1
14.04.2023	17.6	670	Data Lake	1
14.04.2023	2120	601 to 720	Data Lake	120

Sample Data Lake Query to retrieve data:

```
SELECT *
FROM sl18_mqtt_analytics m
WHERE m.year='2023' AND m.month='04' AND m.day='14'
AND (m.tramid LIKE 'SL18TIB0000000670')
AND m.recordtimestamp >= CAST('2023-04-14 13:00:00.000' AS TIMESTAMP)
AND m.recordtimestamp <= CAST('2023-04-14 14:00:00.000' AS TIMESTAMP)
ORDER BY m.recordtimestamp;
```

4.3 Monitoring

In this thesis, we used Prometheus to monitor the resource usage of the SL-18 Components.

4.3.1 Prometheus

Prometheus [8] is an open-source time series database used to collect and visualize metrics. Prometheus can collect all types of metrics directly or through various “metric exporters.” A metric can be CPU usage, memory usage, count/type of MQTT messages, or other measurable variables.

4.3.1.1 Prometheus Exporter Set-Up Guide for AWS EC2

To set up Prometheus for this study, we followed these step-by-step instructions for configuring the Prometheus exporter on AWS to collect and visualize metrics:

i) First, we set up a Prometheus server on an AWS EC2 instance and installed Prometheus, node exporter, and Grafana [9].

ii) Next, we set up the target machines (EC2 instances and ECS clusters) and labeled them as target 01 and target 02 as shown in Figure 8. Then Installed the node exporter on these target machines.

iii) Then we modified the configuration files on the Prometheus server to add the targets and updated the Grafana configuration by adding the Prometheus URL to include it in Grafana.

iv) For detailed instructions on how to set up Prometheus, we used this guide "Set up Prometheus Node Exporter on AWS EC2 Instance | by Aniket Patel | Towards AWS" by Aniket Patel. This guide provides all the necessary details and commands to complete the setup process.

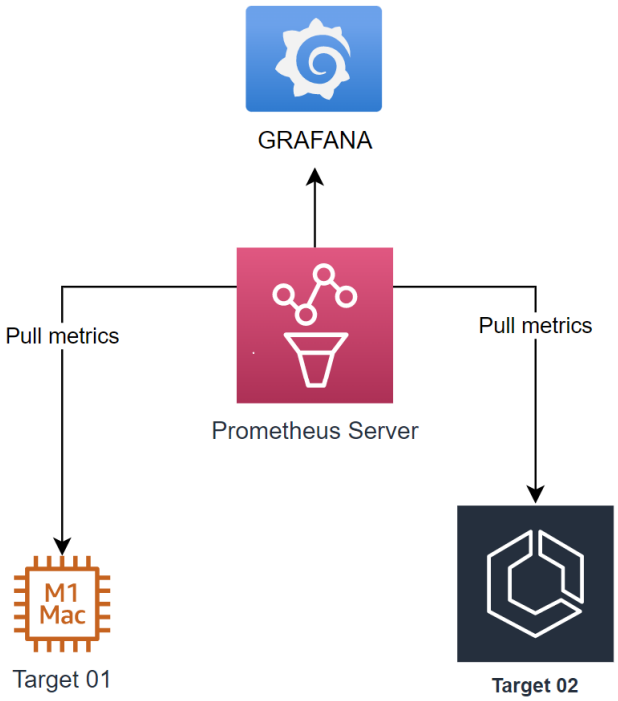


Figure 8:Prometheus Set up to capture metrics from services and integration with Grafana.

4.3.1.2 Architecture

a) Node Exporter is a configurable file that measures server and OS (Operating System) metrics, including RAM, disk space, and CPU utilization. Prometheus has node exporters that are configured into applications or containers as described in the set-up guide. It has a data retrieval worker to pull out the resource usage metrics from the application or container.

b) The collected metrics are stored in a database for time-series data, allowing customization of the analysis period. For instance, in my experimentation on 14-03-2023 and 15-03-2023, I set the time span for the entire dashboard to observe the resource utilization during that specific period.

c) Prometheus also offers a powerful query language called PromQL for complex calculations and aggregations on the collected metrics. The Prometheus Web UI accepts PromQL queries and displays the results. To visualize the results for specific services' resources, I utilized PromQL queries in my analysis.

4.3.1.3 Features

AWS has a CloudWatch monitoring service to monitor the services. But Prometheus is preferred here because some of its features, including:

1. Better Support for containers: Prometheus is designed to have native integration with containers. Cloud watch also has integration with containers but that could be more complex to set up and manage. There are some services in my requirement analysis running on ECS Containers. So, Prometheus is a better choice.

2. Multi-dimensional data model: Prometheus has a powerful multi-dimensional data model that allows the collection and storage of time series data with labels that can be used for querying and grouping. This makes it easier to slice and dice the metrics and create dashboards. CloudWatch uses a simpler namespace-based data model, which is less flexible. Although complex querying is not required for this experimentation, it might be necessary in the future.

   A sample query to find out the maximum utilization of memory by a service:

   ```
   aws_ecs_memory_utilization_maximum
   {job="SL18 Staging/Test",service_name="snapcache-ecs-service"}
   ```

3. Integration with Grafana: Prometheus can integrate with Grafana [9], a powerful visualization tool used to display dashboards for monitoring data. In this experimentation, Grafana is used to visualize all the results.

4. Real-time metrics collection: Prometheus can be used cross-platform and has real-time metrics collection, whereas CloudWatch is an AWS-specific service that shows metrics with a delay of 1-5 minutes. For example, in this thesis, digital twins were run in an EC2 instance, and it was crucial to notice the metrics in real-time when gradually increasing the number of instances to have precise resources for the EC2 instance.

5. Richer query language and broader online community: Prometheus has a richer query language and a broader online community.

Moreover, it is already being used by the Sporveien Cloud Platform team for monitoring purposes.

Chapter 5: Results

We evaluate the behavior of the SL-18 Cloud Platform when the data load was gradually increased from 30, 60, 90, to 120 trams to identify which SL-18 Components need to be scaled up, scaled down, or remain unchanged. The load was gradually increased to carefully monitor the behavior, but the results for the maximum load (120 trams which is 2120 MB per hour) are presented. Prometheus was used to track the utilization of resources by the SL-18 Components.

5.1 Experiment Results

This section describes the results of each shortlisted component after applying the data load of 120 trams. Our thesis's objective is to manage the load of 87 trams while ensuring optimal performance with lower costs. However, the SL-18 IT team is sensitive towards the operations of the services, we have decided to take some extra buffers and use 120 trams. To achieve this goal, we strive to maintain resource utilization at an optimal level, while also being willing to make some compromises on minimal cost.

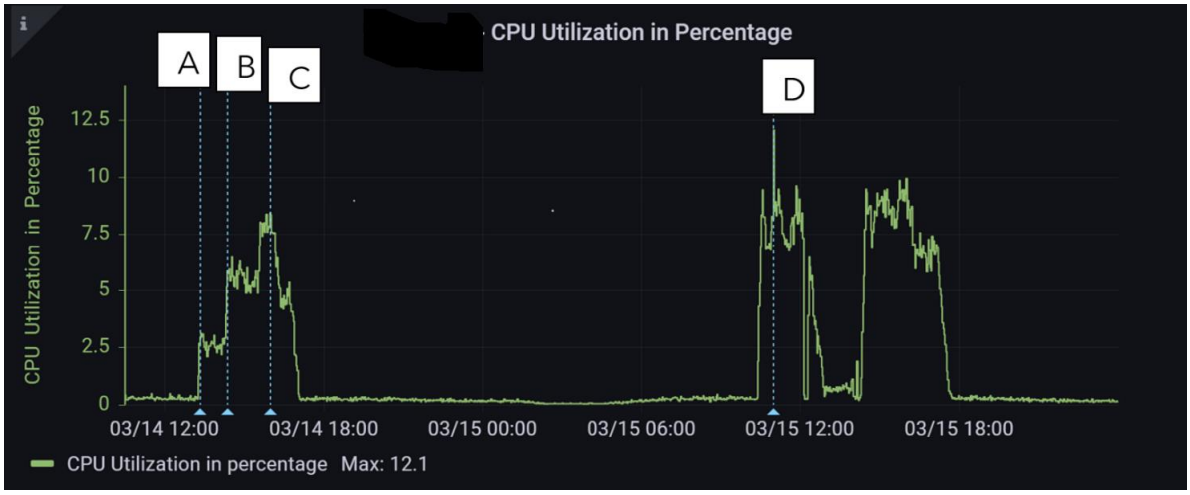


Figure 9: Maximum CPU Utilization of Snap Cache is shown on Y-axis, whereas x-axis shows the date and time, Point A, B, C, and D correspond to the data load of 30,60,90 and 120 trams, respectively.

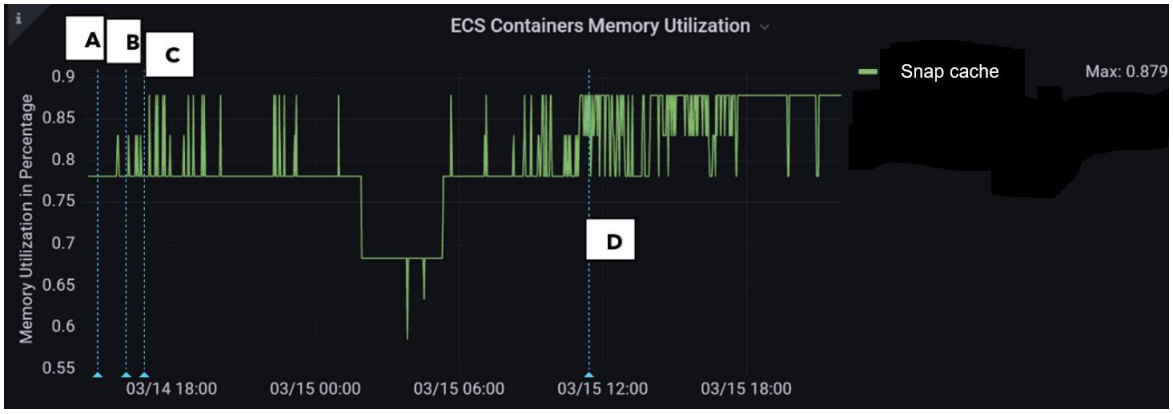


Figure 10: Maximum Memory Utilization of Snap Cache is shown on Y-axis, whereas x-axis shows the date and time, Point A, B, C, and Dare corresponding to the data load of 30,60,90 and 120 trams, respectively.

5.1.1 Results for Components Running on ECS

Figure 9 and Figure 10 shows the CPU and Memory Utilization of Snap Cache. Similarly, resource utilization for all the components was displayed in graphs. But we were only interested in maxima when the data from 120 trams (2120 MB per hour) was coming. We recorded the maximum results in the tables and presented them in this report.

Table 8 shows the maximum resource Utilization of shortlisted SL-18 Components running on Amazon ECS. When the first three components, Snap Cache, Message Latency Tracker, and Deadman's Switch components were tested with maximum data load (2120 MB per hour), their CPU utilization rates were found to be 12.1%, 16.9%, and 4.82%. There is an increase in CPU utilization after applying the load, but the percentage is low. We will define later in this chapter what should be the percentage of CPU, to scale a component.

It is worth noting that these components also exhibit low levels of memory utilization, and the currently allocated memory is already at a minimum level. Also, the Figure 10 shows that the increase in Data load is not having any impact on the memory utilization of Snap Cache, similar trend was seen for other components. The throughput for the network also went to a maximum of 265 KB/seconds in the first three components.

Table 8: Maximum Resource Utilization by ECS containers shortlisted in Requirement Analysis after applying the data load of 120 trams. The first column of the table lists the SL-18 components running on Amazon ECS, the second column shows the allocation of resources, and the remaining columns illustrate the utilization of important metrics after introducing the data load of 120 trams.

SL-18 Components	**Allocated Resources**	**Network -in (KB / second)**	**Network-out (KB / second)**	**CPU**	**Memory Utilization**	**Task Count**
Snap Cache	*5 vCPU, 1 GB memory*	*62*	*88*	*12.1 %*	*1.03%*	*2*
Message Latency Tracker	*5 vCPU, 1 GB memory*	*265*	*114*	*16.9 %*	*1.95%*	*2*
Safety switch	*5 vCPU, 1 GB memory*	*142*	*21*	*4.82%*	*0.879%*	*1*
MQTT FUSION	*1 vCPU, 2GB memory*	*187*	*874*	*79.1%*	*16.7%*	*2*
MQTT RECEIVER	*1 vCPU, 2GB memory*	*271*	*688*	*99.9%*	*3.69%*	*2*

As shown in Table 8 , MQTT Fusion and MQTT Receiver showed CPU utilization rates of 79.1% and 99.1%, respectively, after subjecting them to maximum data load.

The memory Utilization is also a bit higher than the earlier three components, but that was almost same before applying the data load and we are interested in the change. So, it could be concluded from the results that memory was not affected by the increased load.

The throughput requirements are quite high at 688 KB/second and 874 KB/second (which was less than 80 KB/s before applying the load), so these components will be configured accordingly.

5.1.2 Results for Components Running on EC2

Table 9: Maximum Resource Utilization by EC2 instances shortlisted in Requirement Analysis after applying the data load of 120 trams. The first column of the table lists the SL-18 components running on Amazon EC2, the second column shows the allocation of resources, and the remaining columns illustrate the utilization of important metrics after introducing the data load of 120 trams.

SL-18 Components	**Allocated Resources**	**Network - in (KB/ second)**	**Network-out (KB/ second)**	**CPU Utilization**	**Memory Utilization**
MQTT master	*C5n.xlarge*	*100*	*138.6*	*29 %*	13%
Record Producer host	*C5n.xlarge*	*723*	*143.6*	*28.6%*	3.05%
HA-Proxy	*M5n.large*	*240*	*65.3*	*14.9 %*	4.61%
HA-Proxy 2	*M5n.large*	*230*	*86.6*	*5.59%*	4.84%

5.1.2.1 MQTT Master

As shown in Table 9, The MQTT master's CPU consumption currently stands at 29% and shows a notable change after applying the data load. Meanwhile, the network consumption is 100 KB/s inbound and 138.6 KB/s outbound.

There was not any notable change in the memory utilization of this component. It is 13% for MQTT master, but it was 12.2% before applying the load.

5.1.2.2 Record Producer Host

The Record Producer is currently receiving messages from the MQTT broker, and due to the high load, it needs to adhere to higher network latency. During system design, it was recommended that if the anticipated load of 87+ trams result in CPU utilization higher than 60%, the Record Producer should run on a one-step higher instance than

the MQTT master to perform extra tasks, such as message decoration, validation, and network communication. However, this study found that the CPU utilization for the MQTT master, even under maximum load on the current instance, is only 29%, which means it can continue running on the same instance as the MQTT master, although we will describe later that if 29% CPU utilization is optimal. The network in and out for the Record Producer is higher than that of the MQTT broker. It is standing at 723 KB/s (was 185 KB/second before applying the load) inbound and 143.6 KB/s outbound.

There was not any notable change in memory utilization, and it stands at just 3.05%.

5.1.2.3 HA Proxies

Note that the two HA Proxies are not functioning as load balancers in this system but provide fault tolerance capabilities to the MQTT brokers. At present, HA Proxy 1 is using 14.9% of the allocated CPU, while HA Proxy 2 is using only 5.59%.

In terms of network performance, HA Proxy 1 has a network-in rate of 240 KB/s and a network-out rate of 65.3 KB/s, while HA Proxy 2 has a network-in rate of 230 KB/s and a network-out rate of 86.6 KB/s.

The memory Utilization is less than 5% for both HA proxies.

5.1.3 Apache Flink

Table 10: Maximum Resource Utilization by Apache Flink after applying the data load. The first column of the table lists the SL-18 components running on Apache Flink, the second column shows the allocation of resources, and the remaining columns illustrate the utilization of important metrics after introducing the data load of 120 trams.

SL-18 Component	**Allocated Resources**	**CPU**	**Memory Utilization**
Apache Flink	*2 KPUs = 2 core CPU, 8 GB Memory*	*66.8%*	*67.6%*

Apache Flink auto scales according to the data load and does not need a manual intervention to scale the service.

Rules to scale described in AWS Documentation [27]:

1. If CPU Utilization goes more than 75% for 15 minutes, The application automatically Scales Up.

2. If CPU Utilization goes below 10% for 6 hours, The application automatically scales down.

According to the results presented in Table 10, it can be observed that the maximum CPU utilization during experimentation never exceeded 75%, with a peak utilization of 69.8%.

Memory Utilization is 67.6%, but it remained almost same when load was gradually increased from 30,60,90 and 120 trams, dictating that load does not impact the memory.

5.1.4 DynamoDB

Table 11:Maximum Resource Utilization by DynamoDB Table after applying the data load. The first column of the table lists the SL-18 components running on DynamoDB, the second column shows the allocation of resources, and the remaining columns illustrate the utilization of important metrics after introducing the data load of 120 trams.

SL-18 Component	**Allocated Resources**	**Write-capacity-units (WCUs) count**	**Read-capacity-units (RCUs) count**	**Write throttle events count**
DynamoDB table (tram state)	*WCUs = 2500, RCUs= 500*	*2300*	*200*	*190*

Prior to the experiment, the Tram state Table in DynamoDB was allocated with a Write Capacity Units of 1200 and a Read Capacity Units of 500. However, after loading data for 90 trams, log errors were encountered because more resources were required. As a result, the Write Capacity Units were increased to 2500 per second, which resolved the issue. It was consuming 2300 write capacity units per second when data load of 120 trams was applied. The data load had minimal impact on read capacity units and a maximum of 200 Read Capacity Units per second was achieved after applying the load. The reason is that DynamoDB table is being used to store the data temporarily for **2.1.2 tramTracker** service. It is different than the rest of the services because other services are being used to move the data instead of storing it.

5.1.5 Kinesis Data Stream

Table 12:Maximum Resource Utilization by Kinesis Data Stream after applying the data load. The first column of the table lists the SL-18 components running on Kinesis Data Stream, the second column shows the allocation of resources, and the remaining columns illustrate the utilization of important metrics after introducing the data load of 120 trams.

SL-18 Component	**Allocated Resources**	**Incoming data rate**	**Put records rate**	**Incoming data count**	**Get records count**	**Get records rate**
Kinesis Data Stream	*12 shards*	0.431 MB/s	0.407 MB/s	50k per minute	176k per minute	0.458 MB/s

1 shard = 1MB/s writes or 1000 writes in 1 second, 2 MB/s reads or 2000 read count per second.

1. 12 shards can offer maximum of 12 MB/s of incoming data rate, But Table 12 shows that only 0.431 MB/s was utilized after applying the data load of 120 trams.
2. 12 shards can offer maximum of 720k records per minute of incoming data count, But Table shows that only 50k records were written after applying the data load of 120 trams.
3. 12 shards can offer maximum of 24 MB/s of Get records rate, But Table shows that only 0.458 MB/s was utilized after applying the data load of 120 trams.
4. 12 shards can offer maximum of 1440k records per minute of incoming data count, But Table shows that only 176k records were retrieved after applying the data load of 120 trams.

The results clearly show that the allocated resources are more than required.

5.2 Optimization of Metrics

Results presented in section 5.1 Experiment Results clearly demonstrate that CPU is most significantly impacted after applying the data load of trams in most of the components, since that the data is not stored, but transmitted across numerous services and it is small data. Consequently, increased processing power and network is needed to accommodate the demands of this flow, rather than additional storage.

5.2.1 Reasonable CPU Utilization

The paper [12] proposes a technique to determine the best upper utilization threshold for the provisioning process when handling a high increase in load. The technique utilizes an optimization problem with two objectives: minimizing instance usage and minimizing response time. The authors tested the method with various input loads and discovered that the optimal upper threshold for these loads is 80%. While it is also a common practice to leave a buffer rather than utilizing resources to 100%.

However, the reasonable percentage of utilization is **50%** in this study because of the following reasons:

1. There might be a malfunction in services, which could result in the generation of additional data. For example, in April 2023, a malfunction in services at Ruter caused a significant amount of data to be ingested into SL-18 services.
2. Prioritizing functionality over small cost differences is crucial. Sporveien's platform cannot afford downtime, so it is preferable to pay for a higher-level instance rather than experience downtime. Therefore, this study needs to provide an extra buffer for that.

Based on the above factors, the reasonable utilization of CPU for this study is 50%. This percentage was estimated based on the spikes that occurred after the malfunctioning of services in April 2023. Additionally, after discussing with the solution architects of the SL-18 Platform, they also recommended this percentage based on their previous experiences. This number will be utilized as a reference to make recommendations.

CHAPTER 6: Discussion

The SL-18 Cloud Platform is a large system made up of several SL-18 Components that provide IT services on the AWS cloud. To prepare for influx of data from 87 trams, the components that would be affected were identified, and the data load was estimated using information from operational trams stored in a data lake. To simulate the data generated by the trams, a Digital Twin was created by a project developer and modified for this study to run 87+ instances simultaneously. The data generated by the digital twin was ingested into the SL-18 Cloud Platform, and the performance of the shortlisted components was monitored on a unified dashboard. Results were recorded on tables and presented in this report. It was discovered that in most of the components, network and CPU were the most impacted parameters. A reasonable percentage of CPU utilization was defined.

In this discussion, it is important to highlight a crucial aspect, namely that the system's sensitivity and performance take precedence over cost. Any estimates or predictions made must consider this priority. For instance, in our experimentation testing the system with 100 trams may have sufficed, but we opted to test with 120 trams to ensure optimal performance. When selecting the CPU utilization, we also factored in this priority and provided extra buffer. However, other considerations specific to the CPU are described in the relevant section. In summary, we are willing to compromise on minimal costs to ensure the system operates effectively, while still striving to minimize costs.

Finally, we will make recommendations to consider scaling the components according to the results and finding of this study.

6.1 Recommendations

The following section provides recommendations for each SL-18 service identified during the requirement analysis phase. These recommendations are based on the experiment results and findings of this study. Findings of the study suggest that CPU is most significantly impacted after applying the data load of trams, since the data is not stored, but transmitted across numerous services and it is small data.

6.1.1 Components running on EC2

Table 13:Recommendations for components running on EC2.

SL-18 Components	Allocated Resources	Network -in (KB / second)	Network -out (KB / second)	CPU	Memory Utilization	Recommendations
MQTT master	*C5n.xlarge*	*100*	*138.6*	*29 %*	*13%*	***Scale down***
Record Producer host	*C5n.xlarge*	*723*	*143.6*	*31%*	*3.05%*	***Scale down***
HA-Proxy	*M5n.large*	*240*	*65.3*	*14.9 %*	*4.61%*	***Scale down***
HA-Proxy2	*M5n.large*	*230*	*86.6*	*5.59 %*	*4.84%*	***Scale down***

6.1.1.1 MQTT Master

The MQTT master's CPU consumption currently stands at 29%, which falls below the reasonable threshold of 50%. Meanwhile, the network consumption is 100 KB/s inbound and 138.6 KB/s outbound. The MQTT master is currently deployed on a C5n.xlarge instance, which is a compute-optimized instance with improved network throughput. AWS provides a diverse range of C5n instance types. To achieve optimal CPU utilization, it is recommended to scale down the MQTT master instance.

6.1.1.2 Record Producer Host

The CPU consumption of Record Producer currently stands at 31%, which falls below the reasonable threshold of 50%. The network in and out for the Record Producer is higher than that of the MQTT broker and stands at 723 KB/second inbound. It is currently operating on a C5n.xlarge instance, which is a compute-optimized resource with improved network throughput. To achieve optimal CPU performance without compromising the network, it is recommended to scale down the Record Producer, for example, to a C5n.large instance, as the entire range of C5n instances provides enhanced network throughput.

6.1.1.3 HA Proxies

At present, HA Proxy 1 is using 14.9% of the allocated CPU, while HA Proxy 2 is using only 5.59%, which falls below the reasonable 50% threshold.

In terms of network performance, HA Proxy 1 has a network-in rate of 240 KB/s and a network-out rate of 65.3 KB/s, while HA Proxy 2 has a network-in rate of 230 KB/s and a network-out rate of 86.6 KB/s.

Both the HA Proxies are currently running on M5n.large instances. It also provides enhanced Network throughput and sufficient to cope with the Network Load of 230KB/s and it is possible to decrease the CPU cores within the same instance family.

6.1.2 Components running on ECS Containers

Table 14: Recommendations for components running on Amazon ECS

SL-18 Components	**Allocated Resources**	**Network received (KB / second)**	**Network transmitted (KB / second)**	**CPU**	**Memory Utilization**	**Task Count**	**Recommendations**
Snap Cache	*5 vCPU, 1 GB memory*	*62*	*88*	*12.1 %*	*1.03%*	*2*	**Scale down**
Message Latency Tracker	*5 vCPU, 1 GB memory*	*265*	*114*	*16.9 %*	*1.95%*	*2*	**Scale down**
Safety switch	*5 vCPU, 1 GB memory*	*142*	*21*	*4.82%*	*0.879%*	*1*	**Scale down**
MQTT FUSION	*1 vCPU, 2GB memory*	*187*	*874*	*79.1%*	*25%*	*2*	**Scale Up**
MQTT RECEIVER	*1 vCPU, 2GB memory*	*271*	*688*	*99.9%*	*8.62%*	*2*	**Scale Up**

Table 14 shows that when the first three services, Snap Cache, Message Latency Tracker, and Deadman's Switch components were tested with maximum data load, their CPU utilization rates were found to be 12.1%, 16.9%, and 4.82%. This suggests that the CPU cores assigned to these components are more than needed to meet the 50% CPU utilization target.

It is worth noting that these components also exhibit low levels of memory utilization, and the currently allocated memory is already at a minimum level. Therefore, reducing the number of CPU cores should not have a significant impact on memory usage.

As shown in Table 14, MQTT Fusion and MQTT Receiver services have CPU utilization rates of 79.1% and 99.1%, respectively, after subjecting them to maximum data load. These rates are significantly higher than the optimal threshold of 50% and may lead to potential outages or performance issues in the system.

Therefore, it is recommended to scale up these services to avoid such problems. Scaling up will increase the number of CPU cores allocated to these services, allowing them to handle the increased workload without causing any disruptions. By doing so, the system can ensure that it continues to operate smoothly and efficiently even under heavy data loads of 87+ trams.

6.1.3 Apache Flink

Table 15: Recommendations for Apache Flink

SL-18 Component	**Allocated Resources**	**CPU**	**Memory Utilization**	**Recommendation**
Apache Flink	*2 KPUs = 2 cores of CPU and 8GB memory*	*66.8%*	*67.6%*	*Remain Unchanged*

As discussed in detail in the results of the study, it could be concluded that the resource allocation for this service can remain unchanged, as it can auto scale itself.

6.1.4 DynamoDB

Table 16: Recommendation for DynamoDB Table.

SL-18 Component	**Allocated Resources** Before experiments	**Allocated Resources** After experiments	**Recommendation**
DynamoDB table (tram state)	*Write Capacity Units = 1200, Read Capacity Units = 500*	*Write Capacity Units = 2500, Read Capacity Units = 500*	***Already Scaled Up***

This service was scaled up during the experimentation process. Table 16 shows the resources scaled up after the experimentation.

6.1.5 Kinesis Data Stream

Table 17: Recommendation for Kinesis Data Stream

SL-18 Component	**Allocated Resources**	**Incoming data rate**	**Put records rate**	**Incoming data count**	**Get records count**	**Get records rate**	**Recommendation**
Kinesis Data Stream	*12 shards*	0.431 MB/s	0.407 MB/s	50k per minute	176k per minute	0.458 MB/s	Scale Down

Based on the study results, it is evident that the current resource allocation for the component is excessive. In addressing the problem statement, the study recommends scaling down the service. Furthermore, the findings suggest that around 2 shards would be enough to handle the data load. However, it is important to acknowledge that an exact resource allocation cannot be recommended at this time. Further discussions are necessary to validate the numbers, and additional evaluations need to be conducted. It is also a protocol in Sporveien to present the results to stakeholders for approval to implement the recommended changes.

It's worth noting that many of the components have been allocated more resources than they require due to the SL-18 platform's sensitivity. This decision was made to ensure the platform remains operational and doesn't compromise performance at the cost savings. Furthermore, it was based on just theoretical data and methodology. However, the primary goal of this study is to provide recommendations to optimize the cloud platform's scaling capabilities to efficiently handle the data load of 87 trams based on actual data and methodology. It's also important to fine-tune resource allocation to achieve cost savings without compromising on performance.

6.2 Related Work

The aim of this master's thesis is to improve utilization of AWS Cloud platform resources by reducing costs without sacrificing performance and ensuring scalability. During our research, we came across papers with similar titles, but they had different use-cases and approaches which are not applicable to this case. For example, a paper titled "Cost-Aware Cloud Metering with Scalable Service Management Infrastructure" [44] appeared to be relevant to our study, but it differed significantly from our approach. The authors of that paper used machine learning to optimize resource usage based on pre-existing data, while we had to generate data for our study.

6.2.1 Digital Twins

Digital twins have emerged as a powerful tool for simulating and testing complex systems in a virtual environment. The concept originated in the aerospace industry in the early 2000s, where virtual replicas of aircraft systems were created to predict their behavior and performance under various conditions.[32] The term "digital twin" was coined by Michael Grieves, a professor at the University of Michigan, to describe this concept in 2002. Grieves envisioned digital twins as a bridge between the physical and virtual worlds, enabling engineers to design and test systems more efficiently and effectively.[33]

Since then, digital twins have been applied to various fields, including healthcare, energy, and production lines, enabling them to optimize their operations and reduce their environmental impact.[34] The rise of Industry 4.0 and the Internet of Things (IoT) has further fueled the development of digital twins, as sensors and other IoT devices allow data to be collected in real-time, improving the accuracy and precision of the virtual models. [35]

In recent years, the concept of digital twins has gained increasing attention in various industries due to its potential for improving efficiency, reducing costs, and enabling innovation. Digital twin technology involves creating a virtual replica of a physical system, object, or process, which can be used to monitor, simulate, and control the behavior of the physical system in real time [36].

One valuable contribution to the field of transportation systems is the Mobility Digital Twin (MDT) introduced by Wang and Gupta in their paper [31]. The MDT is a virtual replica of a real-world mobility system that uses real-time data to simulate the behavior of the physical system, and has potential applications in route optimization, traffic management, and vehicle fleet management.

Similarly, our study utilizes a digital twin to simulate the behavior of digitally equipped trams and evaluate the scalability and cost-effectiveness of the AWS cloud platform used to collect and utilize data generated by the trams. Our digital twin consists of multiple components that simulate the trams' behavior and need increased power to operate, and the use of a cloud platform provides increased computational or storage power for the digital twin.

The paper by Ziran Wang [37] proposes a framework for utilizing digital twin technology and cloud computing infrastructure to process and analyze real-time data generated by vehicles for advanced driver assistance systems (ADAS). While this study focuses on vehicle-to-cloud communication and ADAS systems, our study focuses on evaluating the behavior of a complex IT platform against an increased data load, highlighting the potential of digital twins as a tool for evaluating and optimizing complex IT systems in various fields, especially transportation.

The challenge of scaling the Digital Twin for evaluating the SL-18 IT Platform was highly specific to the problem and resources presented by Sporveien. Despite thorough research, no relevant literature was found on this topic. The scaling process was carefully adapted to meet the requirements of the evaluation. There were many errors and issues encountered but those were very specific to the system.

Overall, these studies contribute to the growing body of research on digital twin technology and its potential applications in various industries, offering benefits such as improved performance, reduced downtime, saving costs and increased efficiency.

6.2.2 Monitoring on Cloud Platforms

The paper [40] "Monitoring IaaS Using Various Cloud Monitors" by Stephen and Absa explains the significance of monitoring resource utilization in Infrastructure-as-a-Service (IaaS) clouds. The authors stress that tracking resource utilization metrics is crucial to assess the overall health and performance of the cloud infrastructure and to find areas for improvement. Overall, this paper emphasizes the significance of resource utilization monitoring in cloud computing.

This study describes the process of setting up a monitoring system to record the resource usage of each component of SL-18 on AWS Cloud Platform. Then this study thoroughly discusses the importance of the chosen monitoring tool.

6.2.3 Resource Utilization on Cloud

The literature search on resource utilization yielded results aimed at cloud service providers. However, it is crucial for us as users of these services to focus on using resources efficiently, rather than developing the platforms themselves. For instance,

the review [38] "Energy Efficient Live Virtual Machine Provisioning at Cloud Data Centers - A Comparative Study" by Soni, Shalini (2015), published in the International Journal of Computer Applications, discusses the energy efficient provisioning of resources. Our study does not cover how cloud platforms develop their services but emphasizes how to use the existing services more efficiently.

The paper [39] by Buyya et al. suggests techniques for efficient resource provisioning in cloud computing environments. These techniques involve workload characterization, which analyzes resource requirements and usage patterns of applications running in the cloud, and predictive resource provisioning, which forecasts future resource requirements based on workload characterization. The authors also recommend SLA-based resource allocation techniques, such as differentiated resource allocation and priority-based scheduling, to allocate resources based on the importance of applications and SLA (Service Level Agreements) requirements. By using these techniques, cloud services users can achieve efficient resource management while meeting SLA requirements.

Our study aimed to assess the use of resources of the Amazon Web Services (AWS) cloud platform. To do this, we used a digital twin to simulate the behavior of digitally equipped trams that transmit data to a cloud platform. We did not use predictive modeling or SLA techniques mentioned in the paper by Buyya et al. to allocate resources or evaluate the utilization of already allocated resources. Instead, we monitored the maximum data generated by the existing trams and evaluated the resource utilization by the components. Furthermore, the amount of data coming from trams is constant unlike other studies where the amount of data fluctuates. Based on our findings, we made recommendations on whether already provisioned resources need to be scaled up or down or remain the same.

Although our study did not use the predictive modeling or SLA techniques mentioned in the paper [39] by Buyya et al., we were still able to provide insights into how components can be efficiently provisioned with resources in cloud computing environments. Our findings complement the research done by Buyya et al. and highlight the importance of efficient resource allocation in cloud computing. This can lead to cost savings and improved performance, making it a crucial area of research for cloud computing users.

6.3 Challenges Faced

During this thesis, we encountered various challenges that we had to overcome to achieve our objectives. One of the significant hurdles was understanding the existing SL-18 Cloud Platform, which was complex, extensive, and constantly changing based on customer requirements. We had to simplify it and provide a brief overview to the reader, which was another hurdle.

Another challenge we faced was finding relevant literature. While many papers with similar objectives already had data available, they used machine learning models to predict resource usage. In contrast, our study focused on optimizing AWS resources in terms of cost and performance by estimating the maximum possible data output, which does not change drastically, from the 87 digitally equipped trams. As a result, we divided the thesis and found literature relevant to each aspect of the study.

In addition, we also faced difficulty modifying the digital twin as it was developed by someone else on a different Personal Computer. Running the digital twin on our Personal Computer and scaling it for the 87+ simulations was another significant obstacle. However, we overcame these challenges and were able to run the digital twin on a cloud platform. We also pushed the code to a version control platform (GitHub) and made it easier for anyone within Sporveien to run it again on any cloud platform. Due to security and data concerns, we have not shared the code for this study.

Despite all these challenges, we were able to successfully solve the problem.

6.4 Future Work

While this thesis offers recommendations for efficiently providing resources to the SL18-components, determining the precise allocation of resources such as CPU cores, memory, and network capacity is outside the scope of this study. Instead, the Sporveien Cloud Platform team will use the report's recommendations as a basis for discussing the accurate allocation of resources to each component. They will then perform repeated validation and evaluations of each component in a test environment on the recommended changes. After evaluating the entire SL-18 Cloud Platform to ensure it meets the Service Level Agreements, the report's additional findings will be presented to Sporveien's various stakeholders to seek approval. Finally, the necessary changes will be implemented in the production environment.

In addition, while the initial title of the thesis was SL-18 Data Analysis, it has changed because it specifically focuses on scaling the cloud infrastructure to collect data and provide other services. At present, only around 21 trams are operational, but when all 87 trams are in operation and data is stored for a longer period, numerous use cases could be developed to benefit travelers. For instance, data analysis could inform travelers when there is no more space left for cradles or wheelchairs on the current tram and advise them to wait for the next one. The data could also be analyzed to identify the routes that are more crowded and increase the number of trams on those routes. While these are only a few examples of how data analysis could enhance the travel experience, there are many other use cases that could be developed to fulfill the mission of Sporveien and provide greater value to its customers.

6.5 Conclusion

The SL-18 Cloud platform is a big system that provides IT services on the AWS cloud. To prepare for an influx of data from 87 trams, the system's components were identified and analyzed that were supposed to be affected by the data load. Digital Twin was modified to simulate the data generated by the trams. The study observed the system's real-time processing of the simulated data, established a monitoring system, and presented the results in the form of tables. The study found that in most of the component’s network and CPU were the most affected parameters, and reasonable percentage of CPU utilization was identified and solution to cope with network requirements was presented. Based on these findings, we recommend changes to the components to save costs, ensure scalability and optimal performance.

Overall, the thesis emphasizes the need for evaluating the system with the aggregated data load produced by all trams before they are in service and fine-tuning resource allocation to optimize performance and costs. The thesis concludes that the use of a Digital Twin tool for testing and evaluation can provide valuable insights into the optimal resource allocation required for the AWS Cloud Platform to accommodate the data generated by the sensors (trams).

This research provides valuable insights into how to handle data in complex IT systems. It contributes to the existing body of knowledge by providing a detailed analysis of the performance of an IT system that handles data from multiple sources, including sensors on trams. The results of this study can be applied to other systems with similar characteristics, helping organizations to evaluate their Cloud Platform to handle data and provide numerous services. Therefore, this study encourages the further exploration of research in this area.

References

1. Sporveien AS. About Sporveien. Retrieved September 17, 2022, from https://www.sporveien.com/about-sporveien
2. Fremtidens Byreise. New trams for Oslo. September 18, 2022, from https://www.fremtidensbyreise.no/en/prosjekter/new-trams-for-oslo/
3. Fremtidens Byreise. Program to purchase new trams. Retrieved January 3, 2023, from https://www.fremtidensbyreise.no/en/prosjekter/new-trams-for-oslo/purchase-of-new-trams
4. Distrita. (2018, June 28). Finally, Oslo is getting new trams. Retrieved August 12, 2022, from https://distrita.com/news/finally-oslo-is-getting-new-trams/
5. NRK. (2021, January 10). Here is the new Oslo tram. Retrieved September 5, 2022, from https://www.nrk.no/osloogviken/her-er-nye-oslo-trikk-1.15381717
6. Amazon Web Services, Inc. Network Load Balancer. Retrieved January 25, 2023, from https://aws.amazon.com/elasticloadbalancing/network-load-balancer/
7. Kinesis Data Analytics for Apache Flink: How It Works, https://docs.aws.amazon.com/kinesisanalytics/latest/java/how-it-works.html.
8. Logz.io. A Prometheus tutorial for system & Docker monitoring. Retrieved February 20, 2023, from https://logz.io/blog/prometheus-tutorial/
9. Grafana Labs. Introduction to Grafana. Retrieved December 5, 2022, from https://grafana.com/docs/grafana/latest/introduction/
10. Docker. Use Docker Compose. Retrieved March 1, 2023, from https://docs.docker.com/compose/
11. Docker. Use multi-container apps. Retrieved February 2, 2023, from https://docs.docker.com/get-started/07_multi_container/
12. Al-Haidari, F., Sqalli, M., & Salah, K. (2013). Impact of CPU utilization thresholds and scaling size on autoscaling cloud resources. In 2013 IEEE 5th International Conference on Cloud Computing Technology and Science (pp. 256-261). IEEE.
13. O'Dea, R. (2020). Practical evaluation: A better way to assess and improve project outcomes. Journal of Infrastructure, Policy, and Development, 4(2), 1-18.
14. Amazon Web Services. What is AWS? Retrieved October 10, 2022, from https://aws.amazon.com/what-is-aws/
15. Amazon Web Services. Overview of Amazon Web Services. Retrieved October 17, 2022, from https://aws.amazon.com/about-aws/
16. Amazon Web Services. Amazon Elastic Compute Cloud (EC2). Retrieved November 1, 2022, from https://aws.amazon.com/ec2/
17. Janakiram, M. S. V. (2015, August 24). Amazon Elastic Compute Cloud (EC2): A comprehensive guide. InfoQ. Retrieved November 20, 2022, from https://www.infoq.com/articles/aws-ec2-guide/
18. Amazon Web Services. Amazon Elastic Container Service (ECS). Retrieved December 15, 2022, from https://aws.amazon.com/ecs/
19. Amazon Web Services. Getting started with Amazon ECS. Retrieved January 25, 2023, from

https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ECS_GetStarted.html
20. Amazon Web Services.Amazon DynamoDB. Retrieved February 10, 2023, from https://aws.amazon.com/dynamodb
21. Amazon Web Services. (2019). Getting started with Amazon DynamoDB. Retrieved March 1, 2023, from https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html
22. AWS Documentation. (2022). Amazon S3 Overview. Retrieved from https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html
23. Amazon Web Services, Inc. Amazon EC2 instance types. Retrieved January 25, 2023, from https://aws.amazon.com/ec2/instance-types/
24. SAS. (2016, February 3). Analyzing the data lake - The Data Roundtable. Retrieved October 30, 2022, from https://blogs.sas.com/content/datamanagement/2016/02/03/analyzing-the-data-lake/
25. Amazon Web Services. (2021). What is MQTT? - MQTT Protocol Explained. Retrieved December 30, 2022, from https://aws.amazon.com/mqtt/
26. Amazon Web Services. (2021). Amazon Kinesis Data Analytics for Apache Flink product page. Retrieved February 28, 2023, from https://aws.amazon.com/kinesis/data-analytics/flink/
27. Amazon Web Services. (2019, July 12). Scaling Kinesis Data Analytics for Apache Flink Applications. Retrieved March 5, 2023, from https://aws.amazon.com/blogs/big-data/scaling-kinesis-data-analytics-for-apache-flink-applications/
28. AWS. Scaling Kinesis Data Analytics for Apache Flink Applications. Retrieved April 28, 2023, from https://aws.amazon.com/blogs/big-data/scaling-kinesis-data-analytics-for-apache-flink-applications/
29. Dirnfeld, R. (2022). Digital Twins in Railways: State of the Art, Opportunities, and Guidelines.
30. Amazon Kinesis Data Streams Overview. Amazon Web Services, Inc. Retrieved Feb 25, 2023, from https://aws.amazon.com/kinesis/data-streams/
31. Wang, Z., Gupta, R., Han, K., Wang, H., Ganlath, A., Ammar, N., & Tiwari, P. (2022). Mobility digital twin: Concept, architecture, case study, and future challenges. *IEEE Internet of Things Journal*, *9*(18), 17452-17467.
32. Glaessgen, E., & Stargel, D. (2012, april). The digital twin paradigm for future NASA and US Air Force vehicles. In 53rd AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics, and materials conference 20th AIAA/ASME/AHS adaptive structures conference 14th AIAA (p. 1818).
33. Grieves, M. (2014). Digital twin: manufacturing excellence through virtual factory replication. White paper, 1(2014), 1-7.
34. Tao, F., Zhang, H., Liu, A., & Nee, A. Y. (2018). Digital twin in industry: State-of-the-art. IEEE Transactions on industrial informatics, 15(4), 2405-2415.
35. Wang, S., Wan, J., Zhang, D., Li, D., & Zhang, C. (2016). Towards smart factory for industry 4.0: a self-organized multi-agent system with big data-based feedback and coordination. Computer networks, 101, 158-168.

36. Fuller, A., Fan, Z., Day, C., & Barlow, C. (2020). Digital twin: Enabling technologies, challenges, and open research. *IEEE access*, *8*, 108952-108971.
37. Wang, Ziran. “A Digital Twin Paradigm: Vehicle-to-Cloud Based Advanced Driver Assistance Systems.” *Vehicular Technology Conference*, 2020, pp. 1–6, doi:10.1109/VTC2020-SPRING48590.2020.9128938.
38. Soni, Shalini. “Energy Efficient Live Virtual Machine Provisioning at Cloud Data Centers - A Comparative Study.” International Journal of Computer Applications, vol. 125, no. 13, Sept. 2015, pp. 37–42, doi:10.5120/IJCA2015906173.
39. Buyya, R., Garg, S. K., & Calheiros, R. N. (2011, December). SLA-oriented resource provisioning for cloud computing: Challenges, architecture, and solutions. In 2011 international conference on cloud and service computing (pp. 1-10). IEEE.
40. Stephen, A., Benedict, S., & Kumar, R. A. (2019). Monitoring IaaS (Infrastructure as a Service) using various cloud monitors. Cluster Computing, 22(Suppl 5), 12459-12471.
41. Felter, W., Ferreira, A., Rajamony, R., & Rubio, J. (2015, March). An updated performance comparison of virtual machines and linux containers. In 2015 IEEE international symposium on performance analysis of systems and software (ISPASS (International Symposium on Performance Analysis of Systems and Software)) (pp. 171-172). IEEE.
42. Bernstein, D. (2014). Containers and cloud: From lxc to docker to kubernetes. IEEE cloud computing, 1(3), 81-84.
43. Merkel, D. (2014). Docker: lightweight Linux containers for consistent development and deployment. Linux Journal, 2014(239), 2-11.
44. A. Anwar, A. Sailer, A. Kochut, C. O. Schulz, A. Segal and A. R. Butt, "Cost-Aware Cloud Metering with Scalable Service Management Infrastructure," 2015 IEEE 8th International Conference on Cloud Computing, New York, NY, USA, 2015, pp. 285-292, doi: 10.1109/CLOUD.2015.46.

Appendices

Appendix 1

Kinesis Data Streams is a popular service used to ingest and process large volumes of streaming data in real-time. In this comparison, we focus on two modes of Kinesis Data Streams: On-Demand and Provision.

Assuming a data rate of 1000 records per second, with each record being 3KB in size and one month, we calculate the costs for both modes.

In the On-Demand mode, we find that the total cost of data ingestion for one stream is **$621.84**. This includes the data-write charges of $593 and the data stream cost of $28.84.

On the other hand, in the Provision mode, we need three shards to handle the data ingestion rate of 3MB per second. The cost for three shards per day, including write and read operations, is $0.36. Over one month, this amounts to $33.48. Additionally, the cost for putting payloads is $0.014 for 100,000 units of 25KB each.

For the same period, on the same size and quantity of data Autoscaling mode costs 621.84 $ whereas Provisioned mode costs less than 33.5$. Although there were some conversions that were made to equalize days, minutes, and seconds. While it is difficult to make a precise comparison due to differences in parameters and conversions, it is evident that the On-Demand mode is way more expensive than the Provision mode.

Detailed calculations:

Let us assume we have:

Number of Records = 1000/ second,

Size of Each Record = 3 KB,

Period = 1 Month

On-Demand mode:

Data ingested = 1000 records/s * 3KB / record = 3000 KB/ second

3000 KB/ Second = 7413.12 GB/month

Data-write charges for 1 GB / month = $0.08

Data-write charges for 7413.12 GB / month = 7413.12 * $0.08 = 593 $

Assuming we have one stream running.

1 stream per hour costs = 0.04 $

Data stream cost for 1 month = 0.04 * 24 * 31 = 28.84 $

Total ingestion cost in On-Demand mode= 593 + 28.84 = **621.84 $**

Provision mode:

Data ingested = 3000KB / s = 3 MB /s

Ingestion offered in 1 shard = 1MB / s

Number of shards required = Data ingested / Ingestion offered in 1 shard = 3

Shard Hour (1MB/s write, 2MB/s read) costs $0.015.

0.015 * 24 = 0.36 $ per day = 0.36 * 3 shards = 1.08 * 31 days (about 1 month) in a month **=** $33.48

Put Payloads (for 100,000 units, 1 unit = 25 KB) costs $0.014

Total ingestion cost in Provisioned mode=**33.494$**

Appendix 2

To run modified SL-18 Twin on EC2, follow these steps:

1. Start by creating an EC2 instance according to your needs for duration and number of instances.
2. Configure SSH (secure shell) on the EC2 instance to clone the SL-18 Twin repository from Git Hub.
   (To download the modified code of SL-18 Twin from a versioning control system)
3. Install Docker on the EC2 instance.
4. Install Docker Compose on the EC2 instance.
5. Add confidential credentials to the environment file.
6. Edit the run.sh file (from the downloaded code) to specify the number of trams you want to run.
7. Execute ./run.sh in the EC2 terminal to begin running the trams.
8. When you are done running the trams, stop the containers by running docker compose down.
9. Finally, shut down the EC2 instance to avoid incurring unnecessary costs.