Ministry of Higher Education And Scientific Research
Tunis ElManar University
Faculty of Science of Tunis
Department of Computer Science

Moving From a Lambda Architecture Toward a Kappa Architecture in a Transport Processing Data Pipeline

Submitted in partial fulfillment of
the requirements for the award of the degree of

Computer Engineering

Submitted by

Anouer Hermassi

sentiance
Powering the Internet of You

Under the supervision of

Mr. Christoph Liekens, Sentiance

Mr. Sadok Ben Yahia, Faculty of Science of Tunis

July 2018

Acknowledgements

I would first like to thank my project supervisor, Mr. Christoph Liekens, for his guidance during my time at Sentiance. He never gave up on me even when I was discouraged. I have learned a lot from him, and he is and will remain a true inspiration for the rest of my career.

I would like to express my sincere gratitude to my teacher, Mr. Sadok Ben Yahia, for his continuous encouragement and support and his belief that I can reach my goals.

I would like also to show my deepest appreciation to Mr. Bill Rambin, my cultural advisor during my exchange year in the U.S. I learned from him that life is more meaningful when it is driven by passion.

Finally, there are many more people who deserve thanks. So to everyone who helped support me, knowingly and unknowingly, directly and indirectly, you have my appreciation.

Dedications

To my parents and my sister, for their endless love.

To Hamza, for 15 years of laugh and counting.

To Stack Overflow, for saving my semesters when everyone else let me down.

To my failures, for making me stronger.

Contents

List of Figures

Chapter 1

Introduction

The promise of Internet of Things, or IoT, is that all the objects around us are going to adapt themselves to us. They will learn our behavior and use it to really fit into our lives. If they do that, it will take away all the friction we have with our environment. Ultimately, Internet of Things will become Internet of You. We are at the center not the things.

Well, that is a great vision. Unfortunately, that is not where we are today. Installing a smart home, for instance, takes an engineering degree, and it is a full-time job. Sentiance wants to change that. It wants to be part of making the Internet of You a reality, and it believes that the answer is in the data. Not just any data, but the sensor data, the one that is emitted from all the sensors that are embedded in smart devices.

The sensors provide a constant stream of information about how people behave and why they behave the way they do. If we use the right Artificial Intelligence and Machine Learning algorithms, we can extract a lot of knowledge from sensor data. This knowledge can make the Internet of You a reality, and that is important because all of us are part of the Internet of You.

Companies of all sizes and across multiple activity domains see that knowledge a tremendous opportunity. They could be building smart home solutions that make homes smarter and safer. They could be also providing fabulous content on their platforms, and we as consumers expect that content to completely fit into our lives. In all cases, companies need to understand their consumers when they are interacting with them.

Sentiance developed an Artificial Intelligence platform that takes in smartphone sensor data and derives behavior intelligence from it allowing

the contextualization IoT. The scope of this dissertation is an initial effort for moving from a data processing architecture that doesn't provide enough accurate real-time insights, the Lambda architecture, towards a Kappa architecture where real-time is at the heart of business.

We start by giving more context to the project. We then walk through the state-of-the-art technologies and methods with regard to data processing. After that, we provide a detailed analysis of the requirements of our project. We proceed with the design and implementation attributes of the solution we proposed, and we finish with the results of our work and a couple of suggested improvements.

Chapter 2

Project Context

The goal of this chapter is to understand the context of the project with regard to the problem it tackles as well as the company where the project was carried out.

2.1 Project Background

The project can be better understood when presented in its broader context. Sentiance makes use of a data processing model called Lambda architecture which, sure enough, has its advantages as well as its drawbacks. A different way to process data is the Kappa architecture, where basically results can be derived in a more real-time fashion. The definitions of and differences between both of these architectures will come in a later section.

Efforts have been made within Sentiance to move from Lambda to Kappa architecture. The project we are about to present falls in the context of this movement.

2.2 Presentation of the Host Company

2.2.1 What Is Sentiance

Sentiance is a Data Science company turning IoT sensor data into rich insights about people's behavior and real-time context. These insights enable companies to understand how customers go through their everyday lives, discover and anticipate the moments that matter most, and adapt their engagement to real-world behavior and real-time context.

As the Internet of Things gains momentum consumers are rapidly adopting smart objects in their everyday lives. The sensors in these IOT devices generate a constant stream of data about how they behave and why they behave the way they do. Sentiance helps companies make sense of this new and very powerful data. Some of the world's most innovative companies are adopting the Sentiance platform to help them make their customers safer, smarter, healthier and ultimately more profitable.

2.2.2 How Sentiance Works

Sentiance's Artificial Intelligence-based solution learns to reason about human behavior, thereby moving from traditional activity detection to cognitive computing.
The Sentiance platform ingests four data points from the mobile device: GPS, Accelerometer, Gyroscope and Time. This raw data is then translated in three levels of intelligence about a user: Events, Moments and Segments. This is represented in the pyramid below.

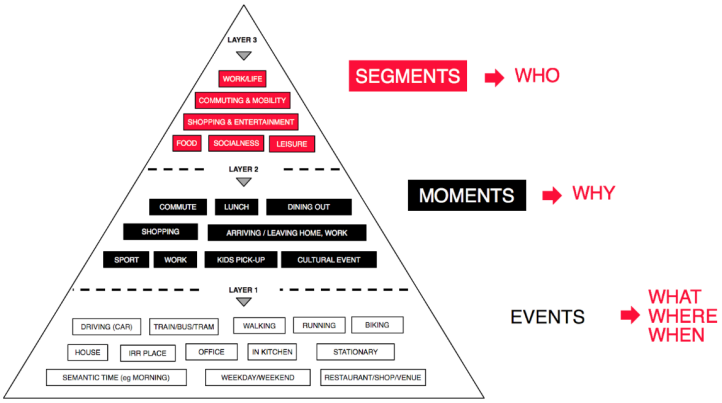

Figure 2.1: Sentiance Pyramid

Events

Events allow Sentiance to sense and interpret the contextual cues of a user. The four main areas we focus on at this level are:

1. Transport classification to detect the mode of transport of the user (walking, biking, car, train, subway, etc.)

2. Automatic home and work detection which gives a baseline of behavioral patterns

3. Semantic time detection where personalized day parts (morning, lunch, evening) are established not only based on time but also taking into account personal routines

4. Geospatial analysis: Route matching and venue mapping based on probabilistic models applied to GPS data.

Moments

Many solutions on the market today directly tie Events to context. Sentiance believes that Events only become relevant when they can be mapped to intent. This means that mobile Events need to be translated to meaningful Moments, reflecting real-life and real-time context and routines.

For example, a sequence of Events such as [walking, shop, walking, shop, walking, coffee shop, walking] becomes interesting only when we assign it a label such as "Afternoon Shopping." If we know that the user goes shopping almost every Saturday afternoon, we can further augment the Moment definition as "Afternoon Shopping Routine."

Another example would be the Event sequence [home, running]. "Running" itself is merely an Event without much meaning. While one user might be in his weekly sporting routine, another might just be running to catch the bus when he is late for work. To put this sequence of Events in context, we need to be able to predict the next most likely Event which might for instance be "At Work."

A Moment is therefore a short-term aggregation of recent Events, combined with predictions that model the user's expected behavior.

Segments

Mapping the Moments in a user's day allows Sentiance to reconstruct the user's timeline based on the mobile Events, as shown in the picture below:

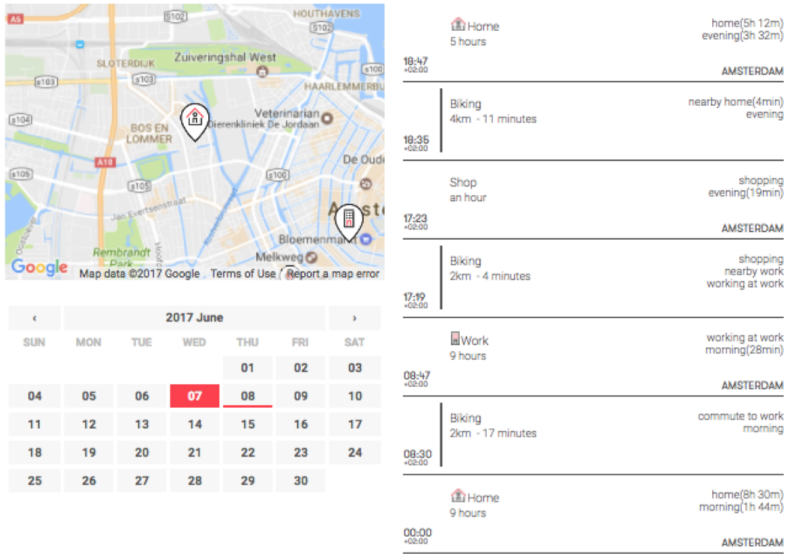

Figure 2.2: User Timeline

This timeline allows Sentiance to define user Segments by clustering users into related groups. Example Segments are for instance "Workaholics" and "Green Commuters." The complete set of Segments a user belongs to makes up his profile.

The following Segments are automatically assigned to each user:

- **Work/lifestyle Segments** - City home, City worker, Early bird, Homeworker, Night worker, Rural Home, Rural Worker, Shopaholic, Town Home, Town Worker, Workaholic, Work Traveler, Full-time Worker, Part-time Worker, Student/Teacher, Uber Parent
- **Geography Segments** - Lives in city_name, Works in city_name
- **Affinity Segments** - Brand Loyalty, Brand Loyalty: Gas Stations, Brand Loyalty: Supermarket, Clubber, Culture Buff, Dog Walker, Quality Time at Home, Restaurant Lover, Sportive
- **Mobility Segments** - Die Hard Driver, Easy Commute, Green Commuter, Heavy Commuter, Long Commute, Normal Commuter, Short Commuter
- **Driver Segments** - Aggressive Driver, Anticipative Driver, Efficient Driver, Illegal Driver, Legal Driver

2.2.3 Who Are Sentiance Clients

The Sentiance platform has a variety of applications and is currently focused on four main business categories: Mobility, Health, Commerce, and Living.

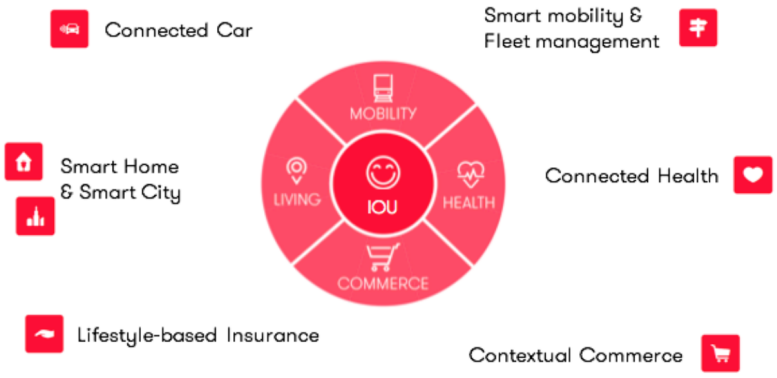

Figure 2.3: Sentiance Platform Use Cases

Scoring Driver Performance for the World's Leading Ridesharing Company
The worlds leading ride sharing company uses the Sentiance platform to score the driving behavior of their drivers based on the sensors in the drivers' smartphones. These sensors measure how smooth, anticipative and legal they drive. And they measure how distracted drivers are by analyzing the movement of the smartphone. This information is used to coach drivers and make them drive more responsibly.

Personalizing In-Car Experiences
PSA/Peugeot used the Sentiance platform to detect the profile and context of drivers before they enter the vehicle. This allowed them to tailor the in-car experience accordingly. Sentiance technology was featured as part of PSA/Peugeot concept car INSTINCT at the 2017 Mobile World Congress.

Supporting the World's Leading Manufacturer of Car Seating and Interiors
Sentiance technology is being used by Faurecia to better understand how factors such as work/life routines, driving behavior, biometrics and weather can adjust in-car experiences. Finding from this ongoing research is used to further develop Faurecia products and services that are used by every major

automotive OEM.

Powering User-Based Insurance (UBI) Programs
RISK (Netherlands) has partnered with Sentiance to introduce the next generation of User Based Insurance (UBI) by incorporating behavioral and contextual data. While the current generation of UBI products is focused on analyzing only car data during a trip, the Sentiance solution tracks people's driving behavior and context by analyzing smartphone sensor data.

Advancing Digital Health through IoT Technology
Sentiance is working with Samsung and Nestle to help pre-diabetes patients eat healthier. The Sentiance platform analyzes sensor data from mobile devices and Samsung wearables. This combination of data is used to identify a patient's eating habits, lifestyle and general routines. The insights form the foundation of personalized nutrition and coaching plans delivered by Nestle's team of nutrition scientists.

Transforming Loyalty Experiences in Commerce
Sentiance is working with a leading grocery retailer in the UK, looking into how the real-time customer context derived from Sentiance platform can enhance overall customer experience, commerce buy flow as well as loyalty program design. Sentiance is also working with world's leading coffee retailer to enhance their mobile order and pay application, and a leading U.S. grocery retailer where Sentiance will analyze customer shopping behaviors and routines and win share of wallet against competitors.

Creating a Contextual Smart Home Hub
Samsung is working with Sentiance to turn their Artik Cloud's open Smart Home platform into a contextual hub. By analyzing signals from home motion and security sensors, Sentiance creates a contextual map of people's homes, determining room adjacency and function (kitchen vs. living room vs. bedroom). The algorithms also learn the household's activity patterns in each room (e.g., cooking, watching TV, showering, cleaning, etc.). This intelligence can be used to reduce the number of false alarms, regulate temperature based on how inhabitants live in the house and help smart devices in the house adapt to the people living there.

2.3 Project Context

2.3.1 Project Statement

The project of the present dissertation involves assisting in the refactoring of one of the sequential data processing pipelines at Sentiance towards a more efficient, scalable processing mesh based on Reactive design principles. The data processing pipeline itself is focused on sensor processing for mobility purposes, for example transport classification, map matching, and driving behavior. The goal of the new processing mesh is to achieve a more cost-effective, more maintainable data processing environment.

2.3.2 Definitions

Reactive Systems

In 16 September 2014, the Reactive Manifesto was published by a group of developers and software architects. This document can be seen as a dictionary of best practices, some of which have been known for a long time. According to Roland Kuhn, one of the early authors of the Reactive Manifesto [1]

> The intention behind the Reactive Manifesto [...] is to condense the knowledge we accumulated as an industry about how to design highly scalable and reliable applications into a set of required architecture traits, and to define a common vocabulary to enable communication about these matters between all participants in that process—developers, architects, project leaders, engineering management, CTOs.

The highly scalable and reliable applications Kuhn describes are what we call Reactive Systems.

The Reactive Manifesto describes Reactive Systems as systems that are: [2]

- **Responsive:** The system responds in a timely manner if at all possible, providing rapid and consistent response times.
- **Resilient:** The system stays responsive in the face of failure.
- **Elastic:** The system stays responsive under varying workload. It can react to changes in the input rate by increasing or decreasing the resources allocated to service these inputs.

- **Message driven:** The system relies on asynchronous message-passing. A message is an item of data that is sent to a specific destination, which means that recipients await the arrival of messages and react to them, otherwise lying dormant.

As of June 2018, more than 22,000 software developers signed the Reactive Manifesto.

Lambda Architecture

The Lambda architecture arose from a blog post by Nathan Marz, the author of the streaming processing framework Apache Storm, back in 2011. The idea is to split complex data processing systems into a real-time component, or speed layer, and a batch layer.
The batch layer computes in-depth results by reading in large batches of data from the dataset. The speed layer computes preliminary real-time results to compliment the in-depth results from batch layer.

An example of a Lambda architecture looks like the following [3]:

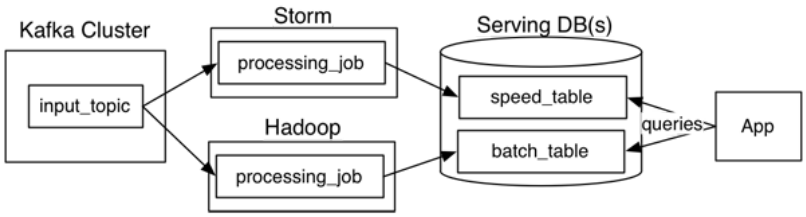

Figure 2.4: Example of a Lambda Architecture

In this example, records of data in a cluster of machines are fed into a batch system (Hadoop) and a stream processing system (Storm) in parallel. The business logic is implemented twice, once in the batch layer and once in the stream processing layer. The serving layer provides answers to queries from both the results of batch jobs and streaming jobs.
Hadoop and Storm are presented in details in a later section.

Kappa Architecture

The term Kappa architecture was coined in July 2014 by Jay Kreps, the author of the messaging system Apache Kafka, in his excellent article

"Questioning the Lambda Architecture" [3]. In this architecture, all the processing occurs in the speed layer, and every reprocessing that was done at the batch layer is simply achieved by running another instance of the streaming application.

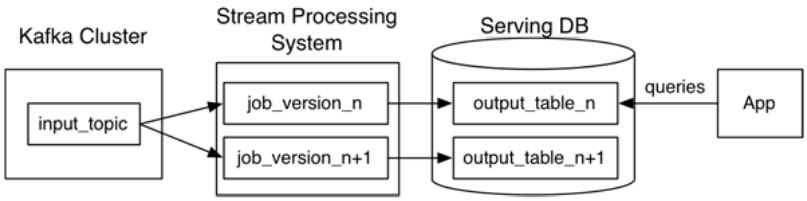

Figure 2.5: Example of a Kappa Architecture

Aquifer

Aquifer is a name for the back-end platform of Sentiance once it will be fully migrated into a reactive platform. The name refers to an underground lake where water can be extracted from several local wells on the surface; as opposed to usual big (data) lakes that act as a single access store for all collected data.

The reactive layer is the streaming part of Sentiance platform that computes results as soon as the data arrives. It basically consists of a set of loosely coupled microservices that use Kafka as a communication layer. The difference with speed layer is that the results are final, without a batch layer that overrules the results after a while.

2.3.3 Problematic

In a Lambda architecture, currently used at Sentiance, the system is modeled according to two pipelines. One pipeline, the speed layer, where a shallow analysis is performed yielding very rough results as a first pass which are presented to the user. The second pipeline, the batch layer, is where the deep analysis is performed which takes a whole lot longer. All the results of calculations are put in a serving layer, basically a key-value store. A key point is that the batch layer always recalculates everything in the master dataset after data is accumulated for hours.

This architecture has a very big advantage for Sentiance which is that it no longer has to deal with errors from the past. If a mistake is made

somewhere in an algorithm or an improvement was made to an algorithm, simply deploying it fixes the errors because everything in the master dataset is recalculated in the batch layer.

However, in such an architecture, everything needs to be implemented twice. Once in the speed layer and once in the batch layer, resulting in two systems to maintain and eventually same code base in both of these systems. The way Sentiance tried to solve this overhead is by abstracting away the code of batch and speed layers behind microservices. The advantage of this approach lies in the ability of using the same functionalities from both layers without having to implement everything twice.

Furthermore, programming in distributed environments using distributed frameworks is complex. The resulting operational complexity of systems implementing the Lambda Architecture is agreed on by everyone doing it.

2.3.4 Solution

In an attempt to mitigate the complexities introduced by the Lambda architecture, and in effort to provide its clients with more real-time analysis results about the behavior of the their customers, Sentiance started to move towards a new generation of architecture. The Kappa architecture puts more emphasis on the real-time streaming processing. This led to the aquiferization initiative that will make Sentiance platform rely more on the reactive layer.

However, moving to a Kappa architecture doesn't mean getting rid of the batch layer completely. Some tasks still require long running batch jobs be performed on the dataset to derive value. An example of such task is the processing that has to be applied to a large population to yield meaningful results. For instance, one of the microservices at Sentiance produces driving scores to evaluate the driving behavior of users. Assigning a correct driving score needs to be done after looking at the large population of users and segmenting it. For this reason, a batch job is necessary.

The present dissertation highlights the tasks that were accomplished while contributing to the movement to a Kappa architecture.

2.3.5 Development Methodology

Sentiance operates in a Squad structure, where all functional development skills are attributed to specific technical squads on an as-need basis.

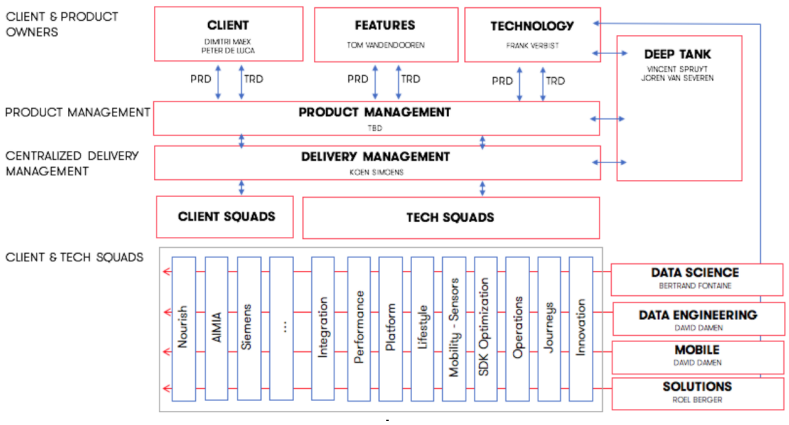

Figure 2.6: Sentiance Squad Structure

The engineering team at Sentiance is divided into 3 groups:

- Data Scientists: Who build mathematical models and come up with algorithms out of sheets of sensor measurements.
- Data Engineers: Who build the Sentiance platform and make sure it is reliable and maintainable. Mobile engineers who build the SDK are also part of the Data Engineering team.
- Solutions: Who do the technical interaction with customers and take care of the integration with the customer's system.

The project was carried out within the Mobility-Sensors, or MobSense, squad. MobSense squad's main focus is well reflected in its name. It basically works on projects that deal with transport and mobility, such as:

- Transport classification: identifying the transport mode of the user.
- Map matching: reconstructing the user's most likely trajectory.
- Driving behavior: determining how well the driver is behaving when a car trip is detected according to the driver segments.
- Phone usage: identifying if the phone is mounted (fixed) or loose (the driver is using it) during a car trip.

- Driver/passenger: detecting whether the user is driving the car or is just a passenger.

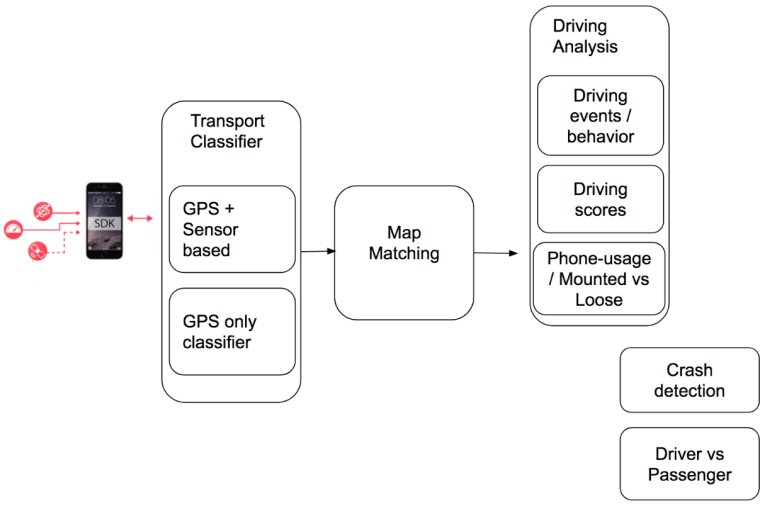

Figure 2.7: MobSense Projects

Sentiance adopts the agile methodology, more specifically Scrum. Agile is a way of executing software development project management. Agile is iterative, which means it uses tight cycles where the team works for a while, sees what it gets, and then revisits all the plans, makes as many changes as it wants, and start again. These iterations are usually called sprints.

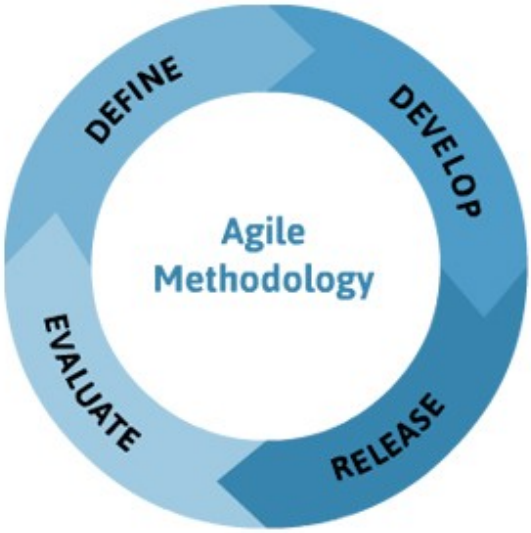

Figure 2.8: Agile Lifecycle

Agile is the total opposite of the waterfall technique, where the team writes a gigantic specification, gets is signed by the client, and then builds a blueprint out of it. Agile is also very streamlined. It favors getting to work and digging in as opposed to having tons of documentation, as well as quick meetings and short documentations.

Conclusion

In this chapter, we gave an overview of the context that defines the project. We also presented the host company and the development organization and methods that allowed us to carry out the project. In the next chapter, we will detail the different trade-offs and concepts that help better understand the broader context of the project.

Chapter 3

State of the Art

In this chapter, we will explore the most relevant concepts and technologies that provide a better context to our project.

3.1 Big Data

Sentiance collects and processes a lot of sensor data. For a microservice like Driving Behavior, 25 samples per second of accelerometer and gyroscope data is needed to be able to do the analysis with an acceptable quality. This results in billions of sensor data points collected every day. This situation has all the characteristics of a Big Data problem.

3.1.1 What Is Big Data

People constantly produce a lot of data, for example via social media, public transport, and GPS. But it goes way beyond that. Every minute, we send over 31 million messages, view more than 2 million videos on Facebook, and upload 300 hours of video to Youtube. Every day, over 1 trillion photos are taken and billions of them are shared online [4]. In total, we produce 2.5 quintillion bytes, or 2.5 million terabytes, of data every day [5]. We call this Big Data.

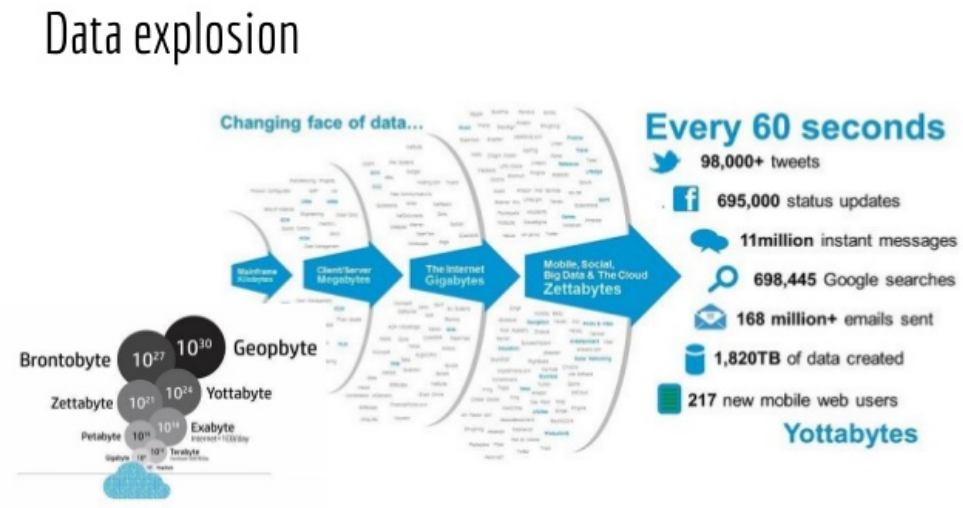

Figure 3.1: Data Explosion

According to Amazon, Big Data can be described in terms of data management challenges that, due to increasing volume, velocity and variety of data, cannot be solved with traditional databases. This brings us to the characteristics of Big Data [6].

3.1.2 The 3Vs

Big Data is defined in terms of 3Vs [7]:

- **Volume:** Volume refers to the amount of data generated through websites, portals and online applications to name a few. Volume encompasses the available data that are out there and need to be assessed for relevance.
- **Velocity:** With Velocity we refer to the speed with which data are being generated.
- **Variety:** Variety in Big Data refers to all the structured and unstructured data that has the possibility of getting generated either by humans or by machines. Examples include texts, pictures, videos, emails, voicemails.

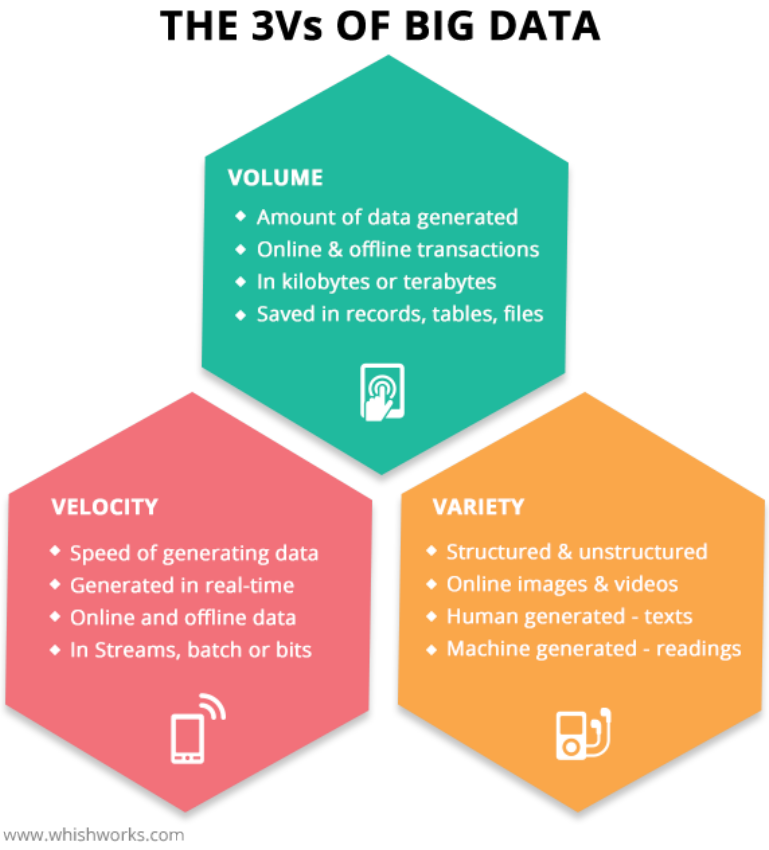

Figure 3.2: 3Vs of Big Data

3.1.3 Big Data in Numbers

- 90% of worldwide data is unstructured.
- Facebook has more than 1.45 billion daily active users. The number of text messages that will be sent today is greater than the population of the entire world.
- More than 290 billion emails are sent every day.
- 97,000 tweets are posted every second.
- More than 250 million photos are uploaded to Facebook every day.
- Viewers spend 2.9 billion hours, or 325000 years, on Youtube every month.

What is actually more important is what can be done with Big Data. Organisations who have an effective information strategy will certainly outperform their peers. This sounds like an opportunity.

3.1.4 Big Data, Big Opportunities

To process Big Data, companies don't need huge computers and servers. They use the Cloud , an endless network of normal servers, and powerful algorithms. For example, video streaming website Netflix analyses the the data of their viewers like popular shows and watching patterns. This way, the company produces the successful series with the perfect combination of actors, directors, and storyline. Moreover, the Big Data of traffic is being analysed to develop cars that can drive accident-free, all by themselves.

The escalation of data can create challenges for many organisations as they struggle to overcome the complexity of this information overload. The defining characteristics of a winning company in the Big Data age is the ability to capture and analyse the wealth of information available and quickly convert it into actionable insights.

Healthcare and life sciences
Big Data insights can improve the quality of patients diagnostic in hospitals by allowing the patient lifecycle management using unstructured data as well as population analysis and drug discovery.

Retail
Big Data enable retailers to deliver a unique customer experience through price optimization and personalised recommendations.

Banking and financial services
Big Data help banks detect fraudulent activity before it happens by employing real-time fraud detection and allowing the storage of historical data for regulatory compliances.

Insurance
It is now possible to dynamically compute insurance premiums based on driving behavior.

Big Data can be considered the most precious raw material in the 2st century. Its power can't be unlocked without using the right tools.

3.2 Apache Kafka

3.2.1 What is Apache Kafka

Apache Kafka is a distributed messaging system providing fast, highly scalable, and redundant messaging through a pub-sub model. Kafka's distributed design gives it several advantages. Kafka was originally developed at LinkedIn in 2011 and has improved a lot since then.

Kafka is highly available and resilient to node failures and supports automatic recovery. In real world data systems, these characteristics make Kafka an ideal fit for communication and integration between components of large scale data systems [8].

Distributed
A distributed system [9] is one which is split into multiple running machines, all of which work together in a cluster to appear as one single node to the end user. Kafka is distributed in the sense that it stores, receives and sends messages on different nodes called brokers.

Horizontally Scalable
Adding a new machine does not require downtime nor are there any limits to the amount of machines we can have in a cluster. The catch is that not all systems support horizontal scalability, as they are not designed to work in a cluster and those that are are usually more complex to work with.

Fault-tolerant
Something that emerges in non-distributed systems is that they have a single point of failure (SPoF). Distributed systems are designed in such a way to accommodate failures in a configurable way. In a 5-node Kafka cluster, you can have it continue working even if 2 of the nodes are down. In this context, a node is a machine.

3.2.2 Kafka Terminology

Broker
Kafka, as a distributed system, runs in a cluster. Each node in the cluster is called a Kafka broker.

Topic
All Kafka messages are organized into topics. Every message is sent to a

specific topic and/or read from a specific topic.

Partition
Kafka topics are divided into a number of partitions. Partitions allow the parallelization of a topic by splitting the data in a particular topic across multiple brokers where each partition can be placed on a separate machine.

Offset
Each message within a partition has an identifier called its offset. The offset is the ordering of messages as an immutable sequence. Kafka maintains this message ordering.

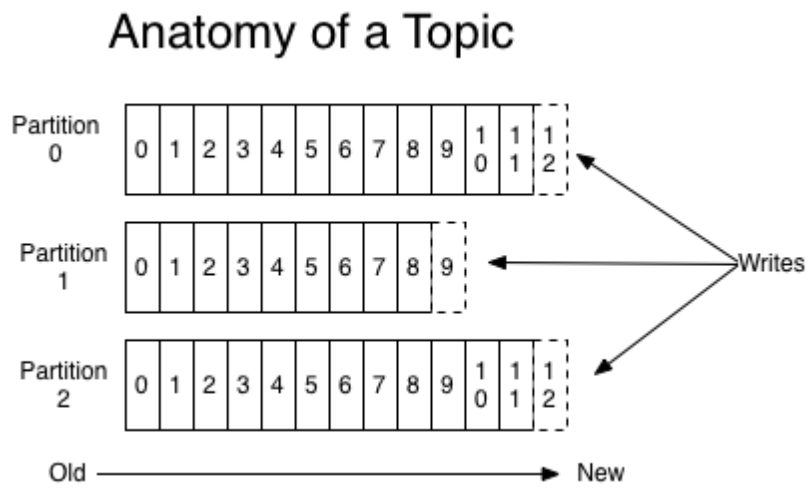

Figure 3.3: Anatomy of Topic

Producer
A Kafka producer is a program that sends messages to one or more Kafka topics.

Consumer
A Kafka consumer is a program that receives messages from a Kafka topic. Consumers can read messages starting from a specific offset and are allowed to read from any offset point they choose, allowing consumers to join the cluster at any point in time they see fit.

Consumer group

Consumers can be organized into consumer groups for a given topic. Each consumer within the group reads from a unique partition and the group as a whole consumes all messages from the entire topic.

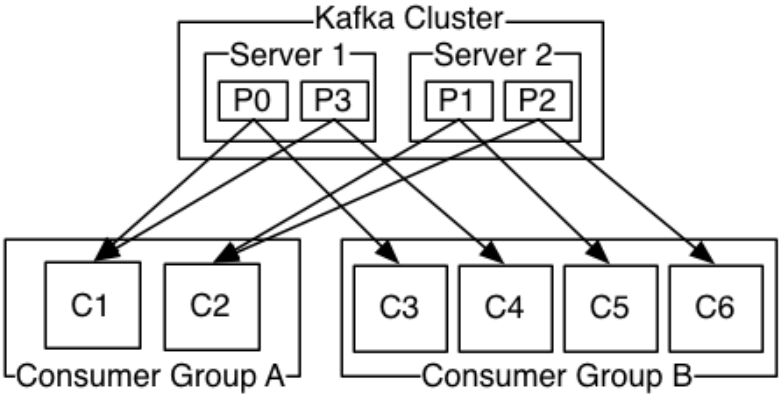

Figure 3.4: Consumers and Consumer Groups

3.2.3 Kafka Use Cases

Some of the popular use cases for Apache Kafka include [10]:

Messaging
In comparison to most messaging systems, Kafka has better throughput, built-in partitioning, replication, and fault-tolerance which makes it a good solution for large scale message processing applications.

Website Activity Tracking
Kafka is able to rebuild a pipeline for tracking user activity using real-time publish-subscribe feeds. Site activity (page views, searches, or other actions) is published to central topics where different activity types are in different topics. These topics can be consumed by applications that perform real-time processing or real-time monitoring.

Log Aggregation
Kafka can be used as a log aggregation solution. Log aggregation is about collecting physical log files that servers generate and storing them in a central place for further processing. Kafka abstraction of these log files is a stream of messages. This allows the support for multiple data sources and distributed

data consumption.

Event Sourcing
This is a style of application design where state changes are logged as a time-ordered sequence of records. Kafka fits with applications built in this style thanks to its support for very large stored log data.

3.3 Apache Hadoop

This section presents one of the biggest players in the Big Data storage and management landscape which is Apache Hadoop.

3.3.1 MapReduce

MapReduce is a parallel programming model for processing large datasets across a cluster of machines. It was first introduced by Google.

MapReduce provides an abstraction that allows developers to program in a distributed computing framework abstracting away all the inherent complexity. It uses 2 operations:

- Map: First phase, and it is performed in parallel on small portions of the dataset.
- Reduce: Second phase, has as input the output of the Map phase. This operation combines the results of the map step to give the final result.

As an example, we start out with a collection of documents or records.

- These documents are sent in a distributed way to many mappers.
- Each mapper performs the same mapping on the respective documents and produces a series of key/value pairs.
- These intermediate results are then shuffled and all key/value pairs with the same key are sent to the same reducer for processing.
- Each reducer produces one final key/value pair for each key.

The following illustrates the word count example using MapReduce:

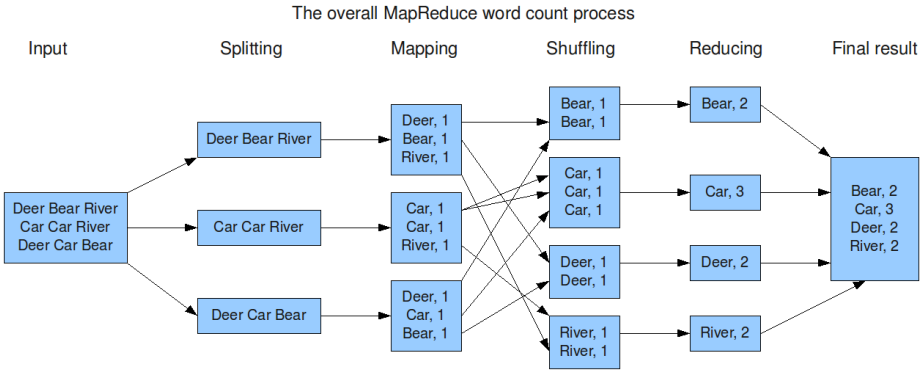

Figure 3.5: Word Count With MapReduce

3.3.2 What is Hadoop

Hadoop is an Apache open source framework for storing and processing large datasets across clusters of servers. It is designed to scale up from a single server to thousands of machines with a very high degree of fault-tolerance.

Hadoop was derived from Google's MapReduce and the Google File System, or GFS. A key aspect of the resiliency of Hadoop clusters comes from the framework's ability to detect and handle failures. HDFS, or Hadoop File System, is a distributed file system than spans all the nodes in a Hadoop cluster for data storage. It allows the distributed storage of Big Data across the cluster. HDFS assmues nodes will fail, so it achieves reliability by replicating data across multiple nodes.
Hadoop is supplemented by an ecosystem of Apache projects.

3.3.3 Hadoop Ecosystem

To complement the Hadoop modules, there is a variety of other projects that provide specialized services. Following are examples of such projects.

Yarn
Yarn sits on top of HDFS, and it stands for Yet Another Resource Negotiator. Yarn is the system that manages the resources on the Hadoop computing

cluster. It is what decides what gets to run tasks when, what nodes are available and which nodes are not.

Pig
Pig is a high-level programming API that allows programmers to write scripts in a syntax very similar to SQL called Pig Latin to query data without needing to write Python or Java MapReduce code.

Hive
Developed at Facebook, Hive is a data warehouse built on top of Hadoop that allows SQL developers to write Hive Query Language, or HiveQL, statements to leverage the Hadoop platform. HiveQL is limited in the commands it understands, but it is still pretty useful. They are broken down by the Hive service into MapReduce jobs and executed across the Hadoop cluster.

Ambari
Apache Ambari gives administrators an overview of their Hadoop clusters including resource utilisation and monitoring.

HBase
HBase is a NoSQL, columnar store. It is a very fast database meant for very large transaction rates. HBase can expose data that is in the Hadoop cluster to other systems.

Oozie
Oozie is responsible for scheduling jobs on the Hadoop cluster. If there is a task that needs to be run on the cluster and involves many different steps and maybe many different systems, Oozie provides a way of scheduling all these steps together into a single job.

Zookeeper
Zookeeper serves as a coordinator in the Hadoop cluster. It is used for keeping track of which nodes are up and which are down. It is a reliable tool for monitoring shared states across the cluster.

A more complete overview of the Hadoop ecosystem is illustrated by the following diagram:

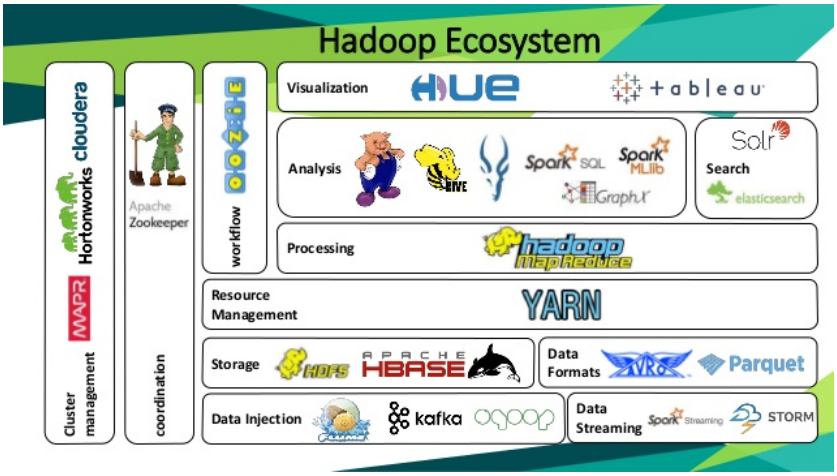

Figure 3.6: Hadoop Ecosystem

A prominent tool in the Hadoop ecosystem is Apache Spark.

3.4 Apache Spark

Apache Spark is an open source, flexible, in-memory framework that allows programmers to handle batch and real-time analytic and data processing workloads.
Coding in MapReduce is challenging. It does require a lot of Java code and mastery of difficult APIs and abstractions. It is quite hard to build long, complex processing jobs in MapReduce.

The developers of Spark at the University of California at Berkeley learned a lot from MapReduce, why it was hard to program, what the performance challenges were, and they addressed them very well. Spark lets developers describe the entire job and execute it in much better parallel form than was ever the case before. It integrates very well with the popular languages in Big Data like Python, Scala, and of course Java and other languages are also well supported.

Spark is deployed as a cluster consisting of a master server and many worker servers. The master server accepts incoming jobs and breaks them down into smaller tasks that can be handled by workers. Spark increases processing efficiency by implementing Resilient Distributed Datasets (RDDs), which are immutable, fault-tolerant collections of objects. RDDs are the foundation for all high-level Spark objects (such as DataFrames and

Datasets).

3.5 Stream Processing

3.5.1 What is a Stream

A stream is often modeled as a an unbounded, continuous real-time flow of records. Theses records are sometimes called facts. They are structured as key/value pairs.

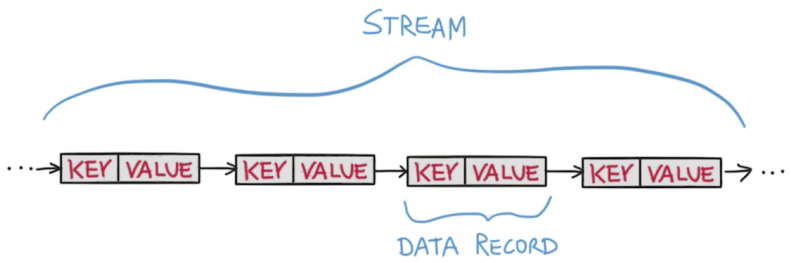

Figure 3.7: Stream of Data

3.5.2 What is Stream Processing

Stream processing is all about processing data in motion, or data as it is produced or received [11]. The larger part of data are produced as continuous streams, such as sensor events, user activity on a website, and financial trades. All these data are generated as a series of events over time. Before stream processing, data was generally stored in a database or a file system. Applications would run batch jobs over the data in order to do computation and analysis.

Data streams are often modeled as unbounded and infinite sequences of records, where each record represents an event or change that happened in time. Typical stream processing applications observe or listen to these flowing streams and query them in near real-time.

3.5.3 Sources of Streaming Data

A popular source of streaming data is IoT data. However, sources of streaming data can not be only found in the IoT area. Many other industries incur streaming data. Some of the most relevant examples include:

- **IoT sensor data**: Industrial machines, consumer electronic.
- **Click streams**: Online shops, self-service portals, comparison portals.
- **Monitoring tools**: System health, resource utilization.
- **Online gaming**: Gamers interactions, reward systems, custom content.
- **Automotive industry**: Vehicle tracking, routing information.
- **Financial transactions**: Fraud detection, trade monitoring and management.

3.5.4 Stream Processing Tools

There is a variety of tools and technologies to choose from when it comes to stream processing. This section gives an overview of the most important of them. This list is not exhaustive.

Kafka Streams

Kafka Streams is a Kafka API for transforming and enriching streams of data. It supports per-record stream processing (no micro-batching) with millisecond latency. The biggest advantage of Kafka Streams is that it comes as part of the Java application, which means there is no separate processing cluster to deploy. Kafka Streams is elastic, highly scalable, and fault-tolerant.

Advantages
- Very lightweight library
- Does not need dedicated cluster
- Inherits the good parts of Kafka

Disadvantages
- Firmly coupled with Kafka, so it can not used without Kafka
- Quite new, not mature enough

Spark Streaming

Spark Streaming [12] is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams. Data can be ingested from many sources such as Kafka and can be processed using complex algorithms expressed with high-level functions like

map, reduce, and join. Processed data can be pushed out to file systems, databases, and live dashboards.

Spark Streaming doesn't provide "real" stream processing. Instead, it uses micro-batching. It means that records received in every few seconds are collected together and then processed in a single mini batch with a delay of few seconds.

Advantages
- Fault tolerant by default due to micro-batch nature [13]
- Optimised for High throughput
- Big community and ongoing improvements

Disadvantages
- Not true streaming
- Highly configurable, with so many parameters to tune

Apache Storm

Apache Storm is a free and open source distributed realtime computation system. Storm applications which process data in real-time are called Storm topologies. A topology is made of 2 types of components: Spout which receives data from the data source and emits it to the rest of the topology. Bolt which performs the processing task on the data.

Advantages
- Very low latency
- Mature product

Disadvantages
- State management is not supported
- No advanced features like aggregation and windowing

Apache Flink

Apache Flink is a framework for stream and batch data processing. Flink supports massively scalable distributed dataflow, which means that developers can write on big clusters and process huge amounts of data. It is written in Java, but it also has Scala and Python APIs.

Flink is agnostic to where data is coming from or where Flink is being executed, or what is called storage and cluster agnostic. It also supports many data sources, and Flink applications can run on multiple platforms.

Advantages
- Low latency and high throughput
- Getting popular among big companies like Uber and Alibaba

Disadvantages
- Not a big community
- Almost used only for streaming, little to no adoption of Flink batch

Conclusion

In this chapter, we gave an overview of the Big Data landscape. We also presented the different trade-offs in data processing and the cutting-edge tools that assist in deriving value from data.
In the next chapter, we will dig deeper into the requirements and specifications of our project.

Chapter 4

Analysis and Requirements

In this chapter, we will detail the requirements of our project and the analysis that led to the choices we took. This chapter is meant to justify these choices and give a clear overview on how we proceeded.

4.1 Ecosystem Overview

In this section, we will provide an overview of the different tools and components that make up the Sentiance platform.

4.1.1 SDK

Sentiance develops 2 SDKs, and Android SDK and an iOS SDK. After customers embed either SDK in their mobile applications, it starts collecting sensor data. The SDK is completely autonomous and can tell if someone is moving or stationary to collect the corresponding sensor data like accelerometer, gyroscope, and GPS coordinates.

The SDK is battery efficient because it knows when to turn on and off the data gathering. It then sends the raw data to the backend API which is a RESTful Node.js system. This communication uses Thrift.

4.1.2 Thrift and Sentiance Fact Model

Apache Thrift is an Interface Definition Language, or IDL, which is used to define cross-language services. It allows to define data types and service interfaces in a simple definition file. Thrift provides a compiler that generates code to be used to easily build RPC clients and servers that communicate seamlessly across programming languages including and not limited

to C++, Java, Python, PHP, Ruby, JavaScript, Perl, and C# [14].

JSON documents are big and bloated and not a good match for the low level data the Sentiance SDK produces. Switching to a binary format as supported by Thrift does reduce the message size on the wire and in storage, and helps address some of the current scalability and performance issues.

Sentiance Fact Model is the data model of Sentiance that uses Thrift to define data types and structures. A Fact is a singular piece information ingested by the Sentiance platform as represented by a Data object from the Sentiance Thrift model. This data model uses Thrift as a serialization format.

4.1.3 Kafka

Kafka is used at Sentiance as the messaging system. The API puts the data it receives from the SDK in a dedicated Kafka topic. All the microservices either read messages, or records, from a source Kafka topic, write messages back to Kafka, or both. For this reason, Kafka is the backbone of Sentiance data pipeline.

Kafka was selected for a couple of reasons:

- **High throughput:** First and foremost, Kafka is able to handle large volumes of data in the Terabytes and beyond.
- **Horizontal scalability:** Kafka is designed to scale out by adding machines to seamlessly share the load.
- **Reliability and durability:** Kafka offers reliable data management, transmission, and durability in the case of failure.
- **Flexible publish-subscribe semantics:** Independent data producing applications send data on a topic, and any interested consuming application could listen in and receive data on that topic which it could process, and in turn, produce on a different topic for others to consume.
- **Adoption:** Many big companies have adopted Kafka to solve their own data problems.

Figure 4.1: Powered by Kafka

4.1.4 Data Processing

Storm

Sentiance uses Apache Storm to implement the shallow analysis at the level of the speed layer.

Spark

Spark is the framework of choice at Sentiance for processing data in the batch layer. Long running jobs are defined to perform computation on this data. Underneath Spark, HDFS is used to store the master dataset taking advantage of the distributed and replicated nature of this file system. The resources used by the Spark jobs are scheduled using Yarn.

4.1.5 SonarQube

SonarQube is an open source platform for continuous inspection of code quality. It performs automatic reviews of code to detect bugs, code smells, and security vulnerabilities. SonarQube offers also reports on duplicated code, coding standards, unit tests, code coverage, and code complexity [15].

Sentiance uses SonarQue to control and maintain the quality of the code base. Custom quality profiles and quality gates are defined for the different projects to ensure an acceptable code quality.

4.1.6 Infrastructure and Deployment

AWS

Amazon Web Services, or AWS, is a global cloud platform that allows developers and businesses to to host and manage services on the Internet. AWS offers compute power, database storage, content delivery and other functionality to help businesses scale and grow [16].

The resources provided by Amazon Web Services are hosted in multiple locations in the world. These locations are composed of regions and Availability Zones. Each region is a separate geographic area. Each region has multiple, isolated locations known as Availability Zones [17].

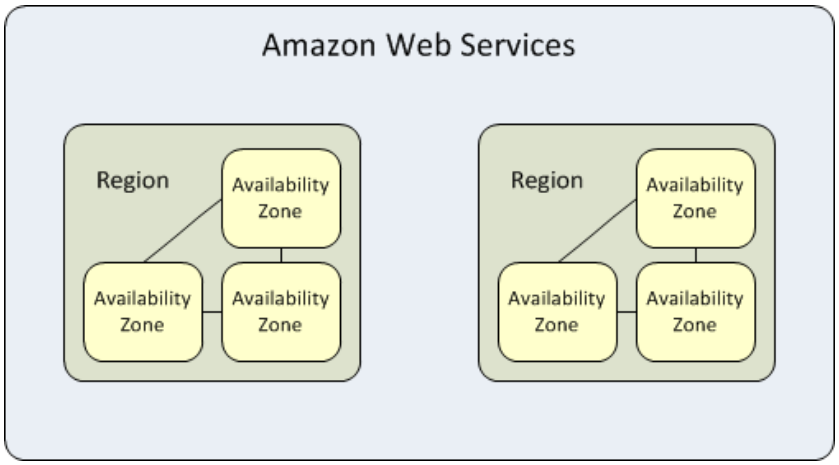

Figure 4.2: AWS Regions and AZs

Sentiance does not have servers. Everything is deployed on AWS cloud, with Ireland as main region. The primary AWS services Sentiance uses are:

- **EC2:** EC2, or Elastic Cloud Compute, provides scalable computing capacity in the Amazon Web Services cloud. An EC2 instance is a virtual machine provisioned with a certain amount of CPU cores, gigabytes of memory, storage space, and network performance.
  Each EC2 instance has a certain group of of technical specifications, and Amazon charges users based on what type of instance they are running.

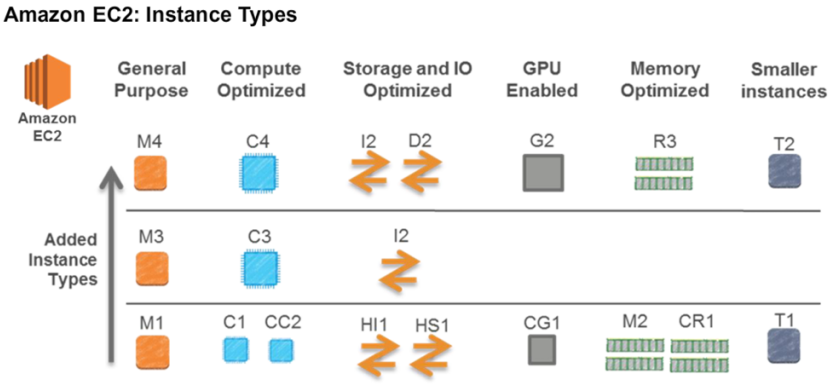

Figure 4.3: AWS EC2 Instance Types

- **S3:** S3, or Simple Storage Service, allows the storage and retrieval of data. S3 stores objects in whichever region the user selects, which can be important for legal or regulatory reasons.
  Objects in S3 are stored in another fundamental structure called bucket. A bucket provides a URL namespace for the objects that are stored on S3.

- **ECS** Elastic Container Service allows to schedule, launch, and run Docker containers across a cluster of EC2 instances. Applications can be built and packaged into containers with Docker and seamlessly deployed into production with ECS.

- **CloudWatch:** CloudWatch is a monitoring service for AWS cloud resources and applications. It enables monitoring for EC2 and other services, and it collects metrics that offer system-wide visibility into the resources utilization.

Docker

Docker is an open source container based technology. A container allows a developer to package up an application and all of its dependencies in an isolated environment. This means that the underlying host and the environment that application runs on is completely abstracted. So Docker separates applications from infrastructure, similar to how virtual machines separate the operating system from the bare metal.

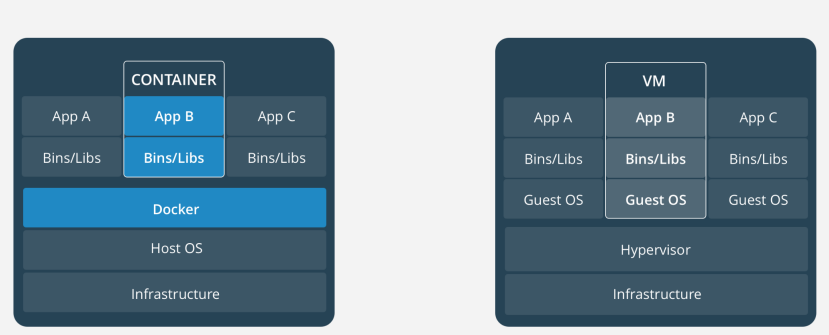

Figure 4.4: Virtual Machine vs. Container

Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs [18].

A Docker image is a read-only template with instructions for creating a Docker container. An image is built by creating a Dockerfile that uses a simple syntax to define the steps needed to create the image and run it. In essence, a container is just a runnable instance of an image.

Docker is used at Sentiance to build microservices images and run containers off them. It was chosen for a number of reasons:

- **Scalability:** Containers are lightweight, which makes scaling up and scaling down fast and easy. More containers can be launched or shut down as needed.
- **Portability:** Dockerized applications can run on any system that has Docker installed. There is no need to build and configure the applications multiple times on different platforms.
- **Continuous deployment and testing:** Docker guarantees consistent environments from development to production. Containers maintain all configurations and dependencies internally. This means the environment in which the application is tested is identical to the one on which the application will run in production.
- **Multi-Cloud:** Docker containers can be run inside an Amazon EC2 instance, Google Compute Engine instance, Rackspace server, VirtualBox, and many other platforms.

- **Isolation:** A containerized application has its own resources that are isolated from other containers. If an application is no longer needed, its container can be simply deleted.
- **Security:** Because of isolation, no container can look into the processes of another container.

CI/CD and Jenkins

The Agile movement formalised several practices that were meant to enable organisations deliver functionalities more often. To enable the project management methodologies introduced by Agile, a collection of technical practices was introduced about the same time. These practices are often referred to as XP, or eXtreme Programming.

Continuous Integration, or CI, is a development practice that requires developers to integrate code into a shared repository several times a day. Each checkin is then verified by an automated build, and feedback allows teams to detect problems early. Practicing CI requires quite of automation especially around testing itself.

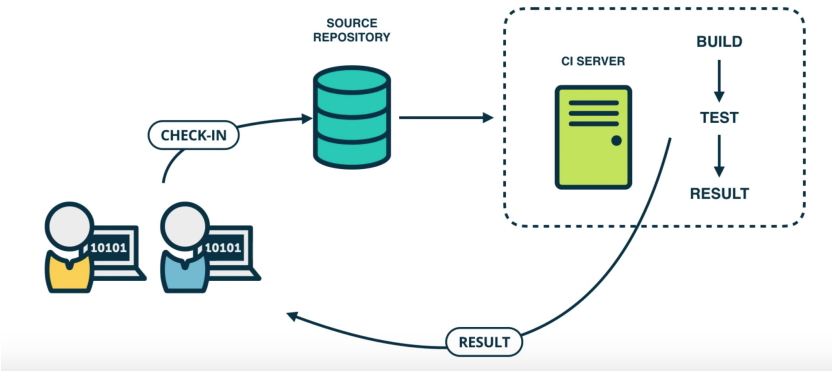

Figure 4.5: Continuous Integration

With solid CI practices in place, developers gain confidence in the quality of their code.

Continuous Deployment, or CD, is about automating software deployment and testing on a regular basis to production-like environments or even production itself. Every build is run through a deployment pipeline.

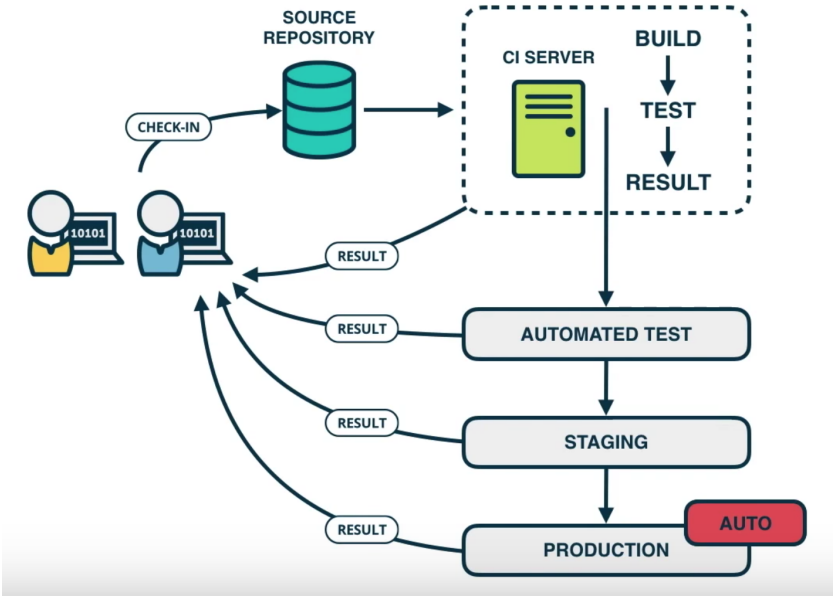

Figure 4.6: Continuous Deployment

With Continuous Deployment, there is no human interaction at all after a code commit. If all the tests pass, it gets deployed to production immediately.

Sentiance uses Jenkins as a CI/CD tool. It is responsible for automatically building, testing, and deploying microservices. Jenkins server runs on an EC2 instance.

4.2 Technical Analysis

In this section, we recall the project statement and analyse its technical requirements.

4.2.1 Project Statement

Our project involves assisting in the refactoring of one of the sequential data processing pipelines at Sentiance towards a more efficient, scalable processing mesh based on Reactive design principles. The data processing pipeline itself is focused on sensor processing for mobility purposes, for example transport classification, map matching, and driving behavior. The goal of the new processing mesh is to achieve a more cost-effective, more maintainable data processing environment.

4.2.2 Motivations

There are 4 primary tasks achieved as part of our project. In this section, we will present the motivations of our work and the solutions provided.

Sequential Transport Processing Pipeline

The data pipeline that processes sensor data for mobility purposes, such as transport classification and driving behavior, is sequential. The different microservices that operate in this data pipeline are represented as follows:

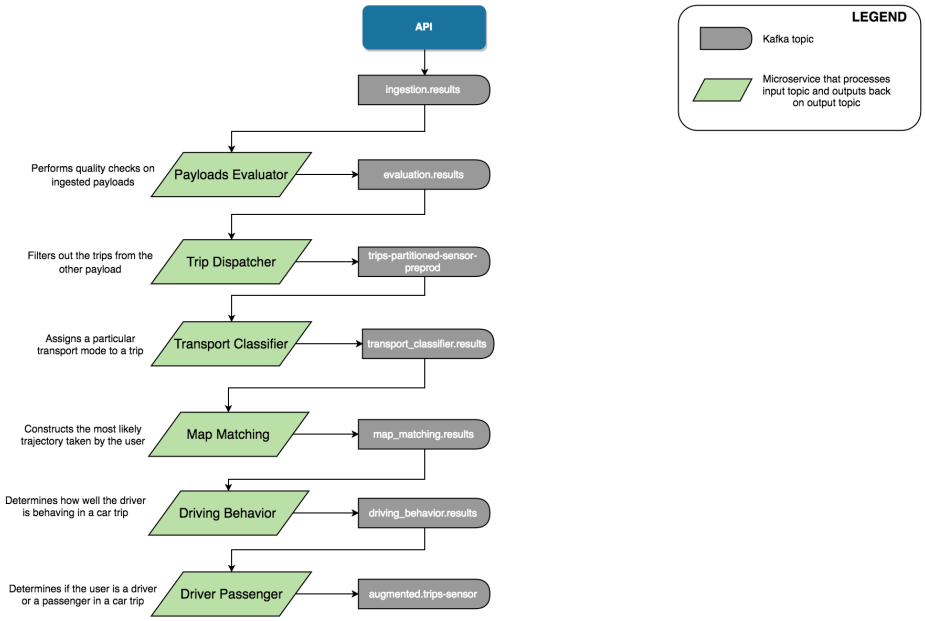

Figure 4.7: Transport Processing Pipeline

This pipeline is not efficient for 3 reasons:

- The messages in the topics contain more data than the microservices need to process.
- Every microservice passes its processing results as well as the messages it consumed from the topic down the pipeline.
- As a consequence, data is duplicated and the messages in topics become bigger in size.

Kafka is designed to handle very large amounts of data, but it is doesn't perform well with messages of big size.

The solution we suggested is to write a streaming application that consumes the output topic of the Trip Dispatcher microservice, extracts only the fields needed by the microservices down the pipeline, and writes this data back to different topics with smaller messages.
The advantage of this solution is that it provides every microservice with an easier way to subscribe only to the topics that it needs to work.
The stream processing library we used is Kafka Streams.

Inconsistent CI/CD

All component releases at Sentiance are done from artifacts built on Jenkins. An artifact is an immutable file generated during a Jenkins pipeline run. A continuous delivery pipeline is an automated description of the process for getting software from code repository right through to users. This process involves building the software in a reliable manner, as well as progressing the built software (called a "build") through multiple stages of testing and deployment [19].

At Sentiance, pipelines are created via the Jenkins user interface. Every pipeline defines the required steps to install the dependencies of the project associated to it, as well as preparing the environment for building, testing, and eventually deploying the code.

In order to introduce a more stable, uniformly structured way of building software, Data Engineers of the MobSense squad created a unified Jenkins shared library to streamline the definition of pipelines in a "Pipeline-as-code" approach instead of the manual.

Large Sensor data in Kafka

The payloads received from the API and stored in Kafka topics contain various information, and sensor data is the biggest in size. In an attempt to further reduce the load on Kafka topics, we implemented a call by reference mechanism in the transport processing pipeline.

According to this mechanism, the messages that are produced to a topic are stored in an external storage system and only references to these

messages are kept in the topic. Dereferencing is performed afterwards in order the retrieve the original messages when needed.

An Overlooked Code Quality

Among the important projects at Sentiance is one that contains the Storm topologies needed to do transport processing on the trips ingested by the clients. A Storm topology is a network of computing and processing nodes of 2 types. Spouts which receive data from the data source and emit it to the rest of the topology. Bolts which perform the processing task on the data.

This project has been in use since 2 years. However, SonarQube reports many bugs, several vulnerabilities, a considerable code duplication percentage, and an extremely low code coverage. The code maintenance has been overlooked.

In order to bring the code quality to an acceptable level, we refactored it, defined a set of tests, and removed the vulnerabilities it contained.

Conclusion

Throughout this chapter, we explored the technical ecosystem that makes up the Sentiance platform. We also gave an overview of the tasks we carried out as part of our project. Next chapter will go into details about these tasks.

Chapter 5

Design and Implementation

The previous chapter introduced the motivations and context of the tasks that comprise our project. In this chapter, we will present the design and implementation details of these tasks.

5.1 Truck Driver

As mentioned previously, the solution we suggested to improve the transport processing pipeline is to write a streaming application that consumes the output topic of the Trip Dispatcher microservice, extracts only the fields needed by the microservices down the pipeline, and writes this data back to different topics with smaller messages. Truck Driver is the codename of this project.

5.1.1 Design

Sentiance Fact Model contains the schemas definitions of the data structures used to communicate between all the different components. One of the most important structures is called Trip.

Conceptually, a Trip structure contains many different fields which hold the data captured by the SDK from the moment a user starts moving until they become stationary. Examples of the Trip fields as defined in the Thrift schema include:

- **sensor_data:** data collected from device sensors such as accelerometer, gyroscope, and magnetometer.
- **waypoints:** location information like latitude and longitude.

- **motion_activities:** user's motion data such as walking, running, and being stationary.
- **metadata:** trip's metadata.
- **m7_data:** sensor data collected from devices equipped with Apple's M7 coprocessor. This field is optional, so a null list is assigned to m7_data for Trips collected through Android phones.

In Truck Driver, we used Kafka Streams to write a streaming application that consumes the Kafka topic containing the trips and produces a number of topics. Each of these topics contains only the data related to one different field of the Trip.

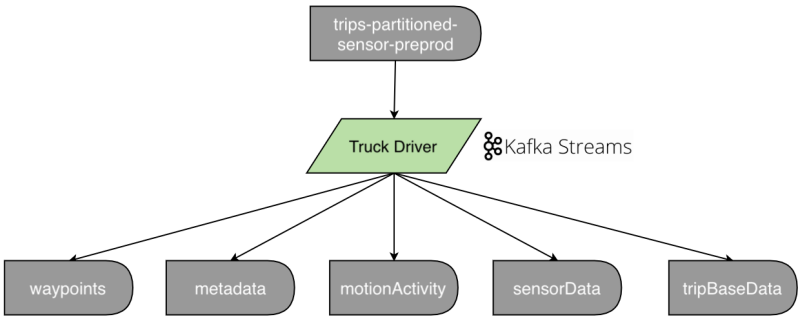

Figure 5.1: Truck Driver

5.1.2 Implementation

Truck Driver was implemented in Java 8 using Kafka Streams library. Kafka Streams was selected for the following reasons:

- The underlying messaging architecture at Sentiance is supported by Kafka, so using a Kafka library was a natural choice.
- There is no separate processing cluster to deploy to use Kafka Streams. The streams code runs inside of a regular Java application.
- Kafka Streams is highly scalable and fault-tolerant. Kafka itself solves the problem of distributing data, and Kafka Streams library solves the problem of distributing the computation.
- Kafka Streams is deployment agnostic, so whether to deploy to containers, virtual machines, or cloud, Kafka Streams is part of the application and gets deployed with the application code.

5.2 Transport Classifier 2

Transport Classifier is the first microservice in the transport pipeline to process the trips. This microservice takes raw data and will try to classify the trip (idle, walking, ...). To do the classification, it uses the transport classifier model created by Sentiance Data Scientists and the GIS model. The entire project is Kafka based. This means that data is read from a Kafka topic and the result is produced to another Kafka topic.

Transport Classifier 2 is a new version of Transport Classifier we worked on that listens to the output topics of Truck Driver. The transport classification requires only the waypoints, motion activities, sensor, and trip base data. Transport Classifier 2 subscribes only to the 4 corresponding topics, as opposed to Transport Classifier that would consume the payloads topic that contains some data it doesn't need.

Transport Classifier 2 is made up of 2 main components:

- Java client: Kafka Streams application that creates 4 streams from the 4 topics and joins them into 1 stream containing all the Trip parts needed for the transport classification logic.
- Python server: Business logic of the service. It receives the joined trips stream from the Java client, performs the transport classification logic, and returns the classification result back to the client.

The communication between the Java client and the Python server is an RPC facilitated by Thrift.

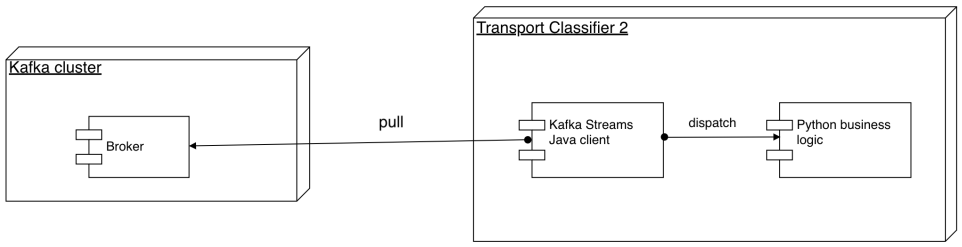

Figure 5.2: Component Diagram of Transport Classifier 2

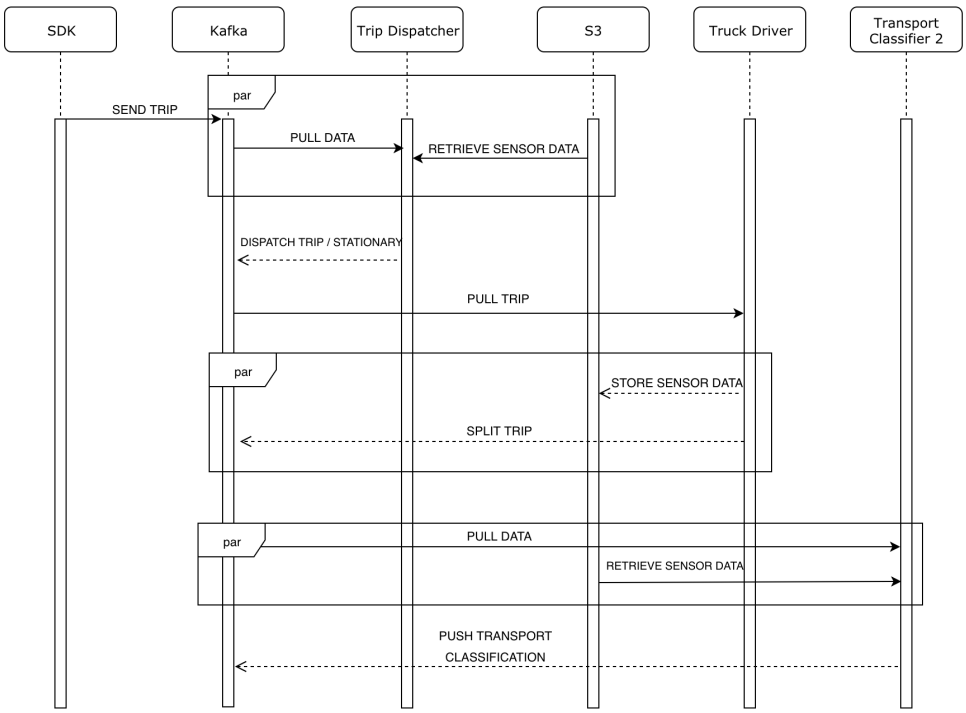

Figure 5.3: Sequence Diagram of Transport Classifier 2

In order to avoid bottlenecks, the Python server implements a feature we called "bus lane". The idea is that messages that take longer than a certain number of seconds to be processed are sent to a bus lane topic. The messages in this topic are processed all over again.

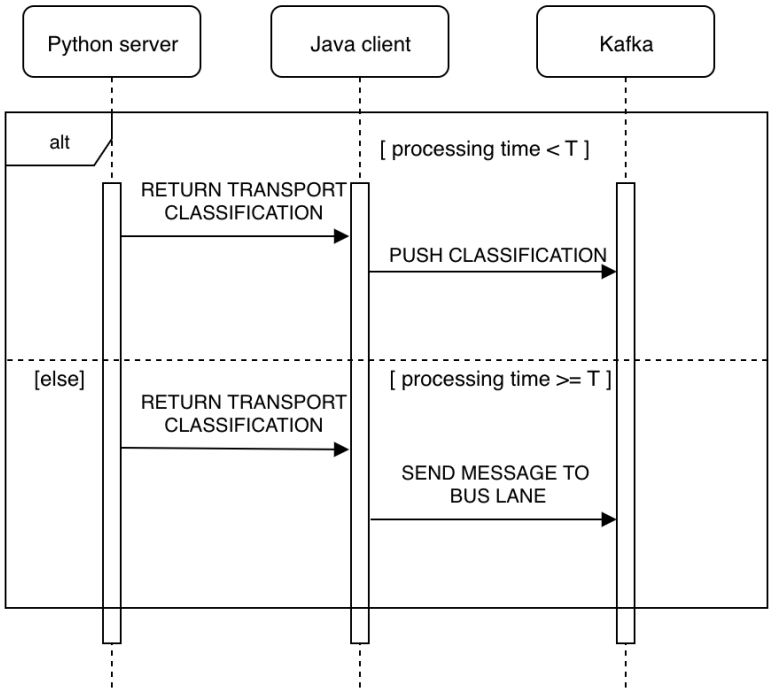

Figure 5.4: Sequence Diagram of Bus Lane

The classification result the Java client receives from the Python server is a stream that is written to a Kafka topic.

5.3 Consistent Jenkins Builds

The initiative we took in MobSense squad aimed at reducing the technical debt that had built up because of the inconsistent way of building software. (explain more what technical this debt is).

Pipeline as Code is a Jenkins feature that makes it possible to define pipelined job processes with code, stored and versioned in a shared source repository. The advantage of this feature is that it allows Jenkins to discover and run jobs for multiple source repositories and branches without the need to manually create and manage the jobs [20].

At MobSense, we did the following:

- Introduced a Jenkins common library of pipeline scripts, which is a set of Groovy class files that contain Pipeline scripts specifying the steps

to execute the jobs: running the unit tests, installing the packages, publishing to Sentiance artifacts server, building docker image, and deploying to the different environments.
These files are usually called Jenkinsfiles. We had different files for different builds such as jar, fat jar, and docker builds.

- Created a pipelines directory at the root of each of our projects. This directory contains the required pipeline files that use the Jenkins shared library Jenkinsfiles.

- At the level of Jenkins interface, we manually set up the build and deployment pipelines. Each of these pipelines specifies the script path where the pipeline file lives in the source repository.

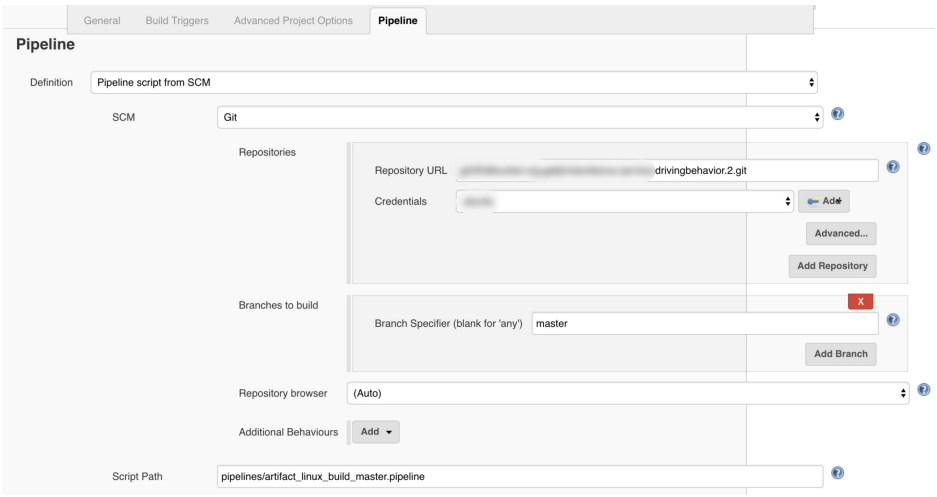

Figure 5.5: Example of Pipeline Script Usage

In this example, the build job of the master branch of Driving Behavior microservice specifies the location of the pipeline script in Script Path box: pipelines/artifact_linux_build_master.pipeline.

5.4 Reference Based Messaging

As we previously mentioned, the trips that the API receives from the SDK and end up in Kafka contain but not limited to waypoints, motion activities, and metadata. Sensor data is the biggest in size. Kafka has a limit on the maximum size of a single message because:

- Large messages increase the memory pressure in the broker.
- Large messages are expensive to handle and could slow down the broker.
- A reasonable message size limit can handle vast majority of uses cases.

Reference based messaging is a technique introduced by LinkedIn to mitigate the limitation of large messages. This method states that, for every large object, the producer stores it in an external storage system, associates a unique identifier, or GUID, to the object, and produces a message in a Kafka topic with the GUID as a value.

In our case, sensor data will be passed by reference meaning the data will be stored on S3 under a GUID and the message on Kafka only contains this GUID, not the sensor data itself.

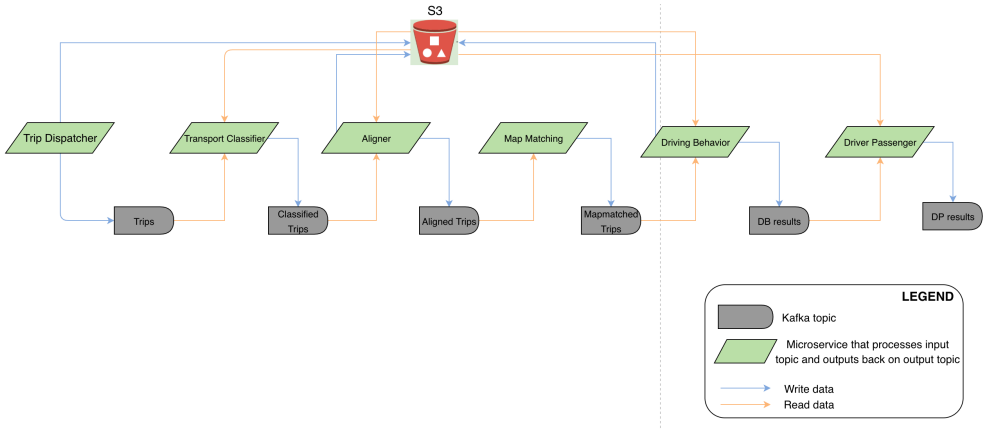

Figure 5.6: Transport Processing Pipeline Using Reference Based Messaging

The pipeline starts with the Trip Dispatcher that stores the original sensor data on S3 for the Transport Classifier to use, and it outputs messages to Trips topic which contain references to the sensor data on S3. This same pattern continues along the pipeline.

Because we rely on Thrift to define our data structures, we had to change the Sentiance Fact Model and some other schemas:

- We created a separate schema for sensor data on S3.
- We adapted the main Sentiance Thrift model to allow the use of references.

A reference is a simple string containing a GUID. A boolean flag was also added to indicate whether sensor data is on S3 or in Kafka. This is required so the Thrift model stays backward compatible, meaning microservices can still run with the new schema and see the fields they expect.

- We created a Java library and a Python library to store and retrieve sensor data from S3.
- We updated Map Matching and Driving Behavior schemas to add the GUID and the boolean flag.

In addition to the schema updates, we had to change Truck Driver as well. It no longer writes the sensor data it extracts from the payloads topic to Kafka. Instead, it uses the same reference based messaging technique.

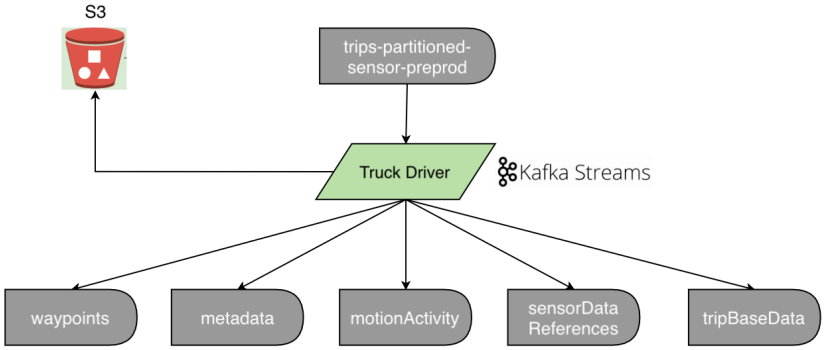

Figure 5.7: Truck Driver Reference Based Messaging

5.5 Transport Processor Topologies Refactoring

5.5.1 What is Refactoring and Why is it Important

Refactoring is the process of changing a software system in such a way that it improves its internal structure without altering the external behavior of the code. It is a disciplined way to clean up code that minimizes the chances of introducing bugs.

The common understanding of software development states that coding is a second phase after design. However, the code will be modified over

time, and the structure of the system according to that design slowly declines. Refactoring is about reworking a bad design into a well-designed code with simplistic steps.

Theimportance of code refactoring can be summed up in the following points:

- Code becomes more readable and maintainable.
- It becomes easy to scale and extend functionalities. Refactoring is required for new features to be implemented properly.
  According to Software Engineering veteran Martin Fowler, when a developer finds they have to add a feature to a program, and the program's code is not structured in a convenient way to add the feature, they first have refactor the program to make it easy to add the feature, then add the feature.
- It gives an opportunity to add comments so it makes more sense.
- Tunes the code to run faster.

5.5.2 Refactoring the Transport Processor

The transport processor Storm topologies is an important piece of code at Sentiance, and it is a crucial part of the speed layer. This project has been in use for 2 years. Because of the lack of maintenance, technical debt started to build up and SonarQube quality checks reported very poor metrics.

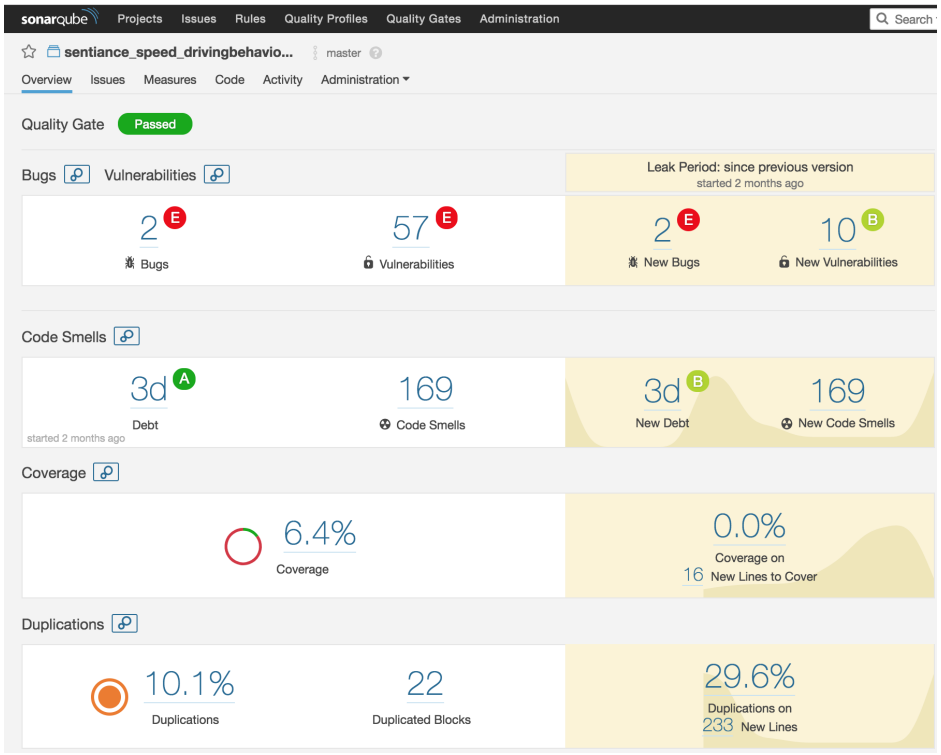

Figure 5.8: SonarQube Quality Metrics

The lowest metric in this quality report is the code coverage, which is a measurement of how many lines/blocks of the code are executed when running the automated tests. It was an alarming sign that the tests we had were not enough.

We started by writing a solid suite of tests. Tests made it easy to catch the bugs and gave an efficient way of controlling the effects of the changes we made to the code. We also restructured the Storm bolts by separating the bolts classes and the logic they implemented that made it possible to have utility classes for common bolts processing. In addition to tests, we removed the code smells. A code smell refers to any symptom in the source code that could possibly mean a deeper problem exists. Code smells are not technically incorrect and do not currently prevent the program from functioning. Instead, they indicate weaknesses in design that may be slowing down development or increasing the risk of bugs or failures in the future. Bad code smells can be an indicator of factors that contribute to technical debt [21].
Restructuring the code contributed in considerably reducing the number of

duplicated blocks.

Note that the quality gate still indicates a green state "passed" despite the poor quality metrics. The reason is that a custom quality gate is defined for legacy code at Sentiance. Legacy code is best defined by Michael Feathers in his book Working Effectively with Legacy Code. According to him, legacy code is code without automated tests.

Conclusion

In this chapter, we went through the design and implementation details of the different tasks of our project. We justified the choices we made and highlighted their importance.

Chapter 6

Results

This chapter highlights the results and achievements for each of our project tasks. It also presents suggested improvements that could be implemented in the future.

6.1 Truck Driver

We ran Truck Driver on MobSense server for 4 weeks. Over 2,250,000 sensor data objects have been written to S3, or 1,2 TB of data.
The project is under SonarQube. 45 unit tests are in place with:

- 0 vulnerabilities
- 0 bugs
- 0 code smells
- 0 duplicated blocks
- 83.3% of coverage which is greater than Sentiance threshold of 80

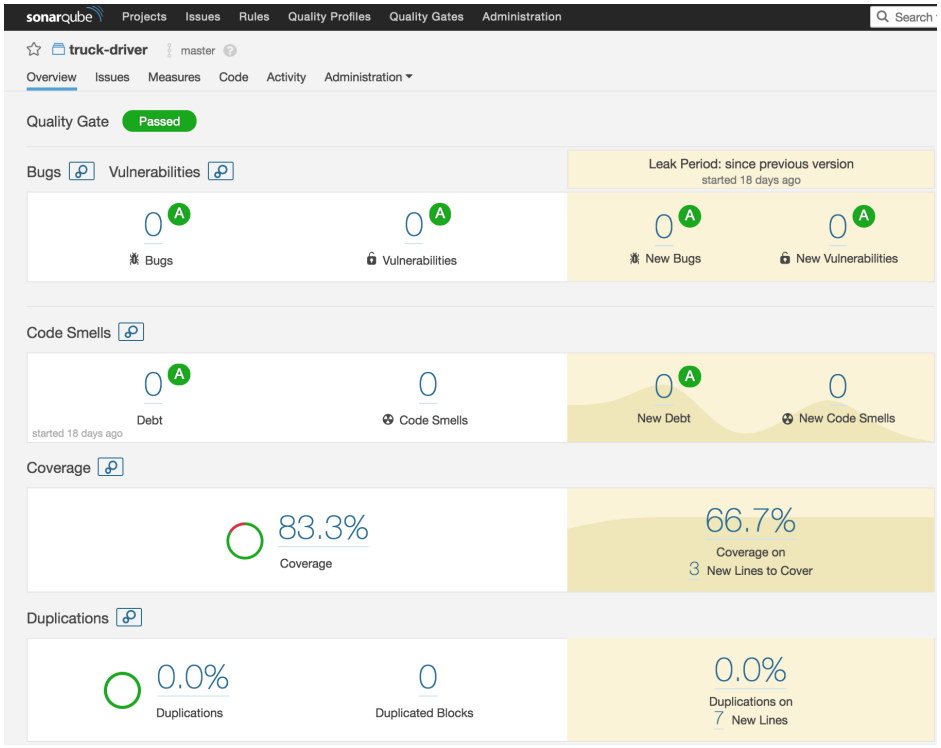

Figure 6.1: Truck Driver Quality Metrics

In addition to the unit tests, we performed a completeness test on Truck Driver to verify that no data is lost. The test topic contains 17,000 messages consumed from trips-partitioned-sensor-preprod topic (the output topic of Trip Dispatcher). The results of the completeness test are the following:

Table 6.1: Truck Driver Completeness Test Results

Topic	**Number of messages**	**% of lost data**
waypoints	17,000	0
metaData	17,000	0
motionActivity	17,000	0
sensorDataRef	17,000	0
m7Data	17,000	0
tripBaseData	17,000	0

% of total lost data	0

6.2 Transport Classifier 2

Transport Classifier 2 is an ongoing work. We were able to run it locally, and there was a successful communication between the Java streaming client and the Python business logic server.

The transport classification result returned back to the streaming application contained a boolean that indicates whether the trip was successfully processed or its processing took longer than a threshold we defined. In the second case, the message was sent to a bus lane topic. We verified that this topic contained records which proved that our logic was functioning correctly.

The Java client and the Python server have been both unit tested. Their quality metrics as reported by SonarQube are the following:

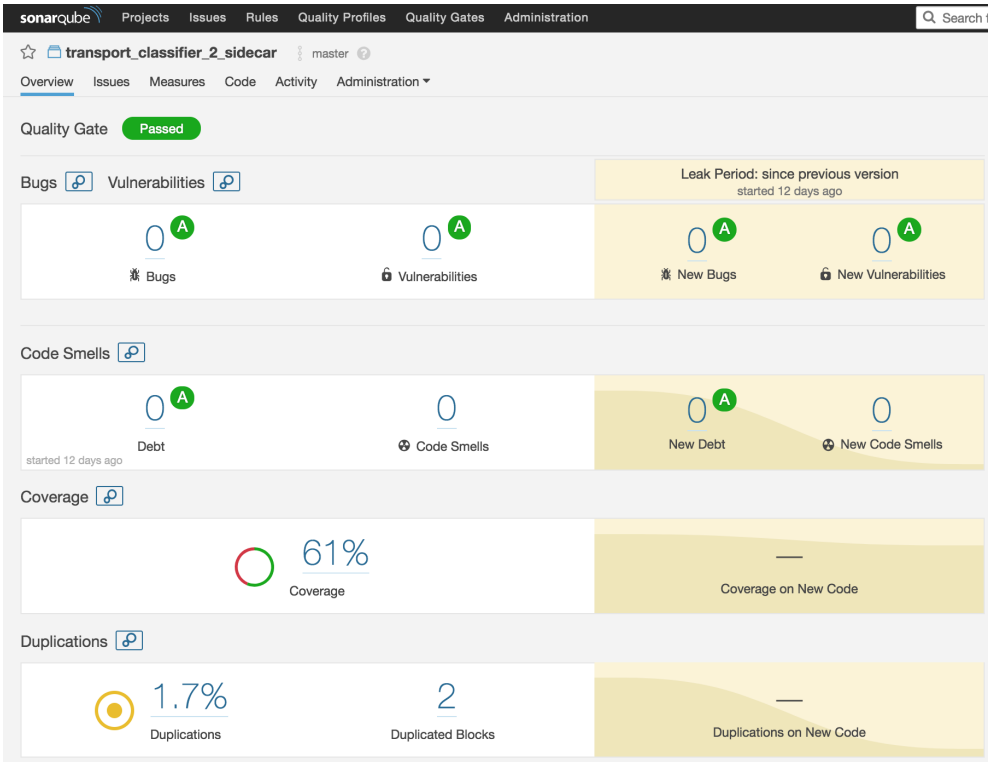

Figure 6.2: Transport Classifier 2 Python Server SonarQube Metrics

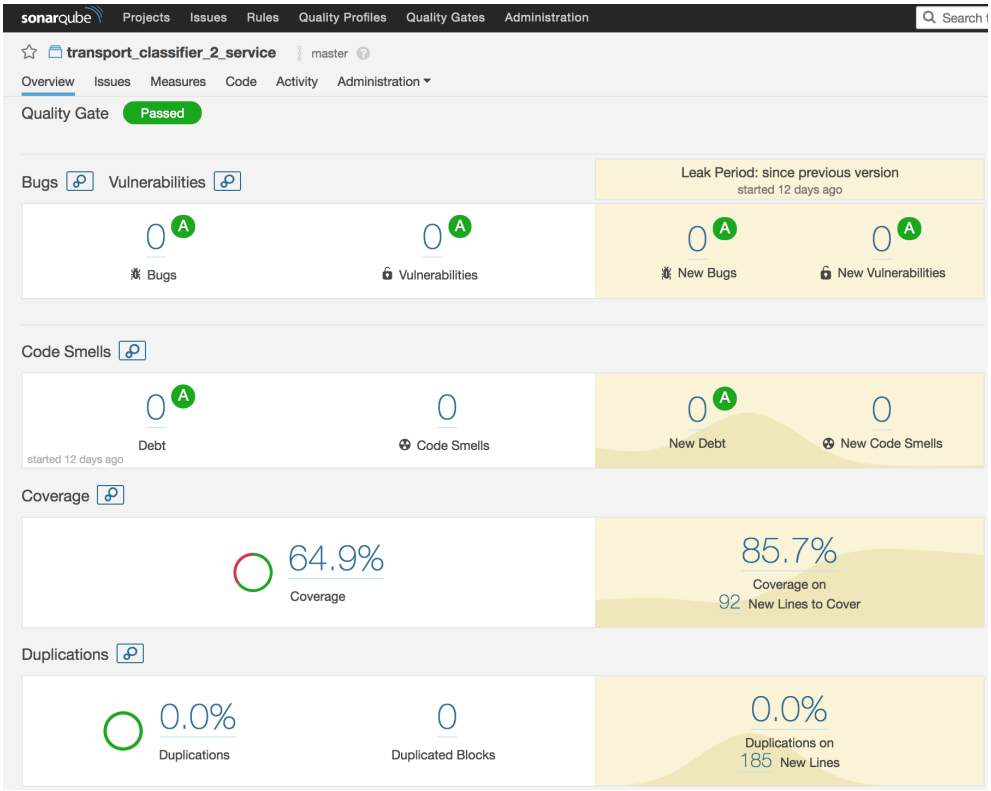

Figure 6.3: Transport Classifier 2 Java Client SonarQube Metrics

6.3 Jenkins Laundry

Jenkins laundry days is the name of the initiative of creating more stable, uniformly structured Jenkins builds.

Over the course of 3 weeks, we achieved the following:

- We fixed 313 builds.
- All these builds are in the Jenkins pipeline format and under version control.
- All builds are reporting quality metrics to SonarQube.

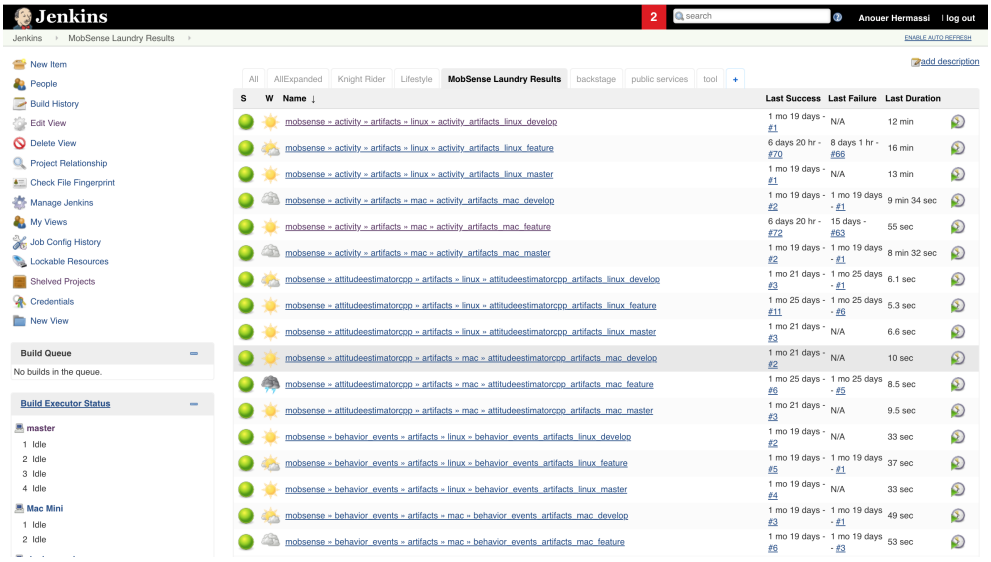

Figure 6.4: MobSense Laundry Results

6.4 Reference Based Messaging

Pushing sensor data to S3 had a considerable impact on the messages size in Kafka.

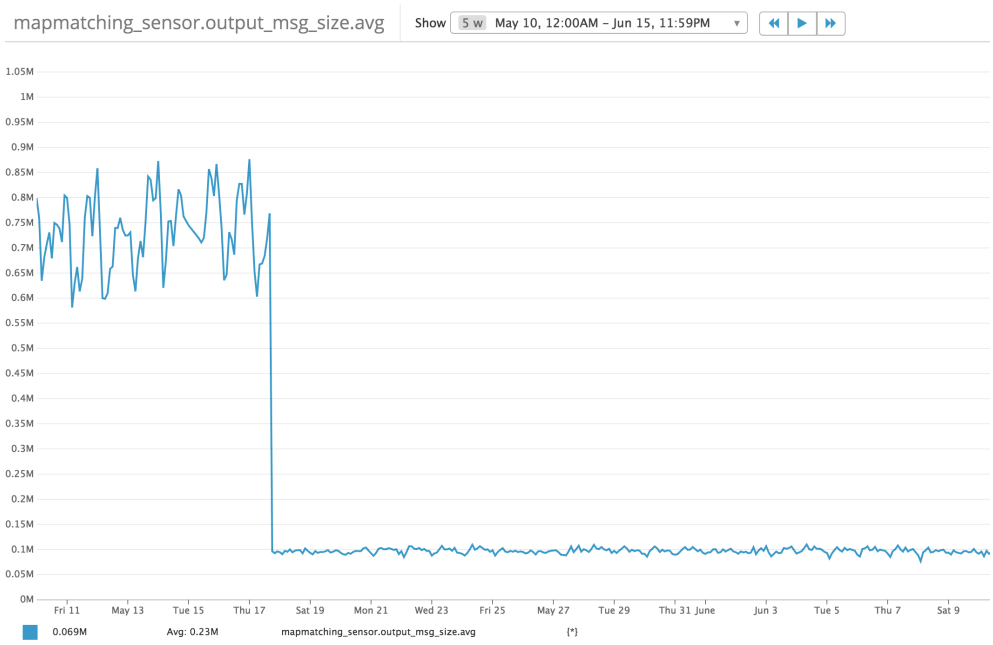

Figure 6.5: Map Matching Average Output Message Size

The average output message size of Map Matching decreased by 8 times.

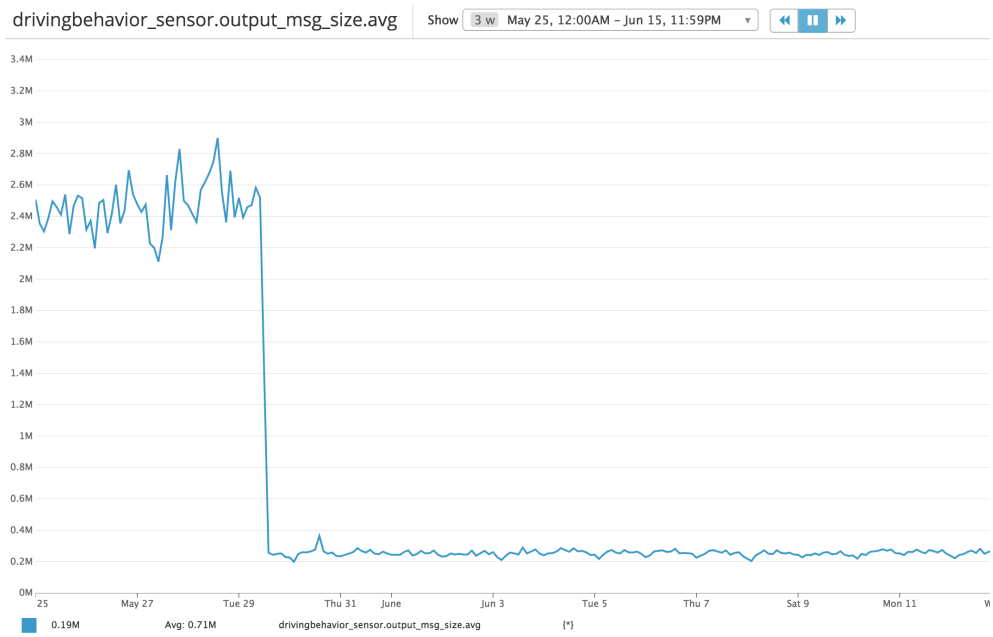

Figure 6.6: Driving Behavior Average Output Message Size

The average output message size of Driving Behavior decreased by 9 times.

6.5 Transport Processor Refactoring

The refactoring efforts of the transport processor Storm topologies yielded the following results:

- 98 unit tests
- From 57 vulnerabilities down to 0
- 0 bugs
- From 169 code smells down to 11
- From 10,1% of duplication down to 0,7%
- From 6,4% of code coverage to 59%

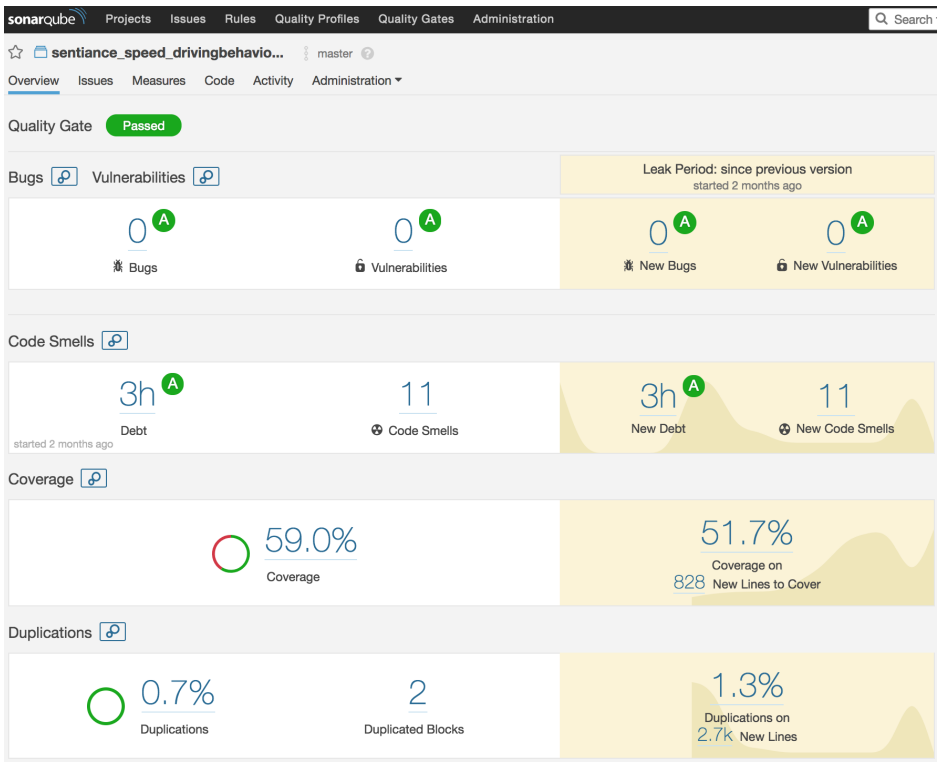

Figure 6.7: Transport Processor Quality Metrics After Refactoring

Certainly, code coverage is still less than the 80% threshold, but it remains an acceptable result.

Conclusion and Future Improvements

Data processing has many trade-offs. It depends on companies' needs and nature of business to whether accumulate the data in preparation for long running batch jobs but with more accurate results, or to process the data on the fly with a questionable accuracy.

The work we accomplished is an attempt to get best of both worlds. While some data analysis tasks still need the characteristics of batch and historical data, more and more Sentiance clients want to receive near real-time insights about their customers.

The different tasks we put together can be considered a Proof of Concept for the first part of refactoring Sentiance data pipeline. The Truck Driver demultiplexer is the basis for such refactoring as it provides a means to select the data that microservices need, hence creating a plugable architecture.
Testing and deploying software is of the same importance as writing code. Streamlining the Jenkins builds helped us create more reliable, maintainable CI/CD procedures.

The work we did is by no means complete. We started working on the integration testing for Transport Classifier 2. This way we ensure that the individual units of the microservice interact correctly. Since designing for failures is one of the crucial rules of distributed computing, we also recommend testing for the different failure scenarios of both Truck Driver and Transport Classifier 2.
Furthermore, we suggest to automate the management of the versions of projects dependencies. At the moment, this is done manually making it more error-prone. A tool like Dependabot is a good candidate for this automation. It periodically checks the dependency files. When outdated requirements are

found in any project, Dependabot corrects the dependencies and awaits for the validation of the project maintainer. This makes programming hassle-free.

References

[1] https://www.quora.com/What-is-the-significance-of-the-Reactive-Manifesto, *(last accessed 20 June 2018)*

[2] https://www.reactivemanifesto.org, *(last accessed 20 June 2018)*

[3] https://www.oreilly.com/ideas/questioning-the-lambda-architecture, *(last accessed 27 May 2018)*

[4] https://www.forbes.com/sites/bernardmarr/2015/09/30/big-data-20-mind-boggling-facts-everyone-must-read/#6afebfa517b1, *(last accessed 25 June 2018)*

[5] https://medium.com/@dpatil/2-5-quintillion-bytes-of-data-are-created-every-day-c908c951f3bf, *(last accessed 25 June 2018)*

[6] https://aws.amazon.com/big-data/what-is-big-data/, *(last accessed 23 June 2018)*

[7] https://www.whishworks.com/blog/big-data/understanding-the-3-vs-of-big-data-volume-velocity-and-variety, *(last accessed 23 June 2018)*

[8] https://sookocheff.com/post/kafka/kafka-in-a-nutshell/, *(last accessed 3 May 2018)*

[9] https://hackernoon.com/thorough-introduction-to-apache-kafka-6fbf2989bbc1, *(last accessed 13 June 2018)*

[10] https://kafka.apache.org/uses, *(last accessed 13 June 2018)*

[11] https://data-artisans.com/what-is-stream-processing, *(last accessed 26 June 2018)*

[12] https://spark.apache.org/docs/2.2.0/streaming-programming-guide.html, *(last accessed 26 June 2018)*

[13] https://www.linkedin.com/pulse/spark-streaming-vs-flink-storm-kafka-streams-samza-choose-prakash/, *(last accessed 27 June 2018)*

[14] https://thrift.apache.org/, *(last accessed 11 March 2018)*

[15] https://en.wikipedia.org/wiki/SonarQube, *(last accessed 25 June 2018)*

[16] https://aws.amazon.com/what-is-aws/, *(last accessed 21 June 2018)*

[17] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html, *(last accessed 21 June 2018)*

[18] https://www.docker.com/what-container, *(last accessed 30 May 2018)*

[19] https://jenkins.io/doc/book/pipeline/, *(last accessed 7 June 2018)*

[20] https://jenkins.io/doc/book/pipeline-as-code/, *(last accessed 7 June 2018)*

[21] https://www.linkedin.com/pulse/sonarqube-55-introduces-code-smells-importance-term-martignano/, *(last accessed 25 June 2018)*