Tomas Klungerbo Olsen
Lars Ivar Ramberg

Service for Data Streaming

Bachelor's thesis in Computer Science
Supervisor: Saleh Abdel-Afou Alaliyat
May 2022

Bachelor's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of ICT and Natural Sciences

NTNU
Norwegian University of
Science and Technology

Tomas Klungerbo Olsen
Lars Ivar Ramberg

Service for Data Streaming

Bachelor's thesis in Computer Science
Supervisor: Saleh Abdel-Afou Alaliyat
May 2022

Norwegian University of Science and Technology
Faculty of Engineering
Department of ICT and Natural Sciences

NTNU
Norwegian University of
Science and Technology

Abstract

In a world where data becomes increasingly plentiful, new solutions are needed to process data as quickly as possible. AutoStore AS has recognized the need to process the data from their automated warehouse installations in a reliable and scalable way. This need provides the basis for the project this paper presents.

By developing a solution for streaming data in real-time, the project described in this paper aims to offer a system for streaming data. By conducting research, testing, and implementation using solid frameworks and methodologies of Scrum, a product is was made that can provide reliable data streaming.

The main product produced as a result of this project provides streaming capabilities that offer reliability and scaling. Although some work remains before the solution is ready to be used in the real world to transport possibly sensitive information, the solution offers a Java-based implementation of a system fully capable of transferring data between endpoints.

In addition to the product, results in terms of research have also been created, which can be helpful to other developers looking to create services for streaming data. In addition to the primary implementation, prototypes have been produced and made publicly available as open-source software, contributing to the global open source community. The project lays a solid foundation for further development into a more secure solution fit for enterprise standards.

Sammendrag

I en verden hvor data blir stadig mer rikelig, er det nødvendig med nye løsninger for å behandle data så raskt som mulig. AutoStore AS har erkjent behovet for å behandle dataene fra deres automatiserte lagerinstallasjoner på en pålitelig og skalerbar måte. Dette behovet danner grunnlaget for prosjektet denne rapporten presenterer.

Ved å utvikle en løsning for strømming av data i sanntid, har prosjektet beskrevet i denne rapporten som mål å tilby et system for strømming av data. Ved å utføre forskning, testing og implementering ved hjelp av solide rammeverk og Scrum metodologi ble det laget et produkt som tilbyr pålitelig datastrømming.

Hovedproduktet produsert som et resultat av dette prosjektet gir mulighet for pålitelig og skalerbar datastrømming. Selv om det gjenstår noe arbeid før løsningen er klar til brukes i den virkelige verden for å transportere mulig sensitiv informasjon tilbyr løsningen en Java-basert implementasjon av et system som er fullt i stand til å overføre data mellom endepunkter.

I tillegg til produktet er det også skapt forskningsmessige resultater som kan være nyttig for andre utviklere som ønsker å lage tjenester for strømming av data. I tillegg til den primære implementeringen, har prototyper blitt produsert og gjort offentlig tilgjengelig som åpen kildekode-programvare. Dette bidrar til det globale åpen-kildekode utviklersamfunnet ved å tilby eksempler og løsninger for lignende prosjekter. Prosjektet legger et solid grunnlag for videre utvikling til en sikrere løsning som temmer overens med bedriftsstandarder.

Preface

The project represented by this paper was made in close collaboration with AutoStore AS, in particual Asle Olsen Gaasø who represented AutoStore and served as the stakeholder representative for the project.

We would like to thank Asle for the open communication, guidance, and feedback on the results produced as a result of this project.

We would also like to thank the staff at the Department of ICT and Computer science at the Norwegian University of Science and Technology, and in partcular our advisor Saleh Abdel-Afou Alaliyat.

This assignment was chosen because of, and made possible due to the willingness of AutoStore to have students develop solutions to their problems. We would like to thank AutoStore as an organization for believing in this project enough to set aside time for discussion during the development period.

Assignment

Assignment text adapted from original assigment text provided by Asle Gaaso Olsen at AutoStore. ”We” in the context of this assignment text refers to AutoStore AS.

Real-time streaming architecture for global data collection and analytics.

AutoStore, founded in 1996, is a warehouse robot technology company that invented and continues to pioneer cube storage automation, the densest order-fulfillment solution in existence. Our focus is to marry software and hardware with human abilities to create the future of warehousing.

The company is global, with over 650 installations in 40 countries in a wide range of industries. All sales are distributed, designed, installed, and serviced by a network of qualified system integrators we call ”partners”.

In order to improve system performance and service capabilities AutoStore is depending on data from their installations. Today this data is gathered using a batch approach, where one every day, the relevant data is transferred from the installation to a central repository.

This assignment involves the following tasks: research relevant technologies and frameworks that exists today that can enable world wide distributed streaming of data to a Cloud storage repository.

- Investigate common fault strategies related to loss of Internet connection or other faults.
- Implement prototype for real-time data distribution
- Implement or configure Cloud services that receives streamed data.
- Research Cloud based analytics frameworks that can enable real-time data analytics on real time data streams.

Table of Contents

List of Figures

List of Tables

Glossary

agile development A collection of iterative change based practices and frameworks. 3

branch Code representing a state of development.. 12

continuous integration A automated code integration practice that assists code change for multiple developers working on the same piece of software. 16, 34, 38, 39

Docker Compose Tool for multi-container management. 17, 27, 28

GitHub Provider of hosting for remote repositores. 21, 28, 29

GitHub actions Github's CI and CD platform. 16, 17

GitLab Provider of hosting for remote repositores. 21

Javadoc Generates documentation from java code into HTML sites. 16, 17, 28, 35

OpenStack Open source cloud software platform. 17, 18, 25

virtual machine Emulation of a computer system through software. 8, 17, 18, 24, 25

Acronyms

API Application programming interface. 28, 33

CD continuous delivery. xiv

CI continuous integration. xiv

HTML HyperText Markup Language. xiv

IDE Integrated development environment. 16

IP Internet Protocol. 18, 24

JVM Java virtual machine. 17

NTNU Norges teknisk-naturvitenskapelige universitet (Norwegian University of Science and Technology). 17, 18, 20, 24, 25

TLS Transport Level Security. 34

VPN Virtual private network. 24

1 Introduction

This chapter introduces the project this report is based upon. The project background, requirements, boundaries, and subject areas are all introduced to provide an understanding of the project and its value. The structure of the report is introduced at the end of the chapter.

1.1 Background

Businesses today generate large amounts of data in their day to day operations. For AutoStore AS, one of the core sources of data are their installations. AutoStore offers an automated warehouse solution which includes robots to handle merchandise, ports that present this merchandise to human operators, and all of the communicative technology necessary for robots, ports, and humans to work efficiently and synchronized. The system is in constant motion, with many moving parts operating in parallel (see Figure 1.1 for a visual example).

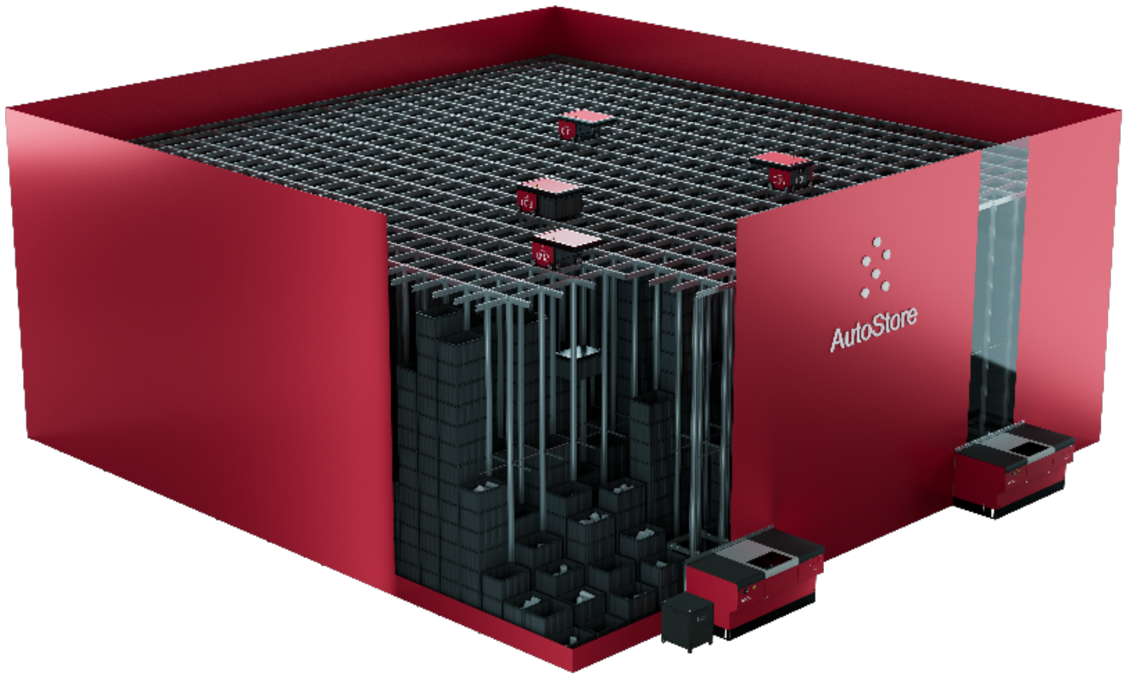


Figure 1.1: AutoStore installation with robots generating event data (TKSL, 2020)

AutoStore currently collects their data in batches at set points during the day. While this is valuable, a real time approach would provide further value and open up new possibilities. Being able to gather all the data generated by a system during operation in real time would enable AutoStore to act on anything happening in a system as it happens and predict system events based on the data.

This project was requested by AutoStore AS on behalf of their Monitoring and Maintenance team, responsible for analyzing data from AutoStore systems. The project is part of a larger initiative within AutoStore to modernize the use and flow of data, introducing real time streaming and analytics.

1.2 Problem Thesis

Analyzing data in real time necessitates real time access. This project was conceptualized as a solution to provide real time access to AutoStore data by streaming it from the locations where it is generated to AutoStore's already existing analytics infrastructure. The end goal is to discover and implement a streaming alternative to the currently existing batch transmissions of data currently used by AutoStore, enabling them to respond and act on their data in real time. To achieve this, a solution will be built that is fault-tolerant, performant, scalable, and secure.

1.3 Requirements

As a product handling data for a public company, there are multiple considerations and requirements that must be accounted for. Firstly, the solution must be secure as it handles highly valuable data. Secondly, it must be scalable to accommodate a company in growth.

Since the solution will target sensitive data of a growing company, the following requirements have been identified:

- Security. Data must not be stolen or manipulated as it is transferred
- Scalability. AutoStore is a company in growth. If the solution does not scale, it will not be a permanent solution

The assignment document provided by AutoStore specifies five main tasks for the project. As the assignment document was formulated by AutoStore, the requirements also reflect their wishes and needs. The tasks mention research of technologies and frameworks, fault strategies during loss of internet connection or other faults, implementation of real-time data distribution, and configuration or implementation of cloud services that receive data. Analytics is also mentioned, but only as a research task.

Based on the project description, the following has been identified as requirements:

- Performance. High-performant streaming with low delay, ideally close to real time.
- Reliability. Data needs to safely travel from source to consumer without loss
- Fault tolerance. Temporary system failure should not lead to loss of data
- Cloud compatibility. The solution needs to be deployable to and accessible from the cloud.

1.4 Boundaries

The group is limited in their access to real AutoStore data. This is to avoid any concerns regarding confidentiality, as well as security. From a confidentiality perspective, it is technically not required for the group to have actual data to implement a general data streaming solution, making it an unnecessary extension of privileges. From the perspective of security, giving the group access means creating one or two extra accounts that can access AutoStore data. These two accounts represent two extra entry-points into the system which should be avoided if possible.

The project has been defined to only concern streaming of data, not analysis. This is a key distinction as it substantially impacts the scope of the solution. The solution need only receive freshly created data, stream the data over the internet, and ensure that the sent data is received intact and secure. While the reduced scope makes the task smaller, it allows for more focus on a secure, stable, and user-friendly streaming solution.

1.5 Subject areas

This assignment covers multiple subject areas related to computer science and software development. A distinction is made between technical subjects, and organizational subjects.

Relevant technological areas are cloud computing, networking and cybersecurity. These three areas can be combined in different ways, but the inclusion of all of them forms the secure distributed computing this assignment is based upon. Figure 1.2 provides a visual representation of the relevant subject areas.

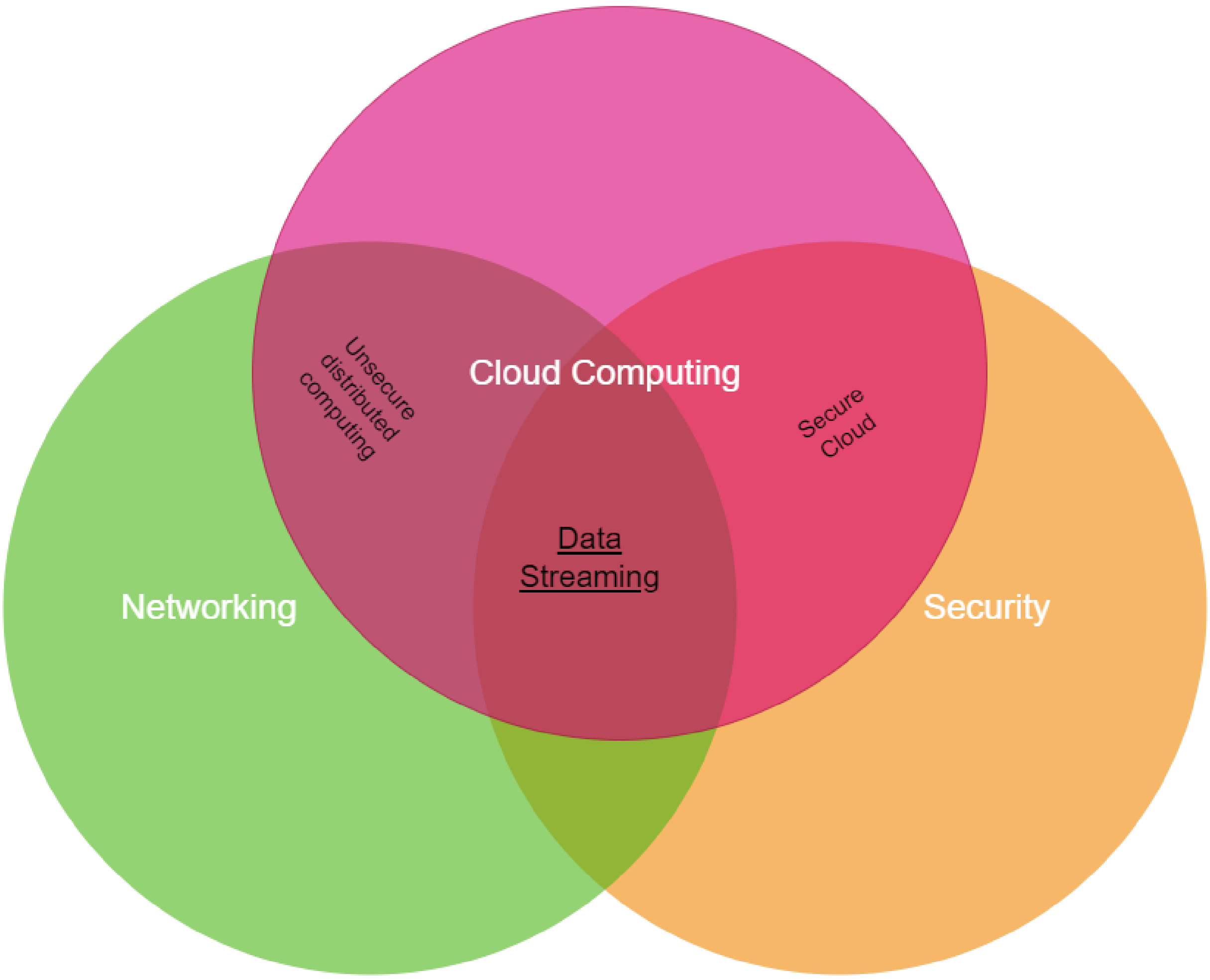


Figure 1.2: Diagram showing the relevant technological subject-areas of the assignment

The non-technical subjects of agile development, agile subjects, testing and prototyping are relevant. These subjects do have technical aspects to them, but have a heavier element of organization than the technical subjects, hence the distinction.

1.6 Structure

This document is divided into 6 main chapters:

Chapter 1: Introduction Introduction to the project with background, problem thesis, boundaries, and requirements.

Chapter 2: Theory and Material Presents the reader to the theoretical background of both the solution and its development

Chapter 3: Method Description of scientific and developmental methods utilized for this project

Chapter 4: Results Project results, presents what was achieved

Chapter 5: Discussion Discussion and reflection, discusses the results in relation to problem thesis

Chapter 6: Conclusion Concludes the thesis as a whole both thematically and in terms of project results and subsequent discussion

2 Theory and material

This chapter presents relevant background-theory for the project. This chapter covers theory related to methodologies, technologies, and tools related to and used during this project. After reading this chapter, the reader should have the necessary theoretical knowledge to understand the tools and technologies described.

2.1 Agile methods and SCRUM

SCRUM is a development methodology commonly used in software development. It is designed to help people, organizations, and teams create adaptive solutions to complex problems (Schwaber and Sutherland, 2020). Scrum is an agile method, allowing for a more flexible and reactive development schedule when compared to traditional waterfall methods by working in an iterative manner where each iteration feeds back into the development of the end product. Figure 2.1 gives a visual overview of Scrum.

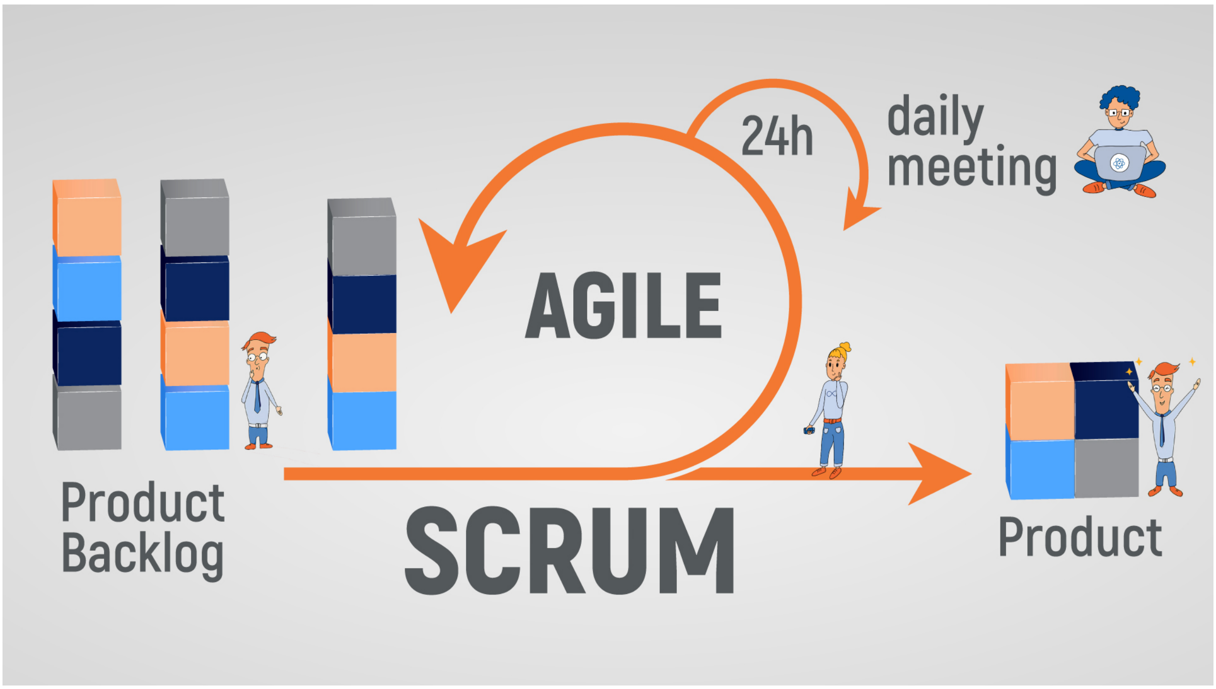


Figure 2.1: Overview of the Scrum methodology (Adam, 2022)

2.1.1 Scrum Sprints

A sprint is a time-boxed period of development where a team works to complete a set amount of tasks (West, 2022). A sprint is usually defined to last from one to four weeks although there are no strict limits to length.

2.1.2 User Stories

User Stories are units of work within the Scrum framework (Rehkopf, 2022a). User Stories help put the end-user at the center of development by formulating software features as user needs or desires. User Stories are usually formulated by expressing the user need in a ”as who, I want what, so that why” way (Paradigm, 2022). Figure 2.2 presents one example of a user story.

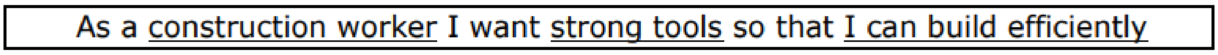

As a construction worker I want strong tools so that I can build efficiently

Figure 2.2: Example of a user story with a ”who”, a ”what”, and ”why”

2.1.3 Backlog

The backlog is the home of all unresolved issues. Any piece of intended functionality will start as an issue in the backlog, before it is eventually selected for implementation and moved from the backlog into a sprint (see section 2.1.1). User stories described in section 2.1.2 are commonly stored in the backlog as issues, although the backlog may contain other tasks as well.

2.1.4 Scrum meetings

In addition to sprints, Scrum is structured around multiple meetings with varying frequencies. These meetings have different purposes and intentions.

The following meetings are commonly included in scrum:

Stand up A small, daily meeting where the development team discuss what each individual member is working on. This is an opportunity for the team members to present any blockers they may have encountered.

Sprint planning A meeting for planning an upcoming sprint. The team decides what issues to take from the backlog and put into the upcoming sprint.

Sprint Review A larger meeting after a sprint has completed. The team presents what results have been achieved, and what each member of the team has been working on. This meeting focuses heavily on the product.

Sprint Retrospective A meeting where the team discuss their process over the last sprint. This meeting focuses on the team and their processes as opposed to the product.

2.1.5 Scrum Roles

Scrum uses roles separate from ordinary job titles to define tasks and responsibilities (Coursera, 2022). Scrum is usually divided into three roles: Product owner, Scrum master and Development team (West, 2022, Coursera, 2022). Figure 2.3 illustrates the different roles and their primary tasks.

The product owner is a bridge between the development team and the stakeholder. The product owner manages the scrum backlog, handles releases, and communicates with stakeholders (West, 2022).

The development team refers to the team creating the product. Development teams in scrum are unique in that their roles are fluent, and a developer in scrum context refers to any technical professional contributing to a product, not only software developers (Fowler, 2019).

The scrum master is a servant leader and scrum advocate within a project West, 2022. The scrum master helps the product owner manage the backlog and deliver value. In addition, the scrum master assists the team by ensuring that work is being done and that blocking tasks are dealt with and removed. (ibid.).

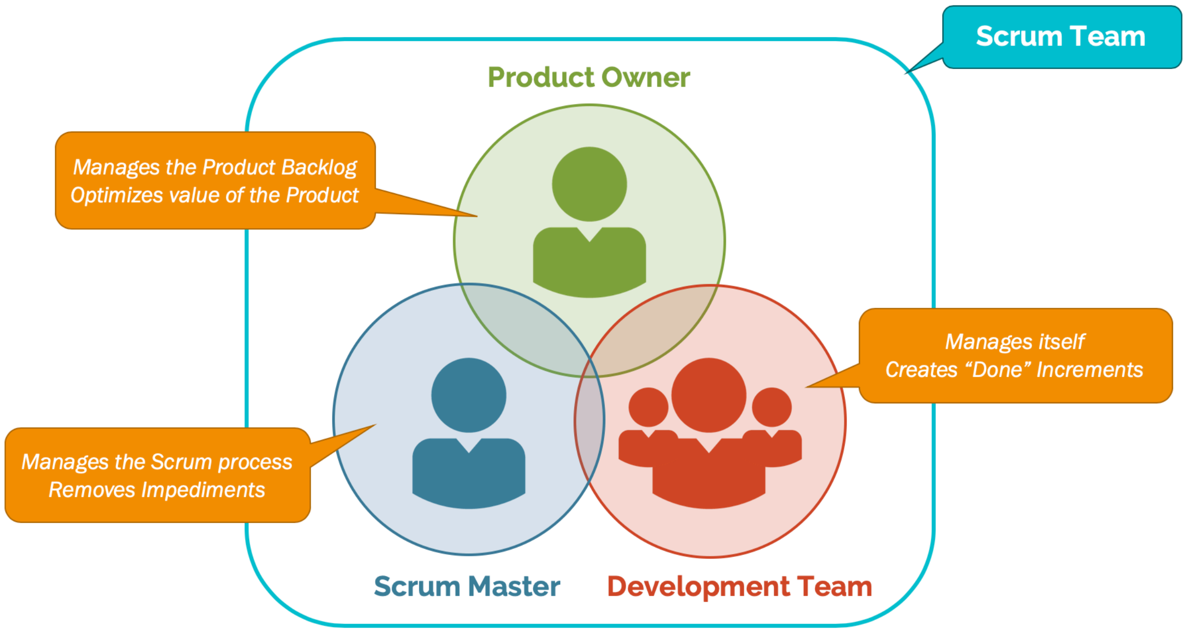

Scrum Team
Product Owner
Manages the Product Backlog
Optimizes value of the Product
Manages itself
Creates "Done" Increments
Manages the Scrum process
Removes Impediments
Scrum Master
Development Team

Figure 2.3: Illustration of the scrum team consisting of a product owner, scrum master, and the development team (Doshi, 2022)

2.1.6 Story Points

Story points are a measure of overall effort required to implement a backlog item (Radigan, 2022). Story points are used to assign an appropriate amount of tasks to a sprint, and to distribute the workload fairly between developers.

2.1.7 Planning Poker

Planning Poker is a way of assigning Story Points to a user story (see section 2.1.2). For each user story in the backlog, the developers estimate an amount of story points individually. The estimation is often performed with playing cards where each card has a value from an agreed-upon limit. These values can be as simple as a linear range or a more complicated range like the Fibonacci sequence. The Fibonacci sequence has the effect of an increasing difference in values which can help expose insecurity when estimating a story. The estimated card is laid on a surface with its back facing up. The card's value is hidden so as not to influence the decision of other team members. When all members have estimated the user story, the cards are flipped upside-down, exposing the estimated values as illustrated in figure 2.4. Discussions regarding why some chose low whereas others chose high values can begin, and convergence towards a final estimate can ensue. The discussion often reveals ideas and concepts for solving a problem that some members did not possess before the discussion. The discussion can also function oppositely and expose issues that might seem easy at first, but that other team members have had difficulties with previously.

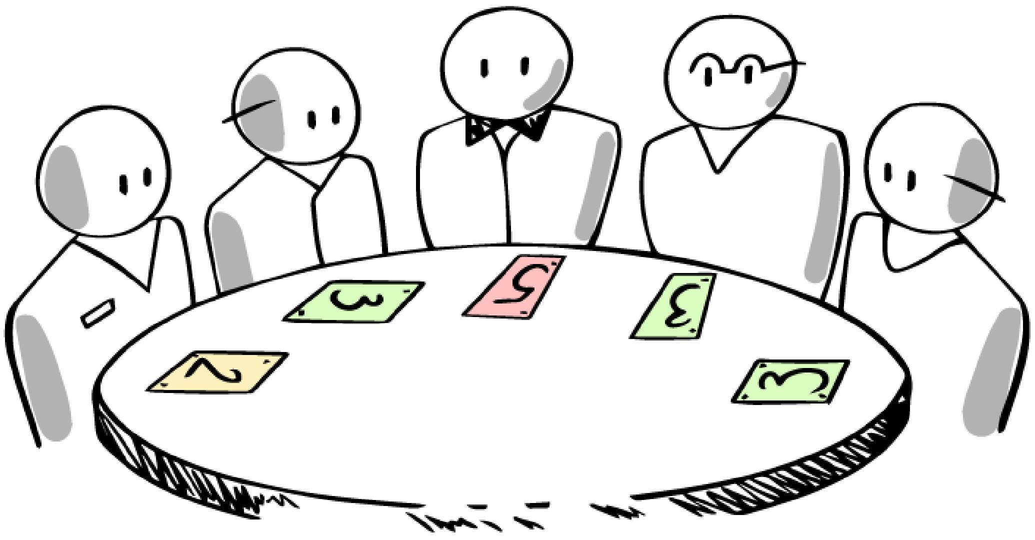


Figure 2.4: Illustration of a development team practicing planning poker (Compliance, 2018)

2.2 Containers and images

A container is a concept in which applications are bundled with their required environment and other dependencies, making it effortless to share across compatible devices (Docker, 2022). A compatible device for containers is a device in which the operating system has an engine to run containers (McCarty, 2018). Compared to similar solutions, which use a virtual machine, containers utilize the kernel of the host operating systems, making them more lightweight. Although containers use the host operating system's kernel, they run as isolated processes in user space for additional security. Figure 2.5 illustrates the difference between virtual machines and containers where a container can be seen running directly on the host operating system from a container engine.

Images function as a template for containers. They contain layers of instructions on how to create container instances. Comparing images to equivalent technology would be that of a snapshot for a virtual machine (Gillis, 2021).

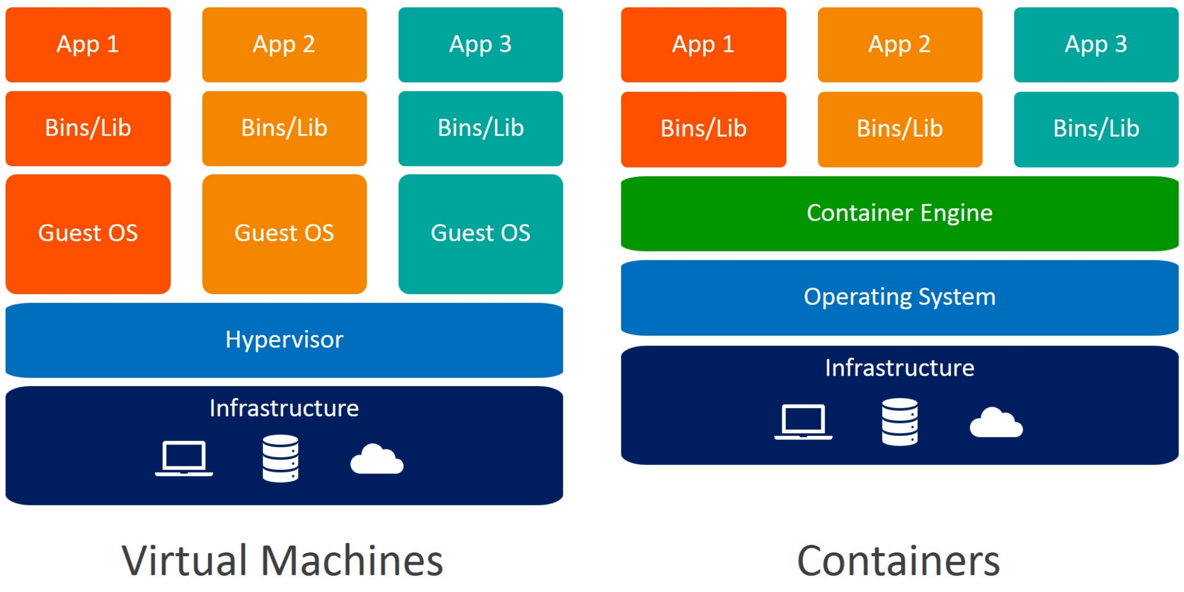


Figure 2.5: Virtual machines vs containers (Weaveworks, 2020)

2.3 Cloud Technologies

The cloud and cloud technologies are a relatively new phenomenon in software development. Although the cloud has existed as a concept of distributed computing since the 1990's (Hoffman, 1993), it gained significant traction after big companies like Amazon, Google, and Microsoft entered the space with their respective services in the 2000's. The cloud refers to a set of computational services on the internet that both software developers and consumers can access.

From the consumer-perspective, cloud technologies allows many technologies and solutions to be available from anywhere in the world without having to install anything on their own device. It is also common to have files stored in the cloud instead of, or in addition to, having them stored on a local device.

For the developers, the cloud offers amorphous black boxes of functionality and services over the internet. Microservices as an example can be many different small applications living on the cloud that can be brought into a larger application to compose applications from multiple small parts.

2.4 Streaming

In computing, streaming refers to the process of transferring data in a continuous stream for immediate processing (Schwaber and Sutherland, 2020). Streaming data provides benefits like the ability to receive data within milliseconds of creation and usage of data without downloading its entire file (commonly used for video and music). Streaming is a central part of the assignment, as the quick reception is at the core of the stakeholder's use-case.

The concept of streaming is closely related to the subject of cloud computing (see section 2.3), in that parts of the streaming infrastructure usually exists as a cloud service or incorporates cloud services. Normally, endpoints exist on client computers, while a middle-man or broker lives on a cloud-server

2.5 Version control

Version control refers to the act of progressively tracking and managing changes in code (Atlassian, 2022). Version control allows developers to track changes in code over time, work on different features without irreversibly changing the source code, and facilitates simultaneous work on different parts of the code.

2.5.1 Git flow

Git flow is a way of managing version control. Git Flow provides a way of organizing branches in order to avoid committing unready changes to the main branch. This type of version control centers around two central branches: main and develop. Every time a change is to be made, a branch is created from develop. A branch can be one of four types:

Feature A branch containing a new feature

Refactor A branch containing changes to an already existing feature that does not fix a bug

Hotfix A branch containing a fix to a bug

Release A branch ready to become a release

All feautre, refactor, and hotfix branches are committed to develop before a release branch is created from develop. The release branch is tagged with a version before it is merged into the main branch. This flow is exemplified in figure 2.6.

Although named after the version control-system Git, Git Flow is technology-agnostic and can be applied to any version-control system that supports branching.

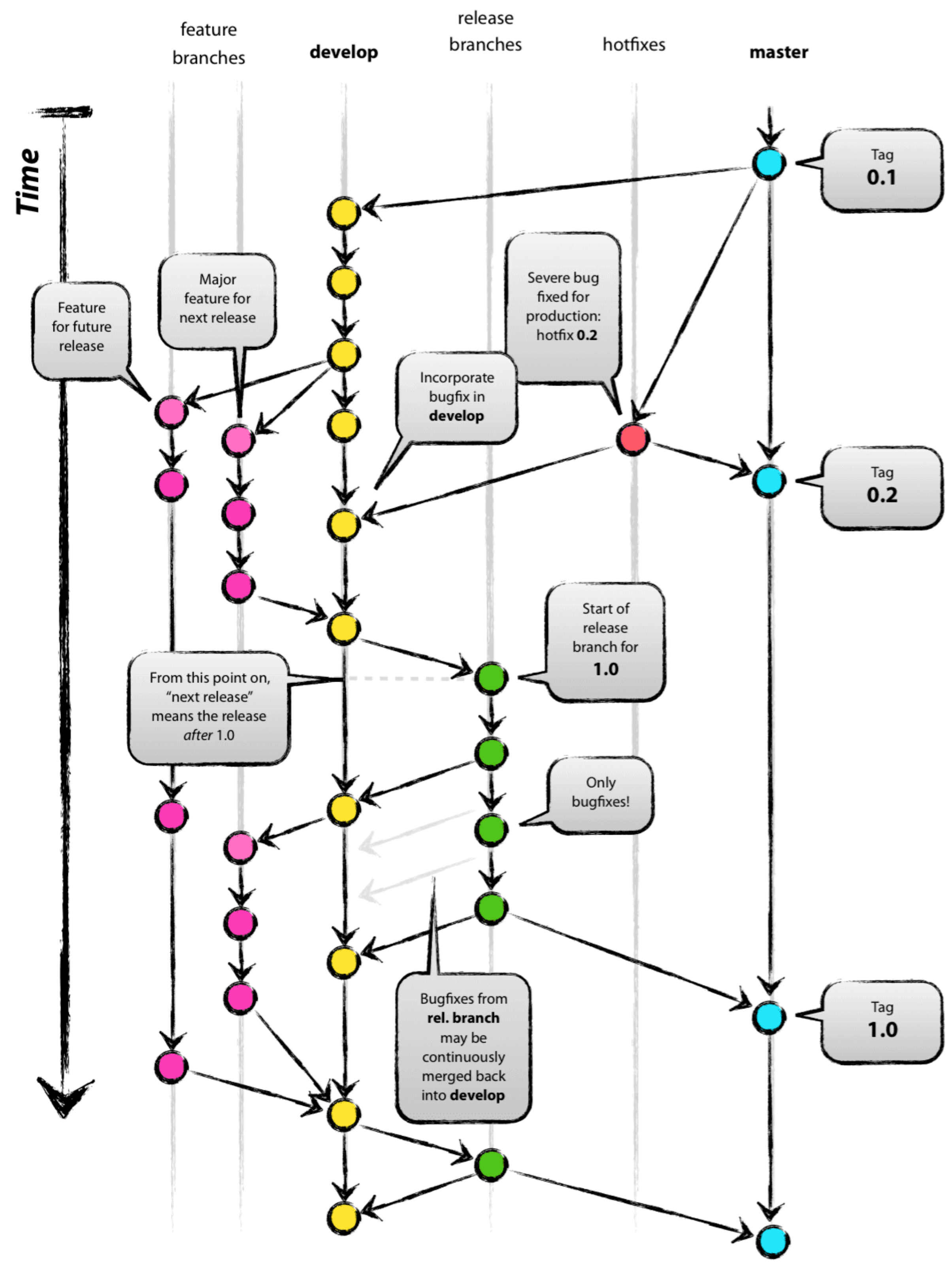

feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix **0.2**
Incorporate bugfix in **develop**
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release *after* 1.0
Only bugfixes!
Bugfixes from **rel. branch** may be continuously merged back into **develop**
Tag 1.0

Figure 2.6: Visual illustration of branching in Git Flow (Driessen, 2010)

2.6 Clean Code

Clean code is a book written by Robert Cecil Martin (Martin, 2009) which investigates good and bad code practices from the author's point of view. Although good and bad code often can be regarded as subjective, many ideas and paradigms can be found in the book which can be implemented to produce simpler code to read. The book covers subjects such as naming conventions, length of methods, clutter comments, and code formatting.

2.7 Pair Programming

Pair programming is a concept with its roots in extreme programming. Pair programming involves co-development by two developers on the same piece of code. One developer, the driver, writes the code while the other, the observer, watches the driver and actively searches for syntactic defects and thinks about the overall direction of the work (Williams, 2001). The pair programming workflow with its driver and observer role has been illustrated in figure 2.7

Pair programming trades the perceived efficiency of working on multiple pieces of code at once with the benefit of having two developers work on the same piece of code to ensure correctness, performance, and maintainability.

Figure 2.7: Illustration of a development team practicing pair programming (Termenji, 2021)

2.8 Code Review

Code review is the process of having one developer inspect and review the code of another. Usually, code reviews are performed before changes in a branch are permanently committed to production code.

Code review provides an extra opportunity for errors to be caught as developers review the code of their colleagues. It also facilitates increased system knowledge among developers as they need to inspect and understand code in parts of the system they are not directly responsible for implementing.

2.9 Encryption

Encryption is the obfuscation and securing of data by changing it in a reversible manner. Encryption when related to computer science and software engineering uses mathematical methods (Chen, 2021) in order to encrypt and decrypt data.

Many modern encryption schemes are based on keys. Keys provide the necessary mathematical information for a program to encrypt and decrypt data, provided the software has the correct keys. There is a distinction between symmetric key cryptography, and asymmetric key cryptography. With symmetric keys, provider and client have the same key and use this key to encrypt and decrypt the data transferred between them. With asymmetric cryptography (see Figure 2.8), clients and providers have public and private keys. Public keys can be distributed to other agents and the private keys are unique to each agent. As an example of this, agent A wants to send an encrypted message to B. A then acquires or uses the public key of B to encrypt the message. A now sends the message, and the private key of B is used to decrypt the message.

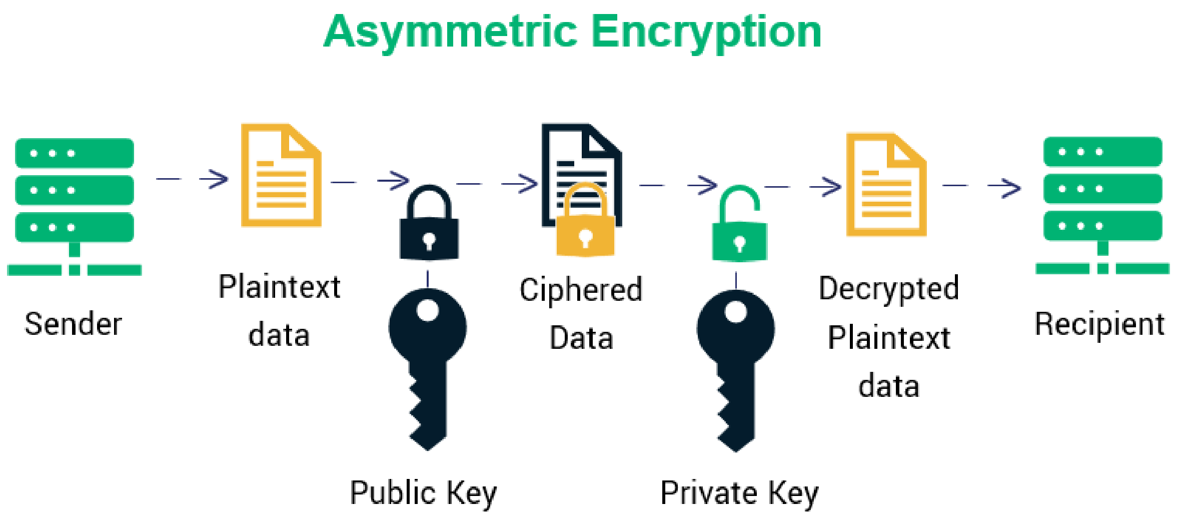


Figure 2.8: Encryption using asymmetric keys (Rajitha, 2021)

2.10 SOLID design principles

The SOLID design principles are a set of principles and guidelines for software engineering and development. It is a subset of the many principles presented by Robert C. Martin in 2000 in his essay ”Design Principles and Design Patterns (Watts, 2020).

SOLID forms an acronym representing each principle. The principles are as follows (Oloruntoba, 2021):

Single responsibility principle Each class should have only one job

Open-closed principle Each class should be open for extension, but closed for modification

Liskov Substitution Principle Every subclass or derived class should be substitutable for their parent class

Interface Segregation Principle A Client should never be forced to implement an interface that it doesn't use.

Dependency inversion Principle Entities must depend on abstractions, not on concretions.

3 Method

This chapter presents all relevant methodology, both scientific and developmental, used during this project. Usage and implementation of the technologies, theories, and concepts described in Chapter 2 are discussed here to illustrate how the project was conducted and how the project reached its result.

Research and implementation were closely related for this project, as part of both research and implementation was the development of prototypes. The prototypes can be considered research in that they were later used to test different frameworks for function and efficiency. They could also be considered part of the development methodology as one of the prototypes were carried on to implementation, and many core decisions were taken based on these prototypes.

3.1 Research method

Although the project was a developmental project, there was a heavy focus on research and documentation of findings. Streaming and related technologies were new areas both for AutoStore and for the group. It therefore became necessary to dedicate time solely for research and discovery to find the best way to solve the problem posed by AutoStore.

3.1.1 Frameworks and Framework Research

Researching and comparing available frameworks was a high priority at the start of the project. As the team had limited previous knowledge of the domain of streaming and related frameworks, a large theoretical analysis was performed and presented to stakeholder. This analysis and its results helped dictate what technologies to use for the prototypes and final implementation.

From the research, a research paper was written and delivered to stakeholder in week 3. The research paper contains general information and facts about each framework to allow for comparison in terms of theoretical capacity, pros, and cons of each framework.

This research paper was later used when discussing with the stakeholder which frameworks should be implemented. This was done to avoid choosing a framework that did not fit the stakeholders use case, lacked functionality, or provided functionality that would not be necessary such as analytics capabilities.

The frameworks chosen for the assignment needed to fit the criteria outlined in Section 1.3. Therefore, the frameworks needed to be fast, secure, and ideally easy to implement. The presence of already existing analytics tools from the stakeholder side also pushed the search in the direction of lightweight frameworks that do not contain any analytics capabilities.

Three frameworks were chosen as a result of the research: Apache Kafka, Apache Pulsar, and Apache Flink. RabbitMQ, Apache Spark and Apache Storm were also considered

RabbitMQ was quickly deemed unfitting for the use-case as it could not offer the required security of data. It does not by default check to see whether messages have been successfully received by the consumer endpoint. Spark and Storm were passed up due to a large amount of unnecessary and unwanted features.

Both Kafka and Pulsar were chosen because they were lightweight frameworks that provide secure messaging without being burdened with un-needed analytics features. Pulsar and Kafka seemed to be the fast and lightweight solutions that the project would need. Additionally, neither framework had any large known security flaws at the time of development.

Flink was the most experimental choice, as Flink contains multiple features that the other frameworks do not and is also compatible with both of the other chosen frameworks as part of a data-chain. This led to a decision of creating a more experimental framework where Flink would be built on top of Pulsar or Kafka. Flink was also chosen because it too promised to be fast and secure, although somewhat less lightweight than the other frameworks.

3.2 Technologies and Developmental Methodologies

3.2.1 Java

Although all the initially chosen frameworks (see Section 3.1.1) supported multiple languages, the group eventually decided on Java for development and implementation.

The most apparent reason was developer experience. Java is a language both members of the development team are highly comfortable with using. Using this experience, the team could rapidly start developing the solution without learning a new language.

Java was also a good choice from a technical standpoint. Java is a cross-platform language, meaning a program compiled in Windows will also work on Mac and Linux. In addition, the ”compile once, run everywhere” feature of the language increases the portability of the solution and makes it a viable option regardless of the end-users operating system.

3.2.2 Code style

Code style was enforced using a formal style guide, as well as a Checkstyle plugin for the IntelliJ IDE.

The style guide details how the members of the development team are expected to formulate their code. While good and maintainable code is possible without a common guide, uniformity is achieved when there is a common guide for the developers to follow.

The Checkstyle file of the project is based on the style-guide. When used, it ensures that the style-guide is being followed by scanning the code in the IDE and highlighting any mismatches between the code and the Checkstyle specifications. Using Checkstyle eliminates a big manual component of maintaining the code, and ensures greater adherence to the style guide.

3.2.3 Continuous Integration

Continuous integration was used to verify that new changes did not break the solution and to automatically generate and publish Javadoc. Continuous integration was implemented using GitHub actions that would run every time code was pushed to the repositories.

Automatic building was implemented to combat regressive failures in the program. Whenever code was pushed to the repository, the continuous integration jobs would use Maven to build the solution and verify that the code was not causing any compilation errors.

Javadoc generation and publishing were specifically set to trigger when a merge request or push to the develop or master branch of the repository occurred. One GitHub action manages the generation of Javadoc for the master branch and pushes the generated documentation to a directory dedicated to documentation related to the official release. In contrast, the GitHub actions that manages the generation of Javadoc for the develop branch pushes the generated documentation to a different directory dedicated to the newest in-development release. Both branches were set to trigger so the development team could access documentation for the latest official release and the newest in-development release. Using Gitflow for managing branches (see Section 2.5.1) leads to the develop branch being pushed to often, while the master branch only has changes pushed to it when a new version is ready. Only generating Javadoc on the master branch would lead to new documentation being rarely available, and having the newest development Javadoc was necessary for the team during development, as the gaps between releases were quite large.

3.2.4 Containers and Docker

Docker containers were chosen for use early in the process before any prototypes had been developed.

Docker was chosen because it would provide a high degree of portability of the solution. Instead of requiring the user to have the correct version of Java or other dependencies installed on their system, it would be possible to define Docker images with all the necessary dependencies defined. A user could then build and run Docker containers directly on their system without downloading any new dependencies directly on their system. Although Java already offers cross-platform support through the way the Java JVM works, Docker provides further compatibility by not requiring Java at all.

In addition to docker, Docker Compose was also utilized. As the system has three different parts that need to communicate with each other, there was a need for a single mechanism that could start all parts and provide a network for them to communicate over. Docker Compose provided both these facilities, and was used actively to test the system locally and rapidly. Docker Compose made testing during development fast and efficient as starting the entire system locally could be done with a single command after the docker-compose file had been properly defined. This saved large amounts of time during development, as the whole solution was frequently built, started, torn down and deleted. Time needed to be spent learning docker compose and writing the related compose-files, but this time investment paid off in time saved during active development.

3.2.5 Virtual Machines and Networking

In addition to containers (see Section 2.2), virtual machines and networks were used during development. NTNU provided the first virtual machine through their OpenStack solution. This virtual machine was used by the development team as a server for the broker, enabling actual tests over the internet with the developed system as opposed to only local testing over the local network.

Microsoft Azure provided the second virtual machine much later in the development process. The team used this virtual machine to host the solution on a publicly reachable IP and for development towards the end of the project. The Azure virtual machine came much later than the OpenStack machine to avoid unnecessary costs related to the project, as Azure can quickly become expensive.

Virtual networks were also used by necessity. The virtual networks' objective was to connect to the virtual machine as it did not have a public IP address. The use of virtual networks gave the group access to their OpenStack virtual machine without being physically present on campus and connected directly to the NTNU network.

3.2.6 Collaboration

Part of the groups methodology was the way they chose to collaborate. The team was geographically distributed with one member being in Ålesund, and another located in Haugesund. This necessitated that the team found ways of collaborating that would transfer well digitally.

The team performed all meetings using chat services like Microsoft Teams and Discord. Microsoft Teams was used for meetings that also involved the stakeholder and advisor, while Discord was used for meetings between the group-members. Teams was used for the large meetings as everyone had an account readily available either through NTNU or AutoStore. Discord was chosen for internal group meetings as it was compatible with the developers operating system of choice (Linux) and provided more effective means of screen-sharing.

Additionally, the group made heavy use of the pair programming described in section 2.7. Pair programming was used as the subject are of streaming, as well as the available frameworks, were new for both group members. It was therefore helpful to be two developers when facing completely new problems and technologies to more quickly resolve issues as they were discovered.

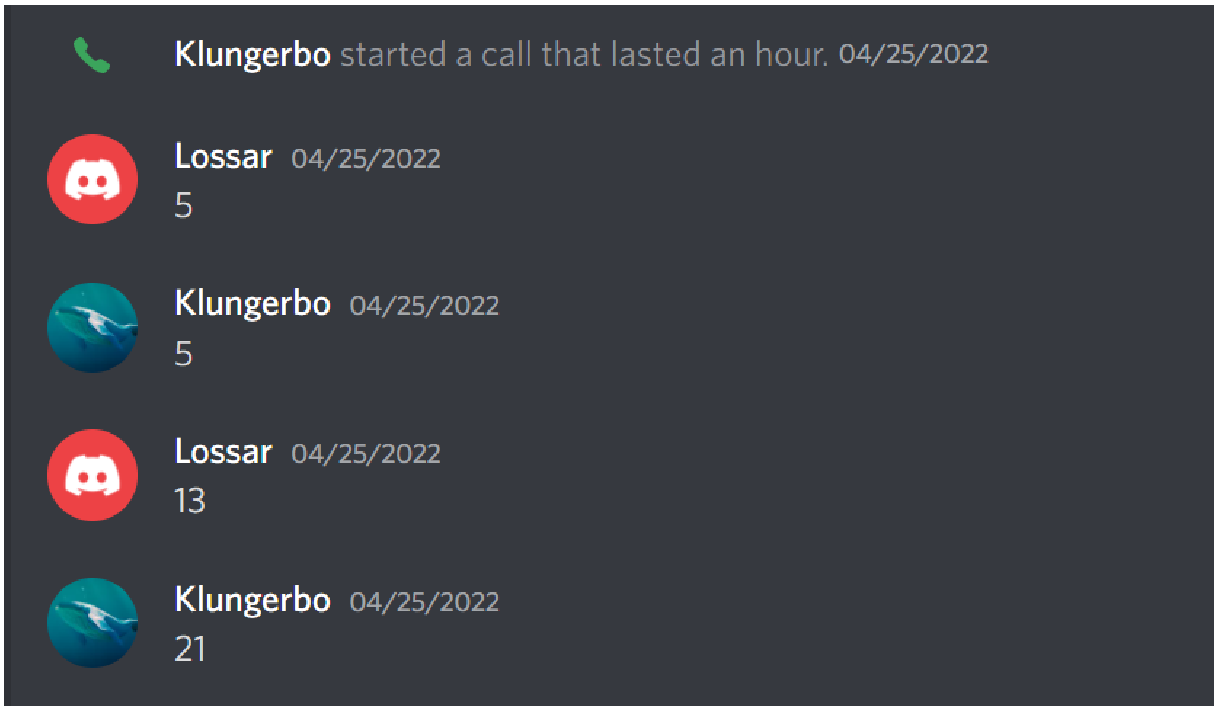
3.2.7 Planning and estimation of work

Planning poker (see section 2.1.7) was used to estimate the amount of work a user story would represent. The team used the Fibonacci numbers for assigning points as the numerical difference that appears when jumping between high-values gives a clear separation in terms of workload. Instead of using physical cards, the team would use text chat in combination with voice chat and screen sharing. The product owner would share their screen and read out the user stories in the backlog, then the points were assigned individually by each member by typing their number in the chat, then on voice cue sending the score to the other member. If the numbers differed, a discussion would start before eventually converging on one agreed upon score. Figure 3.1 shows an example of planning poker in progress, with the members agreeing on the score for the first issue, and disagreeing on the second.

3.2.8 Work distribution

Scrum roles

The distribution of work was even with slightly different responsibilities across the group. As described in section 2.1.5 there are three roles in scrum; product owner, scrum master,


Klungerbo started a call that lasted an hour. 04/25/2022

Lossar 04/25/2022
5

Klungerbo 04/25/2022
5

Lossar 04/25/2022
13

Klungerbo 04/25/2022
21

Figure 3.1: Scrum poker in progress. Voice call is started, issues are read out, and points are assigned.

and development team. All roles were utilized by the group, but some adjustments had to be made due to the small group size.

The roles were decided in the pre-project phase. The decision was made for the group members to have static roles with no rotation of responsibilities. Rotating responsibilities would allow each member to learn from multiple points of view and master different responsibilities, but it would compromise the project as members would have to switch responsibilities and specializations as roles changed. While the learning outcome of this would be greater it would also make the group less efficient. The loss of efficiency was deemed undesirable considering the nature of the project as a product to be delivered to a stakeholder in a timely manner.

The development team for this project only had two members. As such, the traditional role-distribution of scrum had to be circumvented somewhat to accommodate for the small team size. Both members were considered to have the role of development team members, while there was one dedicated scrum master and one dedicated product owner. The distribution was done based on personal preference and perceived best fit of skills for each role. The chosen scrum master was considered the most theoretically knowledgeable and experienced, while the product owner was the member most comfortable with stakeholder communication and backlog management.

Technical Responsibilites

The previous section discussed the work load purely in terms of the scrum framework. In addition to the scrum roles there was a distribution of work on the technical implementation side as well. This distribution was not planned from the outset, but naturally evolved as the project developed.

3.2.9 Work scheduling

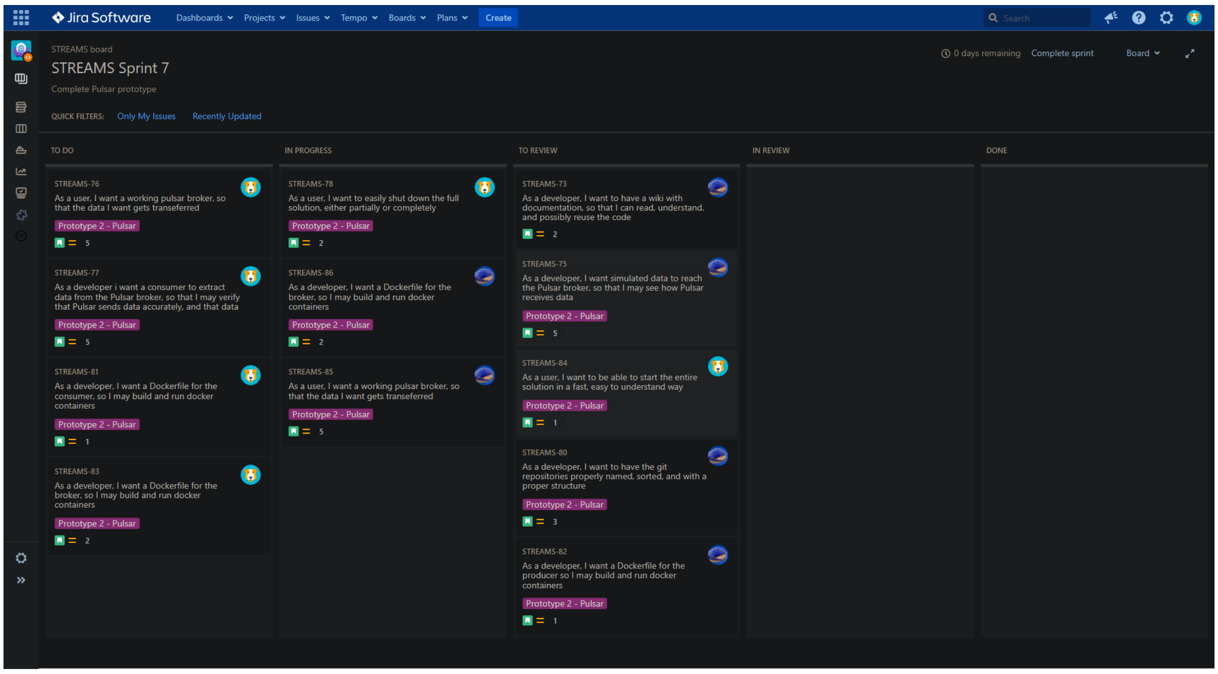
The scheduling throughout development was highly flexible to accommodate other obligations of the development team. As both developers held employment during the duration of the project, alterations were made to regular work weeks to ensure more available time for work during weekends.

As can be seen in Appendix 8, the group determined a work week to be from Monday until Sunday, with meetings every day except Saturdays being mandatory. The extended work week allowed for great flexibility for the team members, as long working days tending to other obligations did not hinder development. Any work that was missed during the week days could be done during the weekend, consequently also avoiding developer burnout by not forcing overly long days.

3.2.10 Issue Tracking System

For managing the issues, backlog, and sprints belonging to the project, a dedicated system was needed. The system needed to be easy to use, easy to configure, and preferably be without a paywall.

The development group chose Jira as their issue tracking system as shown in Figure 3.2. Jira was provided by NTNU free of charge, and the Jira instance provided was managed by university staff. As NTNU owned the Jira instance, it was deemed a safe place to store the backlog and issues without any expense on behalf of the development team.



STREAMS board

STREAMS Sprint 7

Complete Pulsar prototype

TO DO

STREAMS-76
As a user, I want a working pulsar broker, so that the data I want gets transeferred
Prototype 2 - Pulsar
5

STREAMS-77
As a developer i want a consumer to extract data from the Pulsar broker, so that I may verify that Pulsar sends data accurately, and that data
Prototype 2 - Pulsar
5

STREAMS-81
As a developer, I want a Dockerfile for the consumer, so I may build and run docker containers
Prototype 2 - Pulsar
1

STREAMS-83
As a developer, I want a Dockerfile for the broker, so I may build and run docker containers
Prototype 2 - Pulsar
2

IN PROGRESS

STREAMS-78
As a user, I want to easily shut down the full solution, either partially or completely
Prototype 2 - Pulsar
2

STREAMS-86
As a developer, I want a Dockerfile for the broker, so I may build and run docker containers
Prototype 2 - Pulsar
2

STREAMS-85
As a user, I want a working pulsar broker, so that the data I want gets transeferred
Prototype 2 - Pulsar
5

TO REVIEW

STREAMS-73
As a developer, I want to have a wiki with documentation, so that I can read, understand, and possibly reuse the code
2

STREAMS-75
As a developer, I want simulated data to reach the Pulsar broker, so that I may see how Pulsar receives data
Prototype 2 - Pulsar
5

STREAMS-84
As a user, I want to be able to start the entire solution in a fast, easy to understand way
Prototype 2 - Pulsar
1

STREAMS-80
As a developer, I want to have the git repositories properly named, sorted, and with a proper structure
Prototype 2 - Pulsar
3

STREAMS-82
As a developer, I want a Dockerfile for the producer so I may build and run docker containers
Prototype 2 - Pulsar
1

IN REVIEW

DONE

Figure 3.2: Sprint issue board used during development

3.2.11 Project Documentation System

Alongside the use of Jira (see section 3.2.10) Confluence was also used. Confluence provides a repository for documents related to the project. Confluence also communicates with Jira (they are both developed and maintained by Atlassian) allowing for features like referencing Jira issues in Confluence documents, and generating documents in Confluence based on data in Jira.

3.2.12 Milestone plan

Milestone plans were used to direct the development of the project, and to communicate with the stakeholder. The stakeholder suggested a milestone plan that presented week by week what the stakeholder expected to receive from the group. These milestones included multiple reports, programming assets, and time put in for implementing the final solution. The milestone plan was a tool both to communicate with the stakeholder and to guide the group through the project.

3.2.13 Version Control and Git

Version control and Git were essential in keeping the development going smoothly and to keep the project organized with different prototypes.

Gitflow and Changes to code

Gitflow (described in section 2.5.1) was used in the process of development. Every time a new feature was being worked on, a branch was created related to that feature. The feature branch would be updated with relevant commits until the the responsible developer deemed it ready to become part of the develop branch.

After the branch was made ready, the responsible developer would create a pull request in order to merge the branch into the develop branch. The repositories were set up in a way so that review and approval from the other developer was required before the branch could be merged. This was done to ensure code quality, ensure documentation and readability, and to catch errors or bugs that the original developer might not have discovered.

Repository organization

As it was decided early on that this project would require multiple prototypes and code bases it was crucial to have a solid structure within a version control system. Furthermore, the repositories would all need to be in the same place, preferably in a location where they would not share space with other repositories in order to keep the project separate from other projects.

The repo location chosen was GitHub, as it is the repository provider the group has the most experience with. For such a small team, there were also no perceivable benefits in using GitLab or other providers over GitHub for this particular project.

Initially the repositories were stored on the private accounts of the developers, but this both gave a false impression of the streaming service being private projects, and it obfuscated them among other repositories already existing in the personal spaces of the developers.

To alleviate the problems of personal repositories the developers created a GitHub organization. Organizations in GitHub have their own repository space with room for multiple repositories. An organization named ”AutoStreams” was created and used for the project. The organization is publicly accessible, and all repositories belonging to the project are openly available as open source code. The organization and relationships of

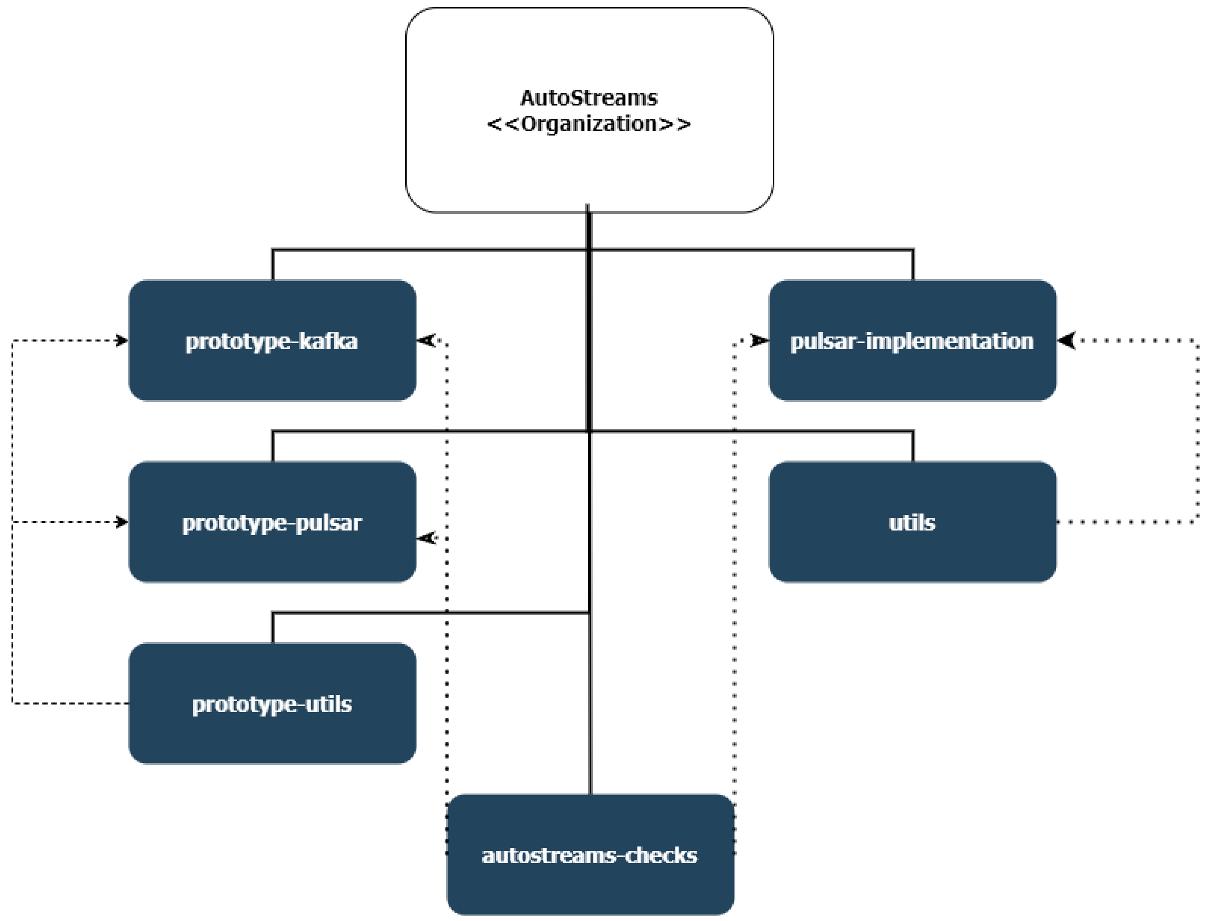


Figure 3.3: Diagram showing the structure of repositories and their relationships. Dotted arrows indicate the implementation of functionality from one repo into another

the repositories within the AutoStreams organization is illustrated in figure 3.3.

This way of organizing he repositories provides clear separations between the projects, a key concern when both prototypes and final implementation need to be implemented, tested, and uploaded to public Version Control.

3.2.14 Utility (util) Repositories

The use of util-repositories (one for prototypes and one for final implementation as illustrated in Figure 3.3 eliminated some code-duplication by extracting shared functionality into shared utility repositories. Additionally, keeping the utilities in a shared repository instead of re-implementing them for every project made the utilities easy to maintain. If a bug was discovered with a utility, the group could be certain that the fault was in the one shared implementation and fix it only there. Had the utils been copied, there would have been a slight chance of breaking alterations as the code is copied and pasted. Additionally, if any fault was found with the utility code, it would need to be changed multiple times across multiple implementations.

3.2.15 Prototyping

Prototyping and experimentation was a substantial part of the project. The need for this prototyping arose from the lack of previous knowledge about the topic both from the side of the student group and AutoStore. The prototyping phase narrowed down the selection of available frameworks to only one that would eventually be used for the final implementation.

Prototype 1: Kafka

The first prototype was implemented using Apache Kafka. Apache Kafka is a streaming framework originally developed and released by LinkedIn in 2011 (Rao, 2011). Kafka works on a publish/subscribe model, with all messages received by a Kafka broker immediately transferred to all subscribed consumers.

The use of Kafka in this prototype was a result of the research performed before development, and the findings presented in the research paper (see section 3.1.1 and Appendix 1). This technology promised high throughput, stability, and good support both from its community of developers and documentation.

Prototype 2: Pulsar

The second prototype was implemented using Apache Pulsar. Apache Pulsar was originally developed by Yahoo, but was given to the Apache foundation in 2016 and subsequently open-sourced Bartholomew, 2020. Pulsar, like Kafka, works on a publish-subscribe model, pushing all messages to all subscribed consumers.

The implementation of Pulsar came as a result of the research performed before development and the findings presented in the research paper (see section 3.1.1 and Appendix 1). Pulsar promised high throughput, stability, and good support from documentation but with a smaller community than Kafka as a result of being newer.

Prototype 3 Flink

The final prototype was planned to be based on Apache Flink in combination with either Kafka or Pulsar. Flink is develop and maintained by Apache and was released in 2011 (Wikipedia, 2022).

As the requirements for the project became clear, this prototype was eventually scrapped. Flink was initially chosen because of its good documentation, great security, and compatability with other streaming frameworks, enabling extra features such as analytics.

As the project progressed it became clear however that no analytics would be required from the solution. The Flink prototype was scrapped, enabling further testing and prototyping of the other frameworks while also avoiding the implementation of features unimportant to the stakeholder. The decision to drop such a feautre-rich framework impacts the project from an open-source standpoint, as it will fit fewer use-cases than if it were to include analytics as well.

The time initially intended to be spent on the Flink prototype was later spent on extra testing of the remaining frameworks, as well as Azure implementation of the other prototypes.

3.2.16 Testing

Testing was a large part of the development process. This is partially because the project involved technologies previously unknown to both the students and stakeholder, and because it was necessary to choose not only a functional framework, but the one that best fit the use case. It was also determined that research alone would not be sufficient to choose the best solution, and so practical testing was done in addition.

Testing for this project were large scale acceptance-tests and performance tests. For the acceptance tests, the system was tested for fault-tolerance in various ways. This was critical to test as the system contains multiple parts that can potentially fail, and because the stakeholder had explicitly outlined the research of fault tolerance strategies as a key goal of the project.

Test Setup

The tests were all performed with the same distributed setup. The setup consisted of two computers and one virtual machine communicating together over the internet. One computer would provide data (a producer component). The other would receive the data (a consumer component) whith a broker managing the data between endpoints ran on the virtual machine. All components ran in Docker for all tests.

As NTNU provided the virtual machine without public IPs, all system components needed to be connected to the NTNU network to communicate. The virtual machine was by default connected to the NTNU network. The computers achieved connection by using VPN to connect to the NTNU network using their student identification. Figure 3.4 illustrates the test setup as a whole.

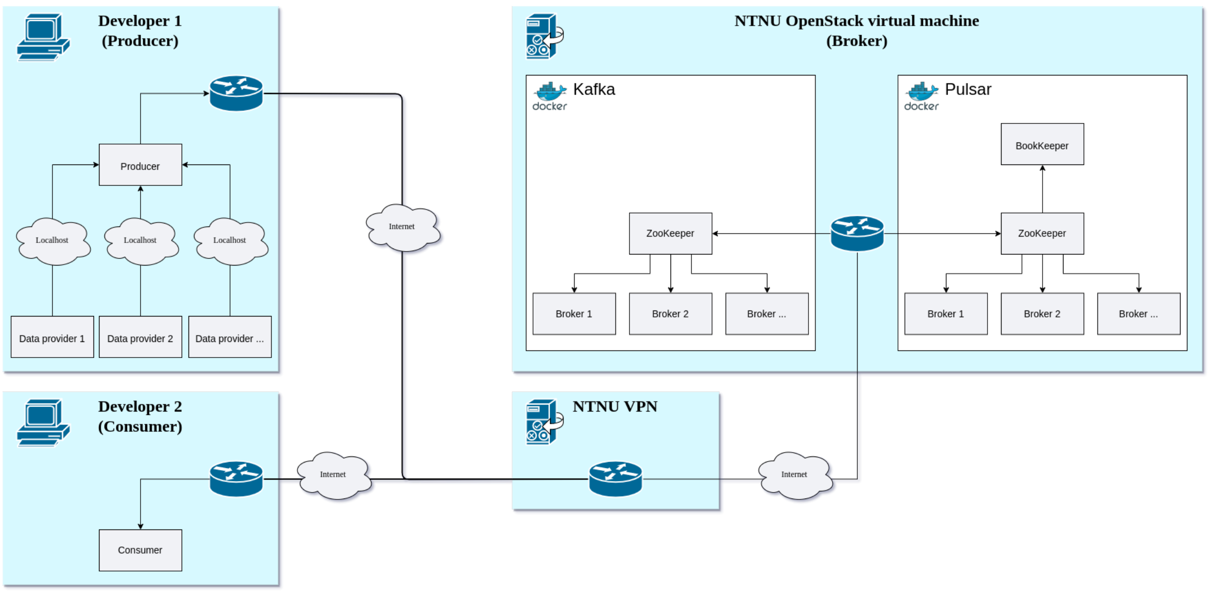


Figure 3.4: Illustration of testing architecture.

Acceptance tests

The system was tested by forcing different parts of the system to fail. For each test, the entire system was started and instructed to continuously process messages. Table 3.1 gives a brief overview of the acceptance tests and what they were intended to test for.

Test type	Component	Goal
Application shutdown	Broker	Assess whether a broker will send all received data even if it shuts down at an inopportune moment
Server shutdown	Broker Server	Assess whether a broker will send all received data even if its server shuts down at an inopportune moment
Disconnect	Broker Network	Assess whether a broker will send all received data even if it loses network connection at an inopportune moment
Disconnect	Consumer Network	Assess whether the system remains stable if the consumer loses connection, and that the consumer can continue from where it left off when reconnecting
Disconnect	Producer Network	Assess whether the system remains stable if the producer loses connection and that the producer can continue sending its data when reconnecting

Table 3.1: Table of acceptance tests and goals

Performance test

A performance test was planned and performed to decide which framework could offer the highest throughput. This was done by using two computers, one from each group member, and an OpenStack virtual machine provided by NTNU. The producer was ran on one computer, the broker on the virtual machine, and the consumer on the second computer. The producing computer then instructed production of either 20,000 or 40,000 messages per second and sent these across the network to the consumer as fast as possible.

For both the producer and the consumer, special code was written to take measurements from the performance test. On both the producer and consumer side, changes were made to log every 100,000 messages sent to console and the time it took to send it, as well as messages per second. This output were used as the results of the test.

Test Report

As part of the milestone plan between the group and AutoStore, a test report was scheduled for delivery in week 15.

4 Results

This chapter presents the results of the project. Results have three categories; scientific, engineering, and administrative.

Scientific results represent results that have no implementations but rather knowledge and findings discovered through the research methods described in section 3.1. For research to be considered a scientific result, it must be documented and made available for others to learn from or use the results later.

Engineering results present the concrete results of the project in terms of products created and code written. The software and all its related code are part of these results. Engineering results also include documentation belonging to the software.

Administrative results relate directly to the group, their organization, and their use of development methodologies. Adherence to development methodologies is also considered an administrative result.

All results are presented alongside necessary documentation as the results are presented and discussed. All documentation is available either as dependencies to this document or in public repositories in the case of code. Upon submitting this paper, the development team will provide the code as files in a folder to be submitted alongside this document.

4.1 Scientific Results

Throughout the project, two significant works of research were undertaken. The first was the research of prototypes at the beginning of the project, and the second was the testing at the end of the project.

A paper was produced after researching multiple frameworks. The research for this paper took place over multiple weeks and the resulting paper covers six frameworks (previously mentioned in Section 3.1.1) evaluated based on scalability, reliability, security, capacity, ability to deliver real-time processing, and compatibility. In addition to documenting the findings for the sake of the development team, the research paper also offers an overview of multiple popular streaming frameworks. In addition to being a part of this document, the research paper was handed over to AutoStore during week 3. This paper is available as Appendix 1.

The testing also produced a document. This document details the tests performed by the team, how they were performed, and under what circumstances. This document presents the strengths and weaknesses of Kafka and Pulsar in terms of reliability and performance, as both stability tests and performance tests are thoroughly documented. In addition to being a part of this document, the paper describing the tests was handed over to AutoStore during week 15. This paper is available as Appendix 2. The raw data from the tests are provided with this document as Appendix 3.

The paper's scientific results can serve others working with streaming frameworks in the future, as the documents can be used as reference materials when trying to select the most suitable framework for a use case. AutoStore in particular, as the stakeholder for this project, can benefit from having these papers available if they should be interested in further developing the existing solution or trying out a different framework. The papers are angled towards AutoStore's use case, and the papers will therefore provide more utility to them than to others.

4.2 Engineering Results

This section presents the engineering results that resulted from this project. Engineering results include style-guides, prototypes, the final solution, and scripts for creating clusters. Documentation for the final implementation is also included in this section.

4.2.1 Style Guide and Checkstyle

Using principles of SOLID design (see Section 2.10) and Clean Code (see Section 2.6) a style guide for the code was developed and utilized by the team. Although not a direct product of engineering, the style guide aided the group in creating maintainable, easily understandable, and technically solid code by setting common standards for the code written during the project.

Additionally, the team actively used Checkstyle configurations (see Section 3.2.2) to ensure similarity in the code written to improve readability and maintainability. Therefore, this Checkstyle configuration file is considered a minor engineering result of the project.

4.2.2 Prototypes

The prototypes produced during the projects are important results from this project. In addition to providing grounds for learning and experimentation, the prototypes stand on their own as functional solutions that can stream real-time data from endpoint to endpoint. Two completed prototypes exist, a Kafka prototype and a Pulsar prototype.

The Kafka prototype was the first one developed and is also the least complete prototype. Many of the core architectural decisions were made with this prototype. This prototype consisted of four components, and these components would be created for all other implementations as well. the four components are as follows:

Data Provider Java application component simulating the generation of data. Sends data to the producer

Producer Java application component that receives data from a local (on the same computer) agent, and sends it to a broker

Broker component provided by the framework-developers that manages the messages on their way from the producer to the consumer

Consumer Java application component. Endpoint for the data. Receives data from the broker. Implemented using a master-slave pattern where one master consumer creates multiple consumer workers

These components and their implementation are further described in Appendix 6.

The prototype has a Java implementation, Docker files, and a Docker Compose file making it possible to run the solution as jars or Docker containers depending on use-case and desire.

Development started on the Pulsar prototype after the Kafka prototype was finished. The Pulsar prototype is slightly different because the APIs for Kafka and Pulsar are different. However, the architecture remains the same, with one data provider simulating data, a producer receiving the data and passing it to a broker before a consumer receives the data from said broker. This prototype also provides Java code, Docker files, and a Docker Compose file enabling multiple ways of running the system.

Both prototypes have been made publicly available in their respective Git repositories. This code may be of use to other developers trying to implement real-time streaming services, as both prototypes provide functional examples of such systems with simulated data.

4.2.3 Final streaming solution

The final streaming implementation stands as the main engineering result of the project. The solution with its three parts, producer, broker, and consumer, provides a usable system for streaming data across the internet at high speeds with high reliability.

The system offers connectivity for the transmission of data through the Producer. The producer will listen to a configurable port on the local network and relay the data it receives to the broker. The broker will re-send the data to the consumer and the consumer will display said data.

The solution has been tested (see Section 4.1) and proven to work both locally and as a distributed system and when ran locally on only one computer.

4.2.4 Cluster Implementation

As the group successfully finished the producer and consumer components, work started on implementing a cluster version of the broker. Moving the broker to a cluster solution was deemed necessary as a cluster would enable horizontal scaling. The ability to scale horizontally with ease is essential for a growing company like the stakeholder of this project.

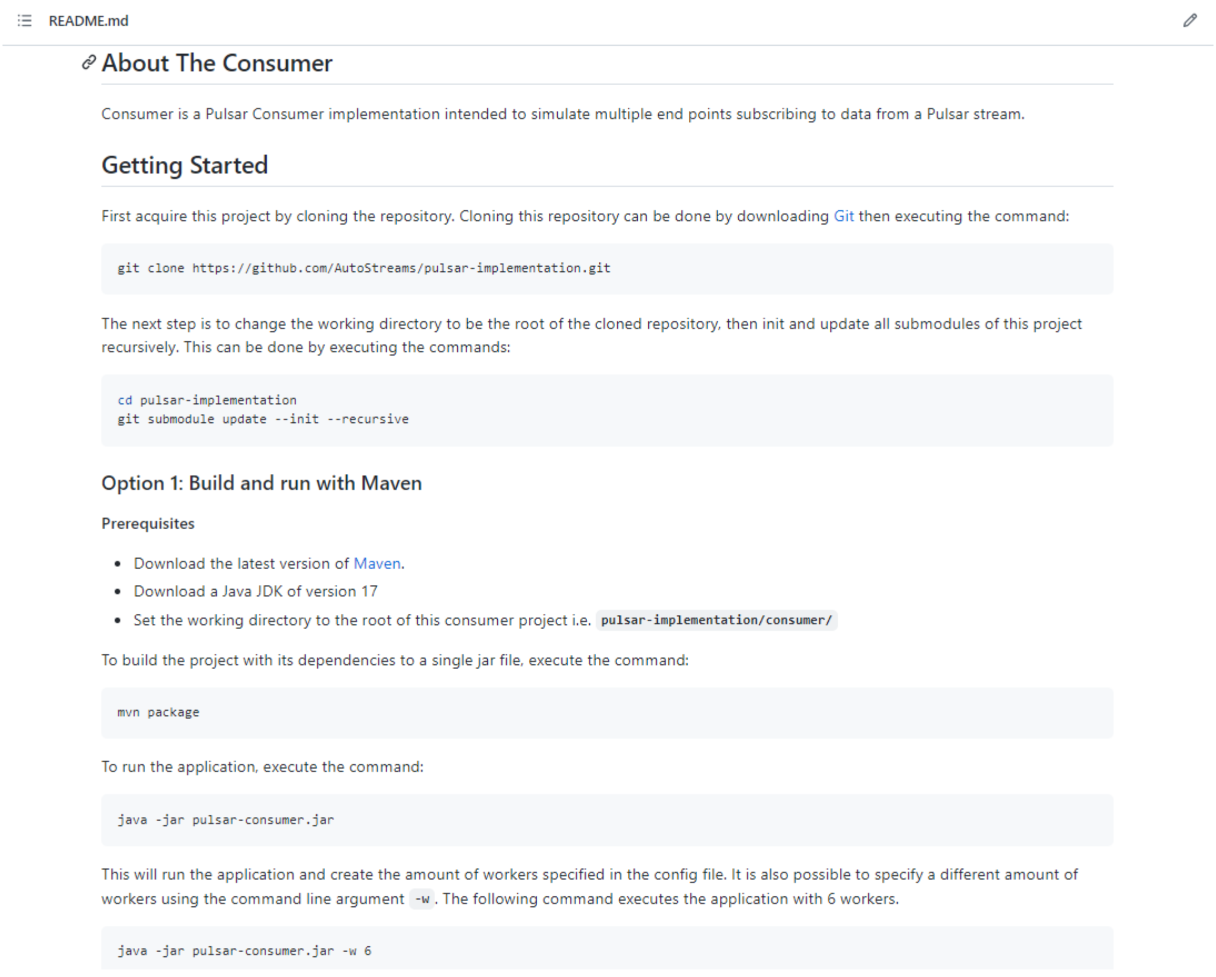
Work was started on the cluster implementation but was left in an incomplete state. The incomplete state is a result of difficulty during the deployment of the cluster. Despite this, two scripts were created to install and run a Pulsar broker cluster. First, the installation script downloads the required Pulsar files. Then, it replaces the configuration files with files written by the development team to allow a cluster to run on the computer executing the script. The run script then starts the cluster with the appropriate configurations. Running these scripts will install and run a cluster. The cluster will accept connections, but the cluster will start failing as soon as connections are received.

4.2.5 Documentation

The developed solutions have been thoroughly documented. Documentation for the solution exists both as a standalone document, and as documentation in the public GitHub repositories of the project. Javadoc based on comments in code is also avaialable.

The standalon documentation is provided alongside this document as Appendix 6 6.2.6. This document describes project architecture, installation, execution, and configuration in detail.

Additionally, there was a great deal of commenting in the code. This was done to ensure



README.md

About The Consumer

Consumer is a Pulsar Consumer implementation intended to simulate multiple end points subscribing to data from a Pulsar stream.

Getting Started

First acquire this project by cloning the repository. Cloning this repository can be done by downloading Git then executing the command:

```
git clone https://github.com/AutoStreams/pulsar-implementation.git
```

The next step is to change the working directory to be the root of the cloned repository, then init and update all submodules of this project recursively. This can be done by executing the commands:

```
cd pulsar-implementation
git submodule update --init --recursive
```

Option 1: Build and run with Maven

Prerequisites

- Download the latest version of Maven.
- Download a Java JDK of version 17
- Set the working directory to the root of this consumer project i.e. `pulsar-implementation/consumer/`

To build the project with its dependencies to a single jar file, execute the command:

```
mvn package
```

To run the application, execute the command:

```
java -jar pulsar-consumer.jar
```

This will run the application and create the amount of workers specified in the config file. It is also possible to specify a different amount of workers using the command line argument `-w`. The following command executes the application with 6 workers.

```
java -jar pulsar-consumer.jar -w 6
```

Figure 4.1: Screenshot of readme belonging to the consumer component

the code could be maintained and develop further by other developers than just the team for this project.

Additionally, all projects have comprehensive readme files available in the public GitHub repositories. The readme files do not go in to as much detail as the document, but provide instructions on how to install and run each component of the system (see 4.1).

Requirement documentation, detailing requirements in terms of user stories domain models, and class diagrams, was produced as part of the project. Appendix 7 presents this requirement documentation.

4.3 Administrative Results

4.3.1 Implementation of Scrum

Scrum was successfully implemented into the development process, although with slight changes compared to the initial plan.

Sprints were all one week long, with 16 sprints for 16 weeks. In addition, the team performed the agreed-upon Sprint retrospectives, Reviews, and Plannings (see Section 2.1.4) every week. The sprint retrospectives were used to identify how the team worked together, reviews to show off work between team members, and planning each week to pick the most critical tasks.

Documentation exists for the Retrospectives and Reviews in the system manual. Every week has a document for its retrospective, but weeks 11, 12, and 13 are missing review documents. It is unknown why these documents are missing, as they were all written collaboratively and stored in Jira. The team may have possibly failed to click ”Publish” after finishing writing the documents, thus losing them upon closing the browser.

Documentation for every week in the form of screenshots also exists. In addition, the team decided to present burndown charts, and velocity charts showing a timeline of issues and story points. Do note that some burndown charts show too large periods because they have later been re-opened and closed to double-check related issues, moving the closed date. Burndown charts for each week are available in the project manual.

4.3.2 Roles and Responsibilities

As mentioned in Section 3.2.8, the group had a scrum master and a product owner based on personal skills. There were also technical roles related to the four components of the system described in section 4.2.2. These roles came naturally as work was distributed, with each team-member specializing in one or more component of the system, with the broker being a shared responsibility. Tomas Klungerbo Olsen served as the scrum master, while Lars Ivar Ramberg was the product owner with both members being considered part of the development team (see section 2.1.5). Table 4.1 gives an overview of the scrum responsibilities, while table 4.2 gives an overview of the technical responsibilities.

Role	Assignee	Description
Scrum master	Tomas Klungerbo Olsen	Responsible for ensuring scrum is followed, and handling blocking tasks. Helps product owner manage the backlog
Product owner	Lars Ivar Ramberg	Responsible for backlog management and communication with stakeholder

Table 4.1: Overview of assigned scrum roles

Component	Assignee	Description
Data provider	Tomas Klungerbo Olsen	Implementation and documentation
Producer	Tomas Klungerbo Olsen	Implementation and documentation
Broker	Tomas Klungerbo Olsen, Lars Ivar Ramberg	Configuration and documentation
Consumer	Lars Ivar Ramberg	Implementation and Documentation

Table 4.2: Overview of technical responsibilities

4.3.3 Milestone plans

Two milestone plans were used during development to guide the team, and to communicate with the stakeholder what they could expect each week.

The first milestone plan laid out milestones for every week of the project. This milestone plan is the one described in the pre-project plan (Appendix 5). A second and final milestone plan was produced later in development after development of the first two prototypes (see Section 3.2.15) had concluded.

4.3.4 Final streaming solution

The final streaming implementation stands as the main engineering result of the project. The solution with its three parts, producer, broker, and consumer, provides a usable system for streaming data across the internet at high speeds with high reliability.

The system offers connectivity for the transmission of data through the Producer. The producer will listen to a configurable port on the local network and relay the data it receives to the broker. The broker will re-send the data to the consumer and the consumer will display said data.

The solution has been tested (see Section 4.1) and proven to work both locally and as a distributed system and when ran locally on only one computer.

5 Discussion

This chapter discusses the results presented in Chapter 4. The chapter explores the results in light of the requirements in order to evaluate whether the result satisfies the requirements and expectations from the stakeholder. The process and methodologies used are also discussed to narrow down what were the strengths and weaknesses of the chosen methodologies. Both strong and weak aspects are discussed in order to have a realistic discussion about the results, and to provide guidance to future developers of similar solutions.

5.1 Scientific discussion

The scientific results and work were done with great success, but with room for more work to be done. The scientific results, as presented in Chapter 4.1 provided a good amount of information on their given subjects and satisfied the requirements from the stakeholder, even if more could still be done.

5.1.1 Framework Research

The framework research was thorough and detailed, which led the development team to make good choices regarding technology choice. In addition to improving the quality of the end-product as a result of informed choices, the document created also conveys the information gained during the research phase in a way that could be re-used by other developers in the future.

Research and Documentation

The research and documentation could be as thorough as it was due to good planning at the start of the project. In the original milestone plan (see Appendix 6) two whole weeks were set aside to undertake this research and the group used these weeks almost exclusively for that purpose. The combination of good planning and the stakeholders view of research as valuable allowed the development team to conduct large amount of research and document their results thoroughly.

The research of the framework could have been improved with even more time to look into more frameworks. The research paper only covers six out of the many frameworks available and it is possible that other frameworks could also have fit the use case. Due to not being aware of their existence, the group did not look into any of the streaming services provided natively in Azure as a service. As the stakeholder explicitly wanted the solution to be Azure compatible, this would have been a helpful thing to investigate. The Azure services remained un-investigated as the group were not aware of their existence until Azure-testing started late in development.

5.1.2 Testing and Documentation of Testing

The testing performed was extensive and valuable, but with additional room for more exploration, mainly in terms of the Performance tests performed.

Stability tests

The stability tests were the most successful part of the testing. The tests managed to confirm that the implemented software modules could manage failures as long as their machines did not go down as well. The instability of the broker upon server shutdown (See section 4.1) was an important discovery made during testing as well.

Performance tests

The performance testing also provided useful information, but could have been expanded and tested system performance in other meaningful ways. An example would be to test the system for capacity to horizontally scale. The current performance test (see section 3.2.16) test the systems ability to handle large loads of data. While testing handling of high data loads is important and gives an indicator of what might happen with multiple connections to a broker, it does not guarantee that the broker will handle multiple simultaneous connections. What was tested was one connection sending hundreds of thousands of messages, what should also have been tested is thousands of connections with a moderate amounts of messages each.

Additionally, the tests should have been performed in a more accurate way in terms of repeatability. The data sent during tests were randomly generated strings as that did not have guarantees in terms of length and size. The strings did have a pre-defined range of lengths, but there was no guarantee that the thousands of messages sent during one test would equal the messages of another in terms of size and complexity. This was alleviated somewhat by the afore-mentioned range of lengths, but to make the tests more equal it would be better to send the copies of the same data many times.

5.2 Engineering Results

The engineering results largely satisfy the requirements posed by the stakeholder. The developed solution can send data over the internet reliably, with good capacity, and from anywhere in the world. However, there are still important features left to be implemented before the solution can be used in the real world.

5.2.1 Streaming and Communication of Data

The streaming aspect of the solution is functional and in adherence to the original assignment from the stakeholder. Data is sent, from geographically distributed computers, and is received in correct order. This aspect was worked on right from the start of the project and has therefore had more development time than any other feature which is likely partially why this feature works reliably and in the way it was intended. The contribution of the streaming framework used for the final solution also helped get the streaming completed, as the streaming was easy to implement using already existing APIs. Strong documentation on the behalf of Apache helped the group quickly use the provided APIs to implement the streaming capabilities.

5.2.2 Security

Security was a requirement not directly requested by the stakeholder but deemed very important by the group as the system would be used to send confidential data related to the installations belonging to AutoStore customers. If parties with bad intentions were to get a hold of this data, they could learn about the inner workings of the AutoStore system which could compromise AutoStores position as their solutions could then be easily copied.

The group attempted to enable the TLS features of the Pulsar framework used for the final solution, but the implementation was unsuccessful. The implementation was attempted using official documentation and by asking for help from fellow students with more experience in the are than the development team. All attempts eventually ended in the broker refusing to accept connection from both the consumer and producer.

There are many possible reasons as to why this failed. The TLS scheme supported by the Pulsar framework uses an asymmetric-key encryption scheme (see section 2.9) so it is possible that the keys were improperly configured on each attempt. Enabling TLS in Pulsar requires a great deal of configuration on the broker, so it is also possible that the configuration of the broker was improperly done by the group.

It is also worthy of note that none of the members of the development team have extensive experience with security. The concepts have been introduced and explained in courses related to networking and mathematics, but neither member has attempted to implement security and encryption in a way similar to the one required by Pulsar. This lack of knowledge should have been recognized by the team earlier, so that more time could have been set aside for implementation and work on the feature could have started earlier.

5.2.3 Implementation of Broker as a Cluster

To maximize the support of horizontal scaling, the broker should be implemented as a scalable cluster that can have more resources added to it as the system requirements expand with more producers connected to the same broker. Work was started on this, but was unsuccessful, despite having produced two scripts that install the required files to run a cluster locally on a computer.

The complexity of clusters was a large contributing factor to why this was not finished. It was also not set as a requirement by the group nor the stakeholder, but it was a desired feature by the development team. The idea was to start by creating a local cluster on a local machine, before distributing the cluster among multiple machines. This would allow machines for different regions to manage connections from multiple regions. This clustering was eventually dropped as the development team failed to make satisfying progress and the stakeholder did not require it, allowing focus on features requested by the stakeholder.

5.2.4 Documentation

The produced documentation is thorough, comprehensive, and in the case of the public repositories freely available to the public. Ensuring good documentation is publicly available is essential for the solution to be used and develop further by other developers. Seeing as the solution is still missing some features, it is extra important that sufficient documentation is produced so that the missing features may be added in the future.

The thoroughness on documentation is a result of a focused effort from the group. The trade-off from this focus is that time that could have been used to develop the solution further was used on documenting the work. Documentation and development must be balanced, as insufficient documentation will make development more difficult.

Documentation was also helped a great deal by the continuous integration job (see Section

3.2.3) automatically generating Javadoc based on the comments present in the code.

Overall the existing documentation should be sufficient both for users looking to download and run the solutions, and for developers who want to develop the solution further.

5.3 Administrative discussion

Administrative results from this project were overall satisfying, although important lessons were learned regarding planning for and managing a small team. The use of scrum and their regular meetings provided a stable framework for collaboration, but the plans made by the team could in hindsight have been made more efficient.

5.3.1 Implementation of Scrum

The implementation of Scrum was largely successful. The team had regular meetings throughout the process to ensure that both members knew what the other member was working on, all important decisions were documented in meeting notes, and the team members managed to keep communication flowing within the group. Some adjustments had to be made with regards to the meeting schedule to accommodate for other obligations of the team members without jeopardizing the project and its development.

Meetings and Meeting Frequency

The meetings had added importance to the group as the group was organized as a distributed team (see Section 3.2.6). The geographical separation inherent to distributed teams necessitates some form of mechanism or scheme to encourage communication between the team as they cannot meet physically and are thus not able to meet in person to discuss and work together.

The meeting frequency and times were altered from what was originally planned as development progressed. The stand-up meetings saw the biggest change in that some of them were omitted in the month of March. During the calendar weeks of 10, 11, and 12 only one stand up meeting was held as opposed to the scheduled six. The primary reason for this was that the group members were busy with obligations in another course and decided to focus on that course in the month of March. The shift of focus led to little work being done during these weeks, which also eliminated the need for daily stand up meetings. The group members also worked together on the other course which meant the group still communicated and could discuss the project if needed. After the course terminated, stand ups were quickly reintroduced as a valuable tool to keep communication flowing during active development.

Sprint retrospectives, reviews, and plannings (see Section 2.1.4) were performed at planned frequency, but varied in terms of what day they were performed. The rule of thumb was to perform these meetings on Sundays as that was considered the end of the work week, but sometimes the meetings would be moved to Mondays to give a bit more time to work on issues. On rare occasions, the meetings would take place later in the week as well if the team members had other obligations they needed to attend to at the beginning of the week that would prevent progress from being made. This flexibility ensured that the meetings would always take place and this was crucial both for the team to evaluate their product during review and their process during retrospective, as well as thoroughly plan for the upcoming sprint during planning. Flexibility was used to the teams advantage to ensure all core scrum meeting activities were performed.

5.3.2 Milestones

As development progressed, the milestones needed adjustments. There were two main reasons for this: the original milestone plan did not account for work in other courses, and it did not account for illness or delays. Other academic obligations pushed the project back slightly, and illness in the development group meant development capacity got halved for a week.

About half-way through development, the milestone plan was revised. Due to delays related to illness and work in other courses, the development time for the third prototype was cut down to one week. The revised plan is available in Appendix 9.

Despite the revision, the development would deviate from the plan shortly after. As described in Section 3.2.15, it became apparent that the three planned prototypes would be reduced to only two as the third framework was deemed unfit for the use case. It became apparent that Flink could not offer any functionality to target the project requirements (see section 1.3) that the other frameworks could not already offer. It was then agreed upon verbally during a meeting with the stakeholder that this prototype would be scrapped, and the entire plan would be moved ahead by one week (week 15's milestone would become week 14, and so on). The week that was going to be used for developing the Flink prototype was instead set aside for testing and implementation in Azure, expanding the time slot from two weeks to three weeks.

6 Conclusion

This chapter presents conclusions and suggestions for further work based on the results presented in Chapter 4 and the discussions of Chapter 5. Conclusions are formed based on the requirements compared to the achieved results.

6.1 Conclusions

Conclusions can be drawn form engineering, scientific, and administrative results of this project.

Comparing with the requirements outlined in section 1.3, the project meets the requirements set by the stakeholder, but is lacking in the requirements enforced by the group. The system is performant, reliable, fault tolerant, and cloud compatible, meeting all stakeholder requirements. From the groups requirements, the solution is scalable, but not secure in any way. Neither encryption nor authentication has been implemented, which was deemed a requirement by the group. As previously expressed in section 5.2.2, this must be implemented before the solution is used by a business or the stakeholder.

The scientific results provide a great amount of researched knowledge, and exceeded the groups initial expectations in terms of scope and results. However, the results should prove useful to future developers, even if there is room to do more research on different streaming frameworks and other types of tests to perform.

Finally for administrative conclusions, there were many correct steps taken and some miss-steps. The flexibility within the group was hugely helpful in ensuring that development of this project would not be haltered by other obligations, but this was of course helped by the fact that the group was very small with only two members. Still, based on how the administration of this project turned out, the flexibility seems to be a good approach for small distributed development teams.

Based on the velocity charts and work estimates the estimation of points using planning poker should be done differently than what was done for this project. Point estimates should be more pessimistic than what was done by the group, especially when a team is working with new technologies or developing a type of product they have little experience with.

6.2 Further work

The following sections cover features that were not implemented with suggestions to improve the product beyond its current state.

6.2.1 Authentication

Verifying access rights to resources is necessary for a system to be secure, as, without it, anyone can access or manipulate them. As authentication was not a part of the agreement between AutoStore and the development team, it has been left out of the current solution.

Pulsar offers documentation and a tutorial on implementing authentication for a producer, broker, and consumer on their official website. The development team would have prioritized authentication if they were to continue beyond the final delivery.

6.2.2 Encryption

Encrypting the data is highly encouraged to send data securely between end devices. The product does not currently support encryption in its current state. The development team attempted different tutorials and documentation despite being unsuccessful.

Further reading of Pulsar's documentation on message encryption for transportation should be done so the product can send secure messages over the network. The development team would have prioritized encryption along with authentication (see section 6.2.1) if they were to develop the solution further.

6.2.3 Pulsar property limitations

Pulsar offers a wide variety of different properties that can be configured. The combination of these properties dictates the behavior of the system. The Java client API specifies these properties through Java objects, making it difficult to load the configuration from a file into the system directly. The current solution reads the properties from a configuration file and converts them to the correct Java object representation.

The development team only converted properties strictly needed for the requested product, making the solution less flexible. Additional property conversion should be implemented to support a broader range of configurations.

6.2.4 Cloud cluster solution

The current version of the project uses the standalone broker solution provided by Pulsar. This solution is not recommended for a system with high growth as it will reach limits in the future. A multi-cluster or a cluster of brokers with their zookeepers and bookkeepers are the recommended solutions as they enable more reliance and throughput. Clusters also open the possibility of brokers scattered around geographical locations allowing quicker access from an installation (producer) to the cloud server (broker). In addition, geographical scattering allows installations to contact other cloud servers when their central cloud server has unexpectedly or expectedly disconnected.

Although the stakeholder did not desire the feature, more in-depth reading of Pulsar's documentation and tutorials to enable clustering would be an interesting feature to implement since achieving clusters for the product would accommodate systems needing massive scaling and geographical scattering.

6.2.5 Infrastructure as code

Defining a virtual machine's resource allocation and configuration through code can have multiple benefits. A resource defined as code can easily be created and deleted upon request, and the infrastructure code can be shared to create duplicates efficiently. The flexibility of this reproduction through code also makes it easier to test the system with automated tools during a continuous integration phase(Morris, 2021)

The team intended to use Terraform to configure their cloud resources. Upon further development, the team would have investigated resource creation and configuration through code with tools like Terraform and Ansible so that the system would be more accomodating to change, testing, and reusability.

6.2.6 Unit testing

Testing code in a program can improve the overall quality of the software. One branch of such testing is unit testing which operates and validates code units. Unit tests are robust when combined with continuous integration as it allows for automated testing of code units during changes to the software.

The development team discussed implementing unit tests for the solution on multiple occasions but decided not to due to the nature of the program being a network application with a producer and a consumer. Integration-testing of the whole system would therefore be more beneficial than unit-testing. This decision was further reinforced as there was an extensive testing phase of the solutions during the development providing integration-testing for both components. However, for future development, the development team would set aside more time to find code that qualifies for unit testing and implement the corresponding tests.

Social impact

The project was made open source and freely available to everyone by using MIT licensing. This licensing enables anyone to use, distribute, and even develop commercial software using the solutions from this project (Initiative, 2022). This is a contribution to the overall open-source community, as the group freely provides example code and documentation for other developers to use and learn from. This access requires no payment, enrollment in programs, or titles, thus helping democratize development across socio-economic backgrounds.

In addition to the solution itself, the choice of stakeholder and assignment is important from a societal and ecological standpoint. Choosing a stakeholder that pollutes and acts in ethically questionable manners would make the developers complicit and accepting of their actions. AutoStore provides a green solution running on electricity, a potentially green and environmentally friendly energy-source.

This project relates to the field of automation in two ways. Firstly, AutoStore as a company offers an automation solution for warehouses that seeks to replace work done by human hands with robot substitutes. Secondly, the streaming project is part of automating the consumption, use, and data within AutoStore. Both these automation cases have their upsides and downsides.

For the warehouse workers, many will be saved from injury and strain as jobs involving heavy lifting will be done by robots. At the same time, this could lead to warehouse-workers losing their jobs to the automated solutions. AutoStore systems still need humans to interact with them. Maintaining the robots and interacting with the ports to receive merchandise must still be done manually, which might mitigate this downside.

AutoStore and other automated warehouse solutions might have a greater negative impact in developing countries. If an international company wants to put a warehouse in such a country, they will need to hire workers to operate the warehouse. Warehouse workers typically do not require education and these jobs can provide income for families that may otherwise not have any other way to earn a living. Taking these low-skill jobs away could impact families that would otherwise depend on them.

Many of the same upside and downsides also apply from the perspective of automating data use. Manual tasks of data analysis will be eliminated with the automation of such tasks. While the risk of human error will be eliminated, it will also eliminate certain roles related to data analysis as more of it is handled by software.

References

Adam, John (2022). *Agile software development with the Scrum framework*. URL: https://kruschecompany.com/agile-software-development-with-scrum-framework/ (visited on 18th May 2022).

Atlassian (2022). *What is version control?* URL: https://www.atlassian.com/git/tutorials/what-is-version-control (visited on 23rd Apr. 2022).

Bartholomew, Chris (2020). *DataStax What is Apache Pulsar?* URL: https://blog.linkedin.com/2011/01/11/open-source-linkedin-kafka (visited on 22nd Apr. 2022).

Chen, James (2021). *Encryption*. URL: https://www.investopedia.com/terms/e/encryption.asp (visited on 15th May 2022).

Compliance, Business Audit (2018). *Planning Poker – Agile Estimation Technique*. URL: https://auditandcompliance.wordpress.com/2018/12/31/planning-poker-agile-estimation-technique/ (visited on 18th May 2022).

Coursera (2022). URL: https://www.coursera.org/articles/scrum-roles-and-responsibilities (visited on 23rd Apr. 2022).

Docker (2022). URL: https://www.docker.com/resources/what-container/ (visited on 19th May 2022).

Doshi, Hiren (2022). *How Do the 3 Scrum Roles Promote Self-Organization?* URL: https://www.scrum.org/resources/blog/how-do-3-scrum-roles-promote-self-organization (visited on 18th May 2022).

Driessen, Vincent (2010). *A successful Git branching model*. URL: https://nvie.com/posts/a-successful-git-branching-model/ (visited on 20th Apr. 2022).

Fowler, Frederik M. (2019). 'The Scrum Development Team'. In: *Navigating Hybrid Scrum Environments: Understanding the Essentials, Avoiding the Pitfalls*. Berkeley, CA: Apress, pp. 39–45. ISBN: 978-1-4842-4164-6. DOI: 10.1007/978-1-4842-4164-6_6. URL: https://doi.org/10.1007/978-1-4842-4164-6_6.

Gillis, Alexander S. (2021). URL: https://www.techtarget.com/searchitoperations/definition/Docker-image (visited on 19th May 2022).

Hoffman, David (1993). *What Is The Cloud*. URL: https://ghostarchive.org/varchive/_a7hK6kWttE (visited on 30th Apr. 2022).

Initiative, Open Source (2022). *The MIT License*. URL: https://opensource.org/licenses/MIT (visited on 19th May 2022).

Martin, Robert Cecil (2009). *Clean code. A Handbook of Agile Software Craftsmanship*. Pearson Education.

McCarty, Scott (2018). URL: https://developers.redhat.com/blog/2018/02/22/container-terminology-practical-introduction# (visited on 19th May 2022).

Morris, Kief (2021). 'What Is Infrastructure as Code?' In: *Infrastructure as code: Dynamic Systems for the cloud age*. 2nd ed. O'Reilly Media, Inc., pp. 9–11.

Oloruntoba, Samuel (2021). *SOLID: The First 5 Principles of Object Oriented Design*. URL: https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design (visited on 15th May 2022).

Paradigm, Visual (2022). *What is User Story?* URL: https://www.visual-paradigm.com/guide/agile-software-development/what-is-user-story/ (visited on 14th May 2022).

Radigan, Dan (2022). *Story points and estimation*. URL: https://www.atlassian.com/agile/project-management/estimation (visited on 22nd Apr. 2022).

Rajitha, Charith (2021). URL: https://medium.com/@rajithacharith/asymmetric-encryption-and-signing-with-keytool-openssl-a014b2ddf01a (visited on 19th May 2022).

Rao, Juan (2011). *LinkedIn Official Blog Open-sourcing Kafka, LinkedIn's distributed message queue*. URL: https://blog.linkedin.com/2011/01/11/open-source-linkedin-kafka (visited on 22nd Apr. 2022).

Rehkopf, Max (2022a). *User stories with examples and a template*. URL: https://www.atlassian.com/agile/project-management/user-stories (visited on 30th Apr. 2022).

Schwaber, Ken and Jeff Sutherland (2020). *The 2020 Scrum Guide*. URL: https://scrumguides.org/scrum-guide.html (visited on 20th Apr. 2022).

Termenji, Artur (2021). *What is Pair Programming and How to Practice it in a Remote Team*. URL: https://railsware.com/blog/what-is-pair-programming/ (visited on 18th May 2022).

TKSL (2020). *AutoStore Develops Router Software, Improving Throughput by 4 Times*. URL: https://www.tksl.co.jp/en/information/2020/1029/000146.html (visited on 18th May 2022).

Watts, Stephen (2020). *The Importance of SOLID Design Principles*. URL: https://www.bmc.com/blogs/solid-design-principles/ (visited on 15th May 2022).

Weaveworks (2020). *Docker vs Virtual Machines (VMs) : A Practical Guide to Docker Containers and VMs*. URL: https://www.weave.works/blog/a-practical-guide-to-choosing-between-docker-containers-and-vms (visited on 18th May 2022).

West, Dave (2022). *Scrum roles and the truth about job titles in scrum*. URL: https://www.atlassian.com/agile/scrum/roles (visited on 22nd Apr. 2022).

Wikipedia (2022). *Apache Kafka*. URL: https://en.wikipedia.org/wiki/Apache_Flink (visited on 19th May 2022).

Williams, L. (2001). 'Integrating pair programming into a software development process'. In: *Proceedings 14th Conference on Software Engineering Education and Training. 'In search of a software engineering profession' (Cat. No.PR01059)*, pp. 27–36. DOI: 10.1109/CSEE.2001.913816.

Appendices

Appendix 1: Research Streaming Frameworks

NTNU | Norwegian University of Science and Technology

Research: Streaming Frameworks, best fit for AutoStore use-case

Norwegian Institute of Science and Technology 2022

on behalf of AutoStore AS

Authors: Tomas Klungerbo Olsen, Lars Ivar Ramberg

Background and purpose

This paper represents the first hand-in to be provided to AutoStore AS as part of the Bachelor Project “Strømmetjenste for Data” at NTNU, 2022. The student group and AutoStore have agreed on a set of milestones and deliverables to be presented during the project, this paper is the first of said deliverables.

This paper presents what the group has learned regarding the needs of AutoStore as a stakeholder and streaming services, both in terms of theoretical details and available frameworks. The stake-holder needs and theory are presented first, before the relevant streaming frameworks are discussed. In addition to presenting the frameworks, this paper also offers an evaluation of each framework, providing an overview of which frameworks will be relevant for the prototyping and possibly implementation stage of the project.

User Needs and Key Considerations

The use-case of AutoStore is unique and challenging. AutoStore is a global company, with hundreds of installations across the globe that generate large amounts of data every moment of the day. Up until now the data from the AutoStore sites have been accumulated over a set period (usually a day) before all the data from said day has been made available the subsequent day. The data is then processed as AutoStore receives the data, in batches of one day. It is desirable to see data and events from AutoStore sites as they happen, thus enabling swift action based on information from the system.

The key objective of this project is to move away from the batch processing by sending the data the moment it is generated as opposed to in one great batch the next day. This assignment focuses fully on the sending of the data, as the processing is already being handled internally by AutoStore. Figure 1 illustrates this separation and gives an overview of the data flow from an AutoStore system to an end-user/consumer of the data.

Based on the information presented above, the following key considerations have been identified:

- Scalability, as AutoStore is growing
- Reliability, as we want reliable data
- Security, as data directly connected to AutoStore assets will be sent
- Real-Time or Close to real-time timeframe, as the data should enable swift action
- Ease of installation and use
- Compatibility, yet generalization (needs to be compatible with AutoStore, but general enough to be used elsewhere)
- High speed and capacity due to large amounts of data

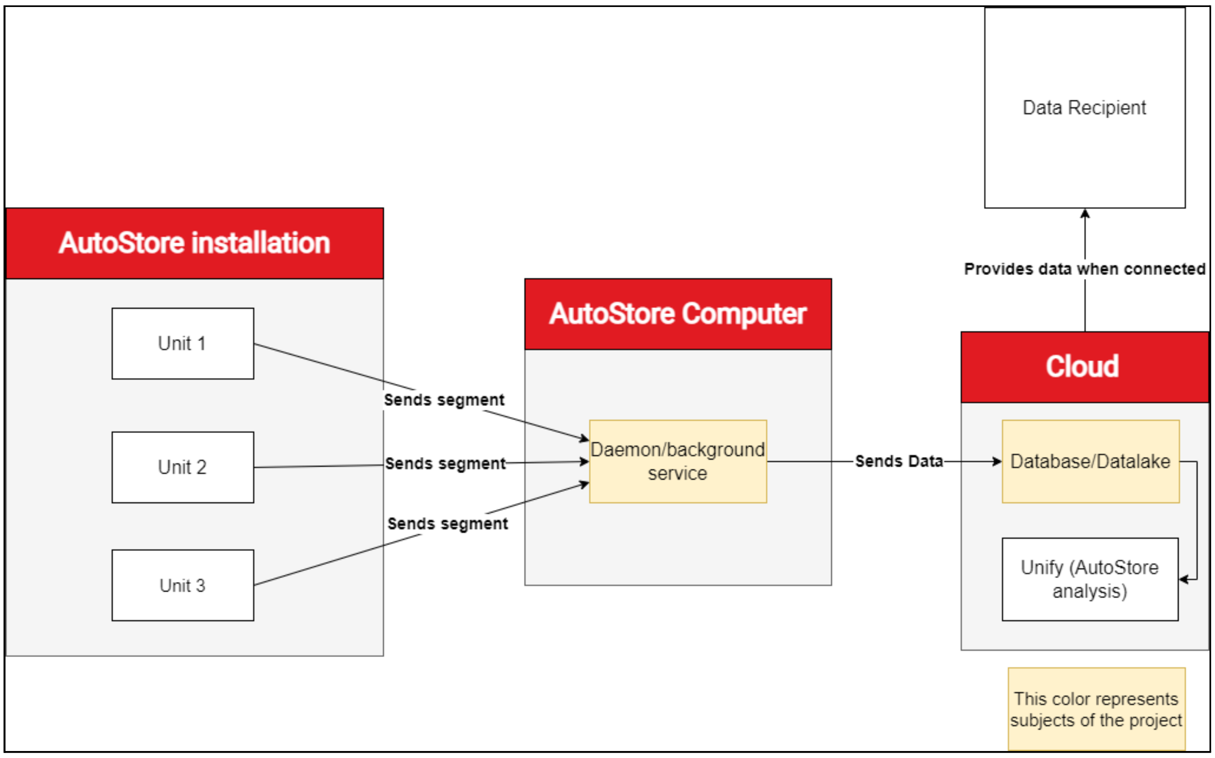


Figure 1: Illustration of data transport from AutoStore to end user. Yellow boxes represent areas covered by the project.

Streaming data, the theory

When streaming data, as opposed to just storing it, the goal is that a user can connect to a channel, stream, or similar, and either receive data real time, or receive parts of data as needed as opposed to downloading all the data and selecting the desired parts. Consider the streaming of video, for example. The user does not have to download a full video before they start watching it, they can watch it in parts as the data of the video reaches their device. They can also select which parts of the video they are interested in, thereby avoiding the sending undesired or uninteresting data to the user.

In this project, the focus will primarily be on the real-time aspect of streaming. This is because the use-case of AutoStore requires this real-time streaming more than streaming of stored data, although both of these functionalities are feasible.

It is assumed that the data being streamed will always be one way. The user will always be receiving data, but not altering or sending data back, at least not in the exact same channel as the streamed data. This means that the data-stream, no matter how many intermediaries there are, will be a Directed Acyclic Graph , meaning that there are no loops in the datastream, and ultimately data is always moving from the source towards the end node.

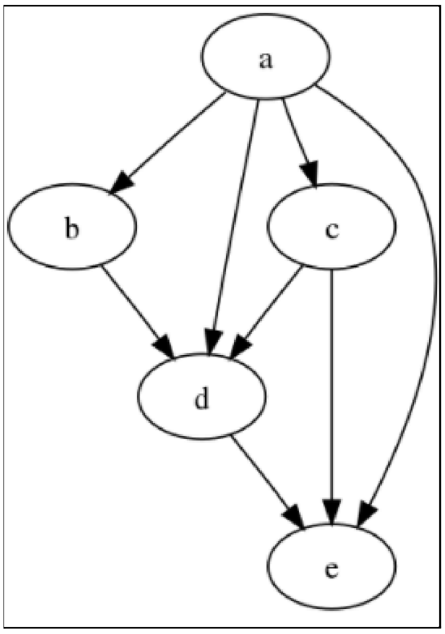


Figure 2: Directed Acyclic Graph, https://en.wikipedia.org/wiki/Directed_acyclic_graph

Frameworks

In the following sections, different frameworks and technologies for streaming are presented. The frameworks are analyzed and evaluated in regards to the key considerations presented in section “User Needs and Key Considerations”.

Framework	**Developer/Maintainer**	**Open Source?**
Apache Kafka	Apache Software Foundation	Yes
Apache Pulsar	Apache Software Foundation	Yes
RabbitMQ	Pivotal Software (now part of VMware)	Yes
Apache Spark	Apache Software Foundation	Yes
Apache Storm	Apache Software Foundation	Yes
Apache Flink	Apache Software Foundation	Yes

Apache Kafka

"Apache Kafka is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications."

- https://kafka.apache.org/

Overview

Apache Kafka serves as an intermediary between producers and consumers, resulting in the decoupling of data streams and services. Kafka focuses on storing events instead of the traditional database way of storing "things." These events are stored in topics which essentially are lists of events with some state associated with them. These topics use redundancy in the form of data replication to mitigate data loss. Topics can be stored for minutes to hundreds of years depending on need, and they can be of any size. Any service can read events from a topic, process the data, and then store the processed data back into a topic. These topics are persistent, making it possible for them to be reread after consumption.

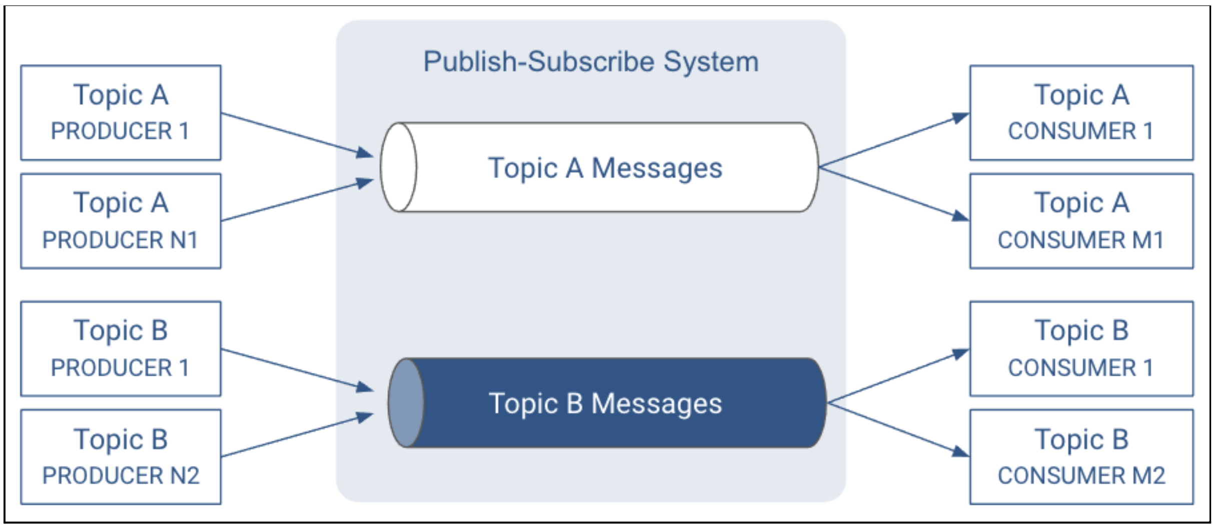


Figure 3: Apache Kafka internals, https://docs.cloudera.com/runtime/7.2.10/kafka-overview/topics/kafka-overview-topics.html

Topics are stored in a Kafka Broker, which is a Kafka server. The combination of multiple Brokers forms a Kafka Cluster. Kafka Zookeeper manages the Kafka Brokers in a cluster both in horizontal scaling and duplication of topics.

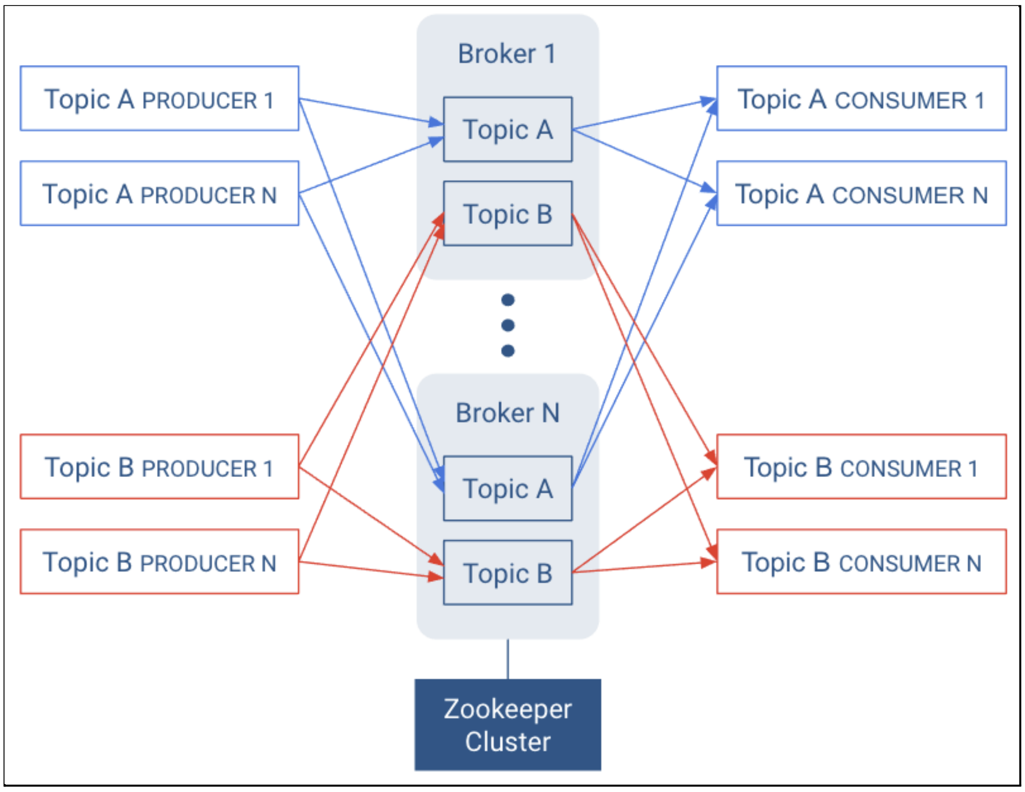


Figure 4: Apache Kafka's Zookeeper to Broker interaction, *https://docs.cloudera.com/runtime/7.2.10/kafka-overview/topics/kafka-overview-brokers.html*

Security

Supported protocols for authentication to brokers in the current version are:

- SSL
- SASL/GSSAPI
- SASL/PLAIN
- SASL/SCRAM-SHA-256 and SASL/SCRAM-SHA-512
- SASL/OAUTHBEARER

SSL can be used for both authentication and encryption of data. This feature is not enabled by default.

Older versions of Apache Kafka and Apache Kafka running on an older version of Eclipse Jetty have known security vulnerabilities that are not present in current releases.

API

Apache Kafka API is designed to use a language-independent protocol. Although the main client maintained by the Kafka project is Java, there are many independent open-source bindings for other languages. A complete list of these supported languages can be found at https://cwiki.apache.org/confluence/display/KAFKA/Clients.

Conclusion

Apache Kafka is a producer-to-consumer streaming framework that sends data as events. The framework is open-source, horizontally scalable, and secure, making it a perfect candidate for this assignment.

Apache Pulsar

"Apache Pulsar is a cloud-native, distributed messaging and streaming platform originally created at Yahoo! and now a top-level Apache Software Foundation project"

- https://pulsar.apache.org/

Overview

Apache Pulsar started as a message queuing system and later expanded into streaming (Kafka vs Pulsar - Performance, Features, and Architecture Compared, n.d.). The framework shares much of the same terminology as Apache Kafka. They both use Apache Brokers as servers and an Apache ZooKeeper to manage the cluster of Apache Brokers. Apache Pulsar also uses the concept of producers and consumers to generate and process data. Apache Pulsar has an addition of an Apache BookKeeper, which it uses to persist data. Apache BookKeepers can separate the storage into tiers, allowing old data storage on cost-efficient solutions (Despot, 2021).

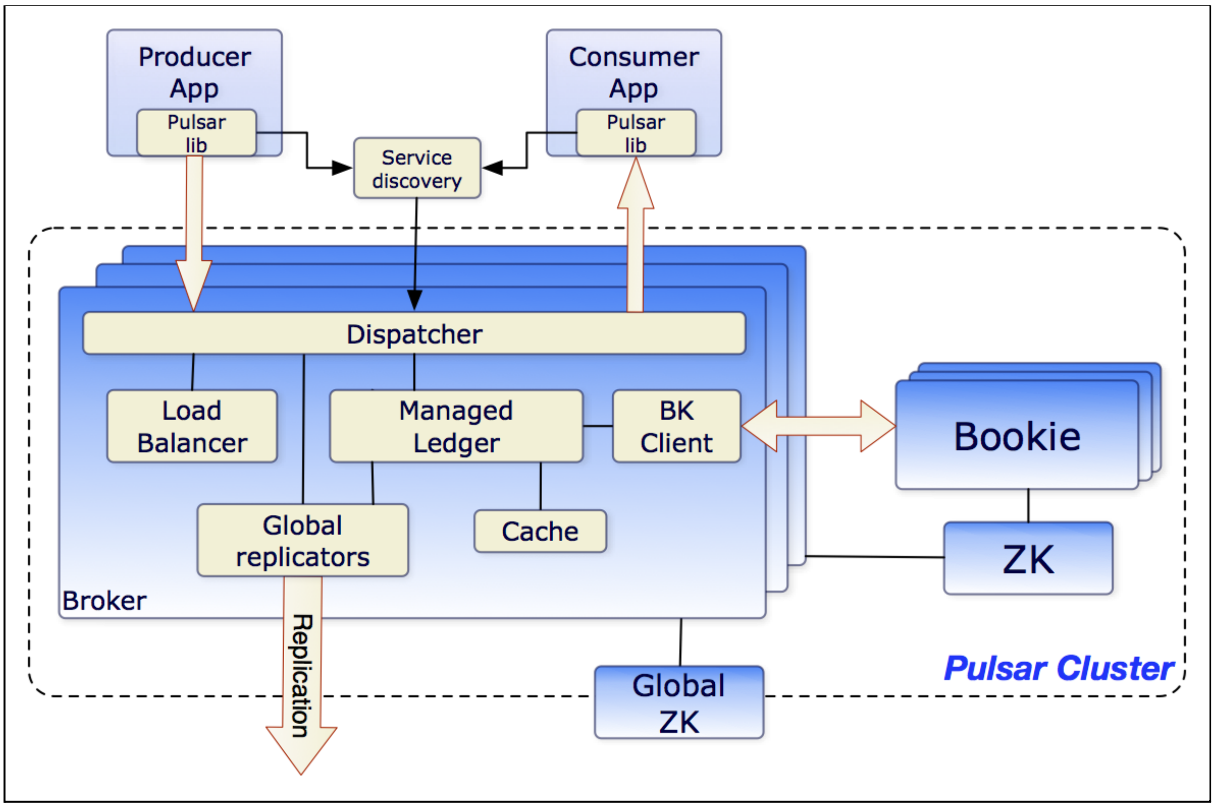


Figure 5: Apache Pulsar internals,
https://memgraph.com/blog/pulsar-vs-kafka

Security

There is currently one known vulnerability in the current version of Apache Pulsar.

“If Apache Pulsar is configured to authenticate clients using tokens based on JSON Web Tokens (JWT), the signature of the token is not validated if the algorithm of the presented token is set to "none". This allows an attacker to connect to Pulsar instances as any user (incl. admins).”

- https://www.cvedetails.com/cve/CVE-2021-22160/

API

There are official client APIs for:

- Java client
- Go client
- Python client
- C++ client
- Node.js client
- WebSocket client
- C# client

There are also unofficial client APIs for:

- Haskell
- Scala
- Rust
- .NET

The WebSocket API can be used by any programming language with a WebSocket library. There is also an Admin API that uses REST accessed by HTTP calls.

Conclusion

Although Apache Pulsar comes with improved storage features compared with Apache Kafka, it falls behind in streaming (Despot, 2021) as it is less developed for that purpose. Therefore, Apache Pulsar is a less suitable candidate for this project. This framework is still worth considering if problems should arise with Apache Kafka.

RabbitMQ

"RabbitMQ is the most widely deployed open source message broker."

- https://www.rabbitmq.com/

Overview

RabbitMQ is a message software that puts messages in queues for consumption. The framework consists of a RabbitMQ Broker that accepts messages from producers, adds the messages to a queue, and passes them along to subscribed consumers when they are ready to consume.

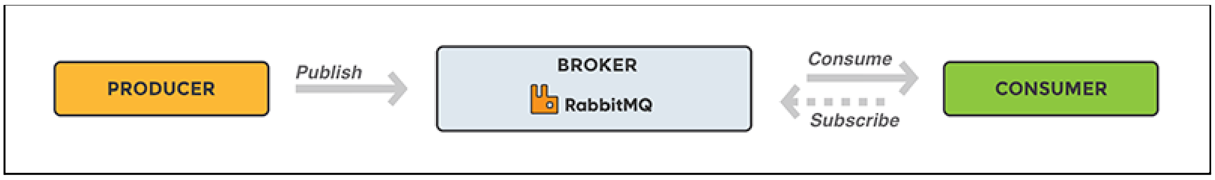


Figure 6: RabbitMQ's Broker system, https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.html

The RabbitMQ Broker contains queues and an exchanger. The exchanger is responsible for sorting messages based on attributes and passing these along to the correct queue.

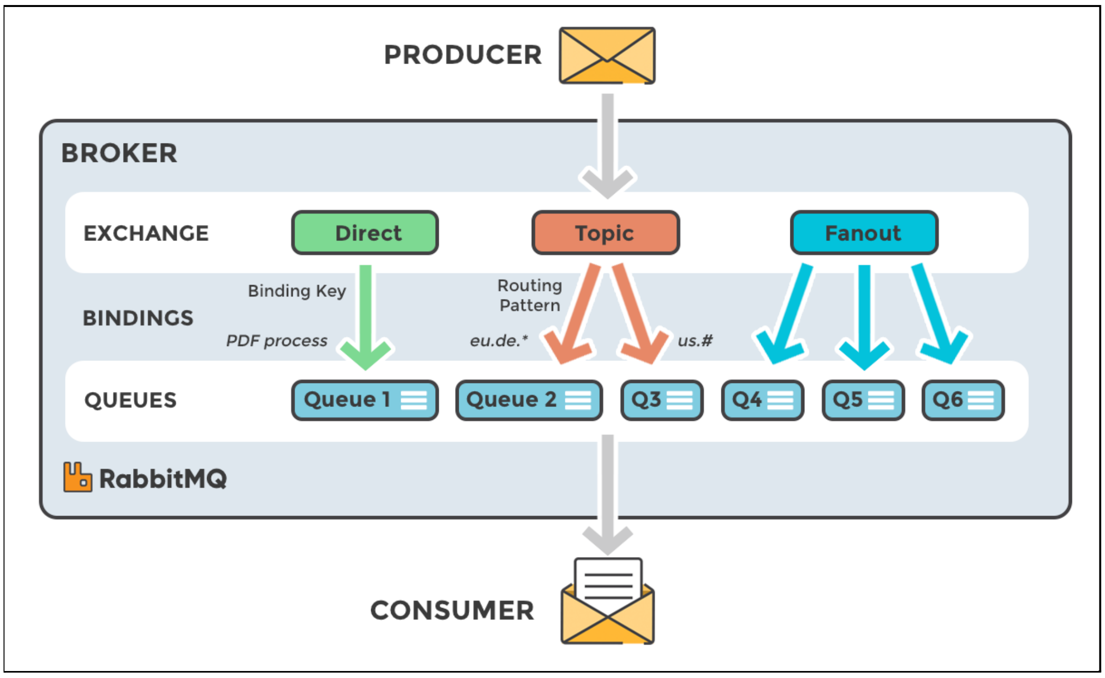


Figure 7: RabbitMQ's exchanger, https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.html

Because RabbitMQ mainly focuses on message queueing compared to streaming, further research into APIs and security has been dropped.

Conclusion

RabbitMQ is a framework that mainly focuses on consuming messages that need to be delayed slightly before being processed. As the project requires persisting the produced data, RabbitMQ is not a candidate.

Apache Spark (Spark Streaming)

"Apache Spark™ is a multi-language engine for executing data engineering, data science, and machine learning on single-node machines or clusters."

- *https://spark.apache.org/*

Overview

Somewhat already in use at AutoStore in the form of databricks. Spark is used for treatment of data, but can also be used to pseudo-stream. The term pseudo-stream is used here, because there is no actual streaming, only really fast batch processing (Levy, 2021).
Spark Streaming works by receiving data from a stream or source of continuous data, which it then processes into batches. These batches are then passed further down the stream to the Spark processing engine, before it is passed on to a server/data lake/recipient. These batches are small, but still large enough to cause delays of milliseconds which should be insignificant for the use case. Spark streaming can be configured to mitigate the batch-streaming, but this is still somewhat experimental (Pointer, 2018).

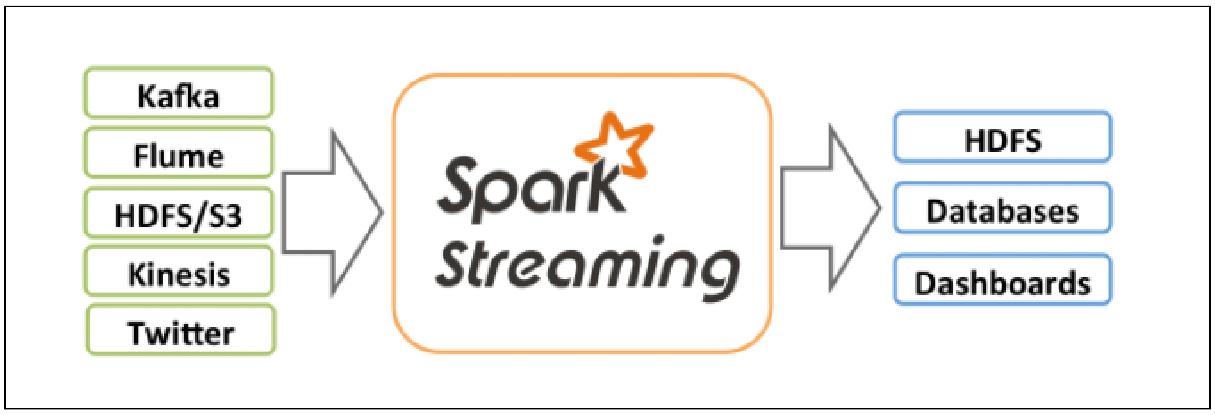


Figure 8: High Level overview of Spark Streaming architecture, https://spark.apache.org/docs/latest/streaming-programming-guide.html

The obvious benefit or detriment depending on the use-case, is that Spark Streaming is inherently tied to the spark engine. The fact that the spark engine is intended to process the data introduces an unwanted and unneeded step for the specific use-case of AutoStore, where processing is not needed.

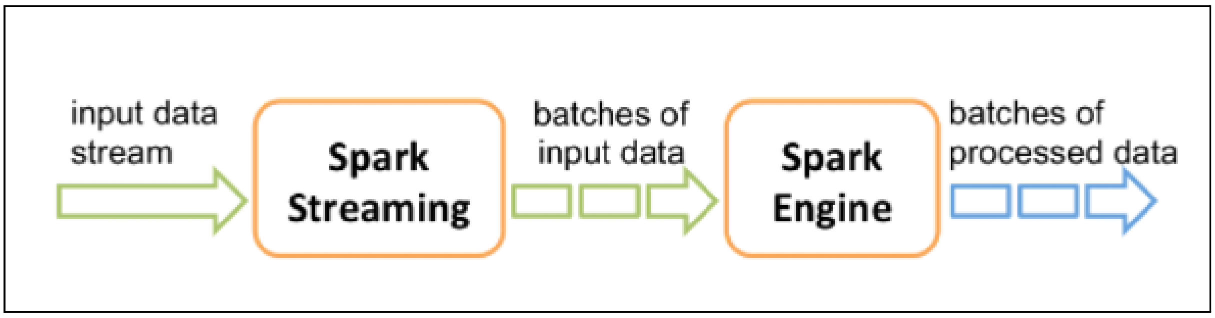


Figure 9: Data stream using Spark Streaming and the Spark Engine, https://spark.apache.org/docs/latest/streaming-programming-guide.html

Security

Spark streaming offers authentication for RPC channels using a shared secret. This does require configuration, but the configuration is well documented, with options for:

- cryptographic key length
- algorithm for encryption
- option to enable SASL-based encrypted communication

There is one known vulnerability in spark related to the shared secret on standalone resource manages. In Spark versions 2.4.5 and older, it is possible to send a specific malicious RPC message to start resources within a Spark cluster. This vulnerability does not apply for other resource managers (I.E Yarn).

API

Spark supports the following languages:

- Scala
- Java
- Python
- R

Spark is widely used, and has a large amount of documentation, simplifying implementation.

Conclusion

Due to unnecessary processing included with the Spark streaming, this framework is unfit for the project, although interesting if the need for processing the streamed data should arise. It is not completely irrelevant as the use of standard Spark functionality can be minimized, but it might prove more efficient to utilize simpler frameworks.

Apache Storm

"Apache Storm is a free and open source distributed realtime computation system. Apache Storm makes it easy to reliably process unbounded streams of data, doing for real time processing what Hadoop did for batch processing. Apache Storm is simple, can be used with any programming language, and is a lot of fun to use!"

- https://storm.apache.org/

Overview

Apache Storm is yet another open source Apache framework. Unlike Spark Streaming, Apache Storm is not tied to or part of a larger product (like Spark).

Apache Storm uses terminology of "Spouts"and "Bolts" to describe the nodes in the graph representing the data stream. Spouts are the source tasks of the stream, and bolts are the processing tasks. The Spout connects to the data source, and communicates with the bolts, sending the data to them. Bolts receive an input stream, processes the contents, and passes it on to other Bolts.

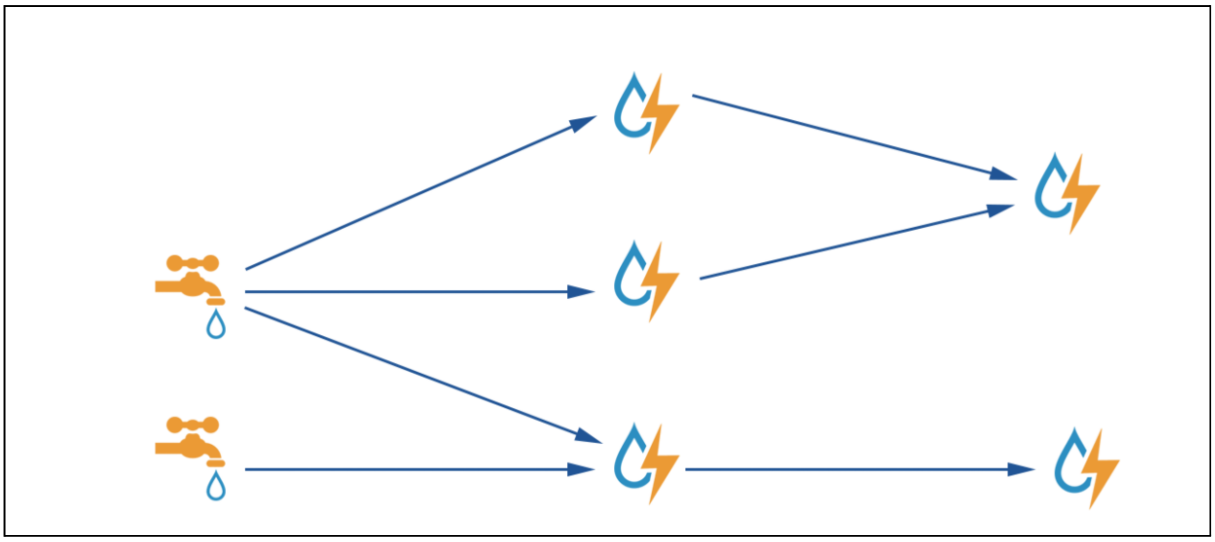


Figure 10: Spouts and Bolts of Apache Storm. Bolts are both intermediaries and end-points, https://storm.apache.org/

Apache Storm is well established and has been in development for many years. However, Storm is primarily a processing service, not a message-broker. The message-broker in Storm is the "Spout" that provides the data. A possible use of Storm could then be as part of a stack, with a message-broker sending the data to Storm, before Storm sends it to storage (*Apache Kafka vs Apache Storm*, 2014). Storm is not a queue the same way that message-brokers are. This means that to ensure reliability, Storm should be used in conjunction with other technologies.

Apache Storm is open source and free to implement. Storm also touts high scalability, and high speed (*Apache Storm*, n.d.), especially when compared to, for example, Spark Streaming, due to micro-batches being used in Storm. Storm is also very well documented, simplifying implementation.

API

Storm is language dependant and supports the following languages:

- JavaScript
- Python
- Ruby

Additionally, there is third party support for c#.

Security

By default, Storm installs without authentication but this can be configured. It is possible to configure for SSL within Storm, and technologies can (and should) be used for authentication and authorization. The built-in security measures of Storm appear somewhat lackluster, so integrating other technologies will be necessary.

Storm has a history of vulnerabilities, but there are no known vulnerabilities in the current version (2.2.1).

Conclusion

Apache Storm is a good candidate as a part of a larger stack. It is possible to use it on its own, but to ensure reliability, a message-broker with queue functionality should be used in conjunction.

Apache Flink

"Apache Flink is a framework and distributed processing engine for stateful computations over unbounded and bounded data streams. Flink has been designed to run in all common cluster environments, perform computations at in-memory speed and at any scale."

- *https://flink.apache.org/flink-architecture.html*

Overview

Yet another Apache product, Flink is a framework that shows great promise. Since its initial release in 2011 Flink has been developed to support large scale streaming of both bounded and unbounded data. Flink has no batches, and its event processing is pure streaming, constantly pushing data out to nodes for processing and handling as events arrive. Flink is also fast compared to other frameworks.

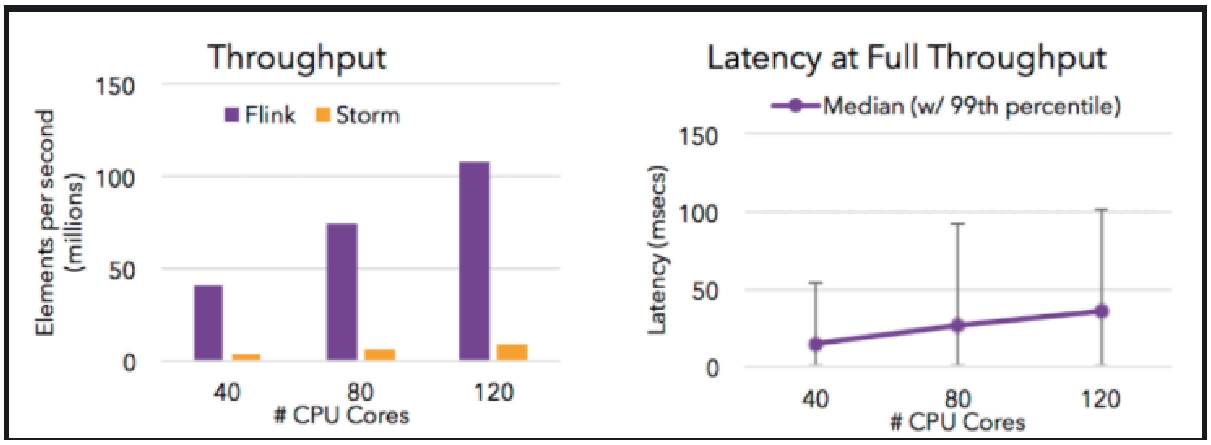


Figure 11: comparison of throughput between Apache Storm and Apache Flink, https://medium.com/@chandanbaranwal/spark-streaming-vs-flink-vs-storm-vs-kafka-streams-vs-samza-choose-your-stream-processing-91ea3f04675b

Flink is a cluster framework and the framework deploys the application either in standalone Flink clusers, or using containers (Flink vs Kafka Streams - Comparing Features, 2016). Typically, Flink is used in conjunction with Kafka, where Kafka serves as the source connecting to Flink, although this is not necessary.

Security

Flink allows for authentication through the use of Kerberos authentication. Other forms of authentication can be used, but Kerberos is the only one actively supported by Flink.

According to CVE Details, Flink is one of the most secure services, as it has few vulnerabilities, and the ones that have been discovered are not considered severe. Furthermore, no vulnerabilities are present in the current version (1.14.3)

API

Flink supports the following languages:

- Java
- Scala
- Python
- SQL

Java is best supported, but the Python implementation (referred to as PyFlink) has also been receiving great support.

Flink offers three layered APIs that each target slightly different use cases. The APIs focus on two areas; Conciseness and Expressiveness, and an increase in one leads to a reduction in the other. Azure connection is also supported, although Hadoop is required.

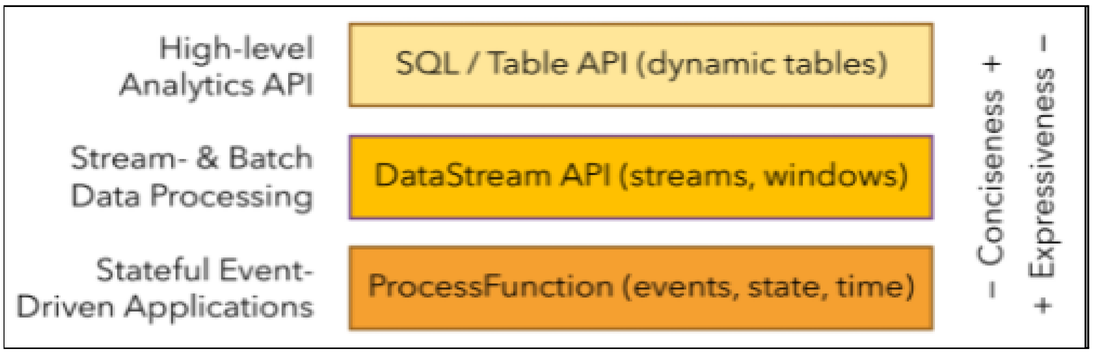

High-level Analytics API
SQL / Table API (dynamic tables)
Stream- & Batch Data Processing
DataStream API (streams, windows)
Stateful Event-Driven Applications
ProcessFunction (events, state, time)
– Conciseness +
+ Expressiveness –

Figure 12: The three API layers, https://flink.apache.org/flink-applications.html

Finally, flink offers connectors for multiple sources like Kafka, but also RabbitMQ, Amazon Kinesis, and Apache Pulsar.

Conclusion

Apache Flink appears to be a good candidate offering real-time streaming, good security, and a robust API, as well as good performance.

Selection for Prototyping

Based on the properties of all frameworks, the following frameworks are suggested for the prototyping stage of the project based on factors such as documentation, availability, performance, and the use-case of the assignment.

Prototype 1, Kafka (barebone prototype)

Prototype utilizing only Kafka, with minimal involvement of other technologies. A pure Kafka prototype will not only form a solid foundation for Kafka in conjunction with other services but can also easily be supported through documentation and online resources due to widespread implementation.

Kafka also has a streaming API with powerful features for handling the data in addition to the message-brokering. This API is a built-in part of Kafka, and this prototype will leverage this library to implement a full stream.

For storage, ksqlDB is the most promising technology. ksqlDB stands for Kafka SQL DataBase, and is built with streaming (in particular with Kafka) in mind. This prototype will utilize ksqlDB to store/receive the stream.

Prototype 2, Pulsar (barebone prototype)

Prototype utilizing Puslar, with minimal involvement of other technologies. Like a pure Kafka prototype, this can lay a foundation for Pulsar in conjunction with other services. Although popular, Pulsar is not as widespread as Kafka, but documentation exists.

For storage, there are multiple options in addition to the ksqlDB mentioned in the section discussing Prototype 1. This prototype will also be used to test multiple different databases in order to discover which database offers the best performance. Database performance will be measured and reported, and the choice of database for the third prototype and final implementation will be based on this analysis.

Prototype 3, Kafka or Pulsar + Flink (stack prototype)

Based on the previous prototypes, this prototype will utilize either Kafka or Pulsar for messaging with Flink for data handling and processing. The processing will be limited to receiving the data and fitting it into a database. Although fitting the data into a database is not a large job, it is still not the main focus of either Kafka or Pulsar. Therefore, it would be interesting to have this prototype in addition to the barebones previous prototypes which utilizes an additional framework concerning data-handling.

Flink has been chosen because it is independant, has few known security risks, is being actively developed, and promises support for both bounded and unbounded data. Flink also supports connectivity with both Kafka and Pulsar. If flink should prove impossible or overly inconvenient to implement, it can be replaced by Storm. Storm is however considerably slower than Flink. Spark will likely not be used, as functionality aside from streaming mostly focuses on analysis of data which is irrelevant for the use-case.

Storage for this prototype will be chosen based on results from the previous prototypes as well. In theory, this prototype should connect to the most efficient database based on previous findings.

References

Levy, E. (2021, December 30). 7 Popular Stream Processing Frameworks Compared. Upsolver. Retrieved February 6, 2022, from https://www.upsolver.com/blog/popular-stream-processing-frameworks-compared

Pointer, I. (2018, March 15). What's new in Apache Spark? Low-latency streaming and Kubernetes. InfoWorld. Retrieved February 6, 2022, from https://www.infoworld.com/article/3262995/whats-new-in-apache-spark-low-latency-streaming-and-kubernetes.html

Apache Storm. (n.d.). Apache Storm. Retrieved February 6, 2022, from https://storm.apache.org/index.html

Despot, I. (2021, November 30). Apache Pulsar vs Apache Kafka - How to choose a data streaming platform. Memgraph. Retrieved February 6, 2022, from https://memgraph.com/blog/pulsar-vs-kafka

Kafka vs Pulsar - Performance, Features, and Architecture Compared. (n.d.). Confluent. Retrieved February 6, 2022, from https://www.confluent.io/kafka-vs-pulsar/

Flink vs Kafka Streams - Comparing Features. (2016, September 2). Confluent. Retrieved February 6, 2022, from https://www.confluent.io/blog/apache-flink-apache-kafka-streams-comparison-guideline-users/

Bibliography

Overview - Spark 2.2.0 Documentation. (n.d.). Apache Spark. Retrieved February 6, 2022, from https://spark.apache.org/docs/2.2.0/index.html

Security - Spark 3.2.1 Documentation. (n.d.). Apache Spark. Retrieved February 6, 2022, from https://spark.apache.org/docs/latest/security.html

Apache Storm. (n.d.). Apache Storm. Retrieved February 6, 2022, from https://storm.apache.org/index.html

Apache Storm : List of security vulnerabilities. (n.d.). CVE Details Retrieved February 6, 2022, from https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-35571/Apache-Storm.html

Apache Flink: Stateful Computations over Data Streams. (n.d.). Apache Flink. Retrieved February 6, 2022, from https://flink.apache.org/

Apache Flink Documentation. (n.d.). Apache Flink. Retrieved February 6, 2022, from https://nightlies.apache.org/flink/flink-docs-release-1.14/

Baeldung. (2021, August 17). Building a Data Pipeline with Flink and Kafka. Retrieved February 6, 2022, from https://www.baeldung.com/kafka-flink-data-pipeline

Apache Flink : List of security vulnerabilities. (n.d.). CVE Details. Retrieved February 6, 2022, from https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-88972/Apache-Flink.html

Apache kafka. Apache Kafka. (n.d.). Retrieved February 6, 2022, from https://kafka.apache.org/

Kafka Introduction. (n.d.). Cloudera. Retrieved February 6, 2022, from https://docs.cloudera.com/runtime/7.2.10/kafka-overview/topics/kafka-overview-intro.html

Apache Kafka : List of security vulnerabilities. (n.d.). CVE Details. Retrieved February 6, 2022, from https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-48980/Apache-Kafka.html

Documentation Security. Apache Kafka. (n.d.). Retrieved February 6, 2022, from https://kafka.apache.org/documentation/#security

Apache Pulsar. (n.d.). Pulsar. Retrieved February 6, 2022, from https://pulsar.apache.org/

Apache Pulsar : List of security vulnerabilities. (n.d.). CVE Details. Retrieved February 6, 2022, from https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-95667/Apache-Pulsar.html

Messaging that just works — RabbitMQ. (n.d.). RabbitMQ. Retrieved February 6, 2022, from https://www.rabbitmq.com/

JOHANSSON, L. (2019, September 23). Part 1: RabbitMQ for beginners - What is RabbitMQ? - CloudAMQP. CloudAMQP. Retrieved February 6, 2022, from https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.html

Kafka vs Pulsar - Performance, Features, and Architecture Compared. (n.d.-b). Confluent. Retrieved February 6, 2022, from https://www.confluent.io/kafka-vs-pulsar/

Despot, I. (2021b, November 30). Apache Pulsar vs Apache Kafka - How to choose a data streaming platform. Memgraph. Retrieved February 6, 2022, from https://memgraph.com/blog/pulsar-vs-kafka

Flink vs Kafka Streams - Comparing Features. (2016, September 2). Confluent. Retrieved February 6, 2022, from https://www.confluent.io/blog/apache-flink-apache-kafka-streams-comparison-guideline-users/

Appendix 2: Prototype Testing documentation

Test Requirements and Results for Prototype Testing, Service for Data Streaming

IDATA2900, Bachelor's in Computer Science, 2022

Tomas Klungerbo Olsen

Lars Ivar Ramberg

Contents

Introduction

This document presents the formal test criteria for, and results from, the testing of prototypes done during the bachelor's in computer science project, "Service for Data Streaming." The document describes in detail the intention, methodology, and environment for the tests, the tests performed, the requirements for the tests, and the results from the tests.

The document introduces the test subjects, test environment, and resources available for the tests. Afterward, test coverage is discussed concerning essential requirements identified for the desired solution. Finally, each test is described in detail, including its purpose, methodology, and results for each test. At the end of the document, the test results are compared considering the outlined criteria to select the best framework for further development.

Test Subjects

Subjects for the tests in this document are the two prototypes developed for the IDATA2900 project "Service for Data Streaming." Two prototypes were produced, one utilizing Apache Kafka and another using Apache Pulsar. Although the prototypes are similar in implementation, the two frameworks differ in how they manage the streaming of data. The purpose of the testing outlined in this document is to discover how these differences manifest in terms of performance and stability, subsequently informing the decision of which framework is best in relation to the requirements for the solution.

Test methodology and framework

Terminology

The two systems tested implement Apache Kafka and Apache Pulsar respectively. For both systems, the implementation is split into the following parts and responsibilities:

1. Data provider: creates data to pass through the system. The data providers simulate the generation of data from the AutoStore system in an actual implementation
2. Producer: forwards data from the data provider to the rest of the system. The producer implements Kafka and Pulsar-related code.
3. Broker: the communication link between producer and consumer. Passes the data from the producer to the consumer with the option to persist data
4. Consumer: receives the data. In the current testing framework, the consumer merely displays received data in the console.

System Overview for Testing

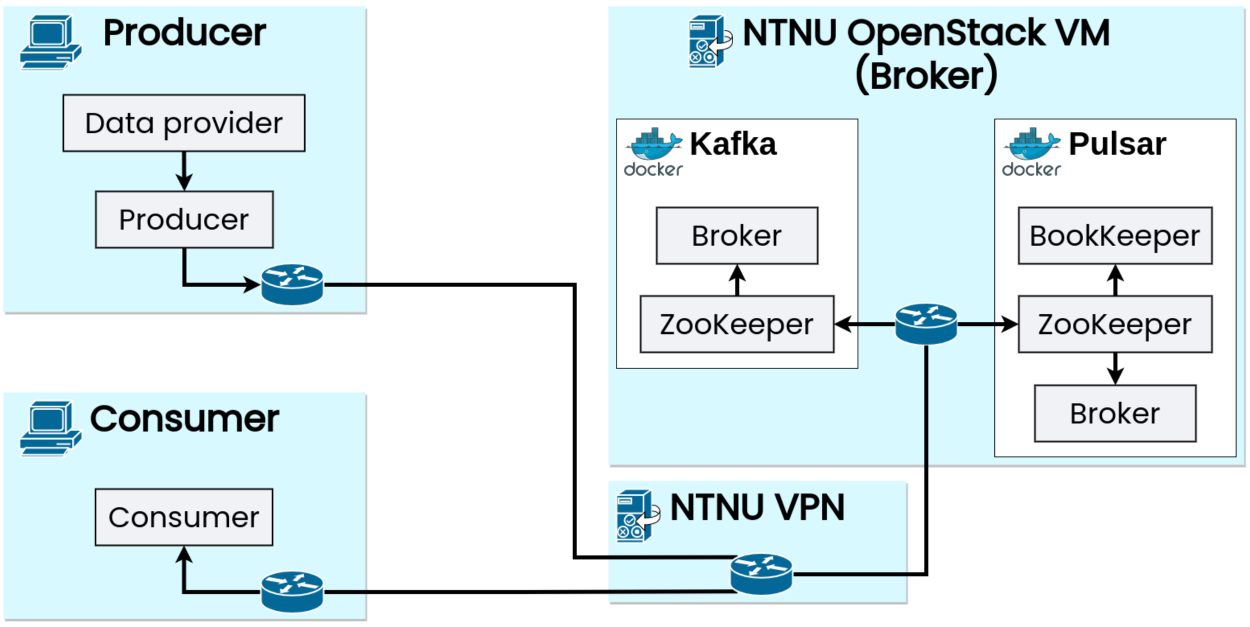


Figure 1 Illustration of testing architecture. Developer one hosts Data Providers and a Producer on their computer. Developer two hosts a consumer on their computer. Both developer and consumer connect to an OpenStack virtual machine through an NTNU VPN. The OpenStack server contains the brokers for both Kafka and Pulsar.

The testing was performed using already acquired and developed technologies. All tests were performed with connections across the internet, with the brokers being on a remote server to simulate an actual use case where the provider, consumer, and broker will be distributed geographically.

The server running the brokers was an NTNU OpenStack virtual machine. To access this machine, it is required to be connected to the NTNU network. This means that the producers and consumers of the solutions also needed to be on the NTNU network when testing the broker. This was achieved by using a VPN connection from the computers belonging to the group to the NTNU network. The architecture described here is illustrated in Figure 1. VPN usage should be considered when viewing the results of the tests, as a VPN introduces overhead that otherwise would not be present without a VPN connection.

Tests were divided into two categories, performance and stability. The performance test measures how effectively the system can perform its task. The stability tests measure the ability to perform said tasks even when faced with unexpected or undesired events like disconnects and program failures.

Test Coverage

The solutions being tested need to meet specific requirements. Briefly summarized, the requirements are:

1. Performance, data needs to be streamed fast
2. Reliability, data needs to safely travel from source to consumer without loss
3. Fault Tolerance, temporary system failure should not lead to loss of data
4. Cloud compatibility, the solution needs to be accessible from the cloud
5. Security, data should be safe from theft and manipulation during transfer
6. Scalability, AutoStore is a company in growth. If the solution does not scale, it will not be a permanent solution

The requirements above are not numbered by importance, urgency, or other metrics. The numbering is present to ease referencing when discussing whether criteria are covered by the defined tests.

Except for 4 and 5, all points are testable by the current prototypes. Points 1, 2, and 3 are directly tested, whereas point 6 is indirectly tested. The thoroughly tested points can easily be tested with a small number of computers and connections and are therefore the ones tested most thoroughly. Testing for scalability (point 6) is also possible and is to an extent done by the performance test, but not to the extent where hundreds or thousands of connections are used. To do this, the group would need multiple computers and servers to force enormous amounts of data from multiple points through the prototyped systems.

It is assumed that cloud compatibility (point 4) will be tested at a later stage in development. As outlined in the milestone plan agreed upon between AutoStore and the group, there will be two weeks of implementation and testing against Azure. During these two weeks, cloud support will be extensively implemented and tested.

As for security, there are two possibilities. Firstly, security can be actively evaluated at a point later in development when authentication and security have been implemented. Secondly, at least for authentication, it is possible that the group can implement a solution compatible with authentications currently used by AutoStore. To what extent already existing security will and can be implemented must be discussed at a later stage of development.

Performance Test

The following test was specifically designed to test the performance of a system. The test does not assess how the system manages unexpected events and assumes the system is working and stable. The question is not “does the system work,” but rather the focus is on “how well does it work.”

Broker race: sending one million messages.

Description

This test covers performance and reliability. The test measures how fast messages in bulk can go from data provider to producer, through a broker, and arrive at the consumer. The data provider continuously creates the messages and sends them to the producer. The time it takes for the consumer to receive one million messages is measured.

This test also serves as a stress test. No parts of the system should fail during high-stress situations. This test simulates such a high-stress situation, with messages being continuously sent and received at a high volume.

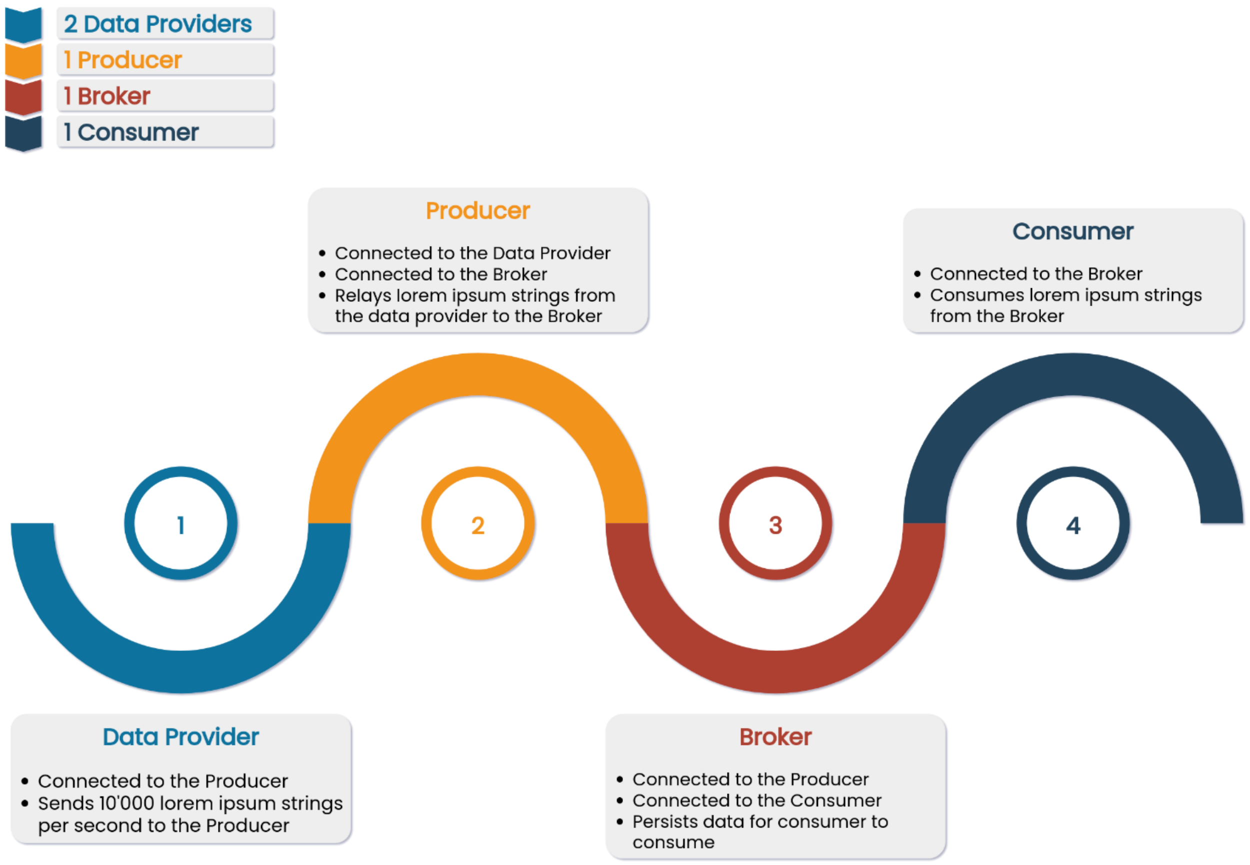

2 Data Providers
1 Producer
1 Broker
1 Consumer
Producer
• Connected to the Data Provider
• Connected to the Broker
• Relays lorem ipsum strings from the data provider to the Broker
Consumer
• Connected to the Broker
• Consumes lorem ipsum strings from the Broker
1
2
3
4
Data Provider
• Connected to the Producer
• Sends 10'000 lorem ipsum strings per second to the Producer
Broker
• Connected to the Producer
• Connected to the Consumer
• Persists data for consumer to consume

Figure 2 High-level systems overview for the one million messages test with twenty thousand messages per second as send rate

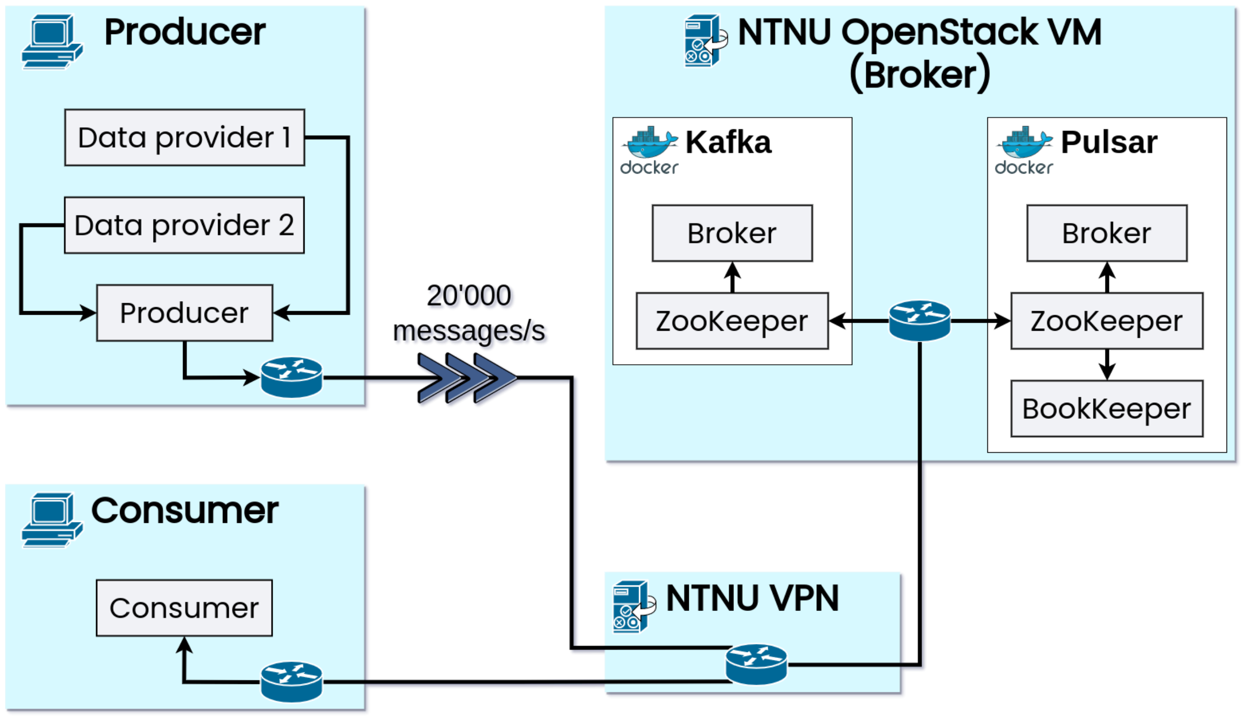


Figure 3 System overview of the one million messages test with twenty thousand messages per second as send rate

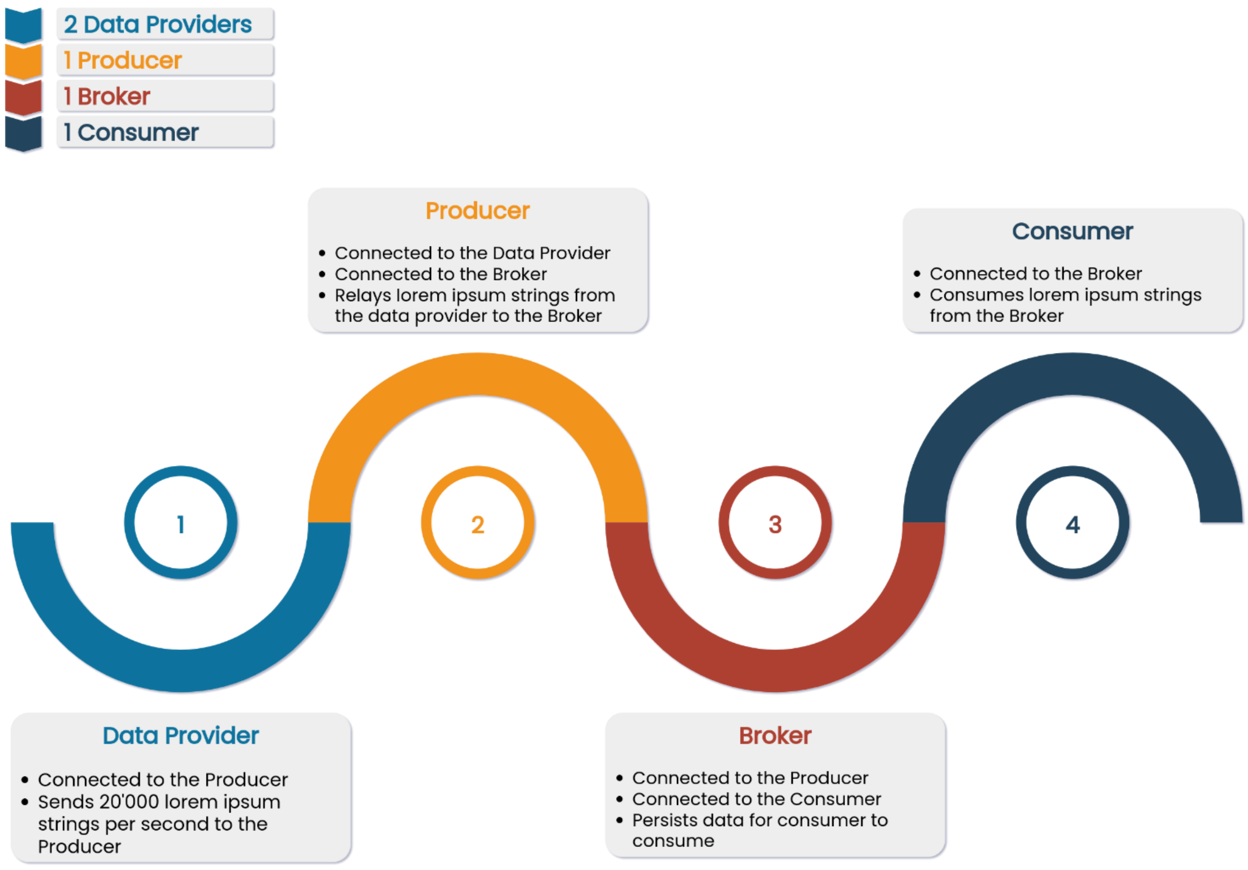

2 Data Providers
1 Producer
1 Broker
1 Consumer
Producer
• Connected to the Data Provider
• Connected to the Broker
• Relays lorem ipsum strings from the data provider to the Broker
Consumer
• Connected to the Broker
• Consumes lorem ipsum strings from the Broker
1
2
3
4
Data Provider
• Connected to the Producer
• Sends 20'000 lorem ipsum strings per second to the Producer
Broker
• Connected to the Producer
• Connected to the Consumer
• Persists data for consumer to consume

Figure 4 High-level systems overview for the one million messages test with forty thousand messages per second as send rate

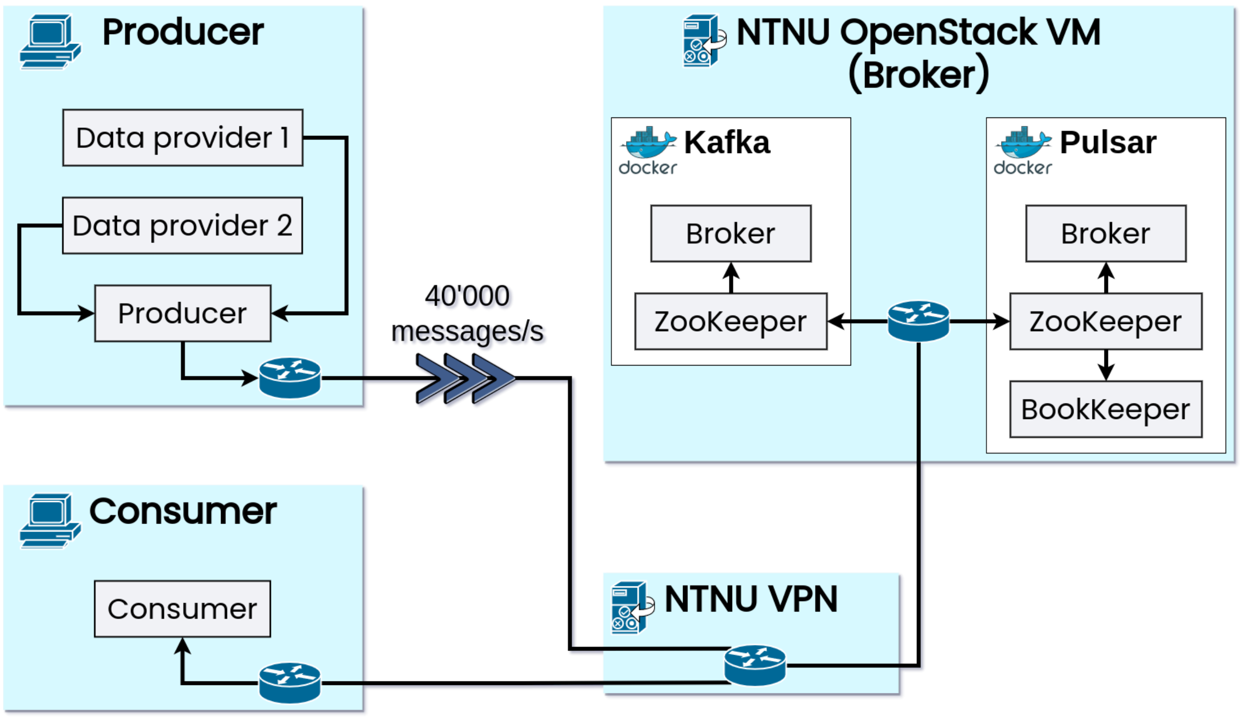


Figure 5 System overview of the one million messages test with forty thousand messages per second as send rate

Methodology

This test was done with two data provider instances, one producer instance, one broker, and one consumer.

The data providers are instructed to produce and send infinite amounts of String messages at set rates to the producer. The producer then forwards these messages as fast as possible to the broker. The broker then passes these messages on to the one consumer that is connected to it.

The production rates tested are twenty thousand and forty thousand messages per second, split equally between the two data providers. For each rate, the time taken to send one million messages is measured five times in five intervals.

The consumer has a timer. The timer records the time of the first received message of an interval, and the time it receives message number one million of its current interval. Both times are saved as Unix timestamps. The times are subtracted to get milliseconds between the two events, then divided by one thousand to get seconds. After a million messages have been received, the time is displayed in seconds in the console, before the timer restarts and times the subsequent million messages, in effect starting its next interval.

To get one numerical result for each test, the average amount of time for the five intervals is calculated. This average is considered the final representative result of how fast one million messages can get from one end of the system to the other.

Success Criteria
The test is successful if all segments are received by the consumer. Additionally, the framework that can do this in the shortest amount of time is considered the most performant in the test.

Additionally, no messages should be lost during the high stress of sending unlimited messages at maximum speed. This also means that neither broker, consumer, nor producer should fail during the test to the point of requiring a manual restart. If any of the components fail and restart themselves, they may still pass the test, but time will be affected by such events.

Results
Both frameworks passed the test. For both Kafka and Pulsar, all messages were sent, and none got lost along the way. Additionally, no parts of the system for either framework failed during the high load.

There is a considerable difference in performance between the two frameworks (see Table 1). In the twenty thousand test, Pulsar was on average 4.73 seconds faster than Pulsar (55.26 seconds vs 59.99 seconds). When the send rate was increased to forty thousand, Pulsar remained faster with 42.37 seconds needed versus 46.37 for Kafka. Although the differences are negligible when considering the enormous amount of data sent in both tests, it is still an indicator that Pulsar will be able to perform better under higher loads in terms of pure throughput.

Pulsar	One million received, 20k send rate	Average messages received per second, 20k send rate	One million received, 40k send rate	Average message per second, 40k send rate
Run one	60.43	16633.4	43.37	23656.6
Run two	53.19	18856.4	42.1	24368.2
Run three	53.38	18828.8	44.31	23156.7
Run four	53.93	18648.2	42.35	23882.7
Run five	55.35	18120.7	39.7	25482.1
Average	**55.26**	**18217.5**	**42.37**	**24109.26**
Kafka	**One million received, 20k send rate**	**Average messages received per second, 20k send rate**	**One million received, 40k send rate**	**Average message per second, Kafka 40k send rate**
Run one	53.01	18903.8	35.03	28683.8
Run two	59.91	16838	48.94	20521
Run three	61.67	16233.1	49.15	20716.1
Run four	62.75	15950	50.63	19850
Run five	62.63	15971.4	48.09	20964.4
Average	**59.99**	**16779.26**	**46.37**	**22147.06**

Table 1 Results of tests sending one million messages through the system

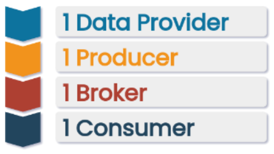
Stability Tests

These tests do not aim to assess how well a system functions, but if it can function at all given specific circumstances. The tests specifically concern fault-handling, whether a system can tolerate parts of it breaking during function. The tests do not measure "how well can a system perform the task," rather focusing on "can the system still function given particular scenarios."

Broker application shutdown

Description

This test covers fault tolerance. The test assesses whether a broker will send all received data even if it shuts down at an inopportune moment.



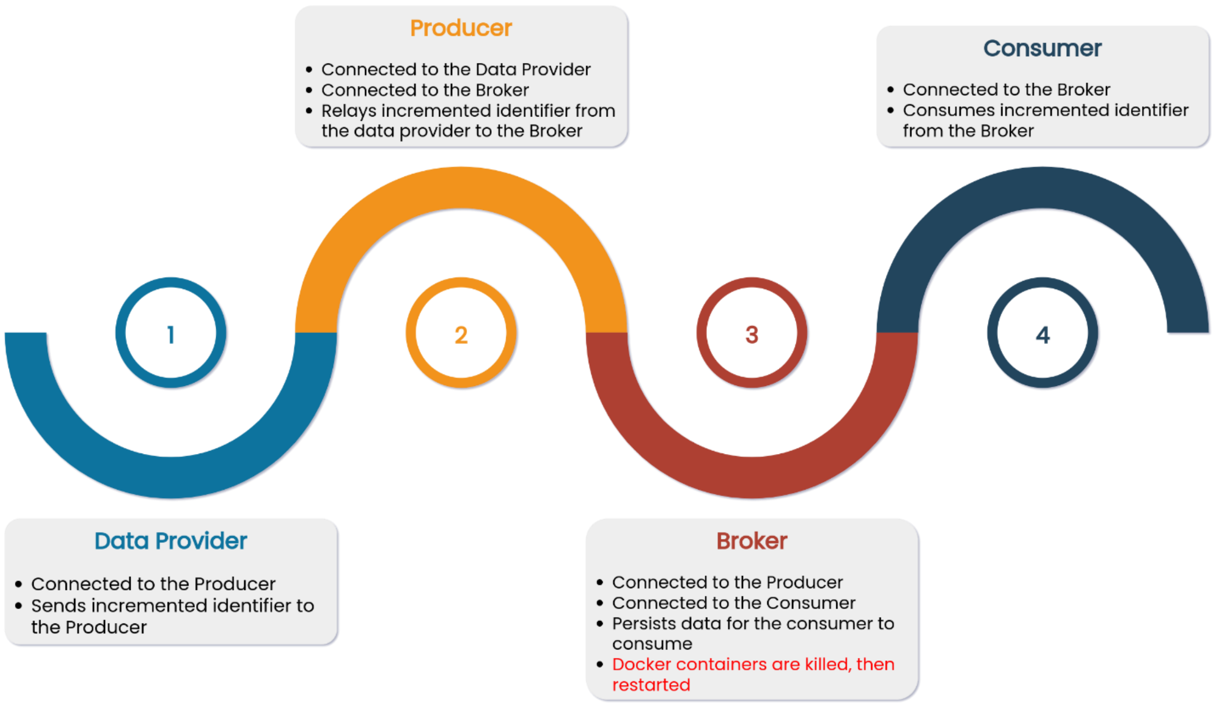

Producer
• Connected to the Data Provider
• Connected to the Broker
• Relays incremented identifier from the data provider to the Broker
Consumer
• Connected to the Broker
• Consumes incremented identifier from the Broker
1
2
3
4
Data Provider
• Connected to the Producer
• Sends incremented identifier to the Producer
Broker
• Connected to the Producer
• Connected to the Consumer
• Persists data for the consumer to consume
• Docker containers are killed, then restarted

Figure 6 High-level systems overview for the broker application shutdown test

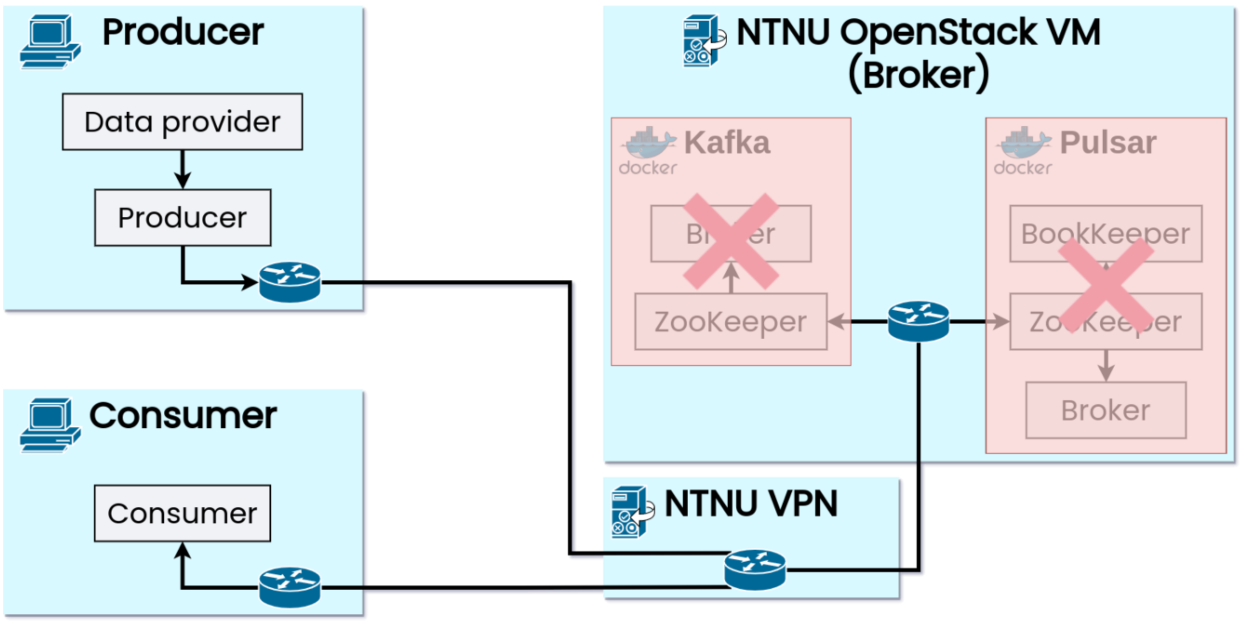


Figure 7 System overview of broker shutdown test. The parts which are shut down are marked with an x and red coloring

Methodology

The test was done using one data provider, one producer, one broker, and one consumer.

After starting the system and ensuring that messages were generated by the producer, sent, and received, the broker was forcefully shut down from the console. To simulate an unexpected and uncontrolled shutdown, the brokers were not gracefully shut down programmatically, but forcefully stopped by killing their processes. Afterward, the broker was restarted after approximately five messages had been sent from the producer.

This test required the broker to persist any unacknowledged messages it received before shutdown, and for the producer to persist messages it could not send while the broker was shut down.

Success Criteria

The test is successful if all generated data reaches the consumer, even after the broker has been restarted.

Additionally, it is a plus if the brokers can restart fast. The time it takes for the consumer to start receiving messages from the brokers after restart should be minimal.

Results

Both frameworks passed this test. In both instances, the brokers could be restarted after a forceful shutdown, and no data was lost. In both instances, only seconds were needed for the brokers to resume sending of data. Neither producer nor consumer encountered irrecoverable problems during this reset, although communication with the broker was impossible while it was shut down.

Broker server shutdown

Description

This test covers fault tolerance. The test assesses whether a broker will send all received data even if its server shuts down at an inopportune moment.

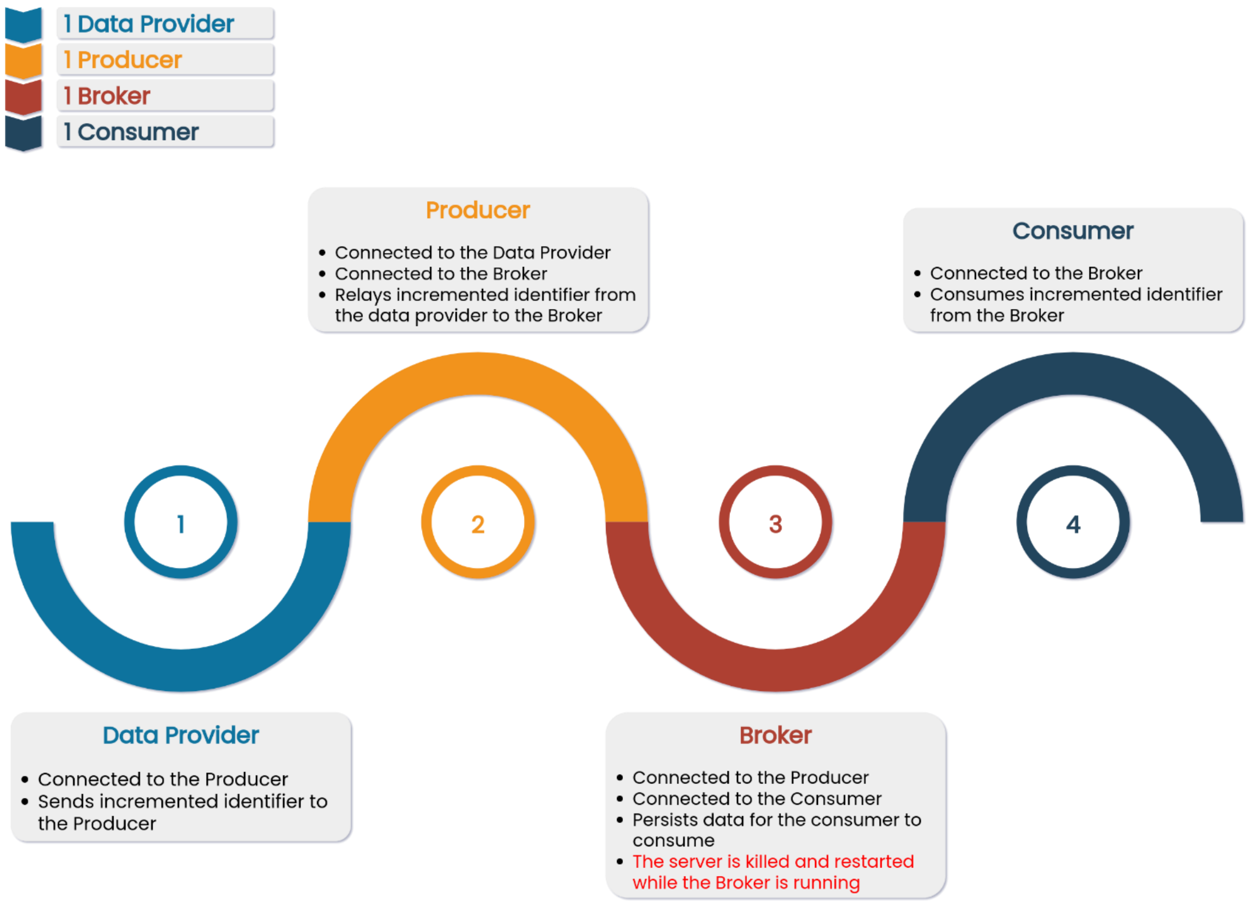

1 Data Provider
1 Producer
1 Broker
1 Consumer
Producer
Connected to the Data Provider
Connected to the Broker
Relays incremented identifier from the data provider to the Broker
Consumer
Connected to the Broker
Consumes incremented identifier from the Broker
1
2
3
4
Data Provider
Connected to the Producer
Sends incremented identifier to the Producer
Broker
Connected to the Producer
Connected to the Consumer
Persists data for the consumer to consume
The server is killed and restarted while the Broker is running

Figure 8 High-level systems overview for the broker server shutdown test

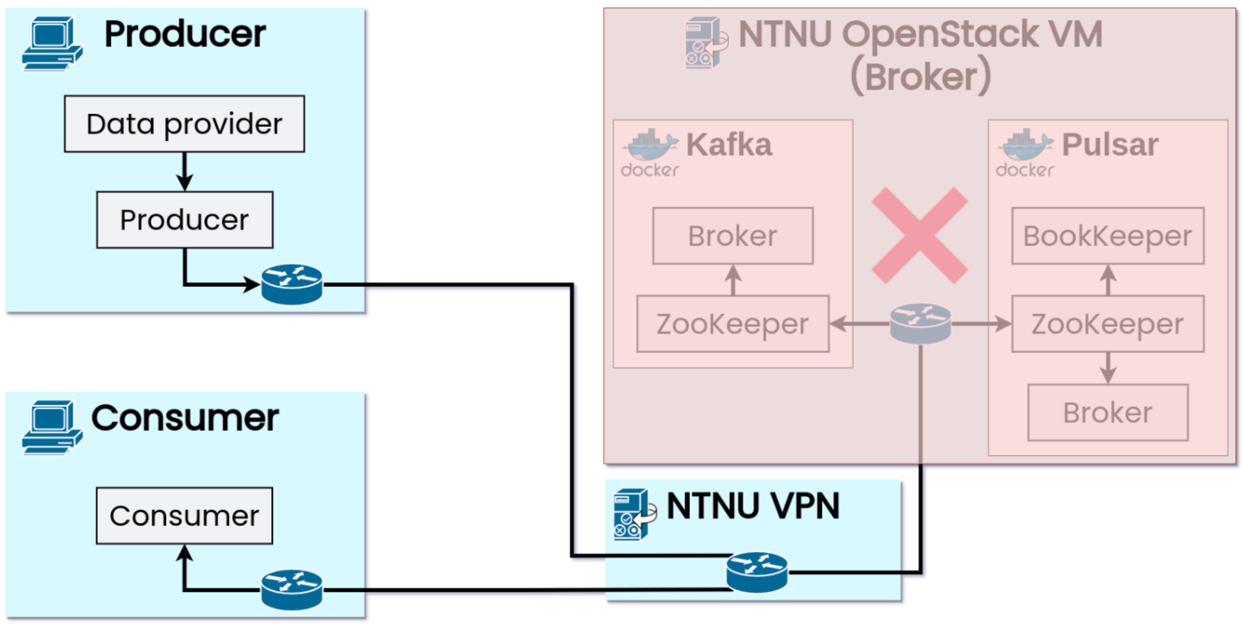


Figure 9 System overview of the broker server shutdown test. The server that was shut down is marked with an X and red coloring

Methodology

This test was done using one data provider, one producer, one broker, and one consumer.

After ensuring connection and data transfer between all parts of the system, the server hosting the broker was rebooted without first shutting down the broker. After restart, the broker was also restarted from the console.

Success Criteria

To pass the test, the broker must resume receiving data from the producer and forward it to the consumer, without loss and corruption. Additionally, it is desirable that the broker restarts sending of data as soon as possible after being rebooted.

Results

Neither framework passed this test, although the extent of failure was slightly different between the frameworks. Kafka completely failed to restart sending data after a server shutdown and threw exceptions upon restart. Pulsar fares a little better, as it sent data after restarting. However, the data sent by Pulsar was incomplete, as data sent from the producer while the server was down never reached the consumer.

Broker network disconnect

Description

This test covers fault tolerance. The test assesses whether a broker will send all received data even if it loses network connection at an inopportune moment.

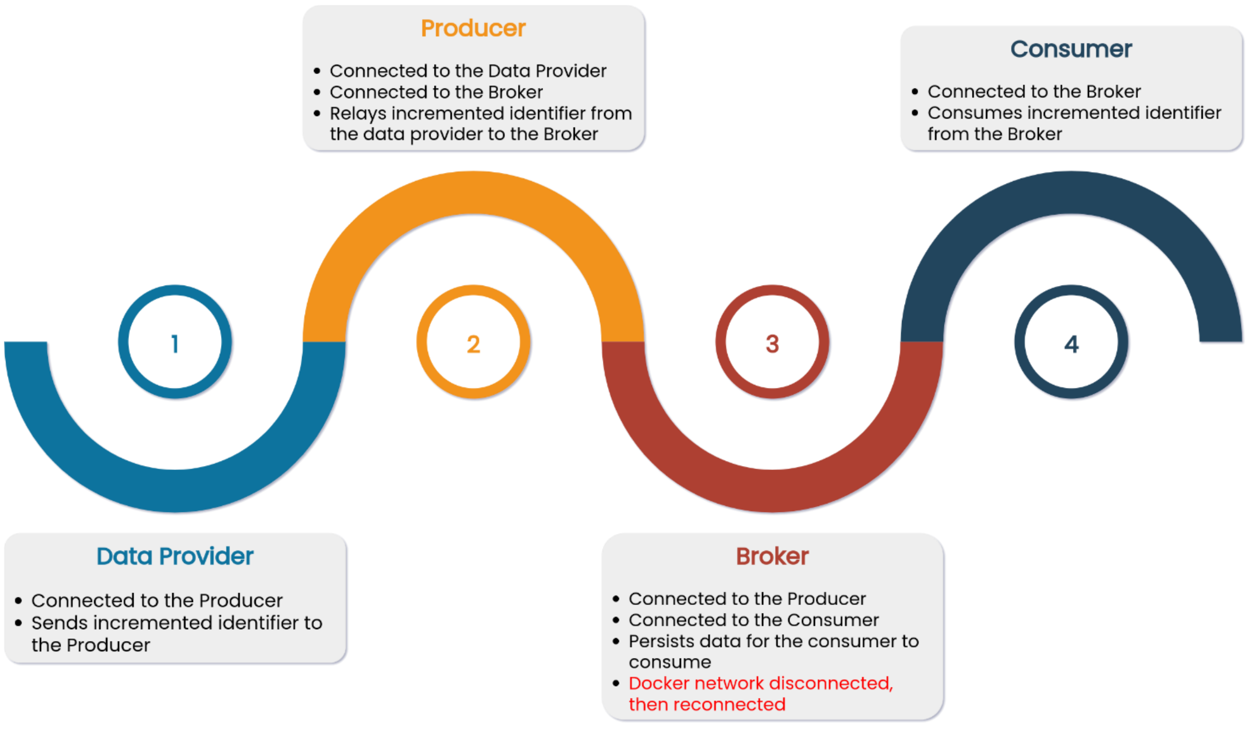

Producer
• Connected to the Data Provider
• Connected to the Broker
• Relays incremented identifier from the data provider to the Broker
Consumer
• Connected to the Broker
• Consumes incremented identifier from the Broker
1
2
3
4
Data Provider
• Connected to the Producer
• Sends incremented identifier to the Producer
Broker
• Connected to the Producer
• Connected to the Consumer
• Persists data for the consumer to consume
• Docker network disconnected, then reconnected

Figure 10 High-level systems overview for the broker network disconnect test

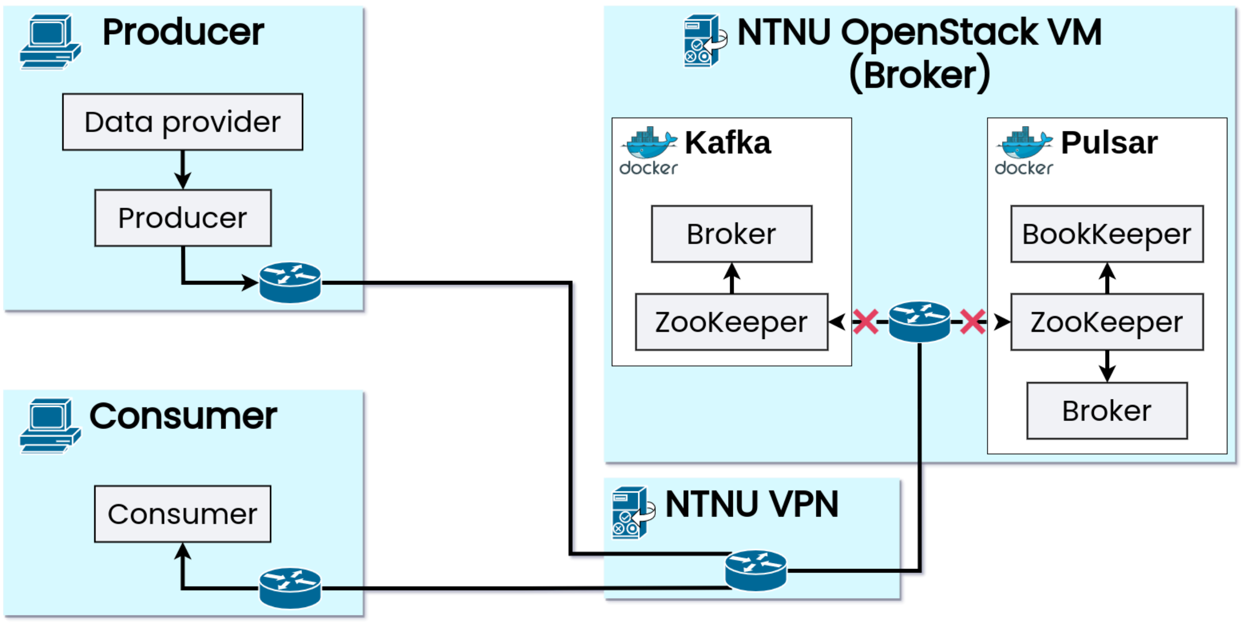


Figure 11 System overview of the broker network disconnect test. The interrupted connections are marked with an x

Methodology

This test was done using one data provider, one producer, one broker, and one consumer.

After starting the system and ensuring that messages were being generated, sent, and received, the broker was disconnected from the internet. Afterward, the broker was reconnected to the internet without restarting the broker itself.

To disconnect the broker from the internet, its docker container was disconnected from its network by issuing a docker disconnect command from the console. To verify that this caused a loss of connection to the rest of the internet, the consumer attempted to reconnect to the broker after the command was issued. After confirming that it was impossible to reconnect, the broker was left disconnected for approximately ten messages, before being reconnected with a docker connect command.

Success Criteria

To pass the test, the broker must restart sending messages when it is reconnected to the network. The messages must arrive at the consumer undamaged and in correct order. Additionally, the time it takes from reconnecting to sending new messages should be minimal.

Results

Both frameworks passed this test. Disconnecting the brokers did not appear to cause any significant problems, and both producers and consumers were able to quickly reconnect to the broker and restart transmission of messages. In terms of time, the frameworks are quite similar, with a small time-gap between reconnecting to the network and transmission of messages.

Unexpected disconnect of consumer

Description

This test covers fault tolerance. The system should be stable, even if a consumer loses connection. This test ensures that even if a consumer either fails or loses connection, the system can remain stable. The consumer should be able to reconnect and continue receiving data.

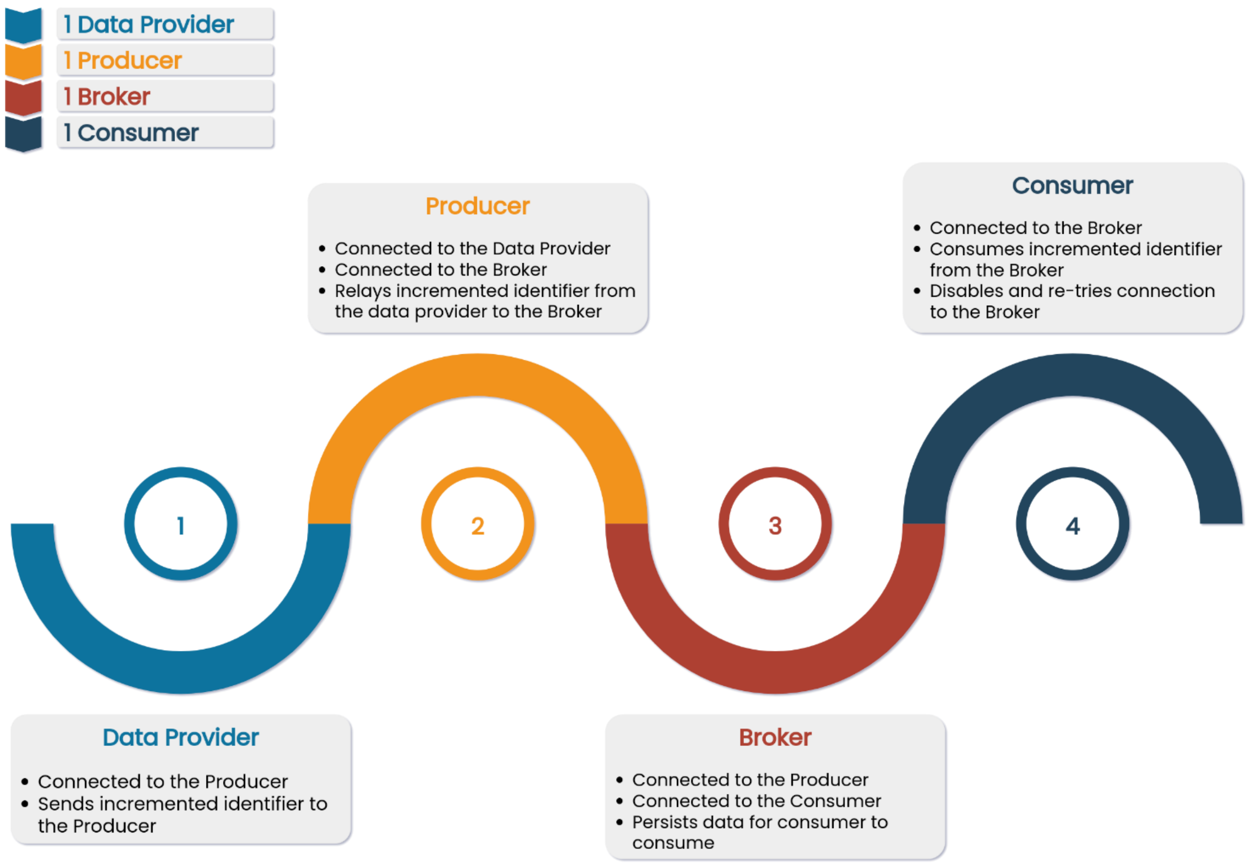

1 Data Provider
1 Producer
1 Broker
1 Consumer
Producer
• Connected to the Data Provider
• Connected to the Broker
• Relays incremented identifier from the data provider to the Broker
Consumer
• Connected to the Broker
• Consumes incremented identifier from the Broker
• Disables and re-tries connection to the Broker
1
2
3
4
Data Provider
• Connected to the Producer
• Sends incremented identifier to the Producer
Broker
• Connected to the Producer
• Connected to the Consumer
• Persists data for consumer to consume

Figure 12 High-level systems overview for the consumer disconnect test

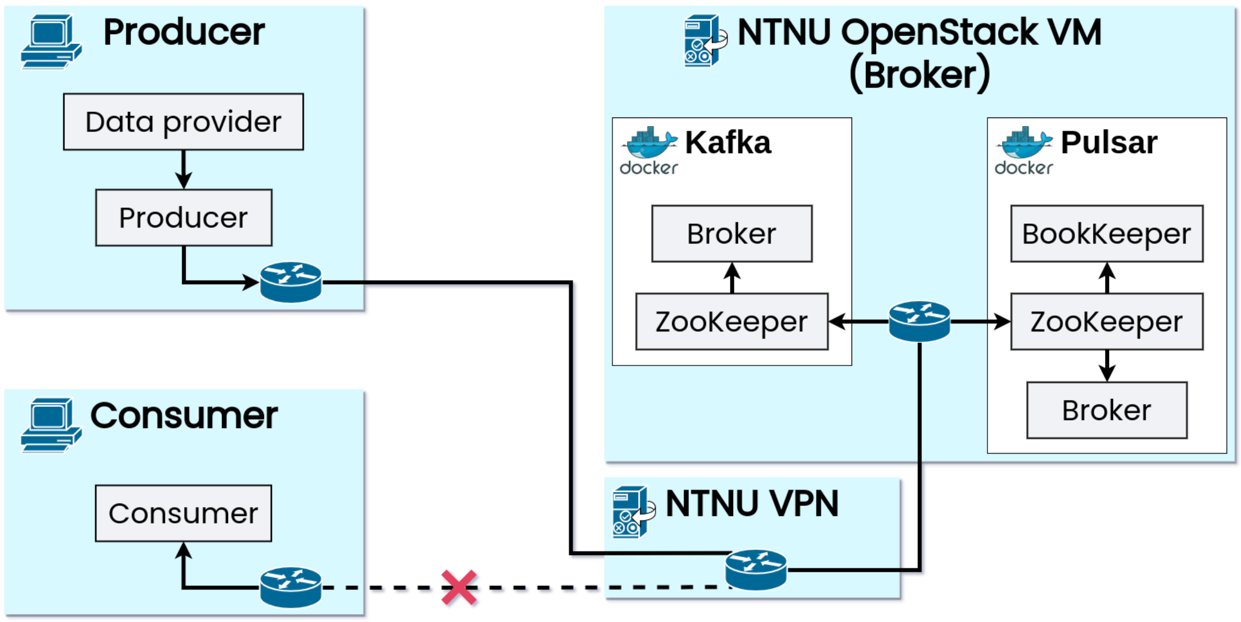


Figure 13 System overview of the consumer disconnect test. The interrupted disconnect is marked with a red x

Methodology

This test was done using one data provider, one producer, one broker, and one consumer.

After connecting all parts of the system and ensuring data transfer, the consumer was disconnected from the network before being reconnected. As the test system was set up over NTNU VPN, the disconnect was simulated by having the computer running the consumer disconnect from the VPN. After generating approximately ten messages, the consumers were reconnected by reconnecting to the VPN.

Success Criteria

The test is passed if the consumer can reconnect and continue receiving data from where it left off. This implies that if the last received segment for the consumer is segment number one hundred, the first segment it receives upon reconnecting should be segment 101, and then all subsequent segments it missed while disconnected. Not only must all messages reach the consumer, but they must also be in correct order.

Results

Both frameworks, given the appropriate configurations, could reconnect and continue to receive messages from where they left off. For Kafka, retries must be issued programmatically, and code must be written to ensure message reception does not go out of order. For Pulsar, it is sufficient to set its subscription type to “Exclusive.” This will guarantee message delivery and correct order. These changes were made to the consumers after they initially failed the test. After the changes, both consumers passed on their first attempts. There was no noticeable difference in time to reconnect between the two solutions, and both solutions appear to start receiving messages instantly when the connection is re-established.

Unexpected disconnect of producer

Description

This test covers fault tolerance. The system should be stable, even if a producer loses connection. This test ensures that even if a producer either fails or loses connection, the system can remain stable. The producer should be able to start resupplying data upon reconnection.

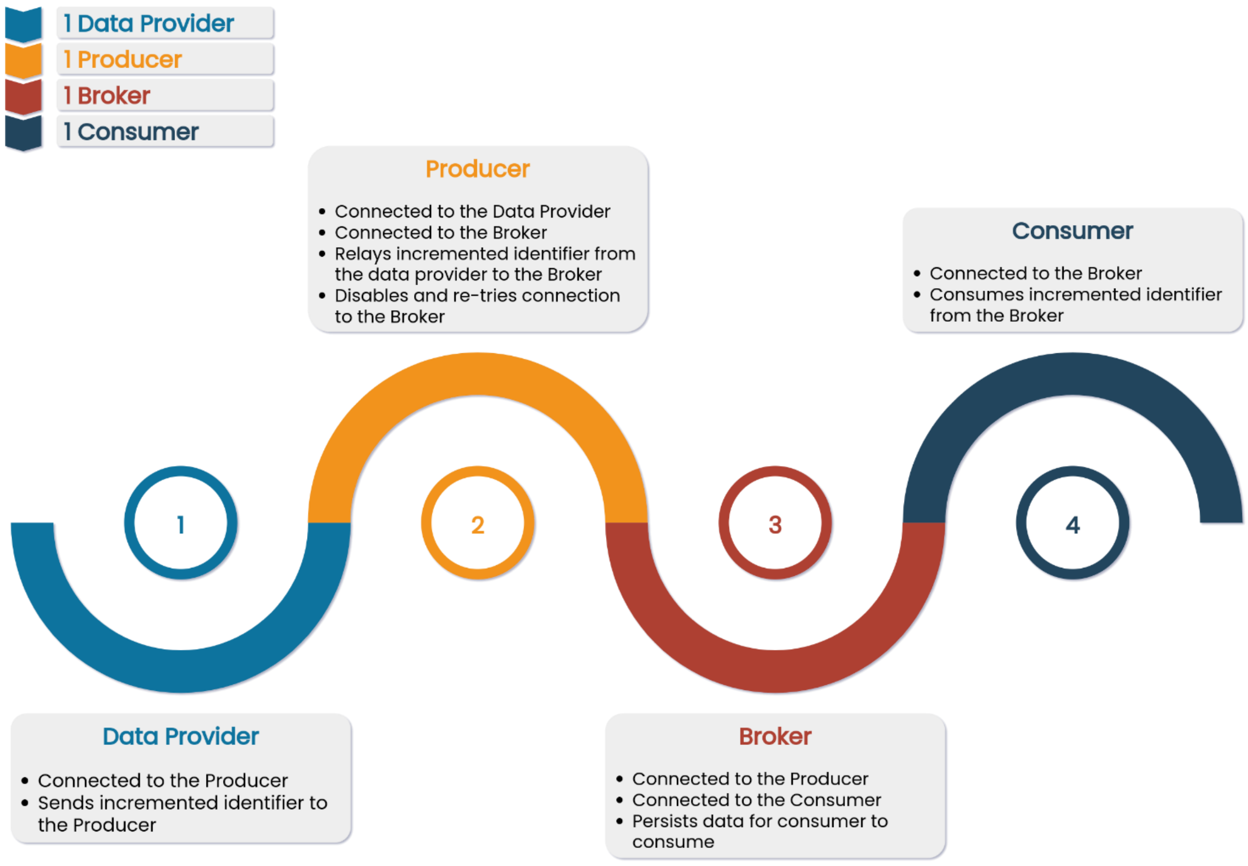

1 Data Provider
1 Producer
1 Broker
1 Consumer
Producer
• Connected to the Data Provider
• Connected to the Broker
• Relays incremented identifier from the data provider to the Broker
• Disables and re-tries connection to the Broker
Consumer
• Connected to the Broker
• Consumes incremented identifier from the Broker
1
2
3
4
Data Provider
• Connected to the Producer
• Sends incremented identifier to the Producer
Broker
• Connected to the Producer
• Connected to the Consumer
• Persists data for consumer to consume

Figure 14 High-level systems overview for the producer disconnect test

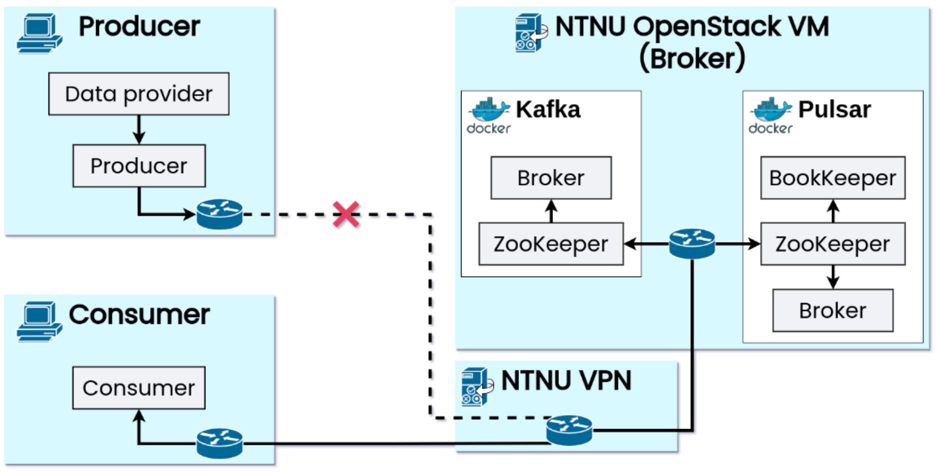


Figure 15 System overview of the producer disconnect test. The interrupted connection is marked with a red x

Methodology

This test was done using one data provider, one producer, one broker, and one consumer.

After connecting all parts of the system and ensuring data transfer, the producer lost connection to the broker, before reconnecting after approximately ten messages had been generated. As the test system was set up over NTNU VPN, disconnection was simulated by simply having the computer running the producer disconnect from the VPN.

Success Criteria

The producer should be able to reconnect and continue to deliver data. The broker should receive the data, and everything should arrive intact to the consumer.

Results

For both frameworks, the producer successfully reconnected and continued passing along its data to the broker, which in turn successfully forwarded it to the consumer. There was no noticeable difference in the time taken to restart sending data, with both frameworks appearing to restart sending close to instantly.

Results

Comparison of results

Both frameworks performed well under the tests outlined in this document. The only significant difference between the two frameworks is in terms of throughput, which is an important metric once the number of messages starts reaching tens of thousands.

Both frameworks are fault-tolerant and able to manage most situations related to restarts and connection failures. The only exception is on server restart where both frameworks fail to appropriately handle the situation, and data is lost as a result.

Both frameworks are also highly performant with send rates surpassing twenty thousand with only one producer, broker, and consumer. However, performance is the only point where there is a noticeable difference between Pulsar and Kafka, as shown in Table 1.

Taking all results into consideration, Pulsar overall outperforms Kafka in the current implementations, although not by a huge margin. It should also be considered that both Kafka and Pulsar were evaluated in an "out-of-the-box" fashion. It is possible that both frameworks can be optimized.

Other considerations when comparing the frameworks

As this test is a part of the decision process of which framework to implement in the final solution, other factors must be considered as well, not just pure test performance.

The most decisive aspect in favor of Pulsar is the lack of persistence in the framework. In the final implementation, the data sent from AutoStore systems will eventually live in an Azure Data Lake. This eliminates the need for persistence in the broker beyond consumption as the data will be stored in the data lake after it has reached the consumer. Kafka by default persists data, while Pulsar does not. There is only a need for data persistence during shortages, not long term, making the default Kafka persistence superfluous.

Conclusion

Based on the results of the tests, Pulsar is the framework best suited for implementation in the final solution. Although Kafka and Pulsar are matched in terms of stability, the additional performance seen is a benefit that needs to be taken into consideration. As AutoStore grows, higher throughput will be necessary and Pulsar provides the highest throughput according to this test. The fact that Pulsar does not persist by default also supports it in the context of this very use case. Pulsar performs well, is stable in most cases, and does not come with unneeded and unnecessary persistence, and is therefore the framework favored for further development

Appendix 3: Data from testing

Pulsar	1 million received, 20k sendrate	Average messages received per second, 20k s	1 million received, 40k sendrate	Average messages received per second, 40k sendrate
Run 1	60.43	16633.4	43.37	23656.6
Run 2	53.19	18856.4	42.1	24368.2
Run 3	53.38	18828.8	44.31	23156.7
Run 4	53.93	18648.2	42.35	23882.7
Run 5	55.35	18120.7	39.7	25482.1
Average	**55.26**	**18217.5**	**42.37**	**24109.26**

Kafka	1 million received, 20k sendrate	Average messages received per second, 20k s	1 million received, 40k sendrate	Average messages received per second, 40k sendrate
Run 1	53.01	18903.8	35.03	28683.8
Run 2	59.91	16838	48.94	20521
Run 3	61.67	16233.1	49.15	20716.1
Run 4	62.75	15950	50.63	19850
Run 5	62.63	15971.4	48.09	20964.4
Average	**59.99**	**16779.26**	**46.37**	**22147.06**

Stability	Pulsar	Kafka
Producer losing connection	🟩	🟩
Consumer losing connection	🟩	🟩
Broker Shutdown from docker	🟩	🟩
Broker Shutdown from server	🟥	🟥
Broker Disconnect server network	🟩	🟩

Disconnect test, Producer, raw data	
ConsumerWorker - Value: 1, Offset: 154	KafkaPrototypeProducer - 1 sent to server
ConsumerWorker - Value: 2, Offset: 155	KafkaPrototypeProducer - 2 sent to server
ConsumerWorker - Value: 3, Offset: 156	KafkaPrototypeProducer - 3 sent to server
ConsumerWorker - Value: 4, Offset: 157	KafkaPrototypeProducer - 4 sent to server
ConsumerWorker - Value: 5, Offset: 158	KafkaPrototypeProducer - 5 sent to server
ConsumerWorker - Value: 6, Offset: 159	KafkaPrototypeProducer - 6 sent to server
ConsumerWorker - Value: 7, Offset: 160	KafkaPrototypeProducer - 7 sent to server
ConsumerWorker - Value: 8, Offset: 161	KafkaPrototypeProducer - 8 sent to server
ConsumerWorker - Value: 9, Offset: 162	KafkaPrototypeProducer - 9 sent to server
ConsumerWorker - Value: 10, Offset: 163	KafkaPrototypeProducer - 10 sent to server
ConsumerWorker - Value: 11, Offset: 164	KafkaPrototypeProducer - 11 sent to server
ConsumerWorker - Value: 12, Offset: 165	KafkaPrototypeProducer - 12 sent to server
ConsumerWorker - Value: 13, Offset: 166	KafkaPrototypeProducer - 13 sent to server
ConsumerWorker - Value: 14, Offset: 167	KafkaPrototypeProducer - 14 sent to server
PRODUCER DISCONNECTS	
.	NetworkClient - Disconnecting from node 1 due to request timeout.
.	NetworkClient - Cancelled in-flight PRODUCE request with correlation id 18 due to node 1 being disconnected
.	NETWORK_EXCEPTION. Error Message: Disconnected from node 1
.	NetworkException: Disconnected from node 1. Going to request metadata update now
ConsumerWorker - Value: 15, Offset: 168	KafkaPrototypeProducer - 15 sent to server
ConsumerWorker - Value: 16, Offset: 169	KafkaPrototypeProducer - 16 sent to server
ConsumerWorker - Value: 17, Offset: 170	KafkaPrototypeProducer - 17 sent to server
ConsumerWorker - Value: 18, Offset: 171	KafkaPrototypeProducer - 18 sent to server
ConsumerWorker - Value: 19, Offset: 172	KafkaPrototypeProducer - 19 sent to server
ConsumerWorker - Value: 20, Offset: 173	KafkaPrototypeProducer - 20 sent to server

Disconnect test, Consumer, raw data	
ConsumerWorker - Value: 1, Offset: 0	KafkaPrototypeProducer - 1 sent to server
ConsumerWorker - Value: 2, Offset: 1	KafkaPrototypeProducer - 2 sent to server
ConsumerWorker - Value: 3, Offset: 2	KafkaPrototypeProducer - 3 sent to server
ConsumerWorker - Value: 4, Offset: 3	KafkaPrototypeProducer - 4 sent to server
ConsumerWorker - Value: 5, Offset: 4	KafkaPrototypeProducer - 5 sent to server
ConsumerWorker - Value: 6, Offset: 5	KafkaPrototypeProducer - 6 sent to server
CONSUMER DISCONNECTS	
NetworkClientNode 2147483646 disconnected.	KafkaPrototypeProducer - 7 sent to server
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4942 due to node 2147483646 bei	KafkaPrototypeProducer - 8 sent to server
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4943 due to node 2147483646 bei	KafkaPrototypeProducer - 9 sent to server
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4944 due to node 2147483646 bei	KafkaPrototypeProducer - 10 sent to server
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4945 due to node 2147483646 bei	KafkaPrototypeProducer - 11 sent to server
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4946 due to node 2147483646 bei	KafkaPrototypeProducer - 12 sent to server
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4947 due to node 2147483646 bei	KafkaPrototypeProducer - 13 sent to server
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4948 due to node 2147483646 bei	KafkaPrototypeProducer - 14 sent to server
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4949 due to node 2147483646 bei	KafkaPrototypeProducer - 15 sent to server
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4950 due to node 2147483646 bei	KafkaPrototypeProducer - 16 sent to server
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4951 due to node 2147483646 bei	KafkaPrototypeProducer - 17 sent to server
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4952 due to node 2147483646 bei	KafkaPrototypeProducer - 18 sent to server
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4953 due to node 2147483646 bei	KafkaPrototypeProducer - 19 sent to server
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4954 due to node 2147483646 bei	KafkaPrototypeProducer - 20 sent to server
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4955 due to node 2147483646 bei	KafkaPrototypeProducer - 21 sent to server
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4956 due to node 2147483646 bei	KafkaPrototypeProducer - 22 sent to server
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4957 due to node 2147483646 bei	KafkaPrototypeProducer - 23 sent to server
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4958 due to node 2147483646 bei	KafkaPrototypeProducer - 24 sent to server
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4959 due to node 2147483646 being disconnected	
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4960 due to node 2147483646 being disconnected	
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4961 due to node 2147483646 being disconnected	
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4962 due to node 2147483646 being disconnected	
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4963 due to node 2147483646 being disconnected	
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4964 due to node 2147483646 being disconnected	
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4965 due to node 2147483646 being disconnected	

NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4966 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4967 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4968 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4969 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4970 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4971 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4972 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4973 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4974 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4975 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4976 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4977 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4978 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4979 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4980 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4981 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4982 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4983 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4984 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4985 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4986 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4987 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4988 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4989 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4990 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4991 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4992 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4993 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4994 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4995 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4996 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4997 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4998 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 4999 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5000 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5001 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5002 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5003 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5004 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5005 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5006 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5007 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5008 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5009 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5010 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5011 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5012 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5013 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5014 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5015 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5016 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5017 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5018 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5019 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5020 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5021 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5022 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5023 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5024 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5025 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5026 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5027 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5028 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5029 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5030 due to node 2147483646 being disconnected					
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5031 due to node 2147483646 being disconnected					

NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5032 due to node 2147483646 being disconnected
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5033 due to node 2147483646 being disconnected
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5034 due to node 2147483646 being disconnected
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5035 due to node 2147483646 being disconnected
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5036 due to node 2147483646 being disconnected
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5037 due to node 2147483646 being disconnected
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5038 due to node 2147483646 being disconnected
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5039 due to node 2147483646 being disconnected
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5040 due to node 2147483646 being disconnected
NetworkClientCancelled in-flight OFFSET_COMMIT request with correlation id 5041 due to node 2147483646 being disconnected
ConsumerCoordinatorGroup coordinator is unavailable or invalid due to cause: null.isDisconnected: true. Rediscovery will be attempted.
NetworkClientNode 1 disconnected.
NetworkClientCancelled in-flight FETCH request with correlation id 4859 due to node 1 being disconnected
NetworkClientCancelled in-flight METADATA request with correlation id 7739 due to node 1 being disconnected
NetworkClientCancelled in-flight FIND_COORDINATOR request with correlation id 7740 due to node 1 being disconnected
NetworkClientCancelled in-flight FIND_COORDINATOR request with correlation id 7741 due to node 1 being disconnected
FetchSessionHandlerError sending fetch request to node 1: org.apache.kafka.common.errors.DisconnectException
ConsumerCoordinatorDiscovered group coordinator 10.212.26.245:9092 (id: 2147483646 rack: null)
ConsumerWorker - Value: 7, Offset: 6
ConsumerWorker - Value: 8, Offset: 7
ConsumerWorker - Value: 9, Offset: 8
ConsumerWorker - Value: 10, Offset: 9
ConsumerWorker - Value: 11, Offset: 10
ConsumerWorker - Value: 12, Offset: 11
ConsumerWorker - Value: 13, Offset: 12
ConsumerWorker - Value: 14, Offset: 13
ConsumerWorker - Value: 15, Offset: 14
ConsumerWorker - Value: 16, Offset: 15
ConsumerWorker - Value: 17, Offset: 16
ConsumerWorker - Value: 18, Offset: 17
ConsumerWorker - Value: 19, Offset: 18
ConsumerWorker - Value: 20, Offset: 19
ConsumerWorker - Value: 21, Offset: 20
ConsumerWorker - Value: 22, Offset: 21
ConsumerWorker - Value: 23, Offset: 22
ConsumerWorker - Value: 24, Offset: 23

Disconnect test broker, docker shutdown

ConsumerWorker - Value: 1, Offset: 0	KafkaPrototypeProducer - 1 sent to server
ConsumerWorker - Value: 2, Offset: 1	KafkaPrototypeProducer - 2 sent to server
ConsumerWorker - Value: 3, Offset: 2	KafkaPrototypeProducer - 3 sent to server
ConsumerWorker - Value: 4, Offset: 3	KafkaPrototypeProducer - 4 sent to server
ConsumerWorker - Value: 5, Offset: 4	KafkaPrototypeProducer - 5 sent to server
ConsumerWorker - Value: 6, Offset: 5	KafkaPrototypeProducer - 6 sent to server
ConsumerWorker - Value: 7, Offset: 6	KafkaPrototypeProducer - 7 sent to server
ConsumerWorker - Value: 8, Offset: 7	KafkaPrototypeProducer - 8 sent to server
ConsumerWorker - Value: 9, Offset: 8	KafkaPrototypeProducer - 9 sent to server
ConsumerWorker - Value: 10, Offset: 9	KafkaPrototypeProducer - 10 sent to server
ConsumerWorker - Value: 11, Offset: 10	KafkaPrototypeProducer - 11 sent to server
DOCKER COMPOSE DOWN	
NetworkClient - Node 2147483646 disconnected.	NetworkClient - Node 1 disconnected.
NetworkClient - Cancelled in-flight OFFSET_COMMIT	NetworkClient - Node -1 disconnected.
NetworkClient - Cancelled in-flight OFFSET_COMMIT	NetworkClient - Node 1 disconnected.
NetworkClient - Cancelled in-flight OFFSET_COMMIT	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient - Cancelled in-flight OFFSET_COMMIT	NetworkClient - Node 1 disconnected.
NetworkClient - Cancelled in-flight OFFSET_COMMIT	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
consumer.internals.ConsumerCoordinator - Group coordinator is unavailable or invalid	NetworkClient - Node 1 disconnected.
NetworkClient - Node 1 disconnected.	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient - Cancelled in-flight FETCH request	NetworkClient - Node 1 disconnected.
NetworkClient - Cancelled in-flight METADATA request	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient - Cancelled in-flight FIND_COORDINATOR request	NetworkClient - Node 1 disconnected.
FetchSessionHandler - Error sending fetch request (sessionId=254417776, epoch=73) to node 1:	NetworkClient - Connection to node 1 could not be established. Broker may not be available.

org.apache.kafka.common.errors.DisconnectException	NetworkClient - Node 1 disconnected.
NetworkClient - Node -1 disconnected.	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient - Node 1 disconnected.	NetworkClient - Node 1 disconnected.
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may no	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient - Node 1 disconnected.	NetworkClient - Node 1 disconnected.
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may no	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 12 sent to server
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may no	KafkaPrototypeProducer - 13 sent to server
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 14 sent to server
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may no	KafkaPrototypeProducer - 15 sent to server
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 16 sent to server
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may no	KafkaPrototypeProducer - 17 sent to server
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 18 sent to server
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may no	KafkaPrototypeProducer - 19 sent to server
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 20 sent to server
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.	
NetworkClient - Node 1 disconnected.	
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.	
NetworkClient - Node 1 disconnected.	
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.	
NetworkClient - Node 1 disconnected.	
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.	
NetworkClient - Node 1 disconnected.	
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.	
NetworkClient - Node 1 disconnected.	
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.	
NetworkClient - Node 1 disconnected.	
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.	
NetworkClient - Node 1 disconnected.	
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.	
NetworkClient - Node 1 disconnected.	
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.	
NetworkClient - Node 1 disconnected.	
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.	
NetworkClient - Node 1 disconnected.	
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.	
NetworkClient - Error while fetching metadata with correlation id 7370 : {Testtopic=LEADER_NOT_AVAILABLE}	
NetworkClient - Error while fetching metadata with correlation id 7372 : {Testtopic=LEADER_NOT_AVAILABLE}	
NetworkClient - Error while fetching metadata with correlation id 7374 : {Testtopic=LEADER_NOT_AVAILABLE}	
NetworkClient - Error while fetching metadata with correlation id 7376 : {Testtopic=LEADER_NOT_AVAILABLE}	
Metadata - Resetting the last seen epoch of partition Testtopic-0	
consumer.internals.ConsumerCoordinator - Discovered group coordinator 10.212.26.245:9092 (id: 2147483646 rack: null)	
ConsumerWorker - Value: 12, Offset: 11	
ConsumerWorker - Value: 13, Offset: 12	
ConsumerWorker - Value: 14, Offset: 13	
ConsumerWorker - Value: 15, Offset: 14	
ConsumerWorker - Value: 16, Offset: 15	
ConsumerWorker - Value: 17, Offset: 16	
ConsumerWorker - Value: 18, Offset: 17	
ConsumerWorker - Value: 19, Offset: 18	
ConsumerWorker - Value: 20, Offset: 19	

Shutdown broker, server shutdown

[Thread-0] INFO com.autostreams.kafka.ConsumerWorker - Value: 1, Offset: 0	KafkaPrototypeProducer - 1 sent to server
[Thread-0] INFO com.autostreams.kafka.ConsumerWorker - Value: 2, Offset: 1	KafkaPrototypeProducer - 2 sent to server
[Thread-0] INFO com.autostreams.kafka.ConsumerWorker - Value: 3, Offset: 2	KafkaPrototypeProducer - 3 sent to server
[Thread-0] INFO com.autostreams.kafka.ConsumerWorker - Value: 4, Offset: 3	KafkaPrototypeProducer - 4 sent to server
[Thread-0] INFO com.autostreams.kafka.ConsumerWorker - Value: 5, Offset: 4	KafkaPrototypeProducer - 5 sent to server
[Thread-0] INFO com.autostreams.kafka.ConsumerWorker - Value: 6, Offset: 5	KafkaPrototypeProducer - 6 sent to server
[Thread-0] INFO com.autostreams.kafka.ConsumerWorker - Value: 7, Offset: 6	KafkaPrototypeProducer - 7 sent to server
[Thread-0] INFO com.autostreams.kafka.ConsumerWorker - Value: 8, Offset: 7	KafkaPrototypeProducer - 8 sent to server

Disconnect-test, Kafka

[Thread-0] INFO com.autostreams.kafka.ConsumerWorker - Value: 9, Offset: 8	KafkaPrototypeProducer - 9 sent to server
[Thread-0] INFO com.autostreams.kafka.ConsumerWorker - Value: 10, Offset: 9	KafkaPrototypeProducer - 10 sent to server
[Thread-0] INFO com.autostreams.kafka.ConsumerWorker - Value: 11, Offset: 10	KafkaPrototypeProducer - 11 sent to server
[Thread-0] INFO com.autostreams.kafka.ConsumerWorker - Value: 12, Offset: 11	KafkaPrototypeProducer - 12 sent to server
[Thread-0] INFO com.autostreams.kafka.ConsumerWorker - Value: 13, Offset: 12	KafkaPrototypeProducer - 13 sent to server
[Thread-0] INFO com.autostreams.kafka.ConsumerWorker - Value: 14, Offset: 13	KafkaPrototypeProducer - 14 sent to server
[Thread-0] INFO com.autostreams.kafka.ConsumerWorker - Value: 15, Offset: 14	KafkaPrototypeProducer - 15 sent to server
[Thread-0] INFO com.autostreams.kafka.ConsumerWorker - Value: 16, Offset: 15	KafkaPrototypeProducer - 16 sent to server
[Thread-0] INFO com.autostreams.kafka.ConsumerWorker - Value: 17, Offset: 16	KafkaPrototypeProducer - 17 sent to server
[Thread-0] INFO com.autostreams.kafka.ConsumerWorker - Value: 18, Offset: 17	KafkaPrototypeProducer - 18 sent to server
[Thread-0] INFO com.autostreams.kafka.ConsumerWorker - Value: 19, Offset: 18	KafkaPrototypeProducer - 19 sent to server
[Thread-0] INFO com.autostreams.kafka.ConsumerWorker - Value: 20, Offset: 19	KafkaPrototypeProducer - 20 sent to server
[Thread-0] INFO com.autostreams.kafka.ConsumerWorker - Value: 21, Offset: 20	KafkaPrototypeProducer - 21 sent to server
SERVER SHUTDOWN	
NetworkClient - Node -1 disconnected.	NetworkClient - Node -1 disconnected.
NetworkClient - Node 1 disconnected.	NetworkClient - Node 1 disconnected.
NetworkClient - Cancelled in-flight FETCH request	NetworkClient - Cancelled in-flight METADATA request with correlation id 25 due to node 1 being disconnected.
NetworkClient - Cancelled in-flight METADATA request	NetworkClient - Node 1 disconnected.
NetworkClient - Node 2147483646 disconnected.	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient - Cancelled in-flight OFFSET_COMMIT	NetworkClient - Node 1 disconnected.
NetworkClient - Cancelled in-flight OFFSET_COMMIT	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient - Cancelled in-flight OFFSET_COMMIT	NetworkClient - Node 1 disconnected.
NetworkClient - Cancelled in-flight OFFSET_COMMIT	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient - Cancelled in-flight OFFSET_COMMIT	NetworkClient - Node 1 disconnected.
NetworkClient - Cancelled in-flight OFFSET_COMMIT	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient - Cancelled in-flight OFFSET_COMMIT	NetworkClient - Node 1 disconnected.
NetworkClient - Cancelled in-flight OFFSET_COMMIT	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient - Cancelled in-flight OFFSET_COMMIT	NetworkClient - Node 1 disconnected.
NetworkClient - Cancelled in-flight OFFSET_COMMIT	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient - Cancelled in-flight OFFSET_COMMIT	NetworkClient - Node 1 disconnected.
NetworkClient - Cancelled in-flight OFFSET_COMMIT	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient - Cancelled in-flight OFFSET_COMMIT	NetworkClient - Disconnecting from node 1 due to socket connection setup timeout. The timeout value is 25140 ms.
NetworkClient - Cancelled in-flight OFFSET_COMMIT	NetworkClient - Node 1 disconnected.
FetchSessionHandler - Error sending fetch request (sessionId=730830382, epoch=108) to node 1:	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
org.apache.kafka.common.errors.DisconnectException	NetworkClient - Node 1 disconnected.
consumer.internals.ConsumerCoordinator - Group coordinator is unavailable	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient - Node 1 disconnected.	NetworkClient - Node 1 disconnected.
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be avail	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient - Node 1 disconnected.	NetworkClient - Node 1 disconnected.
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be avail	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient - Node 1 disconnected.	NetworkClient - Node 1 disconnected.
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be avail	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient - Node 1 disconnected.	NetworkClient - Node 1 disconnected.
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be avail	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient - Node 1 disconnected.	NetworkClient - Node 1 disconnected.
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be avail	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient - Node 1 disconnected.	NetworkClient - Node 1 disconnected.
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be avail	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient - Node 1 disconnected.	NetworkClient - Node 1 disconnected.
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be avail	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient - Node 1 disconnected.	NetworkClient - Node 1 disconnected.
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be avail	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient - Node 1 disconnected.	NetworkClient - Node 1 disconnected.
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be avail	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient - Node 1 disconnected.	NetworkClient - Node 1 disconnected.
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be avail	NetworkClient - Cancelled in-flight API_VERSIONS request with correlation id 26 due to node 1 being disconnected.
NetworkClient - Node 1 disconnected.	NetworkClient - Node 1 disconnected.
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be avail	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient - Node 1 disconnected.	NetworkClient - Node 1 disconnected.
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be avail	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient - Node 1 disconnected.	NetworkClient - Node 1 disconnected.
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be avail	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient - Node 1 disconnected.	NetworkClient - Node 1 disconnected.

NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	NetworkClient - Connection to node 1 could not be established. Broker may not be available.					
NetworkClient - Node 1 disconnected.	NetworkClient - Node 1 disconnected.					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	NetworkClient - Connection to node 1 could not be established. Broker may not be available.					
NetworkClient - Node 1 disconnected.	NetworkClient - Node 1 disconnected.					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	NetworkClient - Connection to node 1 could not be established. Broker may not be available.					
NetworkClient - Node 1 disconnected.	NetworkClient - Node 1 disconnected.					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	NetworkClient - Connection to node 1 could not be established. Broker may not be available.					
NetworkClient - Node 1 disconnected.	NetworkClient - Node 1 disconnected.					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	NetworkClient - Connection to node 1 could not be established. Broker may not be available.					
NetworkClient - Node 1 disconnected.	NetworkClient - Node 1 disconnected.					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	NetworkClient - Connection to node 1 could not be established. Broker may not be available.					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 22 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 23 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 24 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 25 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 26 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 27 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 28 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 29 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 30 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 31 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 32 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 33 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 34 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 35 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 36 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 37 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 38 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 39 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 40 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 41 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 42 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 43 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 44 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 45 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 46 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 47 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 48 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 49 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 50 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 51 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 52 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 53 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 54 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 55 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 56 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 57 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 58 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 59 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 60 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 61 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 62 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 63 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 64 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 65 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 66 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 67 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 68 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 69 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 70 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 71 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 72 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 73 sent to server					

NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 74 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 75 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 76 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 77 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 78 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 79 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 80 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 81 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 82 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 83 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 84 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 85 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 86 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 87 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 88 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 89 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 90 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 91 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 92 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 93 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 94 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 95 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 96 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 97 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 98 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 99 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 100 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 101 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 102 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 103 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 104 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 105 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 106 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 107 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 108 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 109 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 110 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 111 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 112 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 113 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 114 sent to server					
NetworkClient - Cancelled in-flight API_VERSIONS request	KafkaPrototypeProducer - 115 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 116 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 117 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 118 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 119 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 120 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 121 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 122 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 123 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 124 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 125 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 126 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 127 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 128 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 129 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 130 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 131 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 132 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 133 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 134 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 135 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 136 sent to server					

NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 137 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 138 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 139 sent to server					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 140 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	KafkaPrototypeProducer - 141 sent to server					
NetworkClient - Node 1 disconnected.	NetworkClient - Node 1 disconnected.					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	NetworkClient - Connection to node 1 could not be established. Broker may not be available.					
NetworkClient - Node 1 disconnected.	NetworkClient - Node 1 disconnected.					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	NetworkClient - Connection to node 1 could not be established. Broker may not be available.					
NetworkClient - Node 1 disconnected.	NetworkClient - Node 1 disconnected.					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be availa	NetworkClient - Connection to node 1 could not be established. Broker may not be available.					
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 142 sent to server					
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.						
NetworkClient - Node 1 disconnected.						
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.						
NetworkClient - Node 1 disconnected.						
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.						
NetworkClient - Node 1 disconnected.						
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.						
NetworkClient - Node 1 disconnected.						
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.						
NetworkClient - Node 1 disconnected.						
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.						
NetworkClient - Node 1 disconnected.						
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.						
NetworkClient - Node 1 disconnected.						
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.						
NetworkClient - Node 1 disconnected.						
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.						
NetworkClient - Node 1 disconnected.						
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.						
NetworkClient - Node 1 disconnected.						
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.						
NetworkClient - Node 1 disconnected.						
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.						
NetworkClient - Node 1 disconnected.						
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.						
NetworkClient - Node 1 disconnected.						
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.						
NetworkClient - Node 1 disconnected.						
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.						
NetworkClient - Node 1 disconnected.						
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.						
NetworkClient - Node 1 disconnected.						
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.						
NetworkClient - Node 1 disconnected.						
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.						
NetworkClient - Node 1 disconnected.						
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.						
NetworkClient - Node 1 disconnected.						
NetworkClient - Connection to node 1 (/10.212.26.245:9092) could not be established. Broker may not be available.						
NetworkClient - Error while fetching metadata with correlation id 10781 : {Testtopic=LEADER_NOT_AVAILABLE}						
NetworkClient - Error while fetching metadata with correlation id 10783 : {Testtopic=LEADER_NOT_AVAILABLE}						
NetworkClient - Error while fetching metadata with correlation id 10785 : {Testtopic=LEADER_NOT_AVAILABLE}						
NetworkClient - Error while fetching metadata with correlation id 10787 : {Testtopic=LEADER_NOT_AVAILABLE}						
NetworkClient - Error while fetching metadata with correlation id 10789 : {Testtopic=LEADER_NOT_AVAILABLE}						
NetworkClient - Error while fetching metadata with correlation id 10791 : {Testtopic=LEADER_NOT_AVAILABLE}						
Metadata - Resetting the last seen epoch of partition Testtopic-0 to 0 since the associated topicId changed from null to of1l4XPcTguj81wE1Oh2Gg						
consumer.internals.ConsumerCoordinator - Discovered group coordinator 10.212.26.245:9092 (id: 2147483646 rack: null)						
consumer.internals.ConsumerCoordinator - Offset commit failed on partition Testtopic-0 at offset 21:						
consumer.internals.ConsumerCoordinator - Offset commit failed on partition Testtopic-0 at offset 21:						
consumer.internals.ConsumerCoordinator - Offset commit failed on partition Testtopic-0 at offset 21:						
consumer.internals.ConsumerCoordinator - Offset commit failed on partition Testtopic-0 at offset 21:						
consumer.internals.ConsumerCoordinator - Offset commit failed on partition Testtopic-0 at offset 21:						
consumer.internals.ConsumerCoordinator - Offset commit failed on partition Testtopic-0 at offset 21:						
consumer.internals.ConsumerCoordinator - Offset commit failed on partition Testtopic-0 at offset 21:						

consumer.internals.ConsumerCoordinator - Offset commit failed on partition Testtopic-0 at offset 21:	
consumer.internals.ConsumerCoordinator - Offset commit failed on partition Testtopic-0 at offset 21:	
consumer.internals.ConsumerCoordinator - Offset commit failed on partition Testtopic-0 at offset 21:	
consumer.internals.ConsumerCoordinator - Offset commit failed on partition Testtopic-0 at offset 21:	
consumer.internals.ConsumerCoordinator - Offset commit failed on partition Testtopic-0 at offset 21:	
consumer.internals.ConsumerCoordinator - Offset commit failed on partition Testtopic-0 at offset 21:	
consumer.internals.ConsumerCoordinator - Offset commit failed on partition Testtopic-0 at offset 21:	
consumer.internals.ConsumerCoordinator - Offset commit failed on partition Testtopic-0 at offset 21:	
consumer.internals.ConsumerCoordinator - Offset commit failed on partition Testtopic-0 at offset 21:	
consumer.internals.ConsumerCoordinator - Offset commit failed on partition Testtopic-0 at offset 21:	
consumer.internals.ConsumerCoordinator - Offset commit failed on partition Testtopic-0 at offset 21:	
consumer.internals.ConsumerCoordinator - Offset commit failed on partition Testtopic-0 at offset 21:	
consumer.internals.ConsumerCoordinator - Offset commit failed on partition Testtopic-0 at offset 21:	
consumer.internals.ConsumerCoordinator - Offset commit failed on partition Testtopic-0 at offset 21:	
consumer.internals.ConsumerCoordinator - Offset commit failed on partition Testtopic-0 at offset 21:	
consumer.internals.ConsumerCoordinator - Offset commit failed on partition Testtopic-0 at offset 21:	
consumer.internals.ConsumerCoordinator - Offset commit failed on partition Testtopic-0 at offset 21:	
[Thread-0] INFO com.autostreams.kafka.ConsumerWorker - Value: 142, Offset: 21	
[Thread-0] INFO com.autostreams.kafka.ConsumerWorker - Value: 143, Offset: 22	
Broker disconnect, network disconnect	
ConsumerWorker - Value: 1, Offset: 0	KafkaPrototypeProducer - 1 sent to server
ConsumerWorker - Value: 2, Offset: 1	KafkaPrototypeProducer - 2 sent to server
ConsumerWorker - Value: 3, Offset: 2	KafkaPrototypeProducer - 3 sent to server
ConsumerWorker - Value: 4, Offset: 3	KafkaPrototypeProducer - 4 sent to server
ConsumerWorker - Value: 5, Offset: 4	KafkaPrototypeProducer - 5 sent to server
DOCKER NETWORK DISCONNECTED	
NetworkClient -Node 2147483646 disconnected.	NetworkClient - Node 1 disconnected.
NetworkClient -Cancelled in-flight OFFSET_COMMIT	NetworkClient - Cancelled in-flight PRODUCE request with correlation id 9 due to node 1 being disconnected.
NetworkClient -Cancelled in-flight OFFSET_COMMIT	Sender - Got error produce response with correlation id 9 on topic-partition Testtopic-0, retrying (2147483646 attempts left).
NetworkClient -Cancelled in-flight OFFSET_COMMIT	NetworkException: Disconnected from node 1. Going to request metadata update now
NetworkClient -Cancelled in-flight OFFSET_COMMIT	NetworkClient - Node 1 disconnected.
NetworkClient -Cancelled in-flight OFFSET_COMMIT	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient -Cancelled in-flight OFFSET_COMMIT	NetworkClient - Node 1 disconnected.
NetworkClient -Cancelled in-flight OFFSET_COMMIT	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient -Cancelled in-flight OFFSET_COMMIT	NetworkClient - Node 1 disconnected.
NetworkClient -Cancelled in-flight OFFSET_COMMIT	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
ConsumerCoordinatorGroup coordinator is unavailable or invalid due to cause: null.isDisconnected: true.	NetworkClient - Node 1 disconnected.
NetworkClient -Node 1 disconnected.	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient -Cancelled in-flight FETCH request	NetworkClient - Node 1 disconnected.
NetworkClient -Cancelled in-flight METADATA request	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient -Cancelled in-flight FIND_COORDINATOR request	NetworkClient - Node 1 disconnected.
FetchSessionHandler Error sending fetch request to node 1: org.apache.kafka.common.errors.DisconnectExcepti	NetworkClient - Connection to node 1 could not be established. Broker may not be available.
NetworkClient -Node 1 disconnected.	KafkaPrototypeProducer - 6 sent to server
NetworkClient - Connection to node 1 could not be established. Broker may not be available.	KafkaPrototypeProducer - 7 sent to server
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 8 sent to server
NetworkClient - Connection to node 1 could not be established. Broker may not be available.	KafkaPrototypeProducer - 9 sent to server
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 10 sent to server
NetworkClient - Connection to node 1 could not be established. Broker may not be available.	KafkaPrototypeProducer - 11 sent to server
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 12 sent to server
NetworkClient - Connection to node 1 could not be established. Broker may not be available.	KafkaPrototypeProducer - 13 sent to server
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 14 sent to server
NetworkClient - Connection to node 1 could not be established. Broker may not be available.	KafkaPrototypeProducer - 15 sent to server
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 16 sent to server
NetworkClient - Connection to node 1 could not be established. Broker may not be available.	KafkaPrototypeProducer - 17 sent to server
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 18 sent to server
NetworkClient - Connection to node 1 could not be established. Broker may not be available.	KafkaPrototypeProducer - 19 sent to server
NetworkClient - Node 1 disconnected.	KafkaPrototypeProducer - 20 sent to server
NetworkClient - Connection to node 1 could not be established. Broker may not be available.	
NetworkClient - Node 1 disconnected.	
NetworkClient - Connection to node 1 could not be established. Broker may not be available.	
consumer.internals.ConsumerCoordinator - Discovered group coordinator	

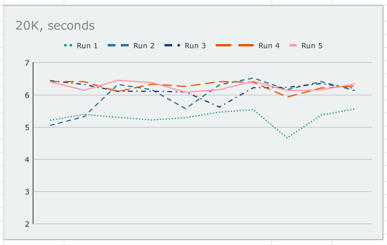
Disconnect-test, Kafka

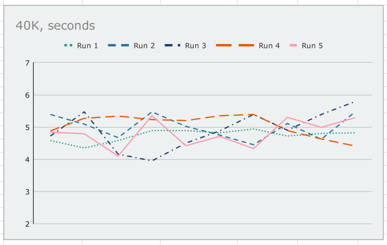
```
ConsumerWorker - Value: 6, Offset: 5
ConsumerWorker - Value: 7, Offset: 6
ConsumerWorker - Value: 8, Offset: 7
ConsumerWorker - Value: 9, Offset: 8
ConsumerWorker - Value: 10, Offset: 9
ConsumerWorker - Value: 11, Offset: 10
ConsumerWorker - Value: 12, Offset: 11
ConsumerWorker - Value: 13, Offset: 12
ConsumerWorker - Value: 14, Offset: 13
ConsumerWorker - Value: 15, Offset: 14
ConsumerWorker - Value: 16, Offset: 15
ConsumerWorker - Value: 17, Offset: 16
ConsumerWorker - Value: 18, Offset: 17
ConsumerWorker - Value: 19, Offset: 18
ConsumerWorker - Value: 20, Offset: 19
```

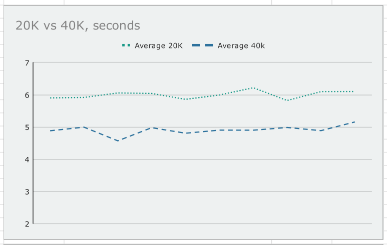
Kafka Producer 1M raw

2 Data providers sending 10 000 msg/s each		
1 Producer in Ålesund		
1 Broker running in Docker container on NTN		
1 Consumer in Haugesund		
Run 1		
Messages	**Seconds**	**Messages per second**
100000	5.21	19201
100000	5.39	18570
100000	5.3	18875
100000	5.22	19149
100000	5.29	18889
100000	5.46	18331
100000	5.54	18063
100000	4.66	21440
100000	5.37	18628
100000	5.56	17988
Run 2		
Messages	**Seconds**	**Messages per second**
100000	5.05	19798
100000	5.32	18804
100000	6.32	15827
100000	6.16	16225
100000	5.57	17966
100000	6.31	15847
100000	6.52	15330
100000	6.16	16244
100000	6.41	15593
100000	6.13	16315
Run 3		
Messages	**Seconds**	**Messages per second**
100000	6.44	15527
100000	6.32	15835
100000	6.11	16358
100000	6.11	16358
100000	6.09	16409
100000	5.62	17780
100000	6.22	16069
100000	6.23	16059
100000	6.35	15748
100000	6.2	16142
Run 4		
Messages	**Seconds**	**Messages per second**
100000	6.41	15612
100000	6.41	15600
100000	6.1	16401
100000	6.33	15807
100000	6.26	15976
100000	6.41	15590
100000	6.39	15651
100000	5.93	16860
100000	6.21	16097
100000	6.27	15948
Run 5		
Messages	**Seconds**	**Messages per second**
100000	6.4	15637
100000	6.14	16299
100000	6.45	15496
100000	6.38	15671
100000	6.08	16458
100000	6.16	16236
100000	6.42	15586
100000	6.13	16302
100000	6.15	16254
100000	6.34	15767

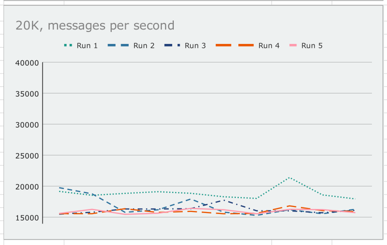
2 Data providers sending 20 000 msg/s eac		
1 Producer in Ålesund		
1 Broker running in Docker container on NT		
1 Consumer in Haugesund		
Run 1		
Messages	**Seconds**	**Messages per second**
100000	4.58	21843
100000	4.35	22977
100000	4.58	21843
100000	4.89	20470
100000	4.89	20445
100000	4.82	20742
100000	4.94	20263
100000	4.72	21190
100000	4.8	20833
100000	4.82	20764
Run 2		
Messages	**Seconds**	**Messages per second**
100000	5.39	18566
100000	5.09	19654
100000	4.67	21427
100000	5.47	18281
100000	5.02	19932
100000	4.75	21070
100000	4.45	22487
100000	5.11	19588
100000	4.62	21626
100000	5.47	18291
Run 3		
Messages	**Seconds**	**Messages per second**
100000	4.72	21190
100000	5.47	18288
100000	4.16	24021
100000	3.95	25342
100000	4.5	22222
100000	4.88	20491
100000	5.39	18542
100000	4.9	20403
100000	5.39	18563
100000	5.79	17259
Run 4		
Messages	**Seconds**	**Messages per second**
100000	4.88	20504
100000	5.27	18960
100000	5.34	18719
100000	5.23	19138
100000	5.2	19219
100000	5.35	18709
100000	5.39	18552
100000	4.89	20449
100000	4.63	21588
100000	4.41	22675
Run 5		
Messages	**Seconds**	**Messages per second**
100000	4.84	20656
100000	4.79	20859
100000	4.1	24414
100000	5.35	18684
100000	4.42	22614
100000	4.71	21231
100000	4.33	23105
100000	5.3	18871
100000	4.99	20052
100000	5.29	18900

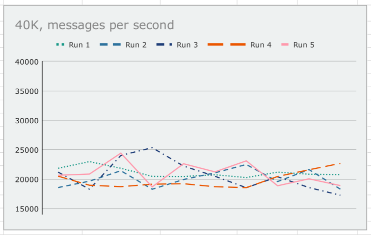


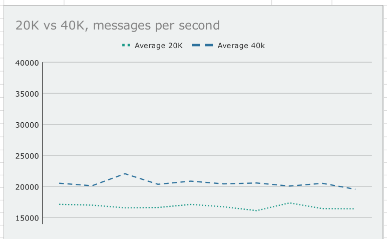




AVG per run 20k	AVG per run 40k
5.902	4.882
5.916	4.994
6.056	4.57
6.04	4.978
5.858	4.806
5.992	4.902
6.218	4.9
5.822	4.984
6.098	4.886
6.1	5.156







AVG per run 20k	AVG per run 40k
17155	20551.8
17021.6	20147.6
16591.4	22084.8
16642	20383
17139.6	20886.4
16756.8	20448.6
16139.8	20589.8
17381	20100.2
16464	20532.4
16432	19577.8

Kafka Producer 1M raw

2 Data providers sending 10 000 msg/s each		
1 Producer in Ålesund		
1 Broker running in Docker container		
1 Consumer in Haugesund		
Run 1		
Messages	**Seconds**	**Messages per second**
100000	5.17	19353
200000	5.52	18129
300000	5.26	19004
400000	5.13	19496
500000	5.3	18867
600000	5.58	17927
700000	5.44	18385
800000	4.83	20716
900000	5.17	19327
1000000	5.61	17834
Run 2		
Messages	**Seconds**	**Messages per second**
100000	5.05	19817
100000	5.29	18910
100000	6.46	15470
100000	6.32	15822
100000	5.26	19018
100000	6.29	15895
100000	6.53	15313
100000	6.17	16217
100000	6.4	15629
100000	6.14	16289
Run 1		
Messages	**Seconds**	**Messages per second**
100000	6.42	15571

2 Data providers sending 20 000 msg/s each		
1 Producer in Ålesund		
1 Broker running in Docker container		
1 Consumer in Haugesund		
Run 1		
Messages	**Seconds**	**Messages per second**
100000	3.44	29044
100000	3.94	25380
100000	3.3	30312
100000	3.53	28336
100000	3.16	31645
100000	3.66	27337
100000	3.28	30487
100000	3.58	27972
100000	3.32	30120
100000	3.82	26205
Run 2		
Messages	**Seconds**	**Messages per second**
100000	4.39	22789
100000	5.09	19642
100000	4.76	21003
100000	5.34	18716
100000	4.96	20165
100000	4.98	20072
100000	4.32	23137
100000	5.04	19860
100000	4.82	20746
100000	5.24	19080
Run 3		
Messages	**Seconds**	**Messages per second**
100000	4.77	20951

100000	6.33	15805
100000	6.11	16355
100000	6.16	16231
100000	6.05	16526
100000	5.61	17825
100000	6.23	16046
100000	6.24	16025
100000	6.33	15790
100000	6.19	16157
Run 1		
Messages	**Seconds**	**Messages per second**
100000	6.44	15540
100000	6.39	15661
100000	6.11	16371
100000	6.32	15832
100000	6.27	15951
100000	6.42	15588
100000	6.38	15681
100000	5.97	16742
100000	6.18	16178
100000	6.27	15956
Run 1		
Messages	**Seconds**	**Messages per second**
100000	6.38	15678
100000	6.18	16178
100000	6.42	15581
100000	6.45	15494
100000	6.08	16452
100000	6.21	16110
100000	6.29	15888
100000	6.14	16278

100000	5.62	17799
100000	4.58	21824
100000	3.63	27525
100000	4.21	23747
100000	4.86	20567
100000	5.41	18501
100000	4.91	20354
100000	5.41	18487
100000	5.75	17406
Run 4		
Messages	**Seconds**	**Messages per second**
100000	4.9	20429
100000	5.31	18850
100000	5.43	18426
100000	5.21	19197
100000	5.11	19580
100000	5.33	18775
100000	5.38	18580
100000	5	19992
100000	4.6	21720
100000	4.36	22951
Run 5		
Messages	**Seconds**	**Messages per second**
100000	5.09	19654
100000	4.54	22021
100000	4.08	24485
100000	5.56	17998
100000	4.21	23747
100000	4.79	20898
100000	4.77	20951
100000	4.83	20712

100000	6.17	16210		100000	4.98	20084	
100000	6.31	15845		100000	5.24	19094	

Disconnect test, Producer, raw data	
ConsumerWorker - Value: 1, Offset: 0	PulsarPrototypeProducer - 1 sent to server
ConsumerWorker - Value: 2, Offset: 1	PulsarPrototypeProducer - 2 sent to server
ConsumerWorker - Value: 3, Offset: 2	PulsarPrototypeProducer - 3 sent to server
ConsumerWorker - Value: 4, Offset: 3	PulsarPrototypeProducer - 4 sent to server
ConsumerWorker - Value: 5, Offset: 4	PulsarPrototypeProducer - 5 sent to server
Producer Disconnect	
.	ClientCnx - Got exception {}
.	NativeIoException: Connection reset by peer
.	ClientCnx - Disconnected
.	ClientCnx - Got exception {}
.	NativeIoException: Connection reset by peer
.	ClientCnx - Disconnected
.	ConnectionHandler - Closed connection- Will try again in 0.1 s
.	ConnectionHandler - Reconnecting after timeout
.	ConnectionPool - Connected to server
.	ConnectionPool - Connected to server
.	ClientCnx - Connected through proxy to target broker at localhost:6650
.	ProducerImpl - Creating producer on cnx
.	ProducerImpl - Created producer on cnx
.	ProducerImpl - Re-Sending 18 messages to server
ConsumerWorker - Value: 6, Offset: 5	PulsarPrototypeProducer - 6 sent to server
ConsumerWorker - Value: 7, Offset: 6	PulsarPrototypeProducer - 7 sent to server
ConsumerWorker - Value: 8, Offset: 7	PulsarPrototypeProducer - 8 sent to server
ConsumerWorker - Value: 9, Offset: 8	PulsarPrototypeProducer - 9 sent to server
ConsumerWorker - Value: 10, Offset: 9	PulsarPrototypeProducer - 10 sent to server
ConsumerWorker - Value: 11, Offset: 10	PulsarPrototypeProducer - 11 sent to server
ConsumerWorker - Value: 12, Offset: 11	PulsarPrototypeProducer - 12 sent to server
ConsumerWorker - Value: 13, Offset: 12	PulsarPrototypeProducer - 13 sent to server
ConsumerWorker - Value: 14, Offset: 13	PulsarPrototypeProducer - 14 sent to server
ConsumerWorker - Value: 15, Offset: 14	PulsarPrototypeProducer - 15 sent to server
ConsumerWorker - Value: 16, Offset: 15	PulsarPrototypeProducer - 16 sent to server
ConsumerWorker - Value: 17, Offset: 16	PulsarPrototypeProducer - 17 sent to server
ConsumerWorker - Value: 18, Offset: 17	PulsarPrototypeProducer - 18 sent to server
ConsumerWorker - Value: 19, Offset: 18	PulsarPrototypeProducer - 19 sent to server
ConsumerWorker - Value: 20, Offset: 19	PulsarPrototypeProducer - 20 sent to server

Disconnect test, Consumer, raw data	
ConsumerWorker - Value: 1, Offset: 0	PulsarPrototypeProducer - 1 sent to server
ConsumerWorker - Value: 2, Offset: 1	PulsarPrototypeProducer - 2 sent to server
ConsumerWorker - Value: 3, Offset: 2	PulsarPrototypeProducer - 3 sent to server
ConsumerWorker - Value: 4, Offset: 3	PulsarPrototypeProducer - 4 sent to server
ConsumerWorker - Value: 5, Offset: 4	PulsarPrototypeProducer - 5 sent to server

CONSUMER DISCONNECTED	
PulsarHandler - Forcing connection to close after keep-alive	PulsarPrototypeProducer - 6 sent to server
ClientCnx - Disconnected	PulsarPrototypeProducer - 7 sent to server
PulsarHandler - Forcing connection to close after keep-alive	PulsarPrototypeProducer - 8 sent to server
ClientCnx -Disconnected	PulsarPrototypeProducer - 9 sent to server
ConnectionHandler -Closed connection	PulsarPrototypeProducer - 10 sent to server
ConnectionHandler - Reconnecting after timeout	PulsarPrototypeProducer - 11 sent to server
ConnectionPool - Connected to server	PulsarPrototypeProducer - 12 sent to server
ConnectionPool - Connected to server	PulsarPrototypeProducer - 13 sent to server
ClientCnx - Connected through proxy to target broker	PulsarPrototypeProducer - 14 sent to server
ConsumerImpl - Subscribing to topic	PulsarPrototypeProducer - 15 sent to server
ConsumerImpl - Subscribed to topic	PulsarPrototypeProducer - 16 sent to server
ConsumerWorker - Value: 6, Offset: 5	PulsarPrototypeProducer - 17 sent to server
ConsumerWorker - Value: 7, Offset: 6	PulsarPrototypeProducer - 18 sent to server
ConsumerWorker - Value: 8, Offset: 7	PulsarPrototypeProducer - 19 sent to server
ConsumerWorker - Value: 9, Offset: 8	PulsarPrototypeProducer - 20 sent to server
ConsumerWorker - Value: 10, Offset: 9	
ConsumerWorker - Value: 11, Offset: 10	
ConsumerWorker - Value: 12, Offset: 11	
ConsumerWorker - Value: 13, Offset: 12	
ConsumerWorker - Value: 14, Offset: 13	
ConsumerWorker - Value: 15, Offset: 14	
ConsumerWorker - Value: 16, Offset: 15	
ConsumerWorker - Value: 17, Offset: 16	
ConsumerWorker - Value: 18, Offset: 17	
ConsumerWorker - Value: 19, Offset: 18	
ConsumerWorker - Value: 20, Offset: 19	

Shutdown docker compose test, Broker

ConsumerWorker - Value: 1, Offset: 0	PulsarPrototypeProducer - 1 sent to server
ConsumerWorker - Value: 2, Offset: 1	PulsarPrototypeProducer - 2 sent to server
ConsumerWorker - Value: 3, Offset: 2	PulsarPrototypeProducer - 3 sent to server
ConsumerWorker - Value: 4, Offset: 3	PulsarPrototypeProducer - 4 sent to server
ConsumerWorker - Value: 5, Offset: 4	PulsarPrototypeProducer - 5 sent to server
ConsumerWorker - Value: 6, Offset: 5	PulsarPrototypeProducer - 6 sent to server
ConsumerWorker - Value: 7, Offset: 6	PulsarPrototypeProducer - 7 sent to server
ConsumerWorker - Value: 8, Offset: 7	PulsarPrototypeProducer - 8 sent to server
SHUTDOWN DOCKER COMPOSE	
ClientCnx - Closed consumer: 0	ClientCnx - Broker notification of Closed producer: 0
ConnectionHandler - Closed connection	ConnectionHandler - Closed connection -- Will try again in 0.1 s
ClientCnx - Disconnected	ClientCnx - Disconnected
ClientCnx - Disconnected	ClientCnx - Disconnected
ConnectionHandler - Reconnecting after timeout	ConnectionHandler - Reconnecting after timeout

ConnectionPool - Failed to open connection	AnnotatedConnectException: Connection refused
ConnectionHandler - Error connecting to broker	AnnotatedConnectException: Connection refused
ConnectionHandler - Could not get connection to broker	ConnectException: Connection refused -- Will try again in 0.2 s
ConnectionHandler -Reconnecting after connection was clos	ConnectionHandler - Reconnecting after connection was closed
ConnectionPool - Failed to open connection	AnnotatedConnectException: Connection refused
ConnectionHandler -Error connecting to broker	AnnotatedConnectException: Connection refused
ConnectionHandler -Could not get connection to broker	AnnotatedConnectException: Connection refused - Will try again in 0.382 s
ConnectionHandler -Reconnecting after connection was clos	ConnectionHandler - Reconnecting after connection was closed
ConnectionPool - Failed to open connection	AnnotatedConnectException: Connection refused
ConnectionHandler -Error connecting to broker	AnnotatedConnectException: Connection refused
ConnectionHandler - Could not get connection to broker	AnnotatedConnectException: Connection refused - Will try again in 0.752 s
ConnectionHandler -Reconnecting after connection was clos	ConnectionHandler - Reconnecting after connection was closed
ConnectionPool - Failed to open connection	AnnotatedConnectException: Connection refused
ConnectionHandler -Error connecting to broker	AnnotatedConnectException: Connection refused
ConnectionHandler -Could not get connection to broker	AnnotatedConnectException: Connection refused -- Will try again in 1.503 s
ConnectionHandler -Reconnecting after connection was clos	ConnectionHandler - Reconnecting after connection was closed
ConnectionPool - Failed to open connection	AnnotatedConnectException: Connection refused
ConnectionHandler -Error connecting to broker	AnnotatedConnectException: Connection refused
ConnectionHandler -Could not get connection to broker	AnnotatedConnectException: Connection refused -- Will try again in 3.047 s
ConnectionHandler -Reconnecting after connection was clos	ConnectionHandler - Reconnecting after connection was closed
ConnectionPool - Failed to open connection	AnnotatedConnectException: Connection refused
ConnectionHandler -Error connecting to broker	AnnotatedConnectException: Connection refused
ConnectionHandler -Could not get connection to broker	AnnotatedConnectException: Connection refused -- Will try again in 5.81 s
ConnectionHandler -Reconnecting after connection was clos	ConnectionHandler - Reconnecting after connection was closed
ConnectionPool - Failed to open connection	AnnotatedConnectException: Connection refused
ConnectionHandler -Error connecting to broker	AnnotatedConnectException: Connection refused
ConnectionHandler -Could not get connection to broker	AnnotatedConnectException: Connection refused -- Will try again in 11.675 s
ConnectionHandler -Reconnecting after connection was clos	ConnectionHandler - Reconnecting after connection was closed
ConnectionPool - Connected to server	ConnectionPool - Connected to server
ConnectionPool - Connected to server	ConnectionPool - Connected to server
ClientCnx - Connected through proxy to target broker	ClientCnx - Connected through proxy to target broker at localhost:6650
ConsumerImpl -Subscribing to topic	ProducerImpl - Creating producer on cnx
ConsumerImpl -Subscribed to topic	ProducerImpl - Created producer on cnx
ConsumerWorker - Value: 9, Offset: 8	ProducerImpl - Re-Sending 25 messages to server
ConsumerWorker - Value: 10, Offset: 9	PulsarPrototypeProducer - 9 sent to server
ConsumerWorker - Value: 11, Offset: 10	PulsarPrototypeProducer - 10 sent to server
ConsumerWorker - Value: 12, Offset: 11	PulsarPrototypeProducer - 11 sent to server
ConsumerWorker - Value: 13, Offset: 12	PulsarPrototypeProducer - 12 sent to server
ConsumerWorker - Value: 14, Offset: 13	PulsarPrototypeProducer - 13 sent to server
ConsumerWorker - Value: 15, Offset: 14	PulsarPrototypeProducer - 14 sent to server
ConsumerWorker - Value: 16, Offset: 15	PulsarPrototypeProducer - 15 sent to server
ConsumerWorker - Value: 17, Offset: 16	PulsarPrototypeProducer - 16 sent to server
ConsumerWorker - Value: 18, Offset: 17	PulsarPrototypeProducer - 17 sent to server
ConsumerWorker - Value: 19, Offset: 18	PulsarPrototypeProducer - 18 sent to server
ConsumerWorker - Value: 20, Offset: 19	PulsarPrototypeProducer - 19 sent to server
	PulsarPrototypeProducer - 20 sent to server

Shutdown server while running test, Broker	
ConsumerWorker - Value: 1, Offset: 0	PulsarPrototypeProducer - 1 sent to server
ConsumerWorker - Value: 2, Offset: 1	PulsarPrototypeProducer - 2 sent to server
ConsumerWorker - Value: 3, Offset: 2	PulsarPrototypeProducer - 3 sent to server
ConsumerWorker - Value: 4, Offset: 3	PulsarPrototypeProducer - 4 sent to server
ConsumerWorker - Value: 5, Offset: 4	PulsarPrototypeProducer - 5 sent to server
ConsumerWorker - Value: 6, Offset: 5	PulsarPrototypeProducer - 6 sent to server
ConsumerWorker - Value: 7, Offset: 6	PulsarPrototypeProducer - 7 sent to server
ConsumerWorker - Value: 8, Offset: 7	PulsarPrototypeProducer - 8 sent to server
ConsumerWorker - Value: 9, Offset: 8	PulsarPrototypeProducer - 9 sent to server
ConsumerWorker - Value: 10, Offset: 9	PulsarPrototypeProducer - 10 sent to server
ConsumerWorker - Value: 11, Offset: 10	PulsarPrototypeProducer - 11 sent to server
ConsumerWorker - Value: 12, Offset: 11	PulsarPrototypeProducer - 12 sent to server
ConsumerWorker - Value: 13, Offset: 12	PulsarPrototypeProducer - 13 sent to server
ConsumerWorker - Value: 14, Offset: 13	PulsarPrototypeProducer - 14 sent to server
ConsumerWorker - Value: 15, Offset: 14	PulsarPrototypeProducer - 15 sent to server
SHUTDOWN SERVER	
ClientCnx -Broker notification of Closed consumer: 0	ClientCnx - Broker notification of Closed producer: 0
ConnectionHandler -Closed connection Will try again in 0.1	ConnectionHandler - Closed connection -- Will try again in 0.1 s
ClientCnx - Disconnected	ClientCnx - Disconnected
ClientCnx - Disconnected	ClientCnx - Disconnected
ConnectionHandler - Reconnecting after timeout	ConnectionHandler - Reconnecting after timeout
ConnectionPool -Connection refused	AnnotatedConnectException: Connection refused
ConnectionHandler - Error connecting to broker	AnnotatedConnectException: Connection refused
ConnectionHandler - Could not get connection to broker	AnnotatedConnectException: Connection refused -- Will try again in 0.187 s
ConnectionHandler - Reconnecting after connection was clo	ConnectionHandler - Reconnecting after connection was closed
ConnectionPool - Failed to open connection	AnnotatedConnectException: Connection refused
ConnectionHandler - Error connecting to broker	AnnotatedConnectException: Connection refused
ConnectionHandler - Could not get connection to broker	AnnotatedConnectException: Connection refused -- Will try again in 0.382 s
ConnectionHandler - Reconnecting after connection was clo	ConnectionHandler - Reconnecting after connection was closed
ConnectionPool - Failed to open connection	AnnotatedConnectException: Connection refused
ConnectionHandler - Error connecting to broker	AnnotatedConnectException: Connection refused
ConnectionHandler - Could not get connection to broker	AnnotatedConnectException: Connection refused -- Will try again in 0.747 s
ConnectionHandler - Reconnecting after connection was clo	ConnectionHandler - Reconnecting after connection was closed
ConnectionPool - Failed to open connection	AnnotatedConnectException: Connection refused
ConnectionHandler - Error connecting to broker	AnnotatedConnectException: Connection refused
ConnectionHandler - Could not get connection to broker	AnnotatedConnectException: Connection refused -- Will try again in 1.522 s
ConnectionHandler - Reconnecting after connection was clo	ConnectionHandler - Reconnecting after connection was closed
ConnectionPool - Failed to open connection	AnnotatedConnectException: Connection refused
ConnectionHandler - Error connecting to broker	AnnotatedConnectException: Connection refused
ConnectionHandler - Could not get connection to broker	AnnotatedConnectException: Connection refused -- Will try again in 3.034 s
ConnectionHandler - Reconnecting after connection was clo	ConnectionHandler - Reconnecting after connection was closed

ConnectionPool - Failed to open connection	ConnectTimeoutException: connection timed out
ConnectionHandler - Error connecting to broker	ConnectTimeoutException: connection timed out
ConnectionHandler - Could not get connection to broker	ConnectTimeoutException: connection timed out -- Will try again in 6.139 s
ConnectionHandler - Reconnecting after connection was clo	ConnectionHandler - Reconnecting after connection was closed
ConnectionPool - Failed to open connection	ProducerImpl - Message send timed out. Failing 30 messages
ConnectionHandler - Error connecting to broker	ConnectTimeoutException: connection timed out
ConnectionHandler - Could not get connection to broker	ConnectTimeoutException: connection timed out
ConnectionHandler - Reconnecting after connection was clo	ConnectTimeoutException: connection timed out -- Will try again in 0.1 s
ConnectionPool - Failed to open connection	ConnectionHandler - Reconnecting after connection was closed
ConnectionHandler - Error connecting to broker	AnnotatedConnectException: No route to host
ConnectionHandler - Could not get connection to broker	AnnotatedConnectException: No route to host
ConnectionHandler - Reconnecting after connection was clo	AnnotatedConnectException: No route to host -- Will try again in 24.625 s
ConnectionPool - Failed to open connection	ConnectionHandler - Reconnecting after connection was closed
ConnectionHandler - Error connecting to broker	AnnotatedConnectException: Connection refused
ConnectionHandler - Could not get connection to broker	AnnotatedConnectException: Connection refused
ConnectionHandler - Reconnecting after connection was clo	AnnotatedConnectException: Connection refused -- Will try again in 49.24 s
ConnectionPool - Connected to server	ProducerImpl - Message send timed out. Failing 31 messages
ConnectionPool - Connected to server	ProducerImpl - Message send timed out. Failing 30 messages
ClientCnx - Connected through proxy	ConnectionHandler - Reconnecting after connection was closed
ConsumerImpl - Subscribing to topic	AnnotatedConnectException: Connection refused
ConsumerImpl - Subscribed to topic	AnnotatedConnectException: Connection refused
ConsumerWorker - Value: 168, Offset: 167	AnnotatedConnectException: Connection refused -- Will try again in 55.853 s
	ProducerImpl - Message send timed out. Failing 30 messages
	ProducerImpl - Message send timed out. Failing 31 messages
	ConnectionHandler - Reconnecting after connection was closed
	ConnectionPool - Connected to server
	ConnectionPool - Connected to server
	ClientCnx - Connected through proxy to target broker at localhost:6650
	ProducerImpl - Creating producer on cnx
	ProducerImpl - Created producer on cnx
	ProducerImpl - Re-Sending 11 messages to server
	PulsarPrototypeProducer - 168 sent to server

Disconnect Docker network while running test, Broker

ConsumerWorker - Value: 1, Offset: 0	PulsarPrototypeProducer - 1 sent to server
ConsumerWorker - Value: 2, Offset: 1	PulsarPrototypeProducer - 2 sent to server
ConsumerWorker - Value: 3, Offset: 2	PulsarPrototypeProducer - 3 sent to server
ConsumerWorker - Value: 4, Offset: 3	PulsarPrototypeProducer - 4 sent to server
ConsumerWorker - Value: 5, Offset: 4	PulsarPrototypeProducer - 5 sent to server
ConsumerWorker - Value: 6, Offset: 5	PulsarPrototypeProducer - 6 sent to server
ConsumerWorker - Value: 7, Offset: 6	PulsarPrototypeProducer - 7 sent to server
ConsumerWorker - Value: 8, Offset: 7	PulsarPrototypeProducer - 8 sent to server
ConsumerWorker - Value: 9, Offset: 8	PulsarPrototypeProducer - 9 sent to server

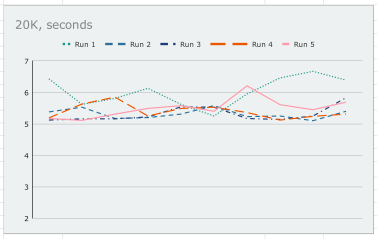
ConsumerWorker - Value: 10, Offset: 9	PulsarPrototypeProducer - 10 sent to server
ConsumerWorker - Value: 11, Offset: 10	PulsarPrototypeProducer - 11 sent to server
ConsumerWorker - Value: 12, Offset: 11	PulsarPrototypeProducer - 12 sent to server
ConsumerWorker - Value: 13, Offset: 12	PulsarPrototypeProducer - 13 sent to server
ConsumerWorker - Value: 14, Offset: 13	PulsarPrototypeProducer - 14 sent to server
ConsumerWorker - Value: 15, Offset: 14	PulsarPrototypeProducer - 15 sent to server
ConsumerWorker - Value: 16, Offset: 15	PulsarPrototypeProducer - 16 sent to server
ConsumerWorker - Value: 17, Offset: 16	PulsarPrototypeProducer - 17 sent to server
ConsumerWorker - Value: 18, Offset: 17	PulsarPrototypeProducer - 18 sent to server
ConsumerWorker - Value: 19, Offset: 18	PulsarPrototypeProducer - 19 sent to server
ConsumerWorker - Value: 20, Offset: 19	PulsarPrototypeProducer - 20 sent to server
DISCONNECT DOCKER NETWORK, BROKER	
ConsumerWorker - Value: 21, Offset: 20	ClientCnx - [10.212.26.245/10.212.26.245:6650] Got exception {}
ConsumerWorker - Value: 22, Offset: 21	NativeIoException: readAddress(..) failed: Connection reset by peer
ConsumerWorker - Value: 23, Offset: 22	ClientCnx - Disconnected
ConsumerWorker - Value: 24, Offset: 23	ConnectionHandler - Closed connection -- Will try again in 0.1 s
ConsumerWorker - Value: 25, Offset: 24	ConnectionHandler - Reconnecting after timeout
	ClientCnx - Got exception {}
	NativeIoException: readAddress(..) failed: Connection reset by peer
	ClientCnx - Disconnected
	BinaryProtoLookupService - Disconnected from server
	ConnectionHandler - ConnectException: Disconnected from server
	ConnectException: Disconnected from server -- Will try again in 0.198 s
	ConnectionHandler - Reconnecting after connection was closed
	AnnotatedConnectException: Connection refused
	AnnotatedConnectException: Connection refused
	AnnotatedConnectException: Connection refused -- Will try again in 0.372 s
	ConnectionHandler - Reconnecting after connection was closed
	AnnotatedConnectException: Connection refused
	AnnotatedConnectException: Connection refused
	AnnotatedConnectException: Connection refused -- Will try again in 0.788 s
	ConnectionHandler - Reconnecting after connection was closed
	AnnotatedConnectException: Connection refused
	AnnotatedConnectException: Connection refused
	AnnotatedConnectException: Connection refused -- Will try again in 1.447 s
	ConnectionHandler - Reconnecting after connection was closed
	AnnotatedConnectException: Connection refused
	AnnotatedConnectException: Connection refused
	AnnotatedConnectException: Connection refused -- Will try again in 3.142 s
	ConnectionHandler - Reconnecting after connection was closed
	ConnectionPool - Connected to server
	ConnectionPool - Connected to server
	ClientCnx - Connected through proxy to target broker at localhost:6650
	ProducerImpl - Creating producer on cnx
	ProducerImpl - Created producer on cnx
	ProducerImpl - Re-Sending 7 messages to server

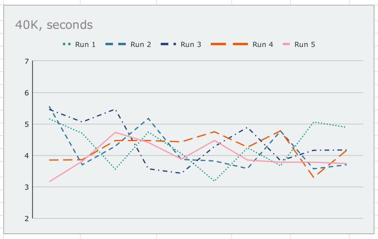
	PulsarPrototypeProducer - 21 sent to server					
	PulsarPrototypeProducer - 22 sent to server					
	PulsarPrototypeProducer - 23 sent to server					
	PulsarPrototypeProducer - 24 sent to server					
	PulsarPrototypeProducer - 25 sent to server					
	PulsarPrototypeProducer - 26 sent to server					
	PulsarPrototypeProducer - 27 sent to server					
	PulsarPrototypeProducer - 28 sent to server					
	PulsarPrototypeProducer - 29 sent to server					
	PulsarPrototypeProducer - 30 sent to server					
	PulsarPrototypeProducer - 31 sent to server					
	PulsarPrototypeProducer - 32 sent to server					
	PulsarPrototypeProducer - 33 sent to server					

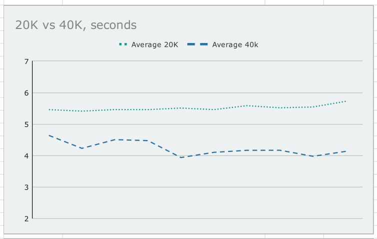
Pulsar Producer 1M raw

2 Data providers sending 10 000 msg/s each		
1 Producer in Ålesund		
1 Broker running in Docker container on NTNU s		
1 Consumer in Haugesund		
Run 1		
Messages	**Seconds**	**Messages per second**
100000	6.43	15554
100000	5.62	17806
100000	5.81	17223
100000	6.13	16310
100000	5.62	17784
100000	5.25	19051
100000	5.95	16818
100000	6.46	15477
100000	6.67	14992
100000	6.39	15644
Run 2		
Messages	**Seconds**	**Messages per second**
100000	5.38	18597
100000	5.53	18083
100000	5.17	19353
100000	5.2	19230
100000	5.31	18828
100000	5.58	17921
100000	5.23	19138
100000	5.25	19043
100000	5.1	19627
100000	5.4	18535
Run 3		
Messages	**Seconds**	**Messages per second**
100000	5.12	19538
100000	5.16	19387
100000	5.16	19376
100000	5.22	19146
100000	5.54	18044
100000	5.53	18092
100000	5.17	19349
100000	5.14	19451
100000	5.25	19043
100000	5.83	17158
Run 4		
Messages	**Seconds**	**Messages per second**
100000	5.19	19252
100000	5.63	17777
100000	5.85	17091
100000	5.25	19040
100000	5.49	18228
100000	5.53	18096
100000	5.36	18660
100000	5.12	19527
100000	5.24	19087
100000	5.31	18835
Run 5		
Messages	**Seconds**	**Messages per second**
100000	5.17	19342
100000	5.11	19573
100000	5.31	18843
100000	5.49	18218
100000	5.58	17934
100000	5.4	18515
100000	6.21	16097
100000	5.61	17828
100000	5.45	18358
100000	5.69	17577

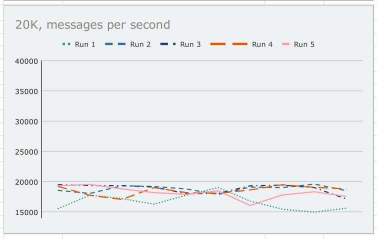
2 Data providers sending 20 000 msg/s each		
1 Producer in Ålesund		
1 Broker running in Docker container on NTNU s		
1 Consumer in Haugesund		
Run 1		
Messages	**Seconds**	**Messages per second**
100000	5.16	19379
100000	4.7	21294
100000	3.56	28129
100000	4.74	21097
100000	4.06	24660
100000	3.18	31446
100000	4.24	23584
100000	3.67	27270
100000	5.06	19766
100000	4.89	20437
Run 2		
Messages	**Seconds**	**Messages per second**
100000	5.56	17975
100000	3.7	27034
100000	4.29	23304
100000	5.17	19353
100000	3.87	25826
100000	3.82	26178
100000	3.58	27940
100000	4.77	20959
100000	3.57	28050
100000	3.7	27056
Run 3		
Messages	**Seconds**	**Messages per second**
100000	5.46	18331
100000	5.06	19762
100000	5.47	18284
100000	3.57	27995
100000	3.43	29154
100000	4.28	23380
100000	4.89	20445
100000	3.83	26089
100000	4.16	24021
100000	4.17	23975
Run 4		
Messages	**Seconds**	**Messages per second**
100000	3.85	26007
100000	3.86	25940
100000	4.47	22391
100000	4.47	22351
100000	4.43	22583
100000	4.75	21065
100000	4.26	23463
100000	4.78	20938
100000	3.3	30312
100000	4.16	24050
Run 5		
Messages	**Seconds**	**Messages per second**
100000	3.16	31635
100000	3.81	26253
100000	4.73	21141
100000	4.41	22696
100000	3.88	25799
100000	4.47	22356
100000	3.85	26001
100000	3.78	26476
100000	3.78	26448
100000	3.74	26773

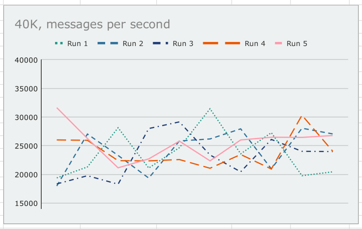




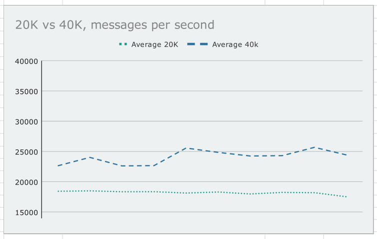


AVG per run 20k	AVG per run 40k
5.458	4.638
5.41	4.226
5.46	4.504
5.458	4.472
5.508	3.934
5.458	4.1
5.584	4.164
5.516	4.166
5.542	3.974
5.724	4.132





AVG per run 20k	AVG per run 40k
18456.6	22665.4
18525.2	24056.6
18377.2	22649.8
18388.8	22698.4
18163.6	25604.4
18335	24885
18012.4	24286.6
18265.2	24346.4
18221.4	25719.4
17549.8	24458.2



2 Data providers sending 10 000 msg/s each		
1 Producer in Ålesund		
1 Broker running in Docker container		
1 Consumer in Haugesund		
Run 1		
Messages	**Seconds**	**Messages per second**
100000	6.4	15622
100000	5.66	17667
100000	5.81	17217
100000	6.12	16350
100000	5.62	17784
100000	5.3	18864
100000	5.94	16823
100000	6.45	15501
100000	6.76	14803
100000	6.37	15703
Run 2		
Messages	**Seconds**	**Messages per second**
100000	5.38	18583
200000	5.51	18142
300000	5.2	19245
400000	5.37	18615
500000	5.14	19447
600000	5.59	17901
700000	5.28	18957
800000	5.65	17714
900000	5.46	18325
1000000	4.62	21635
Run 3		
Messages	**Seconds**	**Messages per second**
100000	5.14	19466
100000	5.16	19394
100000	5.19	19278
100000	5.23	19124
100000	6.22	16084
100000	4.87	20521
100000	5.19	19282
100000	5.18	19312
100000	5.24	19080
100000	5.97	16747
Run 4		
Messages	**Seconds**	**Messages per second**
100000	5.06	19755
100000	5.73	17439
100000	5.82	17176
100000	5.57	17943
100000	5.12	19523
100000	6.17	16196
100000	4.72	21172
100000	5.21	19186
100000	5.22	19164
100000	5.28	18928
Run 5		
Messages	**Seconds**	**Messages per second**

2 Data providers sending 20 000 msg/s each		
1 Producer in Ålesund		
1 Broker running in Docker container		
1 Consumer in Haugesund		
Run 1		
Messages	**Seconds**	**Messages per second**
100000	5.19	19282
100000	4.66	21459
100000	3.59	27894
100000	4.75	21048
100000	4.08	24491
100000	3.13	31918
100000	4.25	23523
100000	3.7	27048
100000	5.08	19669
100000	4.94	20234
Run 2		
Messages	**Seconds**	**Messages per second**
100000	5.48	18234
100000	3.72	26917
100000	4.38	22815
100000	5.28	18925
100000	3.67	27240
100000	3.85	25994
100000	3.59	27855
100000	4.83	20695
100000	3.48	28768
100000	3.81	26239
Run 3		
Messages	**Seconds**	**Messages per second**
100000	5.55	18018
200000	4.97	20140
300000	5.39	18559
400009	3.56	28121
500000	3.47	28826
600000	4.34	23030
700000	4.93	20271
800000	3.72	26874
900001	4.21	23736
1000001	4.17	23992
Run 4		
Messages	**Seconds**	**Messages per second**
100003	3.85	25960
200000	3.95	25303
300004	4.34	23062
400000	4.62	21645
500000	4.38	22857
600000	4.75	21043
700000	4.3	23255
800000	4.7	21272
900000	3.28	30478
1000001	4.18	23952
Run 5		
Messages	**Seconds**	**Messages per second**

Pulsar Consumer 1M raw

100000	5.2	19230		100000	3.22	31065
100000	5.1	19607		200006	3.79	26385
100000	5.35	18695		300007	4.86	20567
100000	5.47	18298		400000	4.35	23004
100000	5.65	17705		500007	4	25018
100000	5.49	18228		600000	4.28	23375
100000	6.17	16207		700003	4.03	24813
100000	5.55	18018		800000	3.61	27670
100000	5.46	18328		900000	3.78	26448
100000	5.92	16891		1000000	3.78	26476

Appendix 4: AutoStreams’ style guide

JavaDoc

Document every class and method, private and public, with JavaDoc. Avoid commenting code within a method. If there is a need to comment code within a method, try to rewrite the method instead.

Example, JavaDoc

This example illustrates the general style of the JavaDoc. Parameter order, spacing, and use of periods should be done the same way in the code as it is in this example.

```
/**
 * This is a description of the class/method.
 *
 * @param1 description of param1.
 * @param2 description of param2.
 * @throws description of exceptions thrown.
 * @return description of return value
 */
Method signature (args) {}
```

Annotations

Do’s

- Use @version and @since on classes (not methods)
- Use @param for every parameter when writing JavaDoc.
- Use @throws when the method throws exceptions.
- Use @return whenever the method returns a value
- Use the same order of annotations as described in the example above

Don'ts

- Use @author

The usage of “var”

Var is not to be used under any circumstance due to the following reasons

- It is difficult to define when and when not to use it, leading to inconsistency
- Not as explicit as using specific types
- Improves readability on GitHub
- More compatible for porting to Java 8 (Var is a Java 10 feature)

Tabs vs spaces

- Indentation of Java files has to be done with 4 spaces.
- The indentation of YAML has to be 2 spaces.

The naming of files and directories

Directories

All directories are lowercase with a hyphen connecting separate words.

Example of a directory name:
kafka-prototype-producer

Java class files

All Java class files are named with CamelCaps. File and class names must match. Naming must be relevant, coherent, descriptive, and grammatically correct.

Example of a Java class name:
DataProducer.java

Git

Branches and Git Flow

Branches shall be structured according to Vincent Driessen's Git Flow, utilizing three main branches:

- Master
- Release
- Develop

Naming a branch

Every branch name is to be written using only lowercase characters. When the name of a branch is composed of multiple words, each word is to be separated with the underscore symbol.

Example of a branch name
branch_name_with_multiple_words

Master branch

The master branch must never be committed to, only merged to from the release branch. The name of the master branch is only “master”.

Can merge into

- None

Can branch off from

- None

Can be pushed to

No

Release branch

Release branches represent new releases of the product. release branches always branch out from develop, never any other branch. The release branch is the final stage before changes are merged into master.

Tagging

Every release has to be tagged with the new version using the command `git tag -a major.minor.patch`

Example of version with major, minor, and patch
1.2.0

Naming

Naming of a release branch has the extra step of adding the major and minor version of the release to the end of the branch name.

Example of name
release-2.1

Can merge into

- Master
- Develop

Can branch from

- Develop

Can be pushed to

No

Develop branch

Develop is the main branch for changes during development. All new features must merge into develop before they can be introduced elsewhere. This centralizes changes to one branch.

Develop merges into Release when the product is approaching a new release.
Develop must never be committed to, only merged with the release, hotfix, and feature branches.

Can merge into

- Master

Can branch off from

- None (initially master)

Can be pushed to

No

Feature Branches

All new work is to be done in a feature branch relating to and named after the feature to be implemented. Feature branches always branch out from develop, never any other branch.

Example of feature branch name:
feature/consumer

Feature branches are always merged into the develop branch.

Can merge into

- Develop

Can branch from

- Develop

Can be pushed to

Yes

Hotfix

Hotfix branches follow the same naming convention as features.

Example of hotfix branch name:
hotfix/infinite_loops

Can merge into

- Develop
- Master

Can branch from

- Master

Can be pushed to

No

Appendix 5: Pre-Project Plan

Tjeneste for Datastrømming
Forprosjektplan

Versjon 1.2

Revisjonshistorie

Dato	Versjon	Beskrivelse	Forfatter
10/01/2022	1.0	Første Utkast	Tomas Klungerbo Olsen Lars Ivar Ramberg
19/01/2022	1.1	Revidert Utkast	Tomas Klungerbo Olsen Lars Ivar Ramberg
28/01/2022	1.2	Ferdigstilt Utkast	Tomas Klungerbo Olsen Lars Ivar Ramberg

Innholdsfortegnelse

1. Mål og rammer

1.1 Orientering

Denne oppgaven ble valgt etter samtale med AutoStore AS. En av gruppemedlemmene er ansatt i AutoStore, og oppgaven ble foreslått etter samtale med ansatte i avdelingen for overvåkning og vedlikehold.

Oppgaven ble valgt fordi den byr på en spennende reell problemstilling, og mulighet til å utforske og implementere moderne teknologier for strømming og behandling av data. I tillegg føler gruppen at de har tilstrekkelig kompetanse til å håndtere en slik oppgave.

1.2 Problemstilling / prosjektbeskrivelse og resultatmål

Målet med oppgaven er å implementere en tjeneste for å strømme data fra AutoStore installasjoner i sanntid. Data fra AutoStore anlegg leveres nå i batcher, der data sendes fra AutoStore anlegg til en sentral server på gitte tidspunkter. Data samles over en hel dag, og all data sendes til den sentrale serveren på et bestemt tidspunkt neste dag.

Relevante problemstillinger vil være utfordringer knyttet til å strømme data over internett. I tillegg til å implementere en slik strømming må løsningen kunne håndtere tap av tilkobling og strømming over svake nettverk. I tillegg må sikkerhet tas i betraktning, da dataene som skal strømmes vil være sensitive. Til slutt må skalering betraktes, da det i dag eksisterer hundrevis av AutoStore anlegg, og flere vil bygges i fremtiden.

Når prosjektet er ferdig skal det minimum være implementert en prototype for datastrømmingen, som skal kunne strømme data fra en AutoStore installasjon til en mottaker.

Hvis prosjektet passerer prototype stadiet vil et større mål være å ha en generell løsning som kan strømme data i sanntid til en mottaker. I forbindelse med denne spesifikke oppgaven vil det bety at flere anlegg vil kunne strømme sin data, og flere mottakere kan motta data fra en eller flere anlegg.

1.3 Effektmål

For gruppen vil følgende mål være relevante:

- Implementere en strømmeløsning som kan strømme faktiske data fra installasjoner
- Lære om og implementere feil-strategier for strømmetjenester (hvordan håndtere tap av tilkobling)
- Utforme en strømmeløsning som er generell nok til å strømme andre data enn kun AutoStore data
- Lære mer om skytjenester og moderne strømmeteknologier

For bedriften vil følgende gevinster og resultater være relevante:

- Benytte seg av strømmetjenesten for å følge opp AutoStore anlegg i sanntid
- Benytte seg av sanntidsinformasjon for å foreta forebyggende vedlikehold
- Bygge videre på strømmetjesten etter prosjektet for å utvide kapasiteten
- Kombinere strømmet sanntidsdata med eksisterende historisk data for å danne et helhetlig bilde av AutoStore anlegg, og hendelser ved anlegg.

1.4 Rammer

Det vil være nødvendig for alle gruppemedlemmer å ha en datamaskin og internett-tilkobling.

Minst en av gruppemedlemmene, helst begge, må ha tilgang til AutoStore sitt interne nettverk for få tilgang til data. Alternativt kan det avtales bruk av dummy-data, gitt at dataene kan modellere use-caset.

For å håndtere datastrømmingen vil det være behov for skytjenester. Det forutsettes av prosjektgruppen at AutoStore kan tilby de nødvendige skytjenestene og rammeverkene for å implementere den tiltenkte strømme-løsningen. Gruppen har avtalt med AutoStore at kostnader for skytjenester kan dekkes underveis gitt at gruppen kommuniserer behov.

2. Organisering

Prosjektgruppen

Tomas Klungerbo Olsen

Student, utvikler for prosjektet, Scrum Master

Lars Ivar Ramberg

Student, utvikler for prosjektet, Product Owner

Veiledere

Saleh Abdel-Afou Alaliyat

Førsteamanuensis, Instituttet for IKT og Realfag ved NTNU Ålesund

Stakeholders/Eksterne oppdragsgivere

Asle Olsen Gaasø

Manager, Monitoring and Maintenance, R&D, AutoStore AS

3. Gjennomføring

3.1. Hovedaktiviteter

Hva	Hvem	Hvorfor	Hvordan	Når	Forutsetninger
Planlegging	Tomas Klungerbo Olsen Lars Ivar Ramberg	Identifisere behov for prosjektet. Planlegge prosjektet	Analyse, samtale, drøfting	Januar, frem til 28.01.2022	Tilstedeværelse, tidlig start for å komme i mål

Skriving av Møtereferater	Dokumentansvarlig/sekretær	Dokumentasjon	Kontinuerlig dokumentasjon. Lagres i Confluence eller Google Drive	Under og etter møter	Tilgang til Confluence
Research	Tomas Klungerbo Olsen Lars Ivar ramberg	Identifisere relevante teknologier basert på behov.	Internettsøk, Lesing av artikler og dokumentasjon. Rapportering av funn	Research-fase 24.01.2022 - 04.02.2022,Kontinuerlig	Oversikt over behov for prosjektet
Utvikling av løsning	Tomas Klungerbo Olsen Lars Ivar Ramberg	Møte prosjektmål og levere produkt til stakeholder	Utvikling, skriving av kode	Etter planlegging frem til innlevering 28.01.2022 - 20.05.2022	Fullført planleggingsfase.
Innlevering av rapport til veileder	Tomas Klungerbo Olsen Lars Ivar Ramberg	Få tilbakemelding på rapport	Ha klart et utkast innen dato	Senest 06.05.2022	Utkast må være klart og leveres.
Møter med stakeholder og veiledning	Tomas Klungerbo Olsen Lars Ivar Ramberg Veileder og Stakeholder	Få tilbakemelding på utvikling	Digitale møter	Annenhver uke med start i uke 2	Møteinnkallinger må sendes ut. Tidsrom må passe for alle deltakere
Innlevering av rapport	Tomas Klungerbo Olsen Lars Ivar Ramberg	Fullføre prosjektet	Innlevering i Inspera	20.05.2022	Prosjektet må være klart, og rapport må være ferdigstilt

Innlevering av kode	Tomas Klungerbo Olsen Lars Ivar Ramberg	Fullføre prosjektet, Presentere resultat til oppdragsgiver	Jobbe målrettet og strukturert med prosjektet	Til oppdragsgiver: 13.05.2022 Til NTNU: 20.05.2022	Prosjektet må være klart, og rapport må være ferdigstilt

3.2. Milepæler

Dato	Hendelse
28.01.2022	Innlevering av forprosjektplan
11.02.2022	Innlevering av research-materiale til AutoStore
01.04.2022	Innlevering av prototyper til AutoStore
06.05.2022	Innlevering av rapport til veileder
13.05.2022	Innlevering av kode til AutoStore
18.05.2022	Innlevering av poster
20.05.2022	Innlevering av rapport

4. Oppfølging og kvalitetssikring

4.1 Kvalitetssikring

For å sikre kvalitet vil det brukes en kombinasjon av intern kvalitetssikring, og ekstern veiledning. Ekstern forstås i denne forstand som utenfor studentgruppen.

Gruppen har definert prosesser for kvalitetssikring i sin arbeidsavtale. I avtalen står det at alt arbeid, både kode og dokumenter, skal kvalitetssikres og godkjennes av gruppemedlemmet som ikke er forfatter. I praksis betyr dette at alt arbeid skal sees av begge medlemmer før det blir en del av endelig løsning eller rapport.

4.2 Rapportering

Rapportering av fremgang skal skje annenhver uke fra starten av prosjektet til slutten av prosjektet. Rapportering skjer til veileder og kontaktperson hos AutoStore AS (Se seksjon 2).

Rapportering vil skje under møter annenhver uke. Her vil fremgang presenteres, og veien videre skal drøftes med både veileder og kontaktperson.

Underveis skal arbeidet også dokumenteres i form av screenshots. Denne rapporteringen vil være en del av den endelige rapporten som leveres i Inspera ved slutten av prosjektet.

Arbeidet skal til slutt presenteres i form av en rapport. Denne rapporten skal leveres inn for vurdering og vil representere slutten av prosjektet.

5. Risikovurdering

De følgende risikoene har blitt identifisert i forarbeidet til prosjektet. Hver risiko har blitt analysert for sannsynlighet og konsekvens. Basert på sannsynlighet og konsekvens har hver risiko blitt gitt en score basert på systemet illustrert i Tabell 1.

	UFARLIG	**MINDRE ALVORLIG**	**ALVORLIG**	**SVÆRT ALVORLIG**	**KATASTROFALT**
SVÆRT SANNSYNLIG	5	10	15	20	25
MEGET SANNSYNLIG	4	8	12	16	20
SANNSYNLIG	3	6	9	12	15
MINDRE SANNSYNLIG	2	4	6	8	10
USANNSYNLIG	1	2	3	4	5

Tabell 1: Oversikt over poengsystem for risikovurdering

I Tabell 1 representerer grønne felter en akseptabel risiko. Oransje felter kan aksepteres dersom det finnes tilfredsstillende tiltak. Røde felter representerer risiko som må elimineres, eller som må støttes med flere tiltak.

Hendelse	Sannsynlighet	Konsekvens	Poengsum	Tiltak
Frafall av et gruppemedlem	Usannsynlig	Katastrofalt	5	Kontinuerlig oppfølging. Arbeidskontrakt.
Teknologisk svikt hos et gruppemedlem	Sannsynlig	Alvorlig	9	Kontinuerlig oppdatering og vedlikehold av utstyr. Ta backups.
Teknologisk svikt hos flere gruppemedlemmer	Mindre Sannsynlig	Katastrofalt	10	Kontinuerlig oppdatering og vedlikehold av utstyr. Ta backups.
Uoverkommelig oppgave-kompleksitet	Usannsynlig	Katastrofalt	5	Skaler oppgave for å tilpasse ferdigheter.
Manglende ressurser fra arbeidsgiver	Usannsynlig	Svært Alvorlig	4	Kommuniser behov til oppdragsgiver tidlig i prosessen. Meld i fra dersom noe mangler så tidlig som mulig.
For sen start på rapport	Mindre Sannsynlig	Alvorlig	6	Start så tidlig som mulig med rapport-skriving.

Langvarig sykdom/ Midlertidig frafall	Sannsynlig	Svært Alvorlig	12	Smittevernstiltak. Følg gjeldende råd og anbefalinger vedrørende COVID-situasjonen.

Appendix 6: System documentation

Service for data streaming

System Documentation

<v1.2>

REVISION HISTORY

Date	Version	Description	Author
29/04/2022	<1.0>	Initial Draft	Lars Ivar Ramberg Tomas Klungerbo Olsen
03/05/2022	<1.1>	Completed the project structure section Documentation section complete	Lars Ivar Ramberg Tomas Klungerbo Olsen
11/05/2022	<1.2>	Update Pulsar implementation directory structure	Lars Ivar Ramberg Tomas Klungerbo Olsen

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

This document serves as the system documentation for the project “Service for Data streaming” at NTNU (Norges Teknisk Naturvitenskapelige Universitet) in Ålesund.

The purpose of the document is to present the structure and features of the solution, and to serve as a guide for installation and use. After reading the document, the reader should have a clear understanding of how the solution is structured, what it can offer to the user in terms of functionality, and how the solution can be installed and used to send and receive data in real-time.

2 Architecture

The system and its architecture are part of a larger intended system. The system represented by this document is solely responsible for transmitting messages and is therefore difficult to present architecturally on its own. For this reason, the architecture is presented here both on its own (Figure 1) and as part of a larger system (Figure 2).

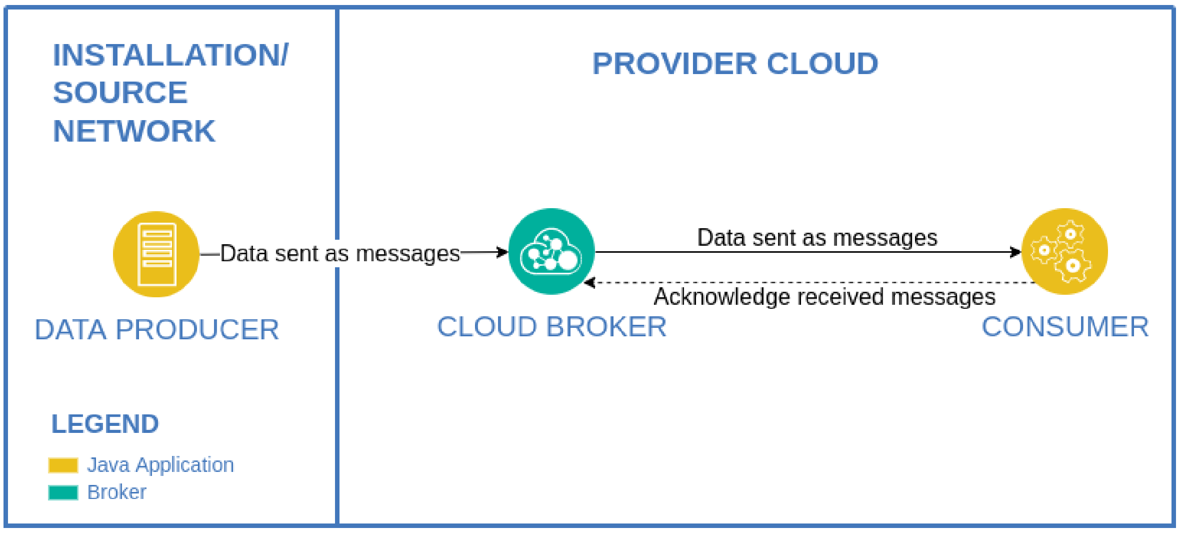


Figure 1 Focused overview of the system.

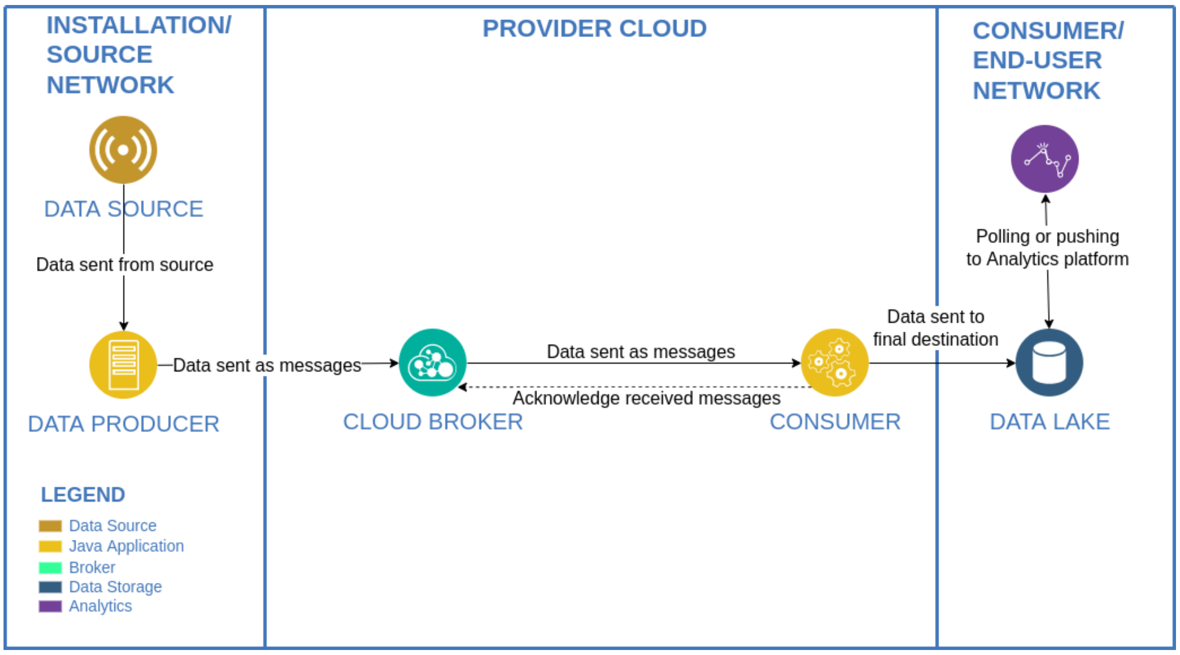


Figure 2 Holistic overview of the system. Illustration places the system in a larger data pipeline.

2.1 Project structure

Service for data streaming is a collection of multiple repositories and projects. Every repository is owned by the organization AutoStreams, which was created to group these related projects logically. One repository was created for the Pulsar prototype, one for the Kafka prototype, and one for the final implementation of the Pulsar solution. AutoStreams also holds some utility repositories shared among the previously mentioned repositories. One utility repository exists for the prototypes and one for the final implementation. The last repository that AutoStreams has is a utility repository containing the check style configuration used by the team.

Java project repositories are organized according to the Maven standard directory layout (Apache Maven Project, 2022). The files and directories that all AutoStreams java project repositories have implemented are as follows:

Name	Type	Usage
pom.xml	file	Defines the project and its dependencies.
src/main/java	Directory	Contains all source files for the project
src/main/resources	Directory	Contains the resource files for the project, which for this project are configuration files

Table 1 List of Maven layout directories and files used by AutoStreams repositories.

Files unrelated to or not strictly bound to the Maven standard directory layout are:

Name	Usage
LICENSE	Defines the license for the project
README.md	Contains information about the submodule project's purpose and guidance for installation
Dockerfile	A YAML file that defines a Docker image of the project. The Docker image can be spun up as Docker containers.

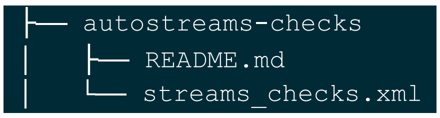
Table 2 List of files used by the AutoStreams repositories not strictly bound to the Maven standard layout structure.

2.1.1 Autostreams checks

Repository name: autostreams-checks
Link: https://github.com/AutoStreams/autostreams-checks

The autostreams-checks repository is a modification of the google Checkstyle configuration (Google checkstyle, 2022). The modification uses four spaces as opposed to the usual two spaces. All AutoStreams's repositories have a dependency on this repository, so new developers can easily find the Checkstyle configuration.

Directory structure:



```
├── autostreams-checks
│   ├── README.md
│   └── streams_checks.xml
```

Figure 3 Directory structure for the AutoStreams/autostreams-checks repository.

2.1.2 Prototype utilities

Repository name: prototype-utils
Link: https://github.com/AutoStreams/prototype-utils

The prototype-utils repository contains utilities shared among projects within other repositories.

Directory structure:

```
├── prototype-utils
│   ├── LICENSE
│   ├── pom.xml
│   ├── README.md
│   └── src
│       └── main
│           ├── java
│           │   └── com
│           │       └── autostreams
│           │           └── utils
│           │               ├── dataprovider
│           │               │   ├── DataProducerHandler.java
│           │               │   ├── DataProducerInitializer.java
│           │               │   └── DataProvider.java
│           │               ├── datareceiver
│           │               │   ├── DataReceiverHandler.java
│           │               │   ├── DataReceiverInitializer.java
│           │               │   ├── DataReceiver.java
│           │               │   └── StreamsServer.java
│           │               └── fileutils
│           │                   └── FileUtils.java
│           └── resources
│               └── simplelogger.properties
```

Figure 4 Directory structure for the AutoStreams/prototype-utils repository.

2.1.3 Utilities

Repository name: utils
Link: https://github.com/AutoStreams/utils

The utils repository contains utilities shared among projects within the final implementation. The utils repository is a divergence of the prototype-utils repository, allowing further development without affecting the prototypes.

Directory structure:

```
├── utils
│   ├── LICENSE
│   ├── pom.xml
│   ├── README.md
│   └── src
│         └── main
│             ├── java
│             │   └── com
│             │       └── autostreams
│             │           └── utils
│             │               ├── dataprovider
│             │               │   ├── DataProducerHandler.java
│             │               │   ├── DataProducerInitializer.java
│             │               │   └── DataProvider.java
│             │               ├── datareceiver
│             │               │   ├── DataReceiverHandler.java
│             │               │   ├── DataReceiverInitializer.java
│             │               │   ├── DataReceiver.java
│             │               │   └── StreamsServer.java
│             │               └── fileutils
│             │                   └── FileUtils.java
│             └── resources
│                 └── simplelogger.properties
└── utils.iml
```

Figure 5 Directory structure for the AutoStreams/utils repository.

2.1.4 Kafka prototype

Repository name: prototype-kafka
Link: https://github.com/AutoStreams/prototype-kafka

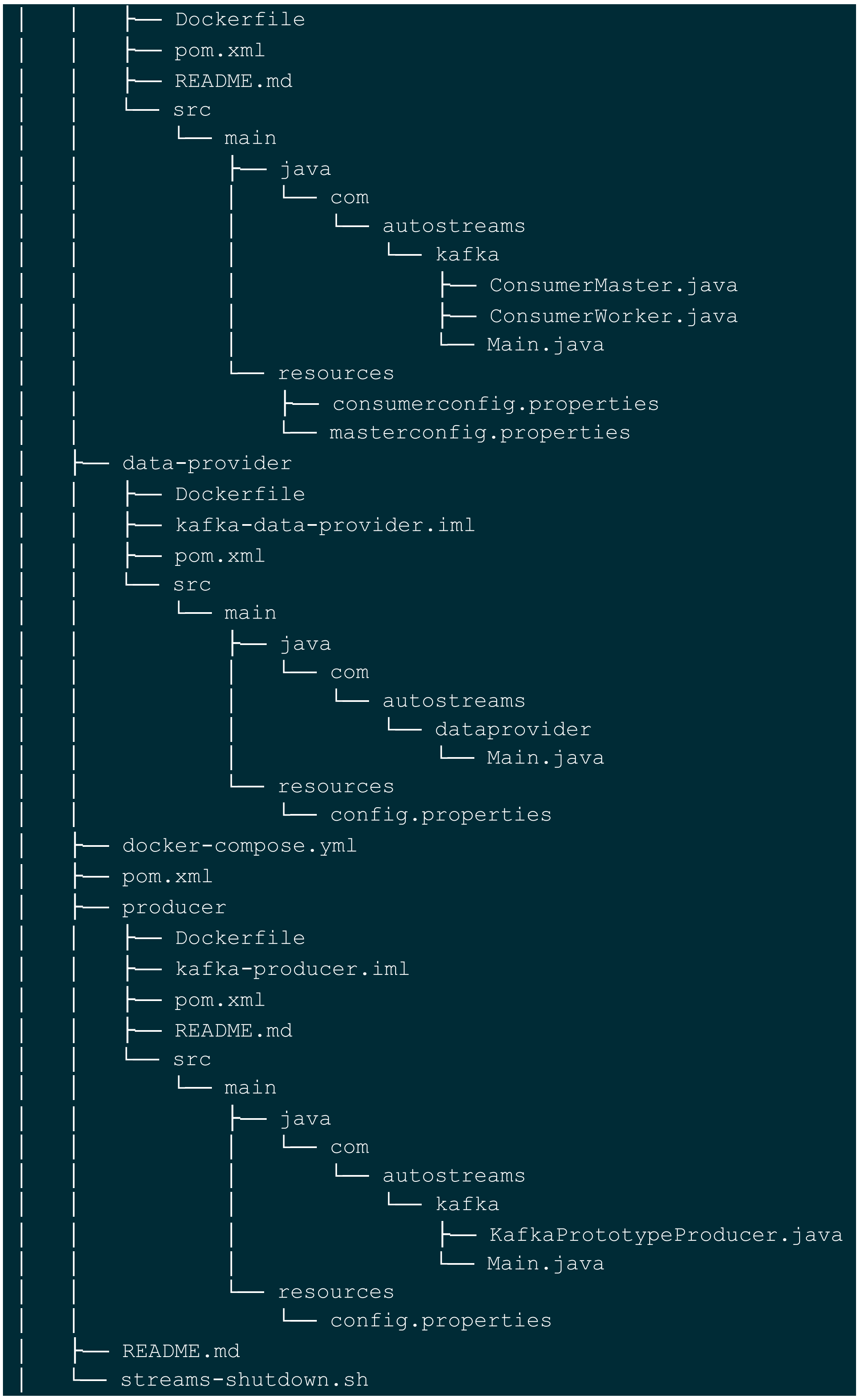
The prototype-kafka repository holds the implementation of the Kafka solution created during the prototype stage. The root directory of this repository contains a parent pom.xml file with sub-project modules. Each sub-project module follows the Maven standard directory layout and is contained in its own unique directory. Furthermore, the root directory contains a docker-compose file to initiate the complete solution locally for testing and a shutdown script to gracefully shut down the consumer and producer while running in a Docker container.

Submodule projects which the Kafka prototype contains are:

- autostreams-checks - checkstyle configuration
- consumer - the Kafka stream consumer
- producer - the Kafka stream producer
- data-provider - creates and sends data to the producer
- broker - the Kafka stream broker

Directory structure:

```
├── prototype-kafka
│   ├── autostreams-checks
│   │   ├── README.md
│   │   └── streams_checks.xml
│   ├── broker
│   │   └── docker-compose.yml
│   ├── consumer
```



```
│   │   ├── Dockerfile
│   │   ├── pom.xml
│   │   ├── README.md
│   │   └── src
│   │       └── main
│   │           ├── java
│   │           │   └── com
│   │           │       └── autostreams
│   │           │           └── kafka
│   │           │               ├── ConsumerMaster.java
│   │           │               ├── ConsumerWorker.java
│   │           │               └── Main.java
│   │           └── resources
│   │               ├── consumerconfig.properties
│   │               └── masterconfig.properties
│   ├── data-provider
│   │   ├── Dockerfile
│   │   ├── kafka-data-provider.iml
│   │   ├── pom.xml
│   │   └── src
│   │       └── main
│   │           ├── java
│   │           │   └── com
│   │           │       └── autostreams
│   │           │           └── dataprovider
│   │           │               └── Main.java
│   │           └── resources
│   │               └── config.properties
│   ├── docker-compose.yml
│   ├── pom.xml
│   ├── producer
│   │   ├── Dockerfile
│   │   ├── kafka-producer.iml
│   │   ├── pom.xml
│   │   ├── README.md
│   │   └── src
│   │       └── main
│   │           ├── java
│   │           │   └── com
│   │           │       └── autostreams
│   │           │           └── kafka
│   │           │               ├── KafkaPrototypeProducer.java
│   │           │               └── Main.java
│   │           └── resources
│   │               └── config.properties
│   ├── README.md
│   └── streams-shutdown.sh
```

Figure 6 Directory structure for the AutoStreams/prototype-kafka repository.

2.1.5 Pulsar prototype

Repository name: prototype-pulsar
Link: https://github.com/AutoStreams/prototype-pulsar

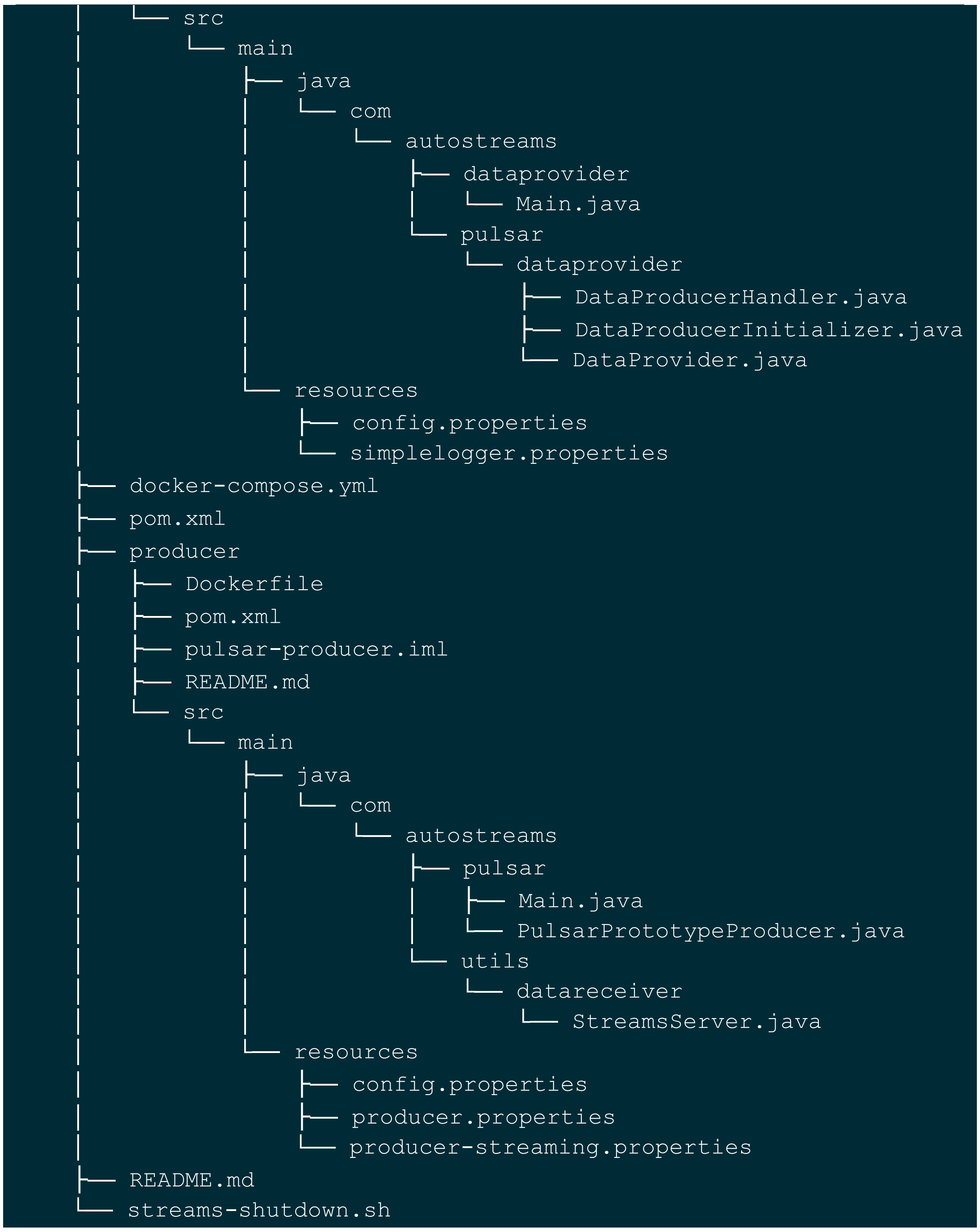
The prototype-pulsar repository holds the implementation of the Pulsar solution created during the prototype stage. Similar to the Kafka prototype, the root directory of this repository contains a parent pom.xml file with sub-project modules. Each sub-project module has its directory following the Maven standard directory layout. The root directory also contains a docker-compose file to initiate the complete solution locally for testing and a shutdown script to gracefully shut down the consumer and producer while running in a Docker container.

Submodule projects which the Kafka prototype contains are:

- autostreams-checks - checkstyle configuration
- consumer - the Pulsar stream consumer
- producer - the Pulsar stream producer
- data-provider - creates and sends data to the producer
- broker - the Pulsar stream broker

Directory structure:

```
└── prototype-pulsar
    ├── autostreams-checks
    │   ├── README.md
    │   └── streams_checks.xml
    ├── consumer
    │   ├── Dockerfile
    │   ├── pom.xml
    │   └── src
    │       └── main
    │           ├── java
    │           │   └── com
    │           │       └── autostreams
    │           │           ├── pulsar
    │           │           │   ├── ConsumerMaster.java
    │           │           │   ├── ConsumerWorker.java
    │           │           │   └── Main.java
    │           │           └── utils
    │           │               └── fileutils
    │           │                   └── FileUtils.java
    │           └── resources
    │               ├── consumerconfig.properties
    │               └── masterconfig.properties
    ├── data-provider
    │   ├── data-provider.iml
    │   ├── Dockerfile
    │   ├── pom.xml
    │   ├── pulsar-data-provider.iml
```



```
│   └── src
│       └── main
│           ├── java
│           │   └── com
│           │       └── autostreams
│           │           ├── dataprovider
│           │           │   └── Main.java
│           │           └── pulsar
│           │               └── dataprovider
│           │                   ├── DataProducerHandler.java
│           │                   ├── DataProducerInitializer.java
│           │                   └── DataProvider.java
│           └── resources
│               ├── config.properties
│               └── simplelogger.properties
├── docker-compose.yml
├── pom.xml
├── producer
│   ├── Dockerfile
│   ├── pom.xml
│   ├── pulsar-producer.iml
│   ├── README.md
│   └── src
│       └── main
│           ├── java
│           │   └── com
│           │       └── autostreams
│           │           ├── pulsar
│           │           │   ├── Main.java
│           │           │   └── PulsarPrototypeProducer.java
│           │           └── utils
│           │               └── datareceiver
│           │                   └── StreamsServer.java
│           └── resources
│               ├── config.properties
│               ├── producer.properties
│               └── producer-streaming.properties
├── README.md
└── streams-shutdown.sh
```

Figure 7 Directory structure for the AutoStreams/prototype-pulsar repository.

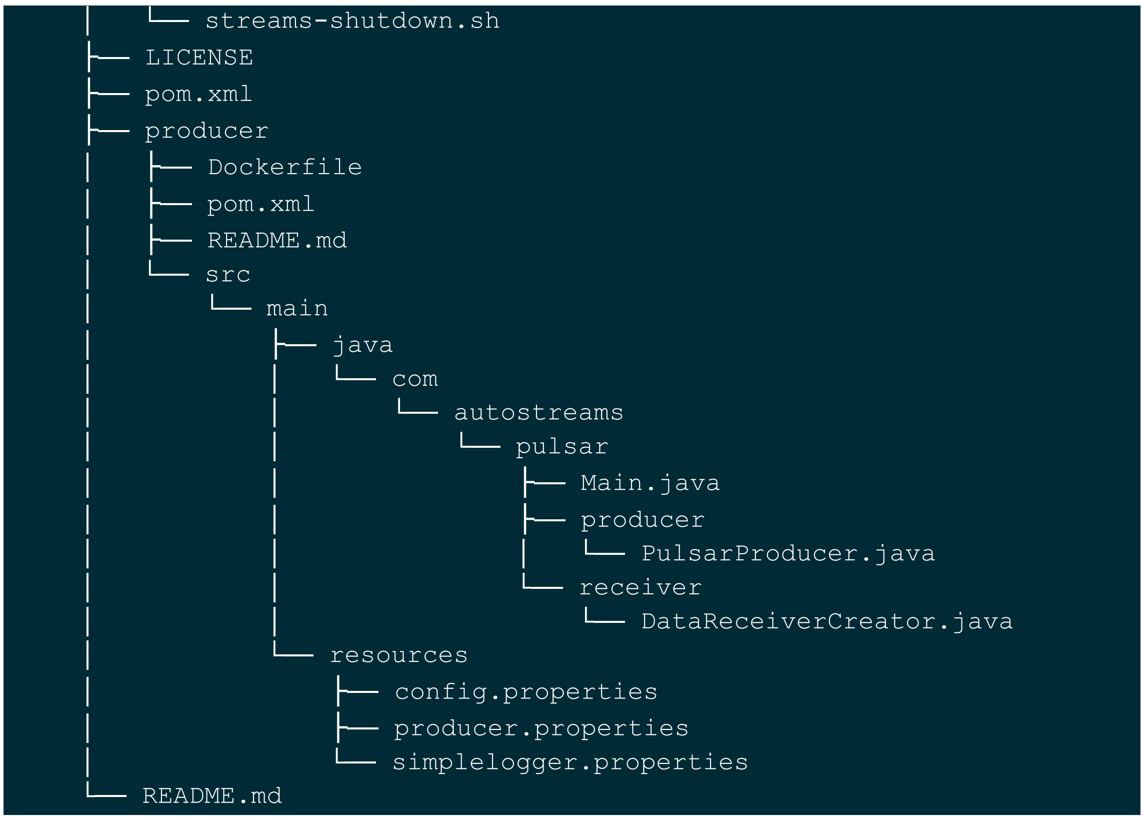
2.1.6 Final implementation - Pulsar

Repository name: pulsar-implementation
Link: https://github.com/AutoStreams/pulsar-implementation

This repository holds the final implementation of the solution. The structure is similar to the Pulsar prototype repository (see section 2.1.5 Pulsar prototype) but with additional features and improvements. There is also an additional directory called example that contains a Docker Compose file that can be spun up to test the complete system locally.

Directory structure:

```
└── pulsar-implementation
    ├── autostreams-checks
    │   ├── README.md
    │   └── streams_checks.xml
    ├── consumer
    │   ├── Dockerfile
    │   ├── pom.xml
    │   ├── README.md
    │   └── src
    │       └── main
    │           ├── java
    │           │   └── com
    │           │       └── autostreams
    │           │           └── pulsar
    │           │               ├── ConsumerMaster.java
    │           │               ├── ConsumerPropertyLoader.java
    │           │               ├── ConsumerWorker.java
    │           │               └── Main.java
    │           └── resources
    │               ├── consumerconfig.properties
    │               ├── masterconfig.properties
    │               └── simplelogger.properties
    ├── example
    │   ├── data-provider
    │   │   ├── Dockerfile
    │   │   ├── pom.xml
    │   │   ├── README.md
    │   │   └── src
    │   │       └── main
    │   │           ├── java
    │   │           │   └── com
    │   │           │       └── autostreams
    │   │           │           └── pulsar
    │   │           │               └── dataprovider
    │   │           │                   ├── DataProducerHandler.java
    │   │           │                   ├── DataProducerInitializer.j..
    │   │           │                   ├── DataProvider.java
    │   │           │                   └── Main.java
    │   │           └── resources
    │   │               ├── config.properties
    │   │               └── simplelogger.properties
    │   ├── docker-compose.yml
    │   ├── README.md
```



```
│   └── streams-shutdown.sh
├── LICENSE
├── pom.xml
├── producer
│   ├── Dockerfile
│   ├── pom.xml
│   ├── README.md
│   └── src
│       └── main
│           ├── java
│           │   └── com
│           │       └── autostreams
│           │           └── pulsar
│           │               ├── Main.java
│           │               ├── producer
│           │               │   └── PulsarProducer.java
│           │               └── receiver
│           │                   └── DataReceiverCreator.java
│           └── resources
│               ├── config.properties
│               ├── producer.properties
│               └── simplelogger.properties
└── README.md
```

Figure 8 Directory structure for the AutoStreams/pulsar-implementation repository.

2.2 CLASS DIAGRAM

As all the three projects (prototypes and final implementation) share similar structure, only one class diagram is presented to avoid redundancy. The class diagram chosen was the Pulsar prototype.

2.2.1 Consumer

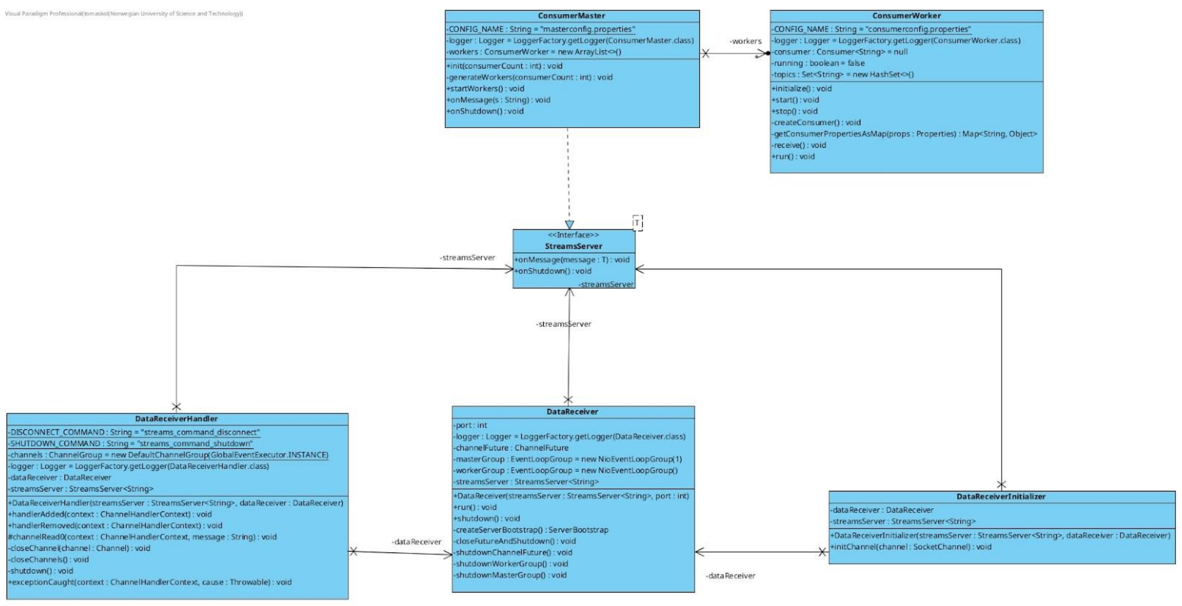

Visual Paradigm Professional(tomaskol(Norwegian University of Science and Technology))
ConsumerMaster
-CONFIG_NAME : String = "masterconfig.properties"
-logger : Logger = LoggerFactory.getLogger(ConsumerMaster.class)
-workers : ConsumerWorker = new ArrayList<>()
+init(consumerCount : int) : void
-generateWorkers(consumerCount : int) : void
+startWorkers() : void
+onMessage(s : String) : void
+onShutdown() : void
-workers
ConsumerWorker
-CONFIG_NAME : String = "consumerconfig.properties"
-logger : Logger = LoggerFactory.getLogger(ConsumerWorker.class)
-consumer : Consumer<String> = null
-running : boolean = false
-topics : Set<String> = new HashSet<>()
+initialize() : void
+start() : void
+stop() : void
-createConsumer() : void
-getConsumerPropertiesAsMap(props : Properties) : Map<String, Object>
-receive() : void
+run() : void
T
<<Interface>>
StreamsServer
+onMessage(message : T) : void
+onShutdown() : void
-streamsServer
-streamsServer
-streamsServer
DataReceiverHandler
-DISCONNECT_COMMAND : String = "streams_command_disconnect"
-SHUTDOWN_COMMAND : String = "streams_command_shutdown"
-channels : ChannelGroup = new DefaultChannelGroup(GlobalEventExecutor.INSTANCE)
-logger : Logger = LoggerFactory.getLogger(DataReceiverHandler.class)
-dataReceiver : DataReceiver
-streamsServer : StreamsServer<String>
+DataReceiverHandler(streamsServer : StreamsServer<String>, dataReceiver : DataReceiver)
+handlerAdded(context : ChannelHandlerContext) : void
+handlerRemoved(context : ChannelHandlerContext) : void
#channelRead0(context : ChannelHandlerContext, message : String) : void
-closeChannel(channel : Channel) : void
-closeChannels() : void
-shutdown() : void
+exceptionCaught(context : ChannelHandlerContext, cause : Throwable) : void
-dataReceiver
DataReceiver
-port : int
-logger : Logger = LoggerFactory.getLogger(DataReceiver.class)
-channelFuture : ChannelFuture
-masterGroup : EventLoopGroup = new NioEventLoopGroup(1)
-workerGroup : EventLoopGroup = new NioEventLoopGroup()
-streamsServer : StreamsServer<String>
+DataReceiver(streamsServer : StreamsServer<String>, port : int)
+run() : void
+shutdown() : void
-createServerBootstrap() : ServerBootstrap
-closeFutureAndShutdown() : void
-shutdownChannelFuture() : void
-shutdownWorkerGroup() : void
-shutdownMasterGroup() : void
-dataReceiver
DataReceiverInitializer
-dataReceiver : DataReceiver
-streamsServer : StreamsServer<String>
+DataReceiverInitializer(streamsServer : StreamsServer<String>, dataReceiver : DataReceiver)
+initChannel(channel : SocketChannel) : void

Figure 9 Class diagram for the consumer Java client.

The consumer is implemented with the Java Pulsar client API (Pulsar, 2022). It implements the StreamsServer interface, which is a server implementation that uses Netty (Netty project, 2022). The consumer uses the server implementation to listen for a shutdown command from localhost. The consumer itself is implemented with the master-worker pattern (Java Design Patterns, n.d.). This pattern allows the consumer master to delegate work amongst multiple consumer workers.

2.2.2 Producer

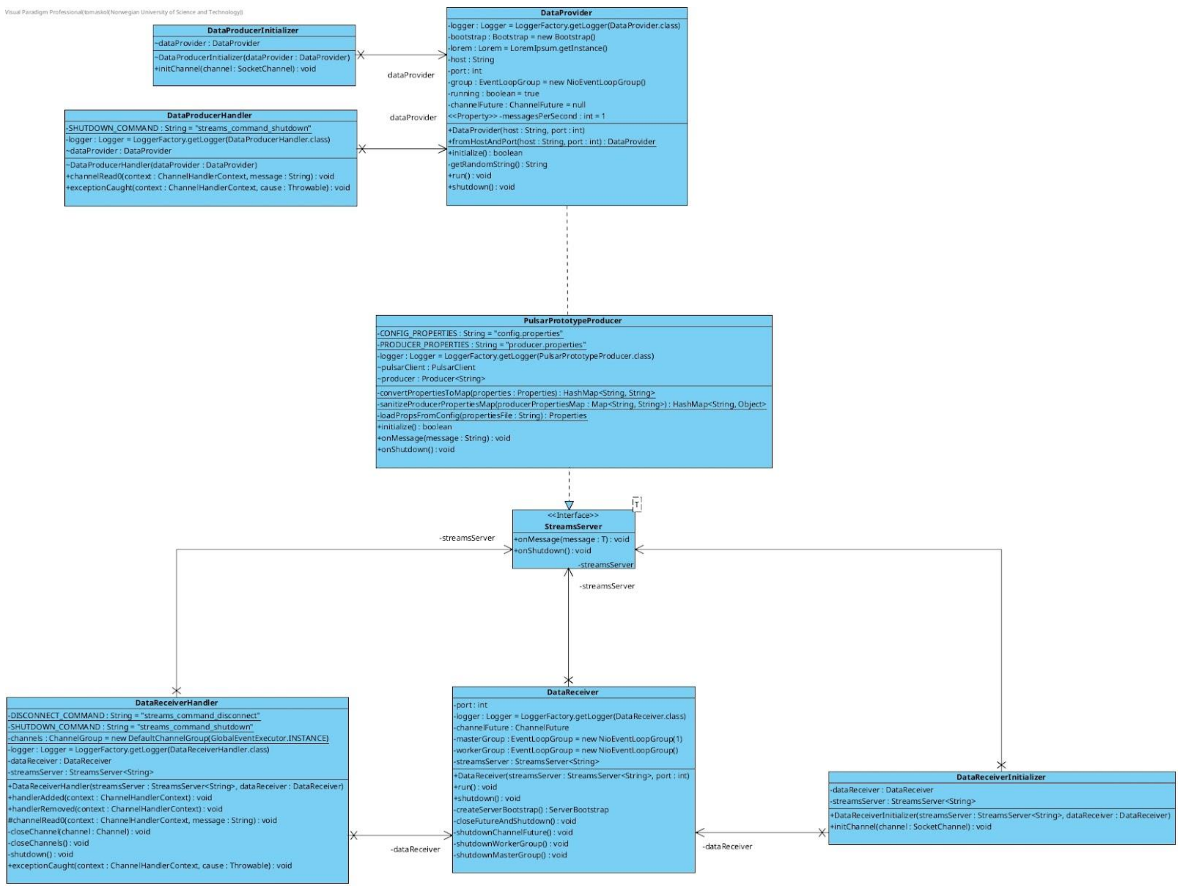

Visual Paradigm Professional(tomaskol(Norwegian University of Science and Technology))
DataProducerInitializer
~dataProvider : DataProvider
~DataProducerInitializer(dataProvider : DataProvider)
+initChannel(channel : SocketChannel) : void
dataProvider
DataProducerHandler
-SHUTDOWN_COMMAND : String = "streams_command_shutdown"
-logger : Logger = LoggerFactory.getLogger(DataProducerHandler.class)
~dataProvider : DataProvider
~DataProducerHandler(dataProvider : DataProvider)
+channelRead0(context : ChannelHandlerContext, message : String) : void
+exceptionCaught(context : ChannelHandlerContext, cause : Throwable) : void
dataProvider
DataProvider
-logger : Logger = LoggerFactory.getLogger(DataProvider.class)
-bootstrap : Bootstrap = new Bootstrap()
-lorem : Lorem = LoremIpsum.getInstance()
-host : String
-port : int
-group : EventLoopGroup = new NioEventLoopGroup()
-running : boolean = true
-channelFuture : ChannelFuture = null
<<Property>> -messagesPerSecond : int = 1
+DataProvider(host : String, port : int)
+fromHostAndPort(host : String, port : int) : DataProvider
+initialize() : boolean
-getRandomString() : String
+run() : void
+shutdown() : void
PulsarPrototypeProducer
-CONFIG_PROPERTIES : String = "config.properties"
-PRODUCER_PROPERTIES : String = "producer.properties"
-logger : Logger = LoggerFactory.getLogger(PulsarPrototypeProducer.class)
~pulsarClient : PulsarClient
~producer : Producer<String>
-convertPropertiesToMap(properties : Properties) : HashMap<String, String>
-sanitizeProducerPropertiesMap(producerPropertiesMap : Map<String, String>) : HashMap<String, Object>
-loadPropsFromConfig(propertiesFile : String) : Properties
+initialize() : boolean
+onMessage(message : String) : void
+onShutdown() : void
T
<<Interface>>
StreamsServer
+onMessage(message : T) : void
+onShutdown() : void
-streamsServer
-streamsServer
-streamsServer
DataReceiverHandler
-DISCONNECT_COMMAND : String = "streams_command_disconnect"
-SHUTDOWN_COMMAND : String = "streams_command_shutdown"
-channels : ChannelGroup = new DefaultChannelGroup(GlobalEventExecutor.INSTANCE)
-logger : Logger = LoggerFactory.getLogger(DataReceiverHandler.class)
-dataReceiver : DataReceiver
-streamsServer : StreamsServer<String>
+DataReceiverHandler(streamsServer : StreamsServer<String>, dataReceiver : DataReceiver)
+handlerAdded(context : ChannelHandlerContext) : void
+handlerRemoved(context : ChannelHandlerContext) : void
#channelRead0(context : ChannelHandlerContext, message : String) : void
-closeChannel(channel : Channel) : void
-closeChannels() : void
-shutdown() : void
+exceptionCaught(context : ChannelHandlerContext, cause : Throwable) : void
-dataReceiver
DataReceiver
-port : int
-logger : Logger = LoggerFactory.getLogger(DataReceiver.class)
-channelFuture : ChannelFuture
-masterGroup : EventLoopGroup = new NioEventLoopGroup(1)
-workerGroup : EventLoopGroup = new NioEventLoopGroup()
-streamsServer : StreamsServer<String>
+DataReceiver(streamsServer : StreamsServer<String>, port : int)
+run() : void
+shutdown() : void
-createServerBootstrap() : ServerBootstrap
-closeFutureAndShutdown() : void
-shutdownChannelFuture() : void
-shutdownWorkerGroup() : void
-shutdownMasterGroup() : void
-dataReceiver
DataReceiverInitializer
-dataReceiver : DataReceiver
-streamsServer : StreamsServer<String>
+DataReceiverInitializer(streamsServer : StreamsServer<String>, dataReceiver : DataReceiver)
+initChannel(channel : SocketChannel) : void

Figure 10 Class diagram for the producer Java client.

Like the consumer, the producer is implemented with the Java Pulsar client API (Pulsar, 2022) and implements the StreamsServer interface as well. The producer uses the server implementation to listen for messages to forward to the broker and a shutdown command from localhost.

3 INSTALLATION AND EXECUTION

This part of the document presents on a component-by-component basis how to configure, install, and run each part of the system. The installation process is described in detail for each component, including necessary prerequisites. Execution is subsequently described. After reading this part of the document, the reader should be able to install and run a complete system capable of sending and receiving data.

Installation and execution of producer and consumer is possible in two ways:

- By using docker images built from Dockerfiles
- By running jar-files built using Maven
- By building and running through an IDE

All three approaches are explained so they are available to readers of this document. Prerequisites are listed in the "Prerequisites" subchapter for each approach, as dependencies will vary based on approach.

The broker is the only non-java part of the system and can also be run in multiple ways. The method presented in this document is taken from the "Run Pulsar Locally" tutorial described in the official Pulsar documentation (Pulsar, n.d.).

To get a fully running system, each part of it must be installed on different computers that can communicate with each other. The intended use is for these computers to be located in different geographical locations with one of them being a broker server.

If desired, it is possible to run everything locally by either using Docker, Docker Compose, or the Java solution. This is recommended if the intention is to test the system, and not to send data over the internet. The Docker-Compose approach is described at the end of this chapter.

3.1 Building and Running Docker Images

Prerequisites

- Docker must be installed on the system to run the Dockerized version of the system

3.1.1 Producer

Installation and Configuration

- Configure the connection URL for the Broker in the Dockerfile belonging to the Producer using the PULSAR_BROKER_URL environment variable:

```
ENV PULSAR_BROKER_URL='pulsar://<BROKER_IP_ADDRESS>:<BROKER_IP>'
```

- Specify other settings such as topic-name, size of pending message buffer, and other options in the environment variables. Below is a table of all available settings, their effect, and defaults. This list is adapted from the official documentation on the official Pulsar websites (Pulsar, N.D).Where possible, the descriptions are taken straight from this source.
    - All descriptions from the official Pulsar website are labeled with *. Note that the default values are based on the defaults of the implemented solution, not the defaults listed on the Pulsar website, although most defaults are based on the official Pulsar defaults.
    - Properties not listed in this document that are listed on the Pulsar website will be set to their corresponding Pulsar defaults

Variable	Description	Default
PULSAR_BROKER_URL	URL for the broker to connect to	broker-1:6650
TOPIC_NAME	Topic to subscribe to. Topics serve as groupings of messages. Consumers with	'Testtopic'

	matching topics will receive the messages sent from the producer	
SEND_TIMEOUT_MS	Message send timeout in ms. If a message is not acknowledged by a server before the sendTimeout expires, an error occurs*.	30000
BLOCK_IF_QUEUE_FULL	If it is set to true, when the outgoing message queue is full, the Send and SendAsync methods of producer block, rather than failing and throwing errors. If it is set to false, when the outgoing message queue is full, the Send and SendAsync methods of producer fail and ProducerQueueIsFullError exceptions occur*. The MaxPendingMessages parameter determines the size of the outgoing message queue*.	false
MAX_PENDING_MESSAGES	The maximum size of a queue holding pending messages. For example, a message waiting to receive an acknowledgment from a broker. By default, when the queue is full, all calls to the Send and SendAsync methods fail unless you set BlockIfQueueFull to true*.	

MAX_PENDING_MESSAGES_ACROSS_PARTITIONS	The maximum number of pending messages across partitions. Use the setting to lower the max pending messages for each partition ({@link #setMaxPendingMessages(int)}) if the total number exceeds the configured value*.	50000
MESSAGE_ROUTING_MODE	Message routing logic for producers on partitioned topics. Apply the logic only when setting no key on messages. Available options are as follows: pulsar.RoundRobinDistribution: round robin pulsar.UseSinglePartition: publish all messages to a single partition pulsar.CustomPartition: a custom partitioning scheme*	RoundRobinPartition
HASHING_SCHEME	Hashing function determining the partition where you publish a particular message (partitioned topics only). Available options are as follows: pulsar.JavastringHash: the equivalent of string.hashCode() in Java pulsar.Murmur3_32Hash: applies the Murmur3 hashing function pulsar.BoostHash: applies the hashing function from C++'s Boost library	Murmur3_32Hash

CRYPTO_FAILURE_ACTION	Producer should take action when encryption fails. FAIL: if encryption fails, unencrypted messages fail to send. SEND: if encryption fails, unencrypted messages are sent*.	Fail
COMPRESSION_TYPE	Message data compression type used by a producer. Available options*: • LZ4 • ZLIB • ZSTD • SNAPPY	zstd

Table 3 Pulsar Producer Docker configuration settings

- After configuring the settings as desired, build the producer docker image from the terminal.
  - Navigate to the folder of the producer:

    ```
    Cd <PATH_TO_PRODUCER>
    ```

  - Run docker build, tag the image as "producer"

    ```
    Docker build -t producer .
    ```

Execution

- Run the built image using docker run:

  ```
  Docker run producer
  ```

The producer will execute inside a Docker container and connect to the broker residing at the specified PULSAR_BROKER_URL (provided the url is valid and there is a broker at the address)

3.1.2 Consumer

Configuration and Installation

- Configure the connection URL for the Broker in the Dockerfile belonging to the Producer using the PULSAR_BROKER_URL environment variable:

  ```
  ENV PULSAR_BROKER_URL='pulsar://<BROKER_IP_ADDRESS>:<BROKER_IP>'
  ```

- Specify other settings such as topic-name, subscription-type, and other options in the environment variables. Below is a table of all available settings, their effect, and defaults. This list is adapted from the official Pulsar documentation (Pulsar, N.D). Where possible, the descriptions are taken straight from this source.

- All descriptions from the official documentation are labeled with *. Note that the default values are based on the defaults of the implemented solution, not the defaults listed on the Pulsar website, although most defaults are based on the official Pulsar defaults.

Variable	Description	Default
PULSAR_BROKER_URL	URL for the broker to connect to	'broker:6650'
TOPIC_NAME	Topic to subscribe to. Topics serve as groupings of messages. Consumers with matching topics will receive the messages sent from the producer	'Testtopic'
SUBSCRIPTION_NAME	Name of the current subscription	'subscription'
SUBSCRIPTION_TYPE	Subscription type Four subscription types are available: • Exclusive • Failover • Shared • Key_Shared	"Shared"
RECEIVER_QUEUE_SIZE	Size of a consumer's receiver queue*. For example, the number of messages accumulated by a consumer before an application calls Receive*. A value higher than the default value increases consumer	1000

	throughput, though at the expense of more memory utilization*.	
ACKNOWLEDGEMENTS_GROUP_TIME_MICROS	Group a consumer acknowledgment for a specified time*. By default, a consumer uses 100ms grouping time to send out acknowledgments to a broker*. Setting a group time of 0 sends out acknowledgments immediately*. A longer ack group time is more efficient at the expense of a slight increase in message re-deliveries after a failure*.	100
CONSUMER_NAME	Consumer name	“Consumer”
ACK_TIMEOUT_MILLIS	Timeout of unacked messages*	0
TICK_DURATION_MILLIS	Granularity of the ack-timeout redelivery*. Using a higher tickDurationMillis reduces the memory overhead to track messages when setting ack-timeout to a	1000

	bigger value (for example, 1 hour)*.	

Table 4 Pulsar consumer Docker configuration settings

- After configuring the settings as desired, build the consumer docker image from the terminal.
  - Navigate to the folder of the producer:

    ```
    cd <PATH_TO_CONSUMER>
    ```

  - Run docker build, tag the image as "producer"

    ```
    Docker build -t consumer .
    ```

Execution

- Run the built image using docker run:

  ```
  Docker run consumer
  ```

The producer will execute inside a Docker container and connect to the broker residing at the specified PULSAR_BROKER_URL (provided the url is valid and there is a broker at the address)

The messages received are printed to the terminal.

3.2 Building and Running With Maven

Prerequisites

- Maven must be installed and available from the terminal
- Java 17 must be installed on the building system

3.2.1 Producer

Configuration and Installation

- Configuration for the producer is done using config.properties and producer.properties located at the following location:pulsar-implementation/producer/src/main/resources
  - Config.properties concerns networking. Producer.properties concern settings related to the producer
- Configure the network connection in config.properties
  - Set the url of the broker:

    ```
    pulsar.broker.url=pulsar://127.0.0.1:6650
    ```

  - Set the port of the producer:

    ```
    listen.port=8992
    ```

- Configure the producer in the producer.properties file. The table below lists all configurable properties. This list is adapted from the official Pulsar documentation (Pulsar, N.D). Where possible, the descriptions are taken straight from this source.

- All descriptions from the source are labeled with *. Note that the default values are based on the defaults of the implemented solution, not the defaults listed on the Pulsar website, although most defaults are based on the official Pulsar defaults.

Variable	Description	Default
topicName	Topic to subscribe to. Topics serve as groupings of messages. Consumers with matching topics will receive the messages sent from the producer	‘testtopic’
sendTimeoutMs	Message send timeout in ms. If a message is not acknowledged by a server before the sendTimeout expires, an error occurs*.	30000
blockIfQueueFull	If it is set to true, when the outgoing message queue is full, the Send and SendAsync methods of producer block, rather than failing and throwing errors. If it is set to false, when the outgoing message queue is full, the Send and SendAsync methods of producer fail and ProducerQueueIsFullError exceptions occur*. The MaxPendingMessages parameter determines the size of the outgoing message queue*.	false
maxPendingMessages	The maximum size of a queue holding pending messages. For example, a message waiting to receive an acknowledgment from a broker.	

	By default, when the queue is full, all calls to the Send and SendAsync methods fail unless you set BlockIfQueueFull to true*.	
maxPendingMessagesAcrossPartitions	The maximum number of pending messages across partitions. Use the setting to lower the max pending messages for each partition ({@link #setMaxPendingMessages(int)}) if the total number exceeds the configured value*.	50000
messageRoutingMode	Message routing logic for producers on partitioned topics. Apply the logic only when setting no key on messages. Available options are as follows: pulsar.RoundRobinDistribution: round robin pulsar.UseSinglePartition: publish all messages to a single partition pulsar.CustomPartition: a custom partitioning scheme*	RoundRobinPartition
hashingScheme	Hashing function determining the partition where you publish a particular message (partitioned topics only). Available options are as follows: pulsar.JavastringHash: the equivalent of string.hashCode() in Java pulsar.Murmur3_32Hash: applies the Murmur3 hashing function pulsar.BoostHash: applies the hashing function from C++'s Boost library*	Murmur3_32Hash

cryptoFailureAction	Producer should take action when encryption fails. FAIL: if encryption fails, unencrypted messages fail to send. SEND: if encryption fails, unencrypted messages are sent*.	Fail
compressionType	Message data compression type used by a producer. Available options*: • LZ4 • ZLIB • ZSTD • SNAPPY	zstd

Table 5 Pulsar producer Maven configurations

- After configuring the settings as desired, build the producer jar from the terminal using maven:
  - Navigate to the folder of the producer:

    ```
    cd <PATH_TO_PRODUCER>
    ```

  - Run maven package to get a runnable jar :

    ```
    Mvn package
    ```

    - If the project is being rebuilt, it is advisable to run the clean command as well:

      ```
      Mvn clean build
      ```

Execution

- Inside the producer folder, there should now be a target folder. Navigate to this folder:

  ```
  Cd target
  ```

  - If the folder is missing, the project may not have been built. Please refer to the previous section "Configuration and Installation".
- Run the jar:

  ```
  Java -jar pulsar-producer.jar
  ```

- The producer is now running and is ready to send data

3.2.2 Consumer

Configuration and Installation

- Configuration for the consumer is done using masterconfig.properties and consumerconfig.properties located at the following location: pulsar-implementation/consumer/src/main/resources
    - masterconfig.properties concerns the consumer master. The only settable property is consumer-count, which decides how many workers the master should produce.
- All settings directly related to the consumer exist in the consumerconfig.properties. Firstly, set the url for the broker using the url property:

```
url=pulsar://23.102.116.13:6650
```

- Configure the consumer in the consumer.properties file. The table below lists all configurable properties. This list is adapted from the official Pulsar documentations (Pulsar, N.D). Where possible, the descriptions are taken straight from this source.

- All descriptions from the official documentation are labeled with *. Note that the default values are based on the defaults of the implemented solution, not the defaults listed on the Pulsar website, although most defaults are based on the official Pulsar defaults.

Variable	Description	Default
topicName	Topic to subscribe to. Topics serve as groupings of messages. Consumers with matching topics will receive the messages sent from the producer	'Testtopic'
subscriptionName	Name of the current subscription	'subscription'
subscriptionType	Subscription type Four subscription types are available: • Exclusive • Failover • Shared • Key_Shared	"Shared"
receiverQueueSize	Size of a consumer's receiver queue*.	1000

	For example, the number of messages accumulated by a consumer before an application calls Receive*. A value higher than the default value increases consumer throughput, though at the expense of more memory utilization*.	
acknowledgementGroupTimeMicros	Group a consumer acknowledgment for a specified time*. By default, a consumer uses 100ms grouping time to send out acknowledgments to a broker*. Setting a group time of 0 sends out acknowledgments immediately*. A longer ack group time is more efficient at the expense of a slight increase in message re-deliveries after a failure*.	100
consumerName	Consumer name	“Consumer”
ackTimeoutMillis	Timeout of unacked messages*	0
tickDurationMillis	Granularity of the ack-timeout redelivery*. Using a higher tickDurationMillis reduces the memory overhead to track messages when setting ack-timeout to a bigger value (for example, 1 hour)*.	1000

Table 6 Pulsar consumer Maven configuration

- After configuring the settings as desired, build the consumer jar from the terminal using maven:

- Navigate to the folder of the consumer:

  ```
  cd <PATH_TO_CONSUMER>
  ```

- Run maven package to get a runnable jar
  ```
  Mvn package
  ```
  - If the project is being rebuilt, it is advisable to run the clean command as well:

    ```
    Mvn clean build
    ```

Execution

- Inside the consumer folder, there should now be a target folder. Navigate to this folder:

  ```
  Cd target
  ```

  - If the folder is missing, the project may not have been built. Please refer to the previous section "Configuration and Installation".
- Run the jar:

  ```
  Java -jar pulsar-consumer-0.1.0-SNAPSHOT.jar
  ```

- The consumer is now running and ready to receive data

3.3 IDE (INTELLIJ)

If desired, the producer and consumer can be run straight from an IDE.

IT is assumed than any IDE that allows code execution through detection of an entry-point will be able to run the solution. However, the only IDE tested is JetBrains IntelliJ and this document only describes the process for IntelliJ.

3.3.1 Prerequisites

- An installed IDE capable of executing programs from source code. IntelliJ is recommended and the assumed IDE in this document.
- Java 17 must be installed.

3.3.2 Producer/Consumer

Configuration and Installation

To configure either the consumer or producer, please follow the steps outlined in the section describing configuration of Maven built versions of the project as the programmes will use the .properties files when run through an IDE.

There is no installation required

Execution

To execute the desired program:

- Open the desired project, consumer or producer, in IntelliJ
- Click the "Start" button. The program will start executing.
    - If the start button is grayed out, the easiest way to start the program is to go to the main file of either the producer or consumer and click the small play button next to the Main class signature or main method signature.

3.4 Broker

The broker is handled differently from the consumer and producer. The broker is provided by pulsar and their documentation is the best resource as to how to run the broker. The broker is run in a cluster with Apache Bookkeeper and Apache Zookeeper to handle persistence and load balancing respectively.

Apache provides a quick and easy way of implementing a broker by downloading a cluster and running it through the command line. The approach described here is the approach described in LINK, but any functioning deployment of the broker will work with the producer and consumer.

3.4.1 Download

Downloading can be done in many ways as described in the official documentation. The method presented here uses wget to get the necessary files.

- Navigate to a location on your computer/environment where you want the broker to be ran from
- Get the files using wget:

```
$ wget https://archive.apache.org/dist/pulsar/pulsar-2.10.0/apache-pulsar-2.10.0-bin.tar.gz
```

- After downloading it, untar the received tarball and cd into the folder:

```
$ tar xvfz apache-pulsar-2.10.0-bin.tar.gz
$ cd apache-pulsar-2.10.0
```

3.4.2 Execution

- Run the broker with the following command:

```
$ bin/pulsar standalone
```

- You will see messages printed out to the console after the broker has booted.
- By default, this broker will run on the following URL:

```
pulsar://<SYSTEM_IP>:6650
```

    - Port can be configured by changing the value of the brokerServicePort variable in the file apache-pulsar-2.10.0/conf/broker.conf:

```
brokerServicePort=<PORT>
```

4 Source code documentation

The prototype-pulsar, prototype-kafka, prototype-utils, utils, and the pulsar-implementation repository has their own GitHub page. The page can be accessed from the link https://autostreams.github.io/<repository-name>/ where <repository-name> is substituted with the repository of interest. To access the documentation, one can append javadoc to the link for the main Javadoc and javadoc-develop for the Javadoc from the development branch. The links to these pages can also be found in the README file for each project.

Example of a full link to the pulsar implementation development javadoc:
https://autostreams.github.io/pulsar-implementation/javadoc-develop/

One can also generate the javadoc locally with Maven, provided Maven is installed. From the root directory of the parent project of interest, issue the command

```
mvn javadoc:aggregate
```

The generated API documentation can then be found in the directory: ./target/site/apidocs

5 Continuous integration

GitHub actions [https://docs.github.com/en/actions] is the continuous integration tool used for all the AutoStream repositories containing a maven project. There are three workflow files for each repository:

- maven.yml - Verifies that all sub-module projects can be built with maven
- javadoc_publisher.yaml - Generates the Javadoc documentation for the master branch and uploads it to GitHub pages related to the project
- javadoc_develop_publisher.yaml - Generates the Javadoc documentation for the develop branch and uploads it to GitHub pages related to the project

REFERENCES

Apache Maven Project, 2022. *Introduction to the Standard Directory Layout.* [Online]
Available at: https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html
[Accessed 11 5 2022].

Google checkstyle, 2022. *Github google_checks.xml.* [Online]
Available at:
https://github.com/checkstyle/checkstyle/blob/master/src/main/resources/google_checks.xml
[Accessed 11 5 2022].

Java Design Patterns, n.d. *Master-Worker.* [Online]
Available at: https://java-design-patterns.com/patterns/master-worker-pattern/
[Accessed 11 5 2022].

JetBrains, n.d. *IntelliJ IDEA: The Capable & Ergonomic Java IDE by JetBrains.* [Online]
Available at: https://www.jetbrains.com/idea/
[Accessed 12 May 2022].

Netty project, 2022. *Netty project.* [Online]
Available at: https://netty.io/
[Accessed 11 5 2022].

Pulsar, 2022. *Pulsar Java client.* [Online]
Available at: https://pulsar.apache.org/docs/en/client-libraries-java/
[Accessed 11 5 2022].

Pulsar, A., n.d. *Pulsar Java client.* [Online]
Available at: https://pulsar.apache.org/docs/client-libraries-java/
[Accessed 12 May 2022].

Pulsar, A., n.d. *Set up a standalone Pulsar locally.* [Online]
Available at: https://pulsar.apache.org/docs/getting-started-standalone/
[Accessed 12 May 2022].

Pulsar, N.D. *Pulsar Java Client.* [Online]
Available at: https://pulsar.apache.org/docs/client-libraries-java/
[Accessed 5 May 2022].

Appendix 7: Requirements documentation

NTNU | Norwegian University of Science and Technology

Service for Data Streaming

Documentation of Requirements

Lars Ivar Ramberg

Tomas Klungerbo Olsen

Table of Contents

1 INTRODUCTION

This document presents the identified requirements for the project "Service for Data streaming" at NTNU (Norges Teknisk Naturvitenskapelige Universitet) in Ålesund. The requirements are identified using a Use Case diagram, user stories, and a domain model. These tools are intended to provide reasoning and background for identified requirements.

The project is a command-line-based solution. Wireframe prototypes are therefore not presented in this document, as no graphical user interface prototypes were developed.

2 USE CASE DIAGRAM

This chapter presents a graphical representation of users and their interactions with the system, a Use Case diagram [1].

The use case diagram is intended to guide the developers during development and communicate an understanding of use cases to the stakeholder. The diagram acts as a common language shared by the development team and the stakeholder.

The users of the system must be identified to create a use-case diagram. For this use case, the following immediate users and their interactions have been identified as:

- **Customers/Installations** generate data passively and provide it to the system
- **Data-analysts** perform analysis on the data provided
- **Service Installers** Install and configure the system at customer sites. The installers may be the same user as the Service Engineer in some cases

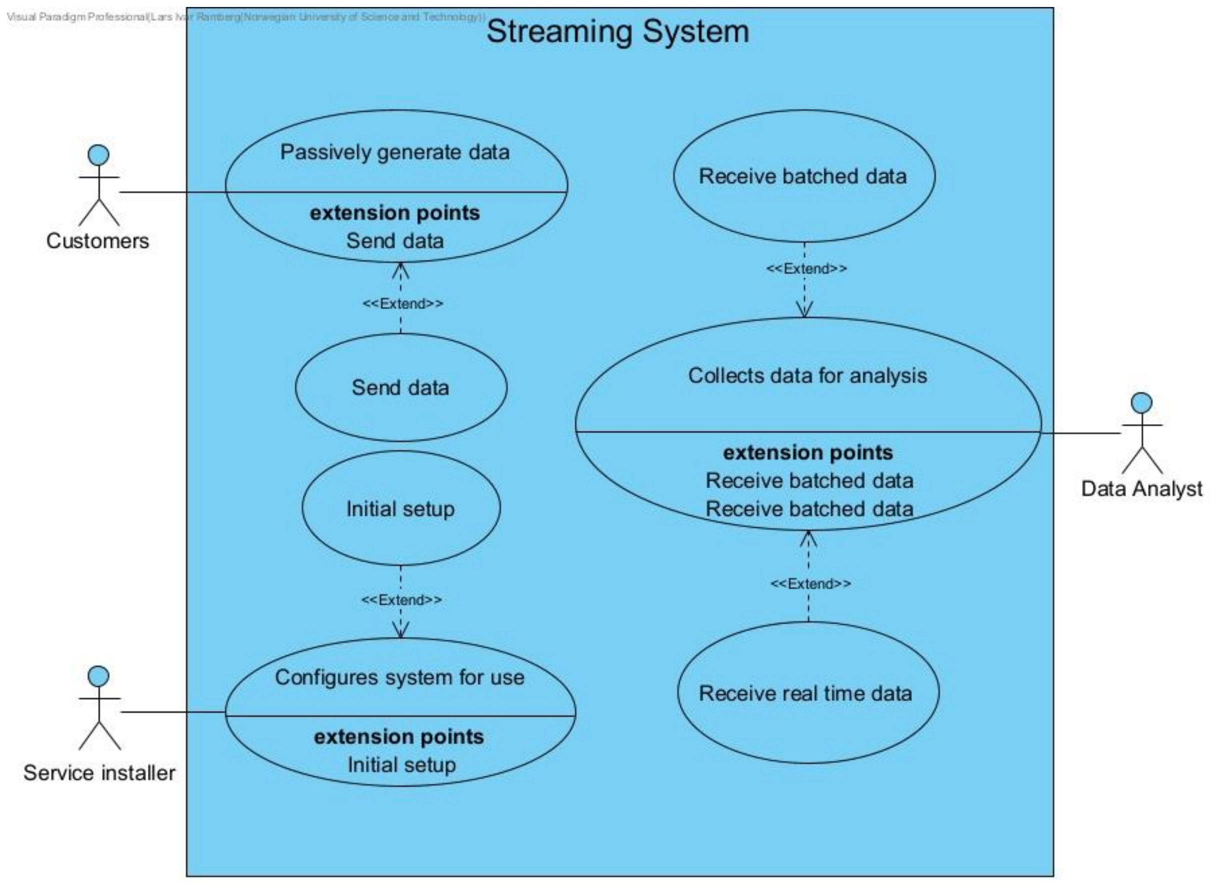


Figure 1 Use Case Diagram of solution

3 USER STORIES

The following section presents relevant user stories for the solution. These user stories differ from the ones used in Jira, as the user stories in Jira have all been defined with a general "user" as opposed to specific users. All user stories do still apply.

3.1 CUSTOMER/INSTALLATION

As a Customer/Installation,
I want the data I generate to be sent from my installation to AutoStore for analysis and action,
So that my system can be properly maintained

Acceptance criteria:

- Mechanism for sending data to Pulsar broker for further sending to consumer
- Mechanism that can run continuously and pass along data as the system generates it

---

As a Customer/Installation,
I want to send my data securely,
So that no actor other than AutoStore can see my data

Acceptance criteria:

- Encryption of sent messages, and functional decryption on the receiving side
- Default settings to encryption, make it either impossible or very difficult to turn encryption off

---

3.2 SERVICE INSTALLER

As a Service Installer,
I want to easily install the Pulsar consumer implementation using a cloud service,
So that messages produced from installations can be consumed

Acceptance criteria:

- Semantic configuration properties in files
- No installation, or easy installation
- Good, thorough documentation and user manual
- Clear error messages in case of misconfiguration

As a Service Installer,
I want to configure the Pulsar consumer,
So that consumption of messages is optimized regarding current needs

Acceptance criteria:

- Ability to configure pulsar consumer

As a Service Installer
I want to configure the Pulsar producer
So that message production is optimized regarding current needs

Acceptance criteria:

- Ability to configure Pulsar Producer

As a Service Installer
access to the Pulsar consumer and producer implementation documentation,
So that additional features can be added

Acceptance criteria:

- Thorough, understandable documentation
- Documentation needs to accompany the packaging of the solution

---

3.3 DATA-ANALYST

As a Data-analyst
I want to receive data from the installation
So that I can perform analysis on the data

Acceptance criteria:

- Mechanism for receiving data transferred from customer sites
- Mechanism for passing the data to systems of analytics¨

As a Data-analyst
I want to receive all the data from the installation
So that my analyses are based on a complete dataset

Acceptance criteria:

- All data sent must be received. Loss should be 0%
- If a piece of data fails to be received on first try, the system must retry until the data is sent
- Delayed data must eventually reach the consumer

---

4 DOMAIN MODEL

Multiple models of the solution were created to improve communication with the stakeholder The first model (See Figure 2) was made relatively early to present the solution to the stakeholder and verify that the team had understood the envisioned architecture properl

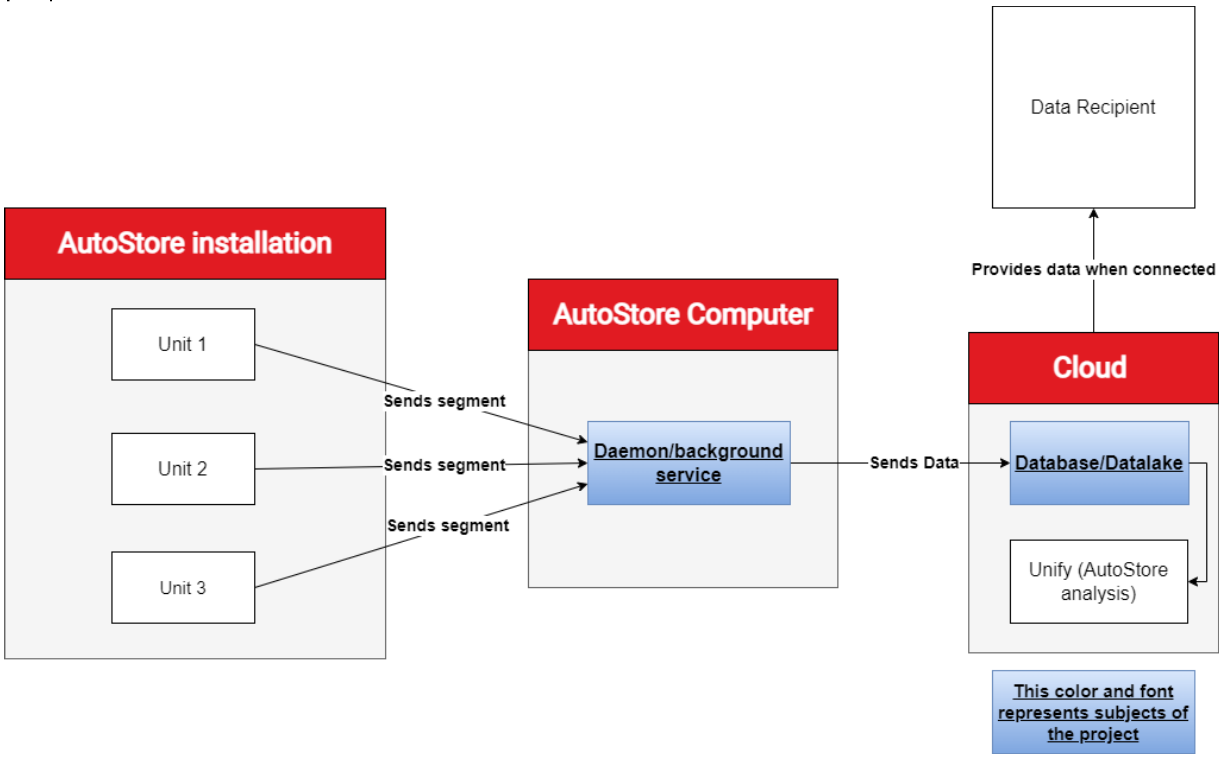


Figure 2 First model made to communicate with stakeholder.

Diagrams were created to show the architecture for the tests. While not strictly domain models, they give an overview of system implementation. A final solution will need to be installed the same way as the tests, with one unit each hosting a producer, consumer, and broker. The model from the testing is included as Figure 3.

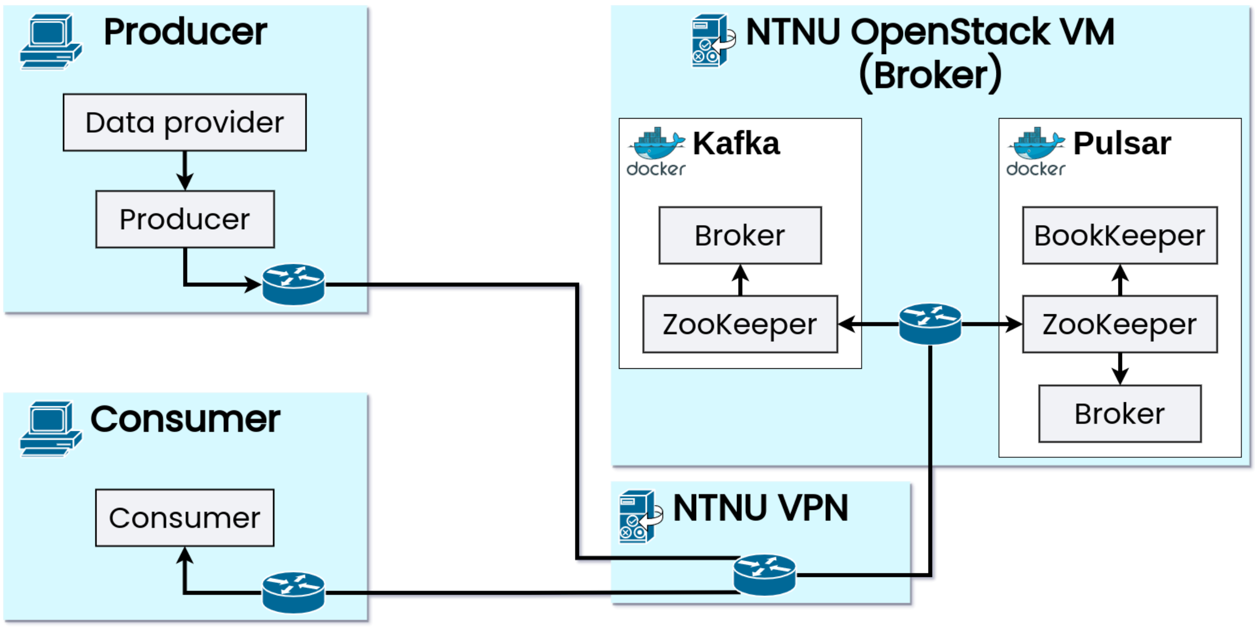


Figure 3 Model created for testing

A more formal domain model was produced using the Visual Paradigm software (see Figure 4). This domain model presents a comprehensive system image with classes, fields, and methods. The domain model complements the higher-level model created for the testing (see Figure 3), as the detailed model shows implementation details of the consumer and producer. However, the broker is omitted as it is configured rather than implemented. It is helpful to have both diagrams to understand the developed solution's implementation and installation.

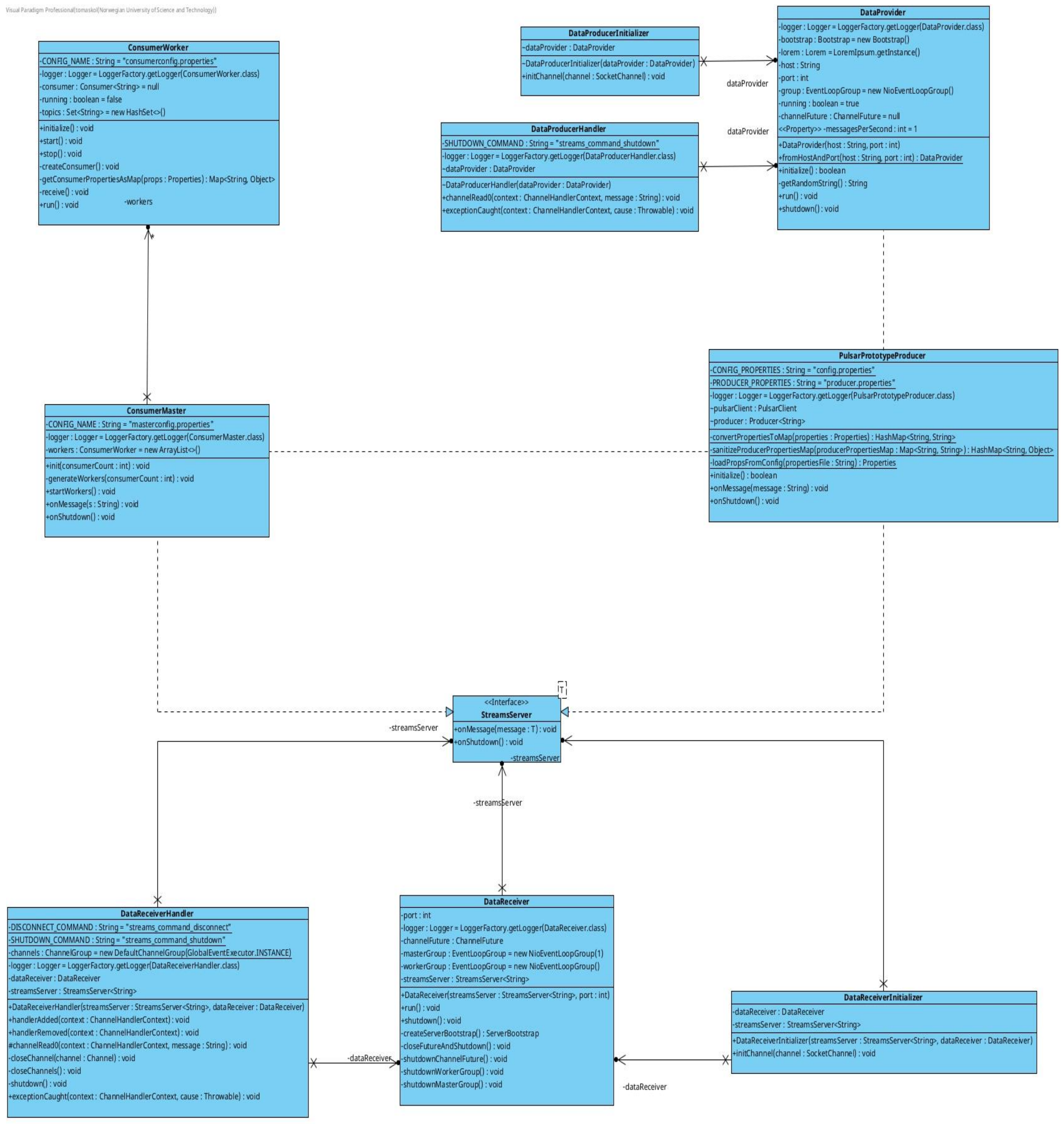

Visual Paradigm Professional(tomaskol(Norwegian University of Science and Technology))
ConsumerWorker
-CONFIG_NAME : String = "consumerconfig.properties"
-logger : Logger = LoggerFactory.getLogger(ConsumerWorker.class)
-consumer : Consumer<String> = null
-running : boolean = false
-topics : Set<String> = new HashSet<>()
+initialize() : void
+start() : void
+stop() : void
-createConsumer() : void
-getConsumerPropertiesAsMap(props : Properties) : Map<String, Object>
-receive() : void
+run() : void
-workers
DataProducerInitializer
~dataProvider : DataProvider
~DataProducerInitializer(dataProvider : DataProvider)
+initChannel(channel : SocketChannel) : void
dataProvider
DataProducerHandler
-SHUTDOWN_COMMAND : String = "streams_command_shutdown"
-logger : Logger = LoggerFactory.getLogger(DataProducerHandler.class)
~dataProvider : DataProvider
~DataProducerHandler(dataProvider : DataProvider)
+channelRead0(context : ChannelHandlerContext, message : String) : void
+exceptionCaught(context : ChannelHandlerContext, cause : Throwable) : void
dataProvider
DataProvider
-logger : Logger = LoggerFactory.getLogger(DataProvider.class)
-bootstrap : Bootstrap = new Bootstrap()
-lorem : Lorem = LoremIpsum.getInstance()
-host : String
-port : int
-group : EventLoopGroup = new NioEventLoopGroup()
-running : boolean = true
-channelFuture : ChannelFuture = null
<<Property>> -messagesPerSecond : int = 1
+DataProvider(host : String, port : int)
+fromHostAndPort(host : String, port : int) : DataProvider
+initialize() : boolean
-getRandomString() : String
+run() : void
+shutdown() : void
ConsumerMaster
-CONFIG_NAME : String = "masterconfig.properties"
-logger : Logger = LoggerFactory.getLogger(ConsumerMaster.class)
-workers : ConsumerWorker = new ArrayList<>()
+init(consumerCount : int) : void
-generateWorkers(consumerCount : int) : void
+startWorkers() : void
+onMessage(s : String) : void
+onShutdown() : void
PulsarPrototypeProducer
-CONFIG_PROPERTIES : String = "config.properties"
-PRODUCER_PROPERTIES : String = "producer.properties"
-logger : Logger = LoggerFactory.getLogger(PulsarPrototypeProducer.class)
~pulsarClient : PulsarClient
~producer : Producer<String>
-convertPropertiesToMap(properties : Properties) : HashMap<String, String>
-sanitizeProducerPropertiesMap(producerPropertiesMap : Map<String, String>) : HashMap<String, Object>
-loadPropsFromConfig(propertiesFile : String) : Properties
+initialize() : boolean
+onMessage(message : String) : void
+onShutdown() : void
T
<<Interface>>
StreamsServer
+onMessage(message : T) : void
+onShutdown() : void
-streamsServer
-streamsServer
-streamsServer
DataReceiverHandler
-DISCONNECT_COMMAND : String = "streams_command_disconnect"
-SHUTDOWN_COMMAND : String = "streams_command_shutdown"
-channels : ChannelGroup = new DefaultChannelGroup(GlobalEventExecutor.INSTANCE)
-logger : Logger = LoggerFactory.getLogger(DataReceiverHandler.class)
-dataReceiver : DataReceiver
-streamsServer : StreamsServer<String>
+DataReceiverHandler(streamsServer : StreamsServer<String>, dataReceiver : DataReceiver)
+handlerAdded(context : ChannelHandlerContext) : void
+handlerRemoved(context : ChannelHandlerContext) : void
#channelRead0(context : ChannelHandlerContext, message : String) : void
-closeChannel(channel : Channel) : void
-closeChannels() : void
-shutdown() : void
+exceptionCaught(context : ChannelHandlerContext, cause : Throwable) : void
-dataReceiver
DataReceiver
-port : int
-logger : Logger = LoggerFactory.getLogger(DataReceiver.class)
-channelFuture : ChannelFuture
-masterGroup : EventLoopGroup = new NioEventLoopGroup(1)
-workerGroup : EventLoopGroup = new NioEventLoopGroup()
-streamsServer : StreamsServer<String>
+DataReceiver(streamsServer : StreamsServer<String>, port : int)
+run() : void
+shutdown() : void
-createServerBootstrap() : ServerBootstrap
-closeFutureAndShutdown() : void
-shutdownChannelFuture() : void
-shutdownWorkerGroup() : void
-shutdownMasterGroup() : void
-dataReceiver
DataReceiverInitializer
-dataReceiver : DataReceiver
-streamsServer : StreamsServer<String>
+DataReceiverInitializer(streamsServer : StreamsServer<String>, dataReceiver : DataReceiver)
+initChannel(channel : SocketChannel) : void

Figure 4 Detailed domain model including classes, interfaces, fields, and methods, as well as their relationships.

5 References

[1] Wikipedia contributors, Use case diagram. Online: https://en.wikipedia.org/wiki/Use_case_diagram. Last visited 29.04.2022

Appendix 8: Internal Work Contract, Development Team

Arbeidskontrakt for “Tjeneste for datastrømming (IDATA2900, ED3)”

Medlemmer: Lars Ivar Ramberg, Tomas Klungerbo Olsen

Innledning

Denne arbeidskontrakten bygger på et sett med typiske mål, oppgavefordelinger, prosedyrer og retningslinjer for interaksjoner for studentarbeider. Arbeidskontrakten er utfylt med *egne* fortolkninger av hva man mener med disse og hvordan man skal oppnå dette. Roller og oppgavefordeling

Dokumentansvarlig/sekretær

Dokumentansvarlig/sekretær har ansvar for å ta møtereferater, screenshots av sprinter, og føre annen relevant dokumentasjon underveis i prosjektet. Denne rollen vil rulleres innad i gruppen på ukentlig basis.

Tomas Klungerbo Olsen tar denne rollen første gang i uke 2, og vil ha denne rollen hver partallsuke.

Lars Ivar Ramberg tar denne rollen første gang i uke 3, og vil ha denne rollen hver oddetalsuke.

Product Owner

Product Owner er bindeleddet mellom oppdragsgiver og utvikler-teamet. Product Owner er ansvarlig for å representere produktet, og har som mål å fatte beslutninger som kommer produktet til gode.

Product Owner har ofte ansvar for backlog og user stories, men i denne oppgaven vil ansvar for backlog og user stories deles mellom gruppemedlemmene. Product Owner vil derfor i hovedsak ha en kommunikativ rolle mot oppdragsgiver, og være ansvarlig for kommunikasjon.

Lars Ivar Ramberg vil være Product Owner for dette prosjektet.

Scrum Master

Scrum Master er ansvarlig for å kommunisere mål og retning til de resterende team-medlemmene. Scrum Master har ikke direkte ansvar for kommunikasjon med oppdragsgiver (dette ansvaret faller på Product Owner).

Scrum Master er en “tjenende leder” for teamet. Teamet skal i utgangspunktet være selvstyrende, og Scrum Master skal da ikke lede teamet direkte. Scrum Master skal påse at Scrum rammeverket blir fulgt og respektert.

Tomas Olsen vil være Scrum Master for dette prosjektet.

Kvalitetssikring

Kode

All kode skal gjennom en “review” før den aksepteres som del av endelig løsning. Rollen som kvalitetssikrer, eller “reviewer” vil tilhøre det gruppemedlemmet som ikke skrev den aktuelle koden. Dette gjøres for å forsikre kvalitet ved at begge gruppemedlemmene ser og godkjenner all kode som skrives.

Dokumentasjon

Kvalitetssikring for dokumenter vil være ansvaret til det gruppemedlemmet som ikke er forfatter av dokumentet.

Prosedyrer

A. *Scrum og utviklingsmetodikk*
    a. Utvikling skal gjennomføres basert på Scrum metodikk.
    b. Hver sprint vil være en uke lange.
    c. Scrum roller og rolle-fordeling er beskrevet i avsnitt "Roller og oppgavefordeling"
    d. En ny sprint starter med sprint planning hver mandag etter daglig stand-up møte (se punkt B "Møter").
    e. Sprint review avholdes hver søndag etter daglig stand-up møte (se punkt B "Møter").
    f. En sprint avsluttes med sprint retrospective hver søndag etter sprint review.
        i. I sprint retrospective skal det reflekteres over prosessen, ikke produktet. Sprint retrospectiven bør derfor ta for seg:
            1. Hva gikk bra?
            2. Hva gikk ikke så bra?
            3. Hvilke konkrete handlinger kan teamet ta for å forbedre det som ikke gikk bra, og videreføre/opprettholde det som har gått bra.
    g. Issue tracking skal skje i prosjektet sitt Jira-space
        i. Alle oppgaver tilknyttet utvikling skal spores (trackes) i Jira-spacet. Oppgaver skal som hovedregel ikke utføres uten å bli sporet i Jira.
    h. All kode utviklet i prosessen skal samles i en GitHub repository.
        i. Den ansvarlige for GitHub repositoriet skal forsikre seg om at repositoriet er tilgjengelig for alle gruppemedlemmer, også etter at prosjektet er over, i minst et år etter sluttdato.
        ii. Dersom det andre gruppemedlemmet ønsker tilgang etter året med tilgjengelighet, må de ta en kopi av repositoriet før tilgjengeligheten avsluttes.
    i. Extreme-Programming metodikk kan implementeres dersom nødvendig.

B. *Møter*
    a. Stand up møter skal gjøres daglig, mandag til fredag og søndag kl 08:00. Disse møtene skjer uten innkalling til faste tider i Microsoft Teams.
    b. Sprint møter skal skje hver uke i form av en sprint review, sprint planning, og sprint retrospective.
    c. Møter med veileder og oppdragsgiver skal skje annenhver uke, med start i uke 2, 2022. Disse møtene vil tilsvare en sprint review med demo, der arbeid vises frem til oppdragsgiver og veileder. Videre prosedyrer for innkalling er beskrevet i punkt C.

C. *Møteinnkalling*
    a. Alle møter som ikke er stand up møter skal ha en formell innkalling.
    b. Møteinnkalling skal skje over mail. Mail sendes til alle gruppemedlemmer hver uke. Annenhver uke inviteres også veileder, kontaktperson hos oppdragsgiver, og annet relevant personell hos oppdragsgiver.
        i. Annet relevant personell kan være veiledere eller interessehavere hos oppdragsgiver.
    c. I møteinnkallingen skal det alltid være en lenke til et Microsoft-Teams rom der møtene holdes.
    d. Møteinnkallingen må inneholde tidspunkt for møtet.

    e. Møteinnkalling skal senest sendes ut tre dager før møtet avholdes.

D. *Varsling ved fravær eller andre hendelser*
    a. Dersom man ikke kan møte skal dette meldes om så snart som mulig før møtet. Varsling skal skje over mail, og varslet skal gå ut til alle andre møte-deltakere.
    b. Følgende ansees som gyldige grunner til fravær:
        i. Uforutsett tap av internettforbindelse
            1. Herunder forutsettes det at gruppemedlemmet tar tiltak for å møte likevel, som for eksempel å bruke mobilnett, eller dra til campus.
        ii. Uforutsette, kritiske tekniske problemer.
        iii. Sykdom
        iv. Ulykke eller alvorlige personskader som hindrer arbeid
        v. Død eller alvorlig sykdom i nære relasjoner
        vi. Dødsfall
    c. Ved fravær uten varsel brukes følgende system for sanksjoner:
        i. Første fravær resulterer i samtale innad i gruppen. Samtalen bør diskutere hvorfor fraværet skjedde, og om det kan gjøres endringer for å unngå flere fravær.
        ii. Andre fravær resulterer i samtale mellom gruppen og veileder. Samtalen bør også her fokusere på å diskutere hvorfor fraværet skjedde, og om noe kan endres for å unngå flere fravær.
        iii. Ved tredje fravær vil det bli iverksatt seperasjon av gruppen.

E. *Dokumenthåndtering*
    a. Alt skal dokumenteres så langt det lar seg gjøre.
    b. Ansvaret for dokumentasjon skal rulleres hver uke (Se seksjon “Roller og Oppgavefordeling”).
    c. Følgende skal alltid dokumenteres:
        i. Møter (møtereferat).
        ii. Beslutninger vedrørende utvikling.
        iii. Sprinter og fremgang, dokumenteres med snapshots underveis.
    d. All dokumentasjon skal så langt det lar seg gjøre skrives og lagres i Confluence. Dokumentasjon som ikke lagres i Confluence skal lagres i en delt Google-drive som alle gruppemedlemmer skal ha tilgang til.
    e. Dokument-ansvarlig har frist til søndag kveld for å fullføre sin dokumentasjon.

F. *Innleveringer av gruppearbeider*
    a. Alle innleveringer vedrørende rapport og dokumentasjon direkte knyttet til rapport, skal gjennomleses og godkjennes av alle gruppemedlemmer før innlevering.
    b. Alle innleveringer vedrørende rapport og dokumentasjon direkte knyttet til rapport skal levers innen gitte tidsfrister.
    c. Gruppearbeider skal være et kollaborativt arbeid der alle gruppemedlemmer deltar i utforming, godkjenning, og innlevering.

G. *Timeregistrering*
    a. All tid brukt på prosjektet, og arbeid direkte relatert til prosjektet, skal registreres
    b. Registrering foregår i Tempo, en plug-in for Jira.

        i. Hver time skal registreres til en issue.
        ii. Hvis arbeid gjøres som ikke kan registreres til en spesifik issue, skal det registreres til en generell issue som kan representere. Arbeidet gjort.
            1. For eksempel vil etter-skriving av møtereferater kunne registreres på en generell issue/task ved navn "Referater" eller "Generelt dokumentarbeid".

Interaksjon

A. *Oppmøte og forberedelse*
    a. Alle gruppemedlemmer skal senest stille opp ved angitt oppmøtetidspunkt informert via mail.
        i. For stand up møter skal gruppemedlemmene senest stille opp ved tidspunktene spesifisert i denne avtalen.
    b. Dersom et gruppemedlem stiller opp etter oppmøtetidspunktet ansees dette som en forsentkomming.
        i. Forsentkomminger behandles på lik måte som fravær (Seksjon "Prosedyre", Punkt D).

B. *Tilstedeværelse og engasjement*
    a. Passiv underholdning tillates under arbeid. Passiv underholdning defineres som:
        i. Musikk
        ii. Video og streaming av video.
    b. Det er opp til hvert enkelt gruppemedlem å påse at underholdningen ikke distraherer fra arbeidet. Dersom underholdningen blir distraherende, selv om den er passiv, skal underholdningen utebli.
    c. Følgende underholdning skal til enhver tid unngås:
        i. Dataspill.
        ii. Facebook og sosiale medier.
        iii. Lesing av materiale ikke relatert til arbeidet (for eksempel nettaviser).
        iv. Vilkårlig surfing.
    d. Under møter er det forventet at alle gruppemedlemmer deltar aktivt og er engasjerte. Andre aktiviteter som ikke er relatert til møtet eller dokumentasjon av møtet skal unngås.

C. *Hvordan støtte hverandre*
    a. Gruppen skal fremme en kultur hvor det er lov å gjøre feil, stille spørsmål, og presentere konstruktiv kritikk.
    b. Alminnelig folkeskikk skal utøves til enhver tid. Trakassering og mobbing skal unngås.
        i. Alvorlige tilfeller av trakassering ansees som avvik og kan ansees som grunnlag for å separere gruppen.
        ii. Ved alvorlige tilfeller skal veileder inkluderes og gruppen skal prøve å løse uenigheter.
        iii. Dersom uenighetene ikke kan løses, selv etter møte mellom gruppemedlemmer og veileder, vil det iverksettes seperasjon av gruppen.
    c. Alle ideer skal høres og behandles med respekt.
        i. Ideer skal være åpen for kritikk, men kritikken skal være saklig og konstruktiv.

D. *Uenighet, avtalebrudd*
   a. Uenighet skal så langt det lar seg gjøre løses ved samtaler innad i gruppen.
      i. Veileder kan involveres dersom uenighetene ikke lar seg løse innad i gruppen.
      ii. Dersom uenigheten fortsatt ikke lar seg løse, vil et siste alternativ være å splitte gruppen. Dette skal unngås så langt som mulig.
   b. Relevante avvik vil også være fravær. Fravær diskuteres og avklares i avsnitt "Prosedyrer" (Punkt D) av denne avtalen.

Appendix 9: Revised milestone plan

Milestones, V 2.0

What	**Description**	**When**
Implementation	Completion of Pulsar prototype (Framework 2)	Weeks 11 – 12
Implementation	Completion of Flink Prototype (Framework 3)	Week 13
Testing	Testing and comparison between frameworks	Week 14
Reporting	Document all finding in above implementation tasks. Start thesis report. Finish research section above and hand over for review to AutoStore.	Week 15 2nd delivery to AutoStore
Implementation	Selection of most relevant framework. More in depth implementation, testing and validation.	Week 16
Implementation	Testing in Azure. Cloud architecture and frameworks available for data consumption. Configuration, implementation and test of cloud consumer and multi-region clients. (Students should use their own Azure accounts, but expenses not covered by university can be reimbursed)	Weeks 17 - 18
Reporting	Finalize report, complete thesis work.	Week 19 3rd and final delivery to AutoStore

Klungerbo/Ramberg

Service for Data Streaming

NTNU
Norwegian University of
Science and Technology