Design and Implementation of a Tool to Collect Execution- and Service-Data of Big Data Analytics Applications

Bachelor's Thesis

for obtaining the academic degree
Bachlor of Science (B.Sc.)

at

Beuth Hochschule für Technik Berlin
Department Informatics and Media VI
Degree Program Mediainformatics

1. Examiner and Supervisor:	Prof. Dr. Stefan Edlich
2. Examiner:	Prof. Dr. Elmar Böhler
Submitted by:	Markus Lamm
Matriculation number:	s786694
Date of submission:	06.09.2016

Ackknowledgements

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Contents

1 Introduction

1.1 Motivation

In preparation of this thesis, I first got in contact with Prof. Dr. Stefan Edlich on January 29th this year and presented an own idea for a topic of a bachelor's thesis. At this time, I was visiting my last course at Beuth Hochschule, the software-project which is spread over two semesters aiming to design and implement a software application in cooperation with software related companies based in Berlin, which was Lieferando in my case, an online food order service. During this project, I got in touch with a lot of technologie like Apache Kafka, Apache Spark, Cassandra, Elasticsearch and Consul, all together well known as "buzzwords" from technology-blogs and magazines.

Because I was interested to learn a bit more about that "big-data-streaming-thing" and especially how to build software using stream processing frameworks, I decided my thesis to be in this big data context and created a working title "Design And Implementation Of A 'Data Processing Pipeline'" To Transform Continous Monititoring Data Streams". The basic idea was to aggregate data from REST RabbitMQ endpoints, send this raw data to a stream processor and create a model which fits the monitoring domain and store this data in a storage system which enables further data analytics.

During the following email correspondence, Prof. Dr. Stefan Edlich he suggested me to get the data from the the streaming platforms components itself, instead of a RabbitMQ queue as my idea suggested. So he presented one of his own topics which was quite congruent to my own idea with the given title *Design and Implementation of a Tool to Collect Execution- and Service-Data of Big Data Analytics Applications*, which I finally choosed to be the one to work out.

This topic is located on germans biggest big data research project "Berlin Big Data Center", which is Prof. Dr. Stefan Edlich a member of. Within the project, a program will be developed, which collects and stores relevant data of streaming platforms like Apache Flink, Apache Kafka or Apache Spark, with the overall aim of which is to build a software that is able to "learn", based on the data that will be collected by the system that will be deisgned in this thesis.

Apache Flink is a "new player" in the plurality of stream processing frameworks. It was initialized by researchers of the Technische Universität Berlin, Humboldt Universität Berlin and Hasso-Plattner Institut Potsdam in 2008 and has emerged from the research project described above. On the 12th of January 2015 Flink became a top level project of the Apache Foundation. In the meantime, the development of Flink is driven by a grown community (216 contributers, 22.08.2016) and a wide range of companies that are actively using it.

1.2 Objective

The main goal of the thesis is a working software system to ingest and store data that can be collected from Apache Flink and Apache Kafka. It will be examined, which data is available and can be collected at all, what data is relevant and how to collect from source systems.

Furhermore, the collected data must be stored in a persistence system to become available for possible consumers like visualization applications, analytical processes or as a data source for applications from the context of Machine Learning for example.

This thesis will not be a deep introduction into big data, stream processing or covers deeper details of the internals of Apache Flink and Apache Kafka. To understand the context this frameworks are located in, the underlying concepts will be explained only briefly.

1.3 Structure of thesis

After a short introduction to the topics and the main goals of the present thesis in this chapter, Chapter 2 discusses the context of stream processing, introduces Apache Flink and Apache Kafka as representatives of widely used stream processing frameworks.

Chapter 3 examines Apache Flink and Apache Kafka regarding to the provided data both of the systems. The different sources for the data collection will be described, as well as the question, which data should be collected and stored in a persistence system regarding to its relevance and data quality. According to the results of the data analysis, the functional and non-functional requirements of the system being developed will be introduced at the end of the chapter.

Based on the requirements elaborated in Chapter 3, Chapter 4 introduces the software solution by giving a detailed conceptional overview of the software components involved and discusses implementation details for selected items.

In chapter 5 we'll see how to setup the technical environment for the usage of the prototype to verify the correct functionality related to the requirements defined in Chapter 4.

The last Chapter 6 covers a conclusion and gives a resumee of the present work.

2 Basic Concepts

After a short introduction to the terminology of Big Data, this chapter will discuss the main characteristics of Big Data Analytics Applications and introduces the concept of stream processing, which is one of the main characteristics of the popular streaming frameworks Apache Flink and Apache Kafka. The underlying concepts both of these systems and how they're used in context of Big Data Analytics will be explained at the end of this chapter.

2.1 Big Data

According to [Nat15] the term "Big Data" is a misleading name since it implies that pre-existing data is somehow small, which is not true, or that the only challenge is the sheer size of data, which is just one one them among others. In reality, the term Big Data applies to information that can't be processed or analyzed using traditional processes or tools.

Another definition comes from the the science historian George Dyson, who was cited by Tim O'Reilly in [ORe13]: *Big data is what happened when the cost of storing information became less than the cost of making the decision to throw it away.*

In the past decade the amount of data being created is a subject of immense growth. More than 30,000 gigabytes of data are generated every second, and the rate of data creation is only accelerating.[Nat15]. People create content like blog posts, tweets, social network interactions, photos, servers continuously log messages, scientists create detailed measurements, permanently.

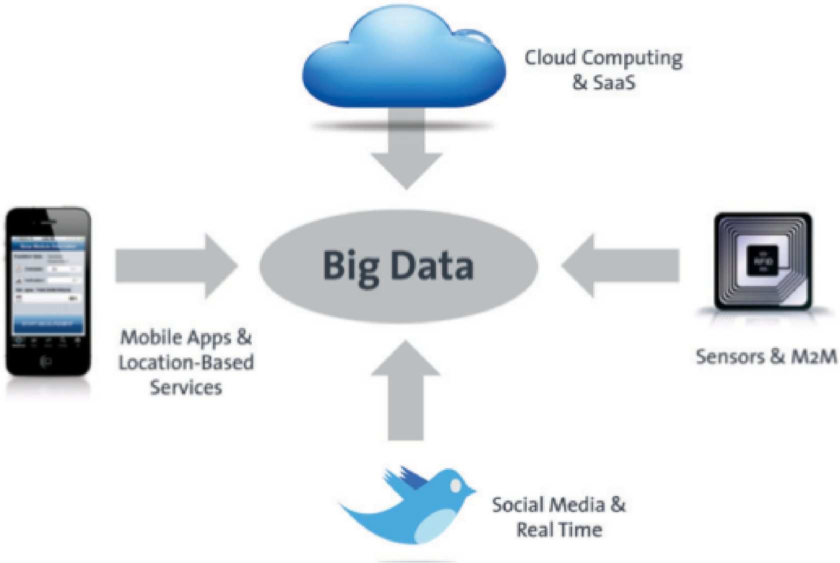

Figure 2.1: Sources of Big Data[Jör14b]

Through advances in communications technology, people and things are becoming increasingly interconnected. Generally referred to as machine-to-machine (M2M), interconnectivity is responsible for double-digit year over year data growth rates. Finally, because small integrated components are now affordable, it becomes possible to add intelligence to almost everything. As an example, a simple railway car has hundreds of sensors for tracking the state of individual parts and GPS-based data for shipment tracking and logistics.[**Ziko12**]

Besides the extremely growing amount of data, an increase in data diversity goes hand in hand. It comes in its raw and unstructured, semistructured or structured form, which makes processing it in a traditional relational system impractical or impossible.[Jör14b] describes, that around 85 percent of the data comes in an unstructured form, but containing valuable information.

According to [Nat15] [**Ziko12**], Big Data is defined by three characteristics:

Volume The amount of data present is growing because of growing amount of producers, e.g. environmental data, financial data, medical data, surveillance data.

Variety Data varies in its form, it comes in different formats from different sources.

Velocity Data needs to be evaluated and analyzed quickly, which leads to new challenges like analysis of large data sets with answers in seconds range, data processing in realtime, data generation and transmission at highspeed.

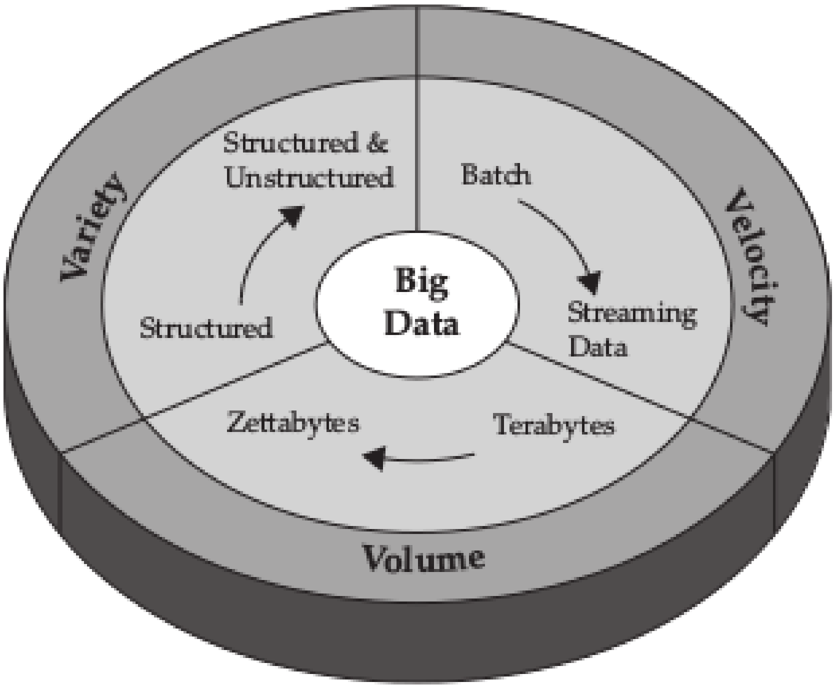

Figure 2.2: The three 'V's of Big Data[**Ziko12**]

A possible definition for Big Data could be derived as follows: *Big Data refers to the use of large amounts of data from multiple sources with a high processing speed for generating valuable information based on the underlying data.*

2.2 Big Data Analytics Applications

Big Data Analytics describes the process of collecting, organizing and analyzing large volumes of data with the aim to discover patterns, relationships and other useful information extracted from incoming data streams [Nat15]. The process of analytics is typically performed using specialized software tools and applications for predictive analytics, data mining, text mining, forecasting and data optimization.

The analytical methods raise data quality for unstructured data on a level that allows more quantitative and qualitative analysis. With this structure it becomes posssible to extract the data that is relevant by iteratively refined queries.

The areas of applications may be extremely diverse and ranges from analysis of financial flows or traffic data, processing sensor data or environmental monitoring as explained in the previous chapter.

The illustration below summarises the six-dimensional taxonomy [Jör14a; Gro14] of Big Data Analytics Applications.

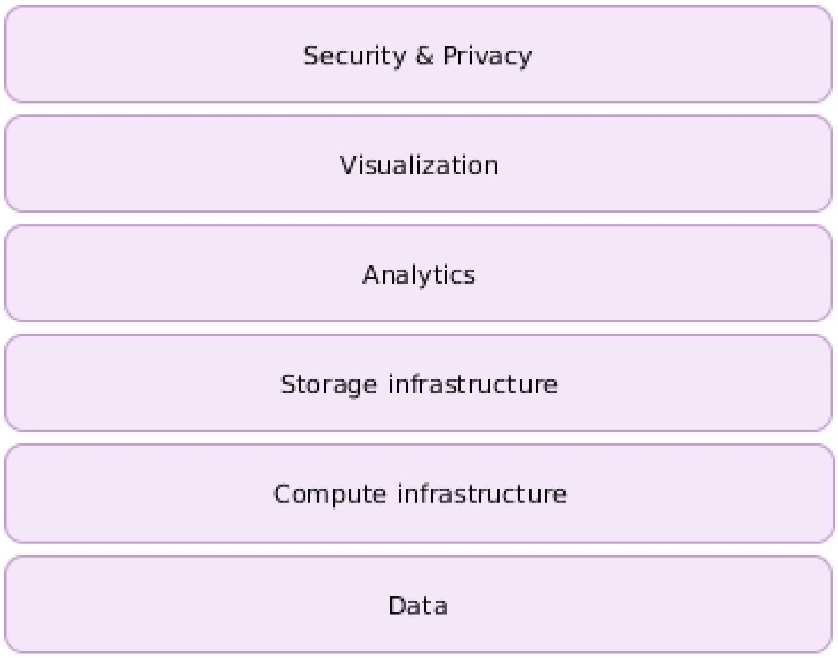

Figure 2.3: Taxonomy of Big Data Analytics Applications [Jör14a; Gro14]

The following section will discus the topic stream processing, which is part of the "Compute infrastructure" layer shown in the figure above.

2.3 Stream Processing

Computing paradigms on big data currently differ on whether the processing will be done in batch mode, or in real-time/near real-time on streaming data. This section is focussed on processing continous data streams in real-time/near real-time and introduces Apache Flink and Apache Kafka as representants of streaming frameworks.

According to [Kle16], stream processing is the real-time processing of data continuously, concurrently, and in a record-by-record fashion in which data is treated not as static tables or files, but as a continuous infinite stream of data integrated from both live and historical sources. Various data streams could have own features. For example, a stream from the financial market describes the whole data. In the same time, a stream for sensors depends on sampling (e.g. get new data every 5 minutes).

The general approach is to have a small component that processes each of the events separately. In order to speed up the processing, the stream may be subdivided, and the computation distributed across clusters. Stream processing frameworks primarily addresses parallelization of the computational load; an additional storage layer is needed to store the results in order to be able to query them.

Benefits of stream processing:

- Accessibility: live data can be used while still in motion, before being stored.
- Completeness: historical data can be streamed and integrated with live data for more context.
- High throughput: large volumes of data can be processed in high-velocity minimal latency.

To introduce a more formal expression, a data stream is described as an ordered pair (S, T) where:

- S is a sequence of tuples.
- T is a sequence of positive real time intervals.

It defines a data stream as a sequence of data objects, where the sequence in a data stream is potentially unbounded, which means that data streams may be continuously generated at any rate [Nam15] and leads to the following characteristics:

- the data arives continous
- the arrival of data is disordered
- the size of the stream is potentially unbounded

After this short introduction to the basics of stream processing, the following sections covers a short introduction of the streaming frameworks Apache Flink and Apache Kafka.

2.3.1 Apache Flink

As described in the documentation [Fli16], *"Apache Flink is an open source platform for distributed stream and batch data processing. Flink's core is a streaming dataflow engine that provides data distribution, communication, and fault tolerance for distributed computations over data streams. Flink also builds batch processing on top of the streaming engine, overlaying native iteration support, managed memory, and program optimization."*

The main components of Flink applications are formed by streams and transformations, in which streams define intermediate results whereas transformations represent operations computed on one or more input streams with one or more resulting streams.

To illustrate the main components of a Flink application, the following code from [Fli16] shows a working example of a streaming application, that counts the words coming from a web socket in 5 second windows:

```
public static void main(String[] args) throws Exception {
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
        DataStream<Tuple2<String, Integer>> dataStream = env
                .socketTextStream("localhost", 9999) *(1)
                .flatMap(new Splitter()) *(2)
                .keyBy(0) *(2)
                .timeWindow(Time.seconds(5)) *(2)
```

```
                .sum(1); *(2)

        dataStream.print(); *(3)
        env.execute("Window WordCount");
    }

    public static class Splitter implements FlatMapFunction<String, Tuple2<String, Integer>> {
        @Override
        public void flatMap(String sentence, Collector<Tuple2<String, Integer>> out) throws Exception {
            for (String word: sentence.split(" ")) {
                out.collect(new Tuple2<String, Integer>(word, 1));
            }
        }
    }
```

Codeauszug 2.1: Basic Apache Flink streaming application

On execution, Flink applications are mapped to streaming dataflows, consisting of streams and transformation operators (3) where each dataflow starts with one or more sources (1) the data is received from and the resulting stream will be written in one or more sinks (3). to.

The following overview summarises the most important streaming features of Apache Flink:

Event time and out of order streams Due to the distributed character of Big Data Analytics Applications, data doesn't arrive necessarily in the order that they are produced. Since version 0.10, Flink provides the concept of "event time", the processing of events by the time they happened in the real world to support "out of order" streams what enables consistently processing of events according to their timestamps.

Windows Flink uses a concept called windows to divide a data stream that can be potentially infinite into finite slices based on the timestamps of elements or other criteria. This division is required when working with infinite streams of data and performing transformations that aggregate elements

Consistency, fault tolerance, and high availability Flink guarantees consistent state updates and data movement in the presence of failures between selected sources and sinks. Such failures include machine hardware failures, network failures, transient program failures, etc. It supports worker and master failover, eliminating any single point of failure by providing a checkpointing mechanism that recovers streaming jobs after failures.

Connectors and integration points Flink integrates with a wide variety of open source systems for data input and output (Kafka, Elasticsearch and others) which makes technical decisions regarding the infrastructure quite flexible.

2.3.2 Apache Kafka

Apache Kafka is publish-subscribe queuing service rethought as a distributed commit log [Kaf16], supporting stream processing with millions of messages per second, durability of messages through disk storage and replication accross multiple machines in clustered environments. It is written in Scala, was initially developed at LinkedIn and follows the distributed character of Big Data Analytics Applications by it's inherent design.

This excerpt from the paper [Jay11] the team at LinkedIn published about Kafka describes the basic principles:

A stream of messages of a particular type is defined by a topic. A producer can publish messages to a topic. The published messages are then stored at a set of servers called brokers. A consumer can subscribe to one or more topics from the brokers, and consume the subscribed messages by pulling data from the brokers. (...) To subscribe to a topic, a consumer first creates one or more message streams for the topic. The messages published to that topic will be evenly distributed into these sub-streams. (...) Unlike traditional iterators, the message stream iterator never terminates. If there are currently no more messages to consume, the iterator blocks until new messages are published to the topic.

A common use case for Apache Kafka in the context of stream processing is the buffering of messages between stream producing systems by providing a queue for incoming and outgoing data. According to the explanation of the concept of data sources and sinks in the section above, Apache Kafka is heavily used as an input source, as well as output sink for the Apache Flink processing dataflow.

The design of Apache Kafka has the following characteristics:

Persistent messages To avoid any kind of data loss, Apache Kafka provides constant-time performance even with very large volumes of stored messages, which is in order of Terrabytes.

High throughput Even on commodity hardware, Apache Kafka supports millions of messages per second.

Distributed by design : Enables Messages partitioning over Kafka servers and distributing consumption over a cluster of message consuming machines

Multiple client support Integration of clients from different platforms, e.g. Java, .NET, PHP, Ruby, Python.

Real Time Messages produced by the producer threads should be immediately visible to consumer threads.

2.4 Summary

TODO

3 Requirements Analysis and Specification

After a short introduction to the basic concepts of Big Data, Big Data Analytics Applications and Apache Flink and Apache Kafka as examples for streaming frameworks, this chapter examines what different kind of data are available for both of the systems and should be transported to a central storage system. According to the results of the data analysis, the functional and non-functional requirements of the software system which is forming the core of the present thesis will be defined.

3.1 Data Analysis

live and historical sources

"COLLECT EVERYTHING!

3.1.1 System data

Observation of cpu-, disk- and memory-utilization, why. Dstat system util introduction

3.1.2 Application data

Apache Flink provides application data via Monitoring REST API, describe REST Since version 1.1.0 new Metrics data via JMX

Representational State Transfer (REST)

Analyze Flinks REST data

Java Management Extensions (JMX)

General JMX explanation, data access MBeansServerConnection

3.2 Data Quality

Define DQ, evaluate quality for data above

3.3 Functional Requirements

Describe "big picture" functionality see [Les14], follows distributed character of Big Data Analytics Applications, provide "on demand" data collection, as much data as possible, realtime?, three main components, break down for:

3.3.1 Collection

collect data in clustered environments

3.3.2 Transport

Scalability with message broker

3.3.3 Persistence

Accessibility for AI, UI applications

3.4 Non-Functional Requirements

3.5 Summary

4 Architecture and Implementation

Distributed system -> distributed collection, cloud environments, microservice architecture, service-discovery, communication via REST, Publish(client) -> Topic <- Subscribe Logstash,

continuous, distributed, time-series "data feed", difference raw and aggregated[Kle16], events as "immutable facts", why? arch uses both, raw because of unknown interests in data, flink-job-index for demonstration of SP.

4.1 Architecture

TODO

4.1.1 Infrastructure Components

Service-Discovery

Registraction for CollectorClients

Message-Broker

Queueing, see Marz15

Transport, "Event-Log", see [Kre13] Collect the streams and make them available for consumption

Indexer

Receive messages from Kafka, roote data, create ES index, why, describe context BDAA

Persistence

ES as search index for time-series based data, easy vizualization with Kibana, why?

4.1.2 Software Components

CollectorClient

The CollectorClient tier is our entry point for bringing data into the system... A module to gather the event streams from data sources.

CollectorManager

Gives overview, uses Consul as service-discovery

CollectorDataProcessor

module to analyze the streams creating derived streams and persist flat data -> data transformation

4.2 Implementation

Introduce software stack, why used?

4.3 The "collect"-algorithm

Java8, CPs, non-blocking streams

4.4 Summary

Maybe Spring alternatives, Lagom, VertX, Play? Maybe collector as agent instead of microservice, alternatives REST, maybe (Web-)Sockets Instrumentation alternative

5 Evaluation

5.1 Local test environment

5.2 Docker environment

5.3 Observations

5.4 Discussion

5.5 Summary

6 Conclusion

TODO

6.1 Summary

6.2 Outlook

List of Figures

List of Tables

List of Source Codes

Bibliography

[Fli16] Flink. *Apache Flink Documentation.* 2016. URL: `http://flink.apache.org` (visited on 08/18/2016).

[Gro14] Cloud Security Alliance - Big Data Working Group. *Big Data Taxonomy.* 2014. URL: `https://downloads.cloudsecurityalliance.org/initiatives/bdwg/Big_Data_Taxonomy.pdf` (visited on 08/18/2016).

[Jay11] Jun Rao Jay Kreps Neha Narkhede. *Kafka: a Distributed Messaging System for Log Processing.* 2011. URL: `http://research.microsoft.com/en-us/um/people/srikanth/netdb11/netdb11papers/netdb11-final12.pdf` (visited on 08/21/2016).

[Jör14a] u.a Jörg Bartel Axel Mester. *Big Data Technologien – Wissen für Entscheider.* 2014. URL: `https://www.bitkom.org/Publikationen/2014/Leitfaden/Big-Data-Technologien-Wissen-fuer-Entscheider/140228-Big-Data-Technologien-Wissen-fuer-Entscheider.pdf` (visited on 08/06/2016).

[Jör14b] u.a Jörg Bartel Dr. Bernd Pfitzinger. *Big Data im Praxiseinsatz – Szenarien, Beispiele, Effekte.* 2014. URL: `https://www.bitkom.org/Publikationen/2012/Leitfaden/Leitfaden-Big-Data-im-Praxiseinsatz-Szenarien-Beispiele-Effekte/BITKOM-LF-big-data-2012-online1.pdf` (visited on 08/06/2016).

[Kaf16] Kafka. *Apache Kafka Documentation.* 2016. URL: `http://kafka.apache.org` (visited on 08/18/2016).

[Kle16] Martin Kleppmann. *Making Sense of Stream Processing.* First edition. Sebastopol, CA 95472: O'Reilly Media, Inc., 2016. ISBN: 978-1-491-94010-5.

[Kre13] Jay Kreps. *The Log: What every software engineer should know about real-time data's unifying abstraction*. 2013. URL: `https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying` (visited on 08/18/2016).

[Les14] Tammo van Lessen. "Wissen, was läuft - Mit Laufzeitmetriken den Überblick behalten". In: *Javamagazin* 10000.11 (2014), pp. 48–52.

[Nam15] Dmitry Namiot. "On Big Data Stream Processing". In: *International Journal of Open Information Technologies* 1 (2015), pp. 48–51.

[ORe13] Tim O'Reilly. *George Dyson's Definition of "Big Data"*. 2013. URL: `https://plus.google.com/+TimOReilly/posts/Ej72QmgdJTf` (visited on 08/18/2016).

Image resources

[Nat15] James Warren Nathan Marz. *Big Data - Principles and best practices of scalable real-time data systems*. Shelter Island, NY 11964: Manning Publications Co., 2015. ISBN: 978-1-617-29034-3.

A

A.1 Diagrams

A.1.1 Use Case diagram

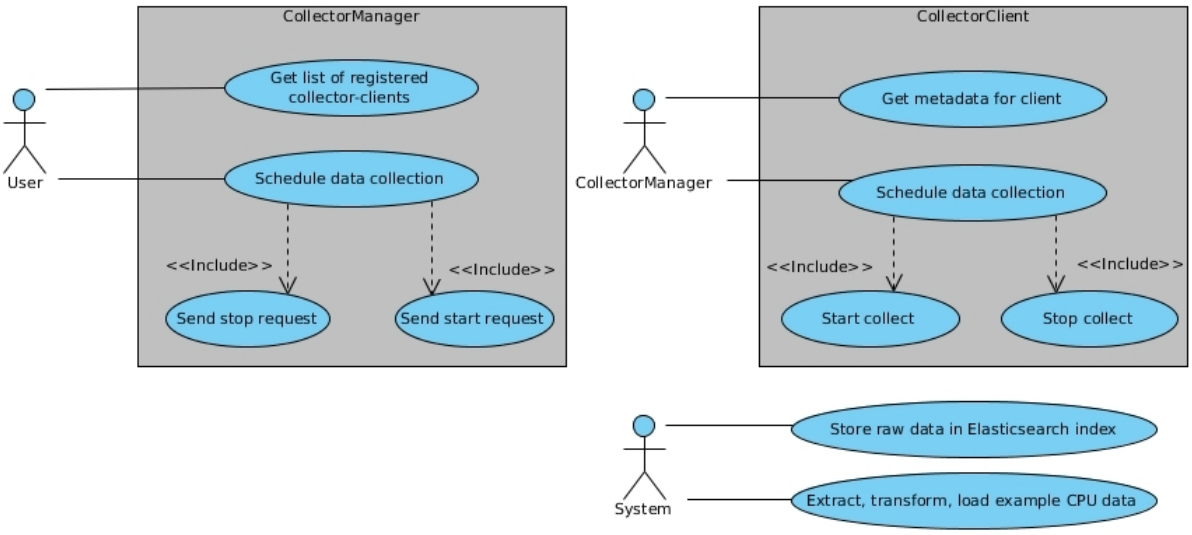

Figure A.1: Use Case Diagramm

A.1.2 Class diagrams

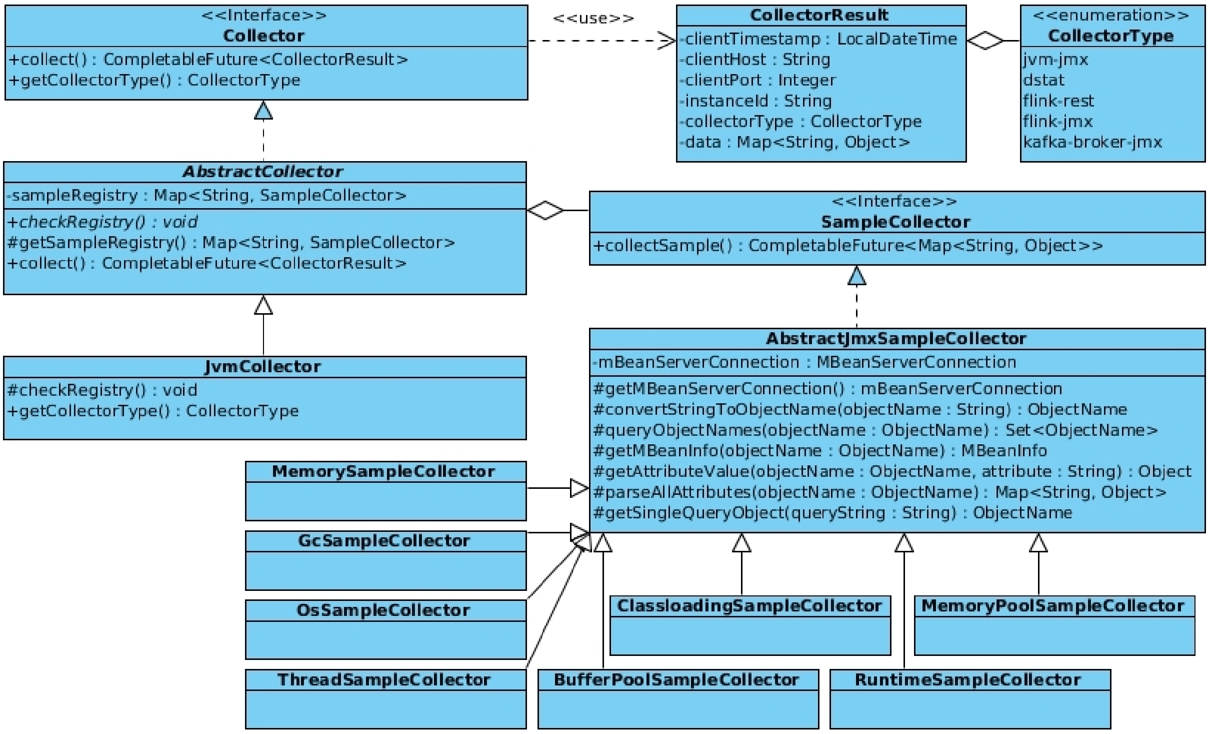

Figure A.2: Class diagram 'JvmCollector'

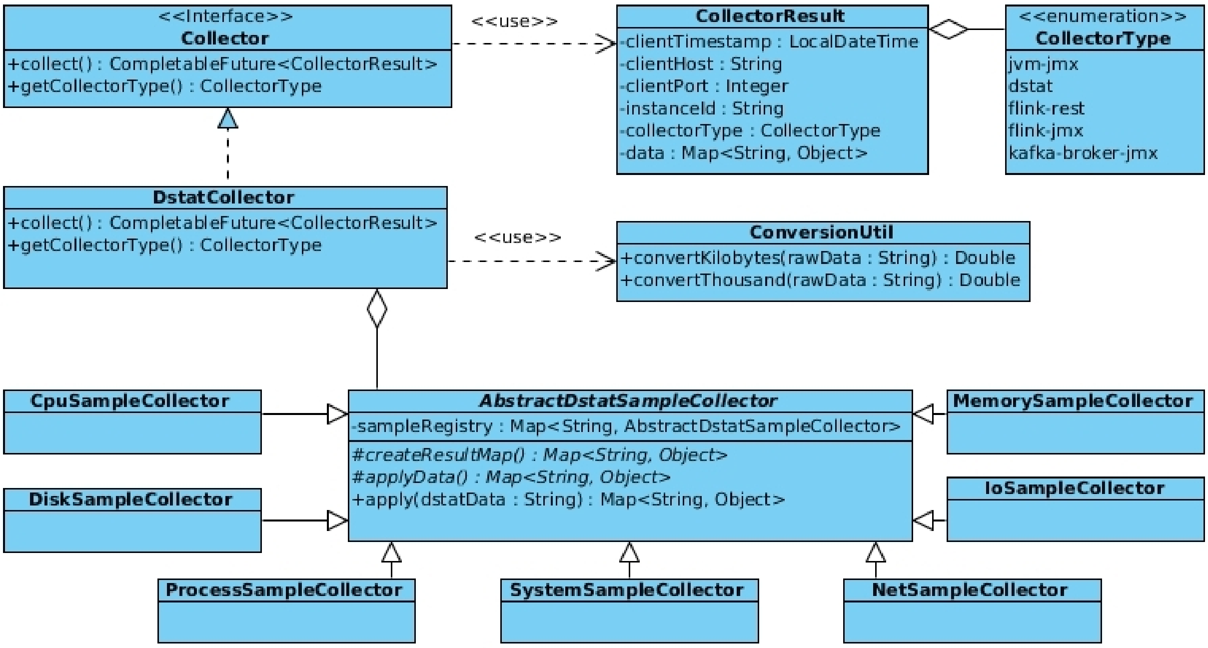

Figure A.3: Class diagram 'DStatCollector'

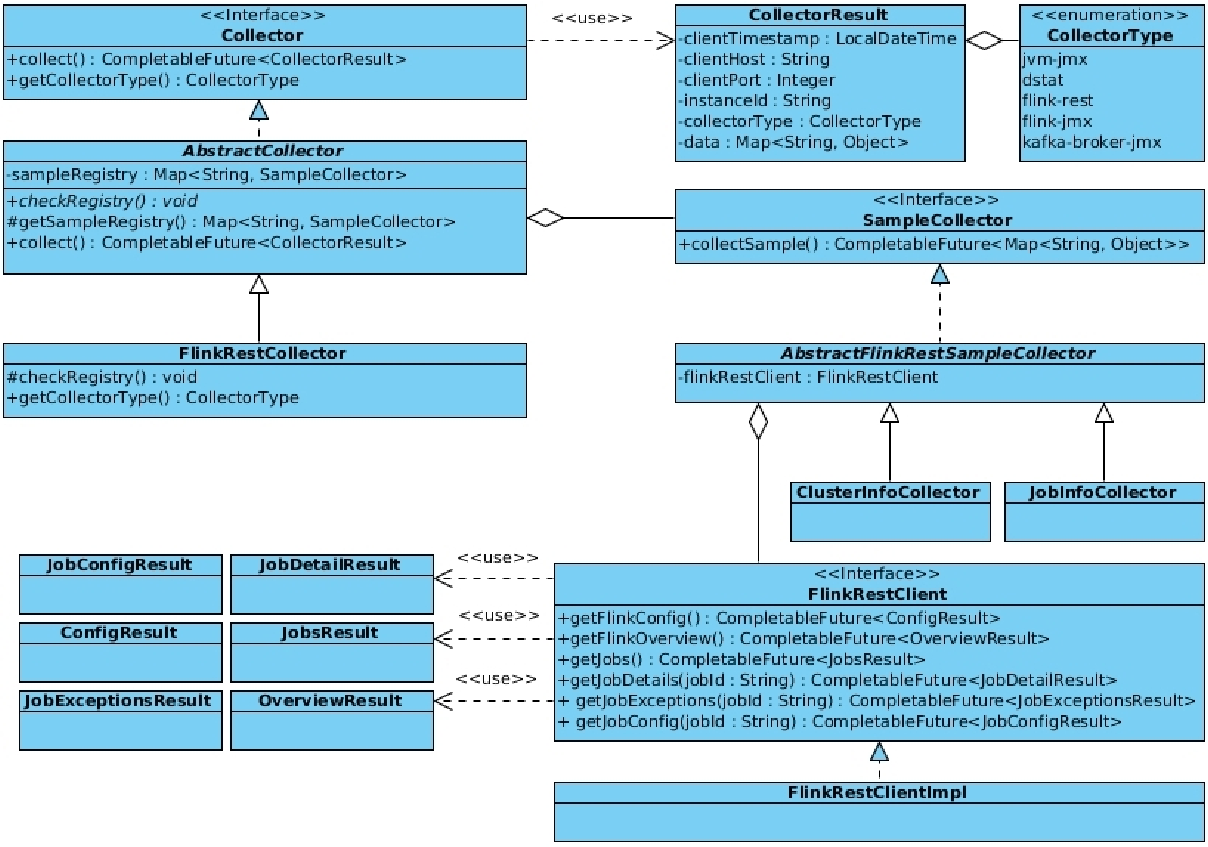

Figure A.4: Class diagram 'FlinkRestCollector'

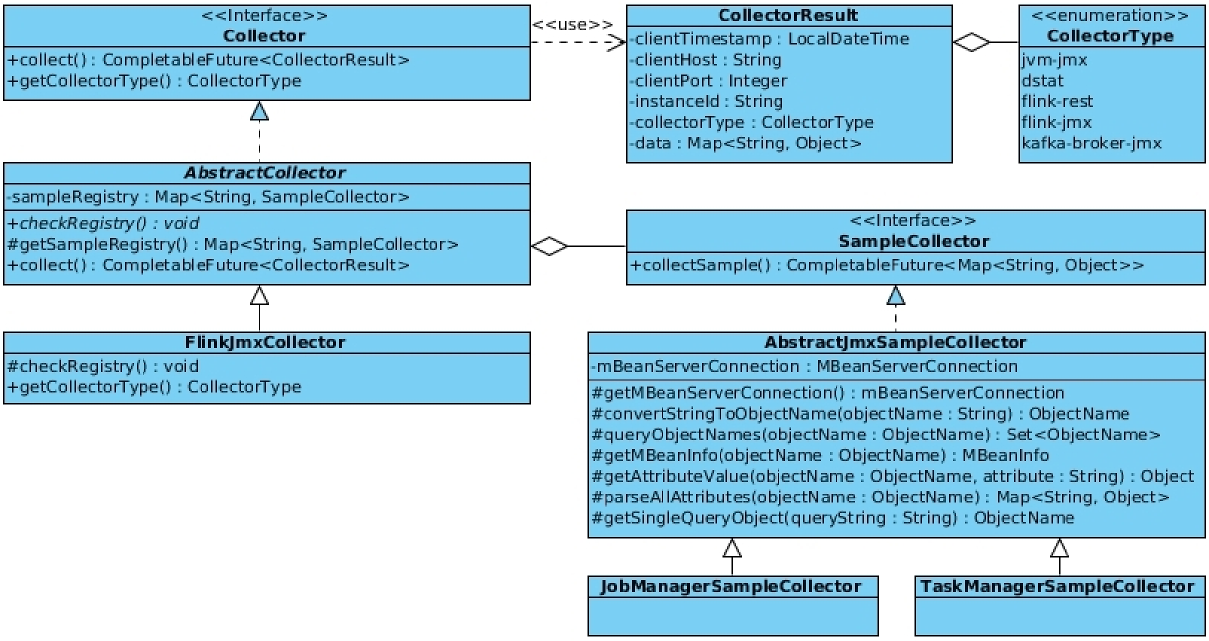

Figure A.5: Class diagram 'FlinkJmxCollector'

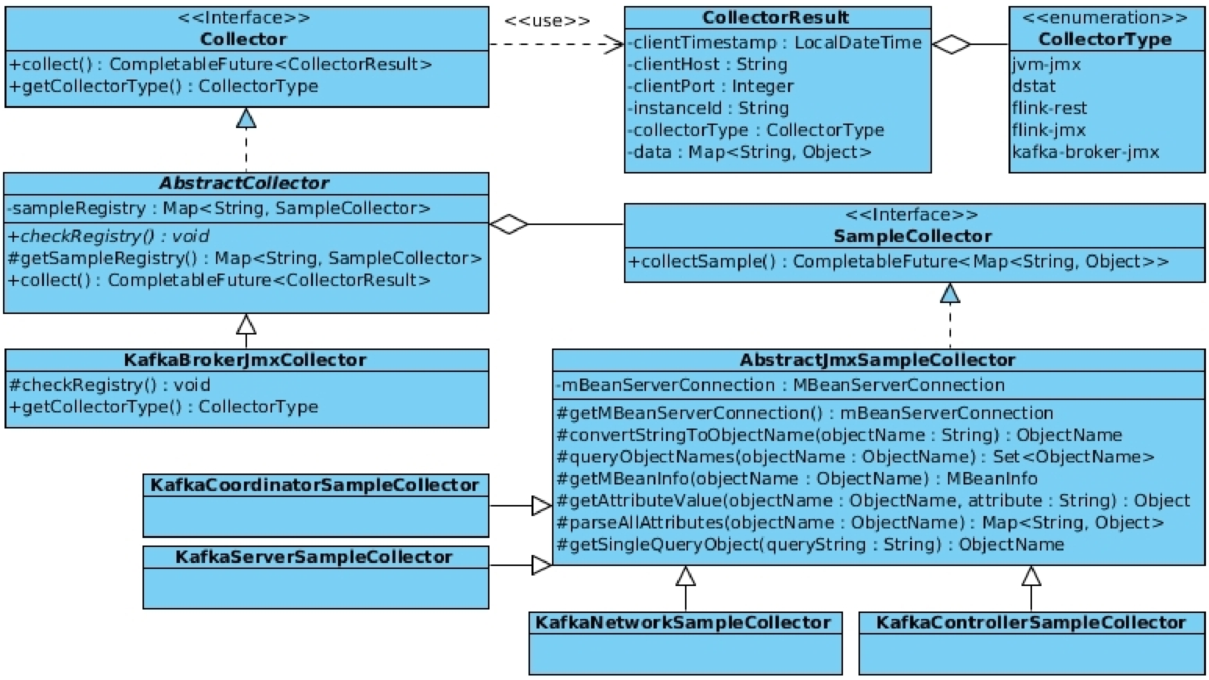

Figure A.6: Class diagram 'KafkaBrokerJmxCollector'

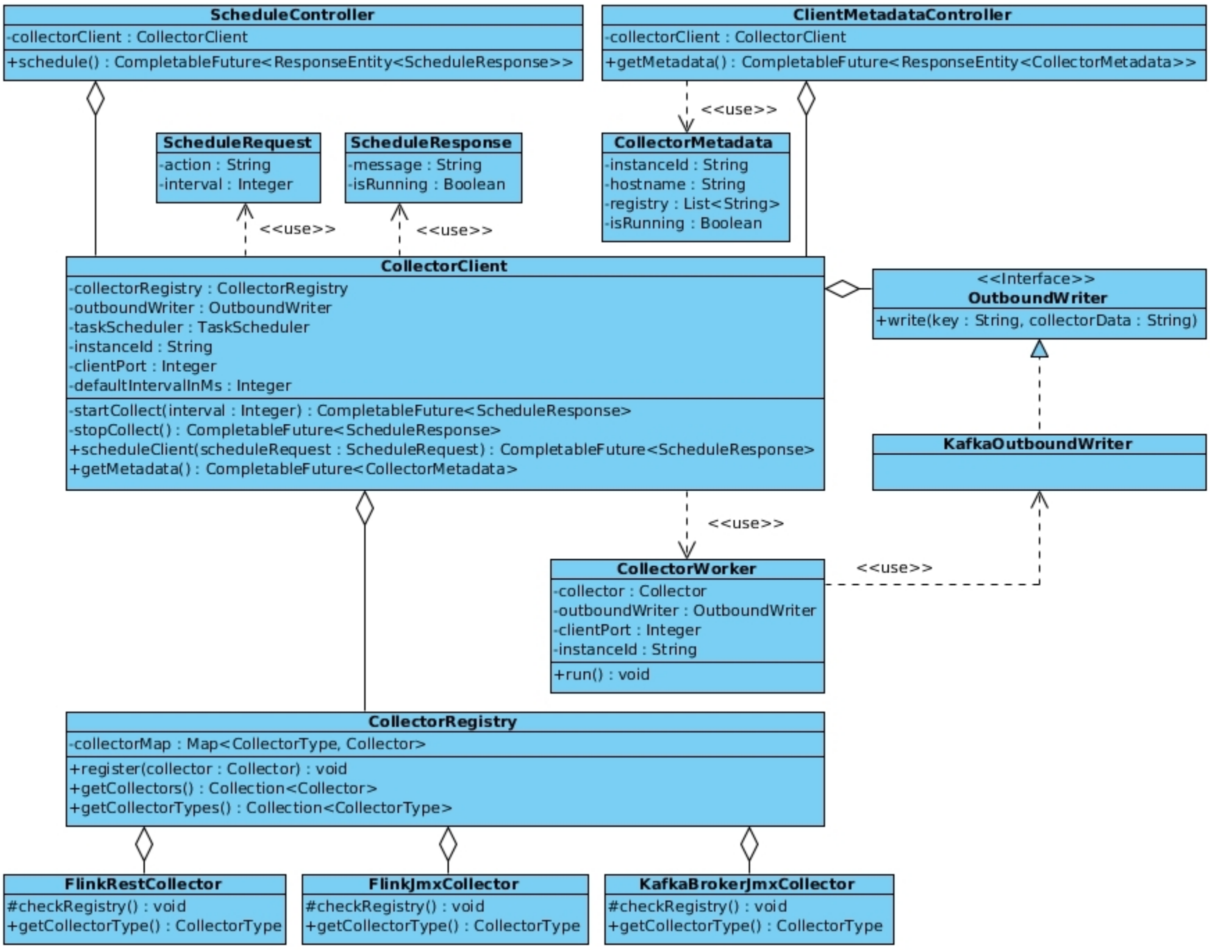

Figure A.7: Class diagram 'CollectorClient'

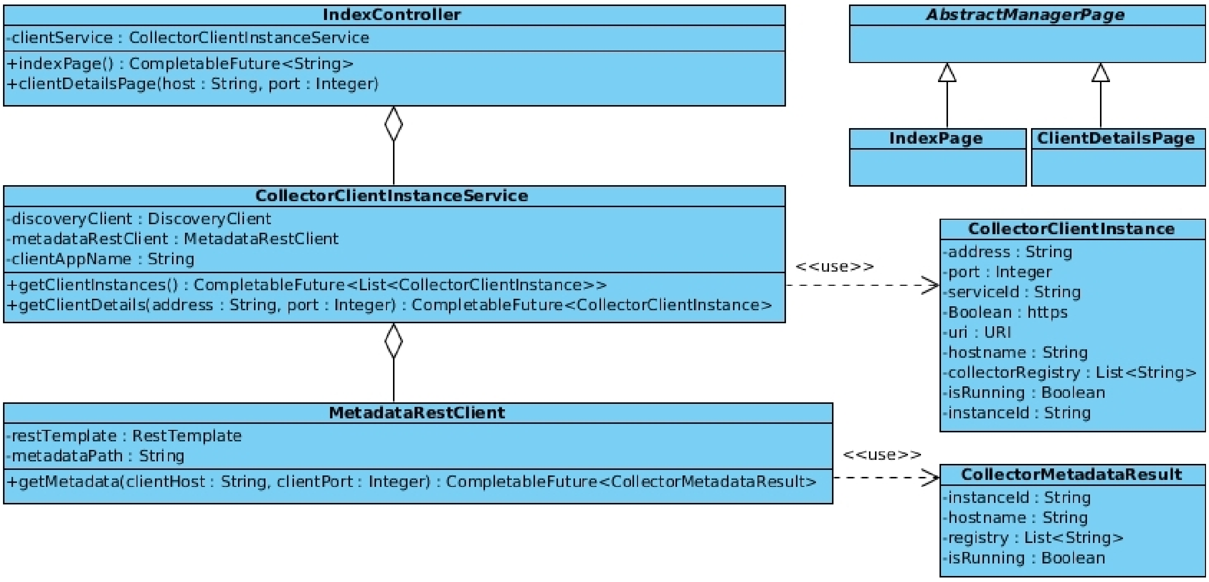

Figure A.8: Class diagram 'CollectorManager'

A.1.3 Sequence diagrams

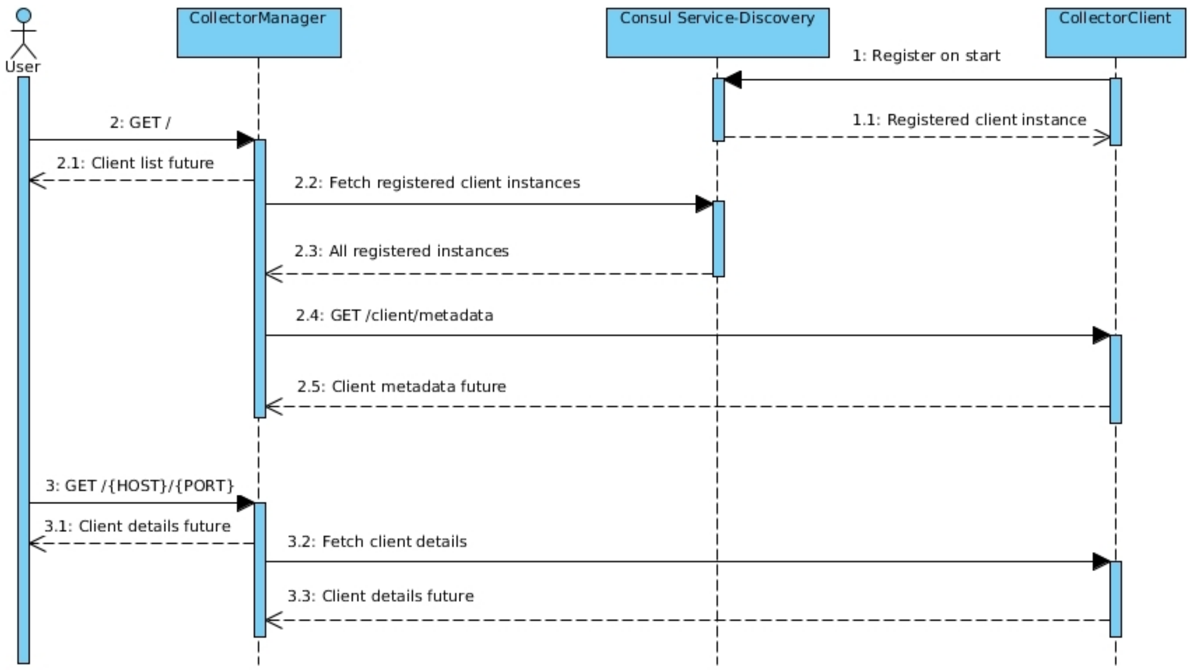

Figure A.9: Sequence diagram 'Client discovery'

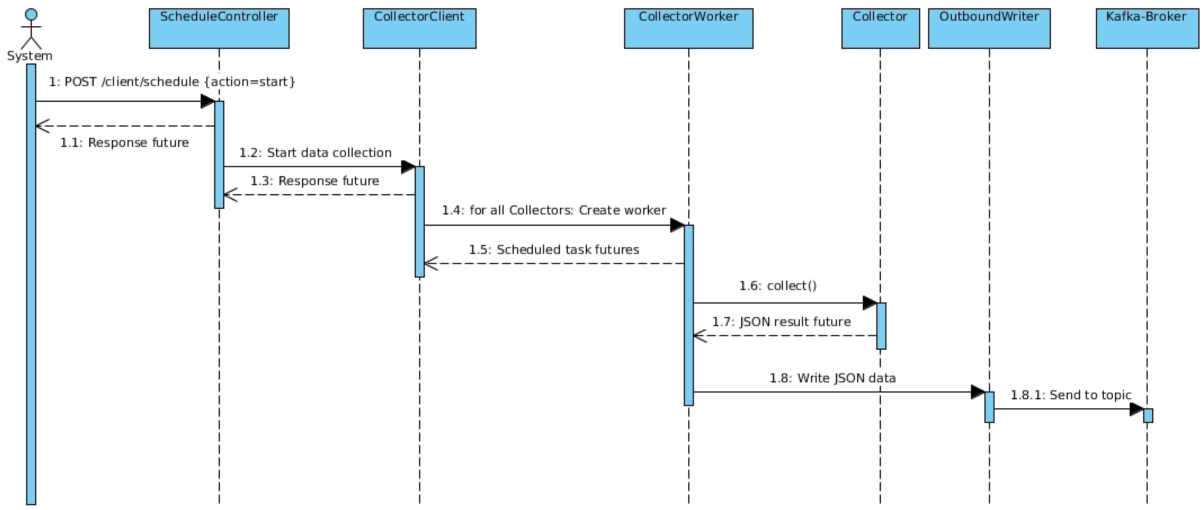

Figure A.10: Sequence diagram 'Client scheduling'

A.1.4 Component diagram

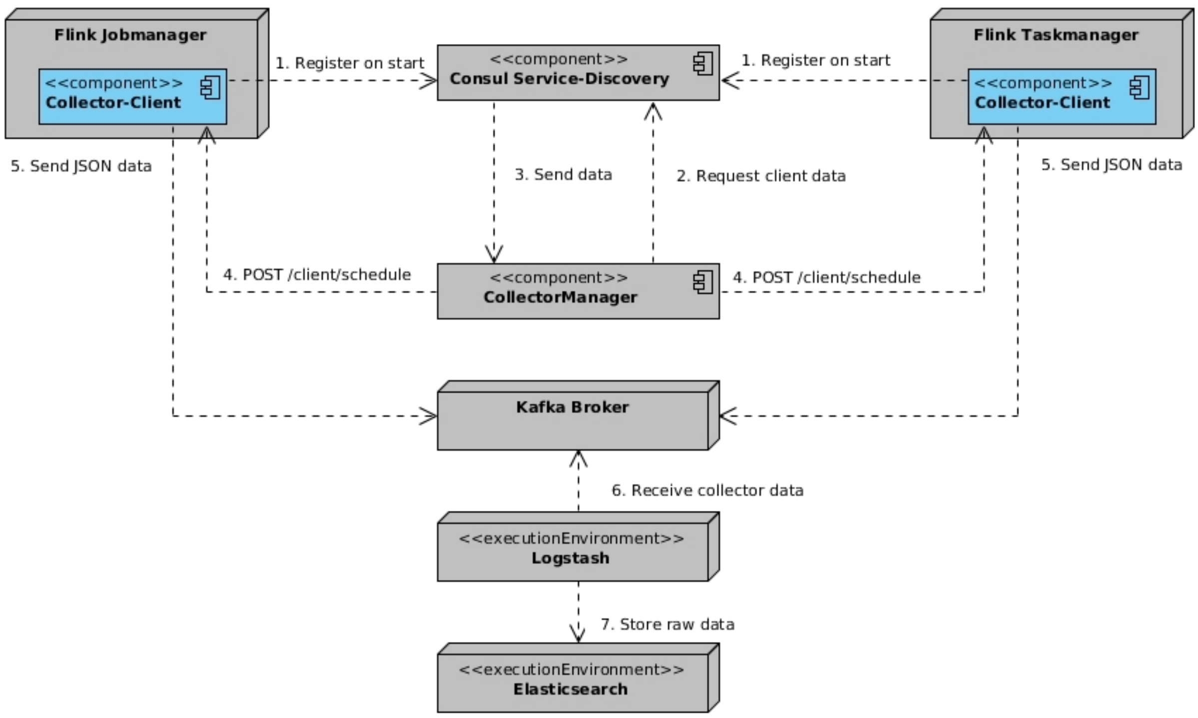

Figure A.11: Component diagram

A.1.5 Deployment diagram

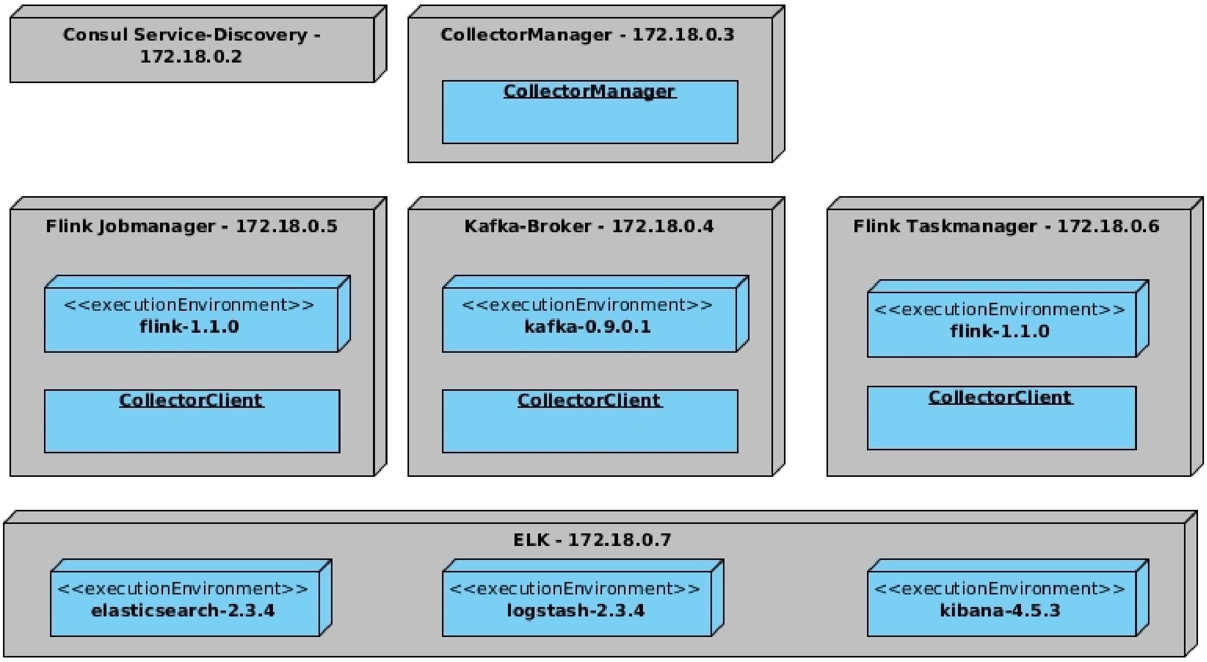

Figure A.12: Deployment diagram

A.2 Tabelle

A.3 Screenshot

A.4 Graph

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Stadt, den xx.xx.xxxx

Max Mustermann