Yevgen Zinchenko

Big Data and Google Cloud Platform

Usage to Create Interactive Reports

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

29 April 2017

Author(s) Title Number of Pages Date	Yevgen Zinchenko Big Data and Google Cloud Platform: Usage to Create Interactive Reports 34 pages 29 April 2017
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Development
Instructor(s)	Juha Koskela, CTO Patrick Ausderau, Principal Lecturer

This study attempts to show how the use of Big Data products on Google Cloud Platform can be a possible solution to process considerable amounts of data. The focus of this thesis is on Big Data processing and modern tools offered on Google's technology stack. The goal of this project is to create a backend that operates with the increasing amounts of incoming data.

Big Data products that are considered in this work are Cloud Pub/Sub, DataFlow and BigQuery. Cloud Pub/Sub offers guaranteed delivery of messages up to 100 million messages per second. DataFlow gives a possibility to concentrate on the logic flow of the application while underlying infrastructure will handle parallel data processing. BigQuery provides storage with quick access to Terabytes of data.

The outcome of this work is a new backend that utilizes Big Data products from Google. The implementation of the backend allows processing of high traffic without performance penalties. This work provides a clear guideline of design and implementation of systems with similar architecture that process big amounts of data.

The new backend offers clear market advantages in processing more data where a company can expand to new markets without the risk of an overload. In future projects it might be really interesting to analyse the data with Google's machine learning products.

Keywords	Big Data, Cloud Pub/Sub, DataFlow, BigQuery, SaaS, PaaS, Google Cloud Platform

Contents

Abbreviations

ack ID	Acknowledgement Identifier.
API	Application Programming Interface.
ATDD	acceptance test-driven development.
CPU	Central Processing Unit.
ETL	Extract Transform Load.
GFS	Google File System.
I/O	Input/Output.
IaaS	Infrastructure as a Service.
JVM	Java Virtual Machine.
NoSQL	Not only SQL.
PaaS	Platform as a Service.
RAM	Random Access Memory.
SaaS	Software as a Service.
SLA	Service Layer Agreement.
SQL	Structured Query Language.

Glossary

App Engine	App Engine is Platform as a Service (PaaS) on Google Cloud Platform.
Big Data	Big Data means large and various sets of information that cannot be proceed with old tools or the cost and time that they need will be unacceptable.
BigQuery	BigQuery is a serverless tool for quick access to the needed data in a data warehouse.
Cloud computing	Cloud computing is a model that implements convenient, omnipresent access to resources that can be accessed with minimal interaction with the service provider.
Cloud Pub/Sub	Cloud Pub/Sub is the implementation of the Pub/Sub pattern on Google Cloud Platform.
DataFlow	DataFlow is a programming model that process big amounts of data in real time and operates with both batch and streaming data.
Docker	Docker is a container platform, used for virtualization.
Google Cloud Platform	Google's implementation of cloud services.
Google Compute Engine	Google Compute Engine is Infrastructure as a Service (IaaS) on Google Cloud Platform that operates with virtual machines.
Google Container Engine	Google Container Engine is IaaS on Google Cloud Platform that uses Docker to run containers.
Pub/Sub	Pub/Sub is a design pattern. Shorthand from the publisher and subscriber. The publisher sends data, the subscriber receives it.

1 Introduction

The amount of data is increasing every day. After switching from the analogue carriers of information to the digital ones it began to grow even more rapidly than earlier. Nowadays petabytes of information are generated every day [1]. To describe that phenomenon the term “Big Data” was introduced. Big Data is a quite often used term and it was first presented in the early 2000s [2]. It describes the same information as previously but in much bigger amounts. The importance of storing, processing and analysing of it grows every day. The old tools such as relational databases cannot operate fast with huge volumes of data. So the modern challenges are to increase the processing speed and to process bigger volumes of data, to handle a variety of the different information that might be without a structure or be semi-structured. So there is a need to make powerful tools that will be fast and reliable and at the same time will operate fast with the increasing amounts of information. Google Cloud Platform is one of those tools. With its products such as Cloud Pub/Sub[1], BigQuery[2] and DataFlow[3] it gives opportunities to work with Big Data in a comfortable way in a cloud environment.

The case organization of my study is ZEF[4]. It is a tech company operating in software development. The main focus is the development of the modern web-based cloud services. The company offices are based in Espoo and Oulu. It currently has more than 30 employees. There are research and development, marketing and sales and service teams. The dream and purpose of ZEF is to revolutionise decision-making. The products are interactive trivias, polls, quizzes that can be used through a web browser or a mobile application. They are used by other companies to get feedback on their products or to ask their users about new changes or just to entertain people with interactive content. The developers are focused mainly on enhancing the product by making new features, improving the performance and introducing a more user-friendly interface. [3]

The business challenge is to make the new product work with big volumes of data, be-

[1]https://cloud.google.com/pubsub/docs/overview Cloud Pub/Sub overview
[2]https://cloud.google.com/bigquery/what-is-bigquery BigQuery overview
[3]https://cloud.google.com/dataflow/docs/ DataFlow overview
[4]http://zef.fi ZEF web page

cause the amount of information generated by the users of those products has increased significantly during the past couple of years and it expected to grow exponentially during the next few years. Also the requirements of the market to process bigger amounts of data faster without data loss makes the company management search for new solutions. At the same time the costs of the product should decrease because of high competition in this market. Therefore ZEF is searching for a solution that would give advantages in comparison with competitors in the cost-effectiveness, speed and reliability of their product.

Currently developers are in the intermediate phase. They are making the new reporting backend alongside with the maintenance of the old one and are implementing a comfortable switch from old to a new one. In this transition phase both backends are available for use to avoid any data loss and to provide a smooth migration.

According to the business challenge this study tries to answer the following question: How can ZEF successfully implement a new reporting backend to analyse a large amount of information using Big Data products on Google Cloud Platform?

2 Theoretical Background

2.1 Big Data

The amount of generated information is increasing from day to day. To describe it the term ‘Big Data’ was introduced. It describes a large and various sets of information that cannot be proceed with the old tools or cost and time that it needs will be unacceptable. Size is not only attribute that used to categorise data, and thus Laney (2001) used three Vs:

- Volume: size of data is mostly over a terabyte
- Velocity: speed of incoming data and time needed to process it
- Variety: different types of data sets

The data expansion in three dimensions is shown in figure 1. [1]

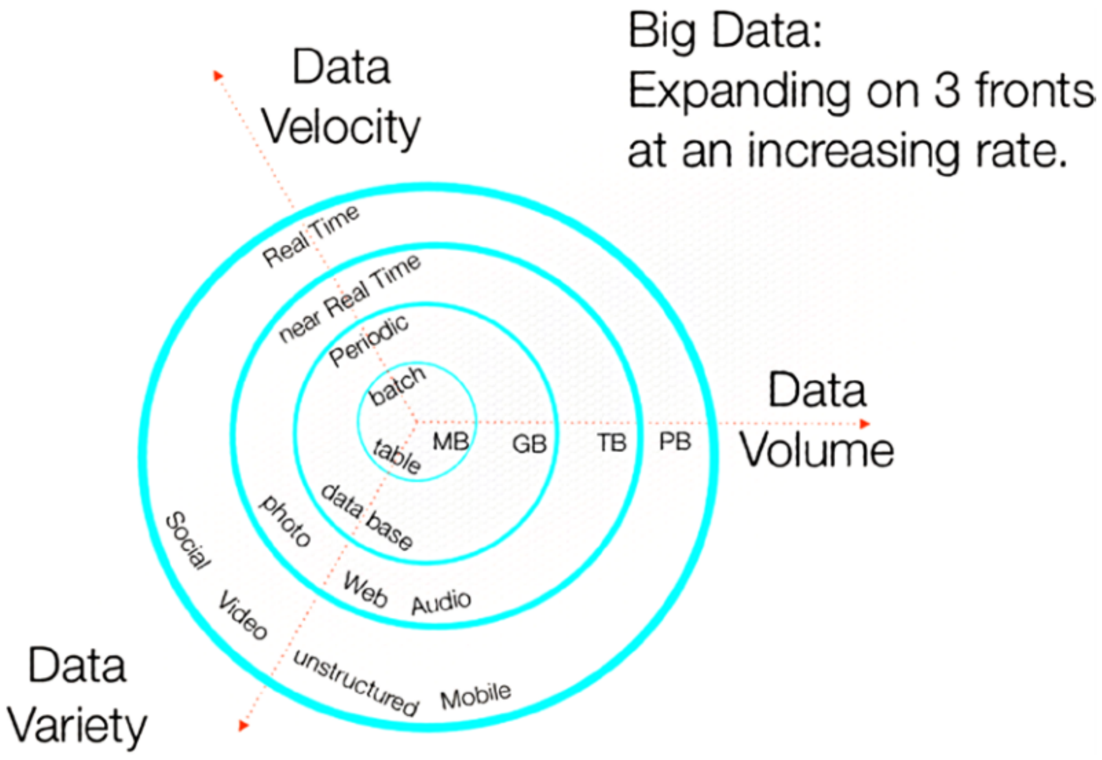

Figure 1: The 3 V theory. Copied from Diya Soubra 2012 [4]

So why did the importance of Big Data increase in a short period of time? First of all, many companies had user profiles with a lot of data on each customer. A good example is social media like Facebook and LinkedIn which track the behaviour of their users online, based on their preferences, friends/connections, joined communities and the like. After analysis of that data suggestions of new friends, groups and targeted advertisements are shown individually. The second reason was that companies with big amounts of data, such as Google and Facebook, went public. It means that now it is possible to get big amounts of data for any company and run its own algorithms to generate useful metadata. The third reason was that people expected to have suitable suggestions based on their preferences everywhere, not just in Google or Amazon products. [5]

With the increased amounts of data the old tools cannot give the needed result. For example the relational databases are not an option for the unstructured data. Based on that a question appears about the characteristics that a new tool should have. A system that successfully manipulates with Big Data needs to meet the following requirements. First of all, it should be able to collect and classify the information, then shift the data to the storage without the loss of it. The storage of this system has even more requirements such as it should not be located on a single server, but it should be distributed on hundreds of servers and be scalable. Also it proposes the data redundancy and the automatic backups even in the case of a failure in the hardware. Plus the whole system should have different tool sets, such as logging tools, reporting tools, Extract Transform Load (ETL) tools, scheduling tools, monitoring tools and the system should process all the data in parallel. A functional graphical interface that is easy to use would be a good addition to all these requirements. [1]

2.2 Cloud Computing

According to Mell and Grance (2011) cloud computing is a model that implements convenient, omnipresent access to the resources that can be accessed with minimal interaction with the service provider. The resources can be various like infrastructure, servers, storage, applications or services. The main characteristics of cloud computing are the following:

- On demand self-service. Server time and storage capacities are provisioned automatically when needed without human interaction.
- Broad network access. Services and infrastructure are available via standard protocols on networks.
- Resource pooling. Multiple users at the same time can use the same physical and

virtual resources as in a multi-tenant model.

- Rapid elasticity. In the case of a rapid increase in the demand of the needed service it will scale as much as requested by the user.
- Measured service. Every resource can be controlled and monitored and the user will be billed according to the usage of the service. [6]

The most common types of cloud computing today are: Software as a Service (SaaS), PaaS and IaaS as shown in figure 2. In the SaaS model users receive a ready-made software application that is deployed and runs on the infrastructure in the cloud. There is no need from the user to know how to maintain or deploy an application. It is possible to use the application from the browser, mobile application or other suitable interface. SaaS is also know as On-Demand software. The most well known example of it is any of the cloud email services, for instance Gmail. [7]

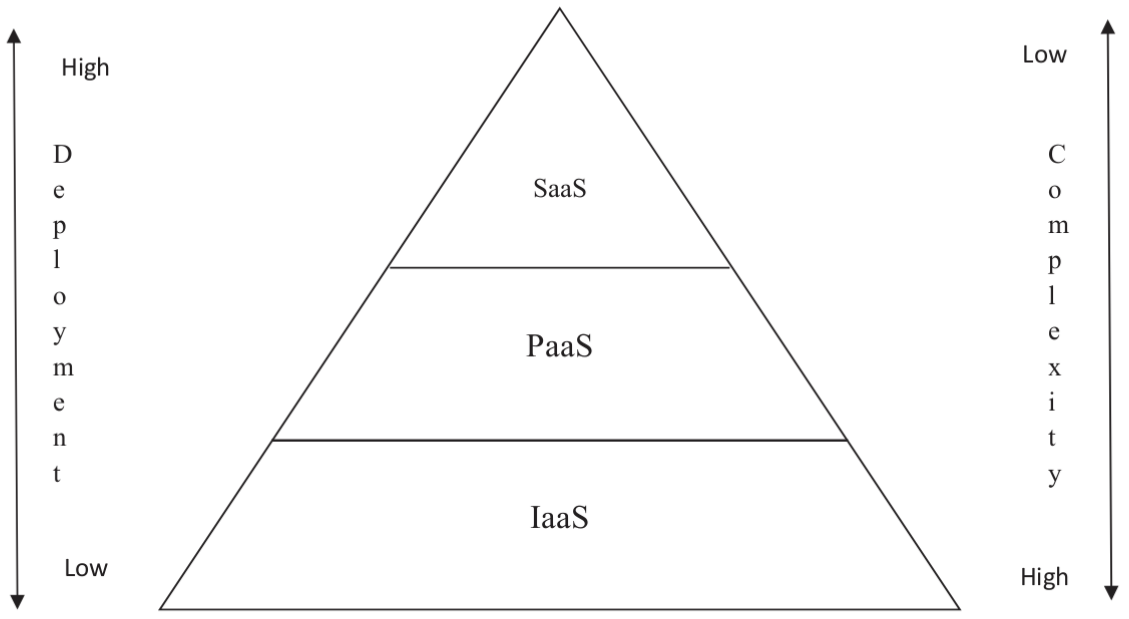

Figure 2: Comparison of the cloud services. Copied from Srinivasan 2014 [7]

In the PaaS model users deploy their own applications on the cloud platform. The cloud service provider gives the underlying infrastructure like servers, storage, network, also provides management and maintenance plus a platform on top of the infrastructure where users can deploy and run their code. On the other hand, it requires from the consumer to have the required IT specialists to maintain and update applications and also to use the only supported programming languages, libraries. [6]

In the IaaS model only the underlying infrastructure, its physical security and support is given. It is also called “Utility computing”, because a company buys only the needed computing resources. It is possible to install any operating system, run all types of the applications, use any programming language and libraries. On the other hand a company should worry about information security, because in this case it is its own responsibility. [7]

The cloud services can be divided by the deployment method. There are public, private, hybrid and community clouds. When a company uses the public cloud it has access to any hardware and software resources that the provider offers often with the pay-as-you-go model. It is an affordable model for small and medium-sized businesses, so they are able to focus on the development of the product instead of setting the servers. Also big companies can benefit from the public cloud especially using computing resources in the peak hours and cloud data centers to store big amounts of data. [8]

On the contrary to the public cloud, the private cloud is only used by one company. It is managed by the company’s employees, has its own firewall and is accessible only to users with the needed permissions. It is also expensive, so not many companies can afford it [9]. A hybrid cloud is a mix of the public and private cloud. Mostly it has the infrastructure as the public part to have more scalability and its own application as the private to have more control of the security [10]. Community cloud computing provides the power of the public cloud and the security features of the private cloud, but only for some segment of industry or to some certain community [11].

2.3 Google Cloud Platform

Google is one of the companies that has both its own infrastructure and software tools, but Google’s approach is to make innovations on every layer no matter whether it is hardware, software or network. Its infrastructure is not only data centers around the world but also advanced networks with thousands of kilometers of the fiber optic cables and even its own cable at the bottom of the Pacific Ocean.

Google has designed energy-efficient servers in a way that they removed unnecessary parts such as video cards and use power supplies that are more energy-efficient. Also not only servers are energy-efficient but the whole data, because it is built of modules and each module includes servers and cooling equipment. In addition the data centers are carbon neutral, what reduces power supply even more. All those factors reduce the price to the end user, which makes a big marked advantage. [12]

Google has released a large amount of innovative software. The Google File System (GFS) was released in 2002 and is used for fast access to the data from the big clusters of the distributed commodity hardware [12]. GFS is based on the principle that any disk can crash, so data is stored in multiple locations and it will still be available to users after a failure [13]. Then MapReduce improved the speed of processing even more. It splits the big task into small pieces that can be proceed in parallel. Apache Hadoop is an open-source analogue of MapReduce that was released later by the community. Later BigTable was released to store large amounts of information on different servers and have multiple copies, which avoid any data losses. In figure 3 interaction between Big Data products on Google Cloud Platform is shown. [12]

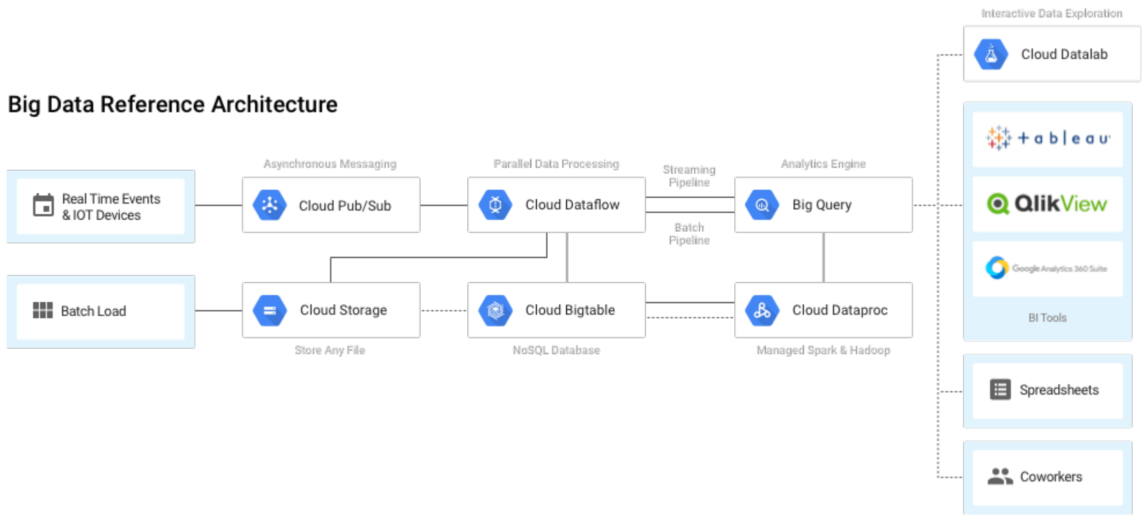

Figure 3: Big Data products in Google Cloud Platform. Copied from [14]

GFS , MapReduce and BigTable improved Big Data processing, but there were still problems that only products from Big Data Stack 2.0 were able to solve. Colossus comparing with GFS provides big performance improvements, but its implementation and documentation is closed for non-Google employees. Megastore is a Not only SQL (NoSQL) database that makes copies of data immediately to different data centers. The spanner resolves problems when there are restrictions, so the data can be only be stored in certain locations, for example only in the EU. BigQuery has a quick access to the needed information just in seconds, Dremel is backend for BigQuery, which process petabytes of information from GFS, Colossus or other file systems in short period of time. The timetable of software releases is shown in figure 4. [12] [13]

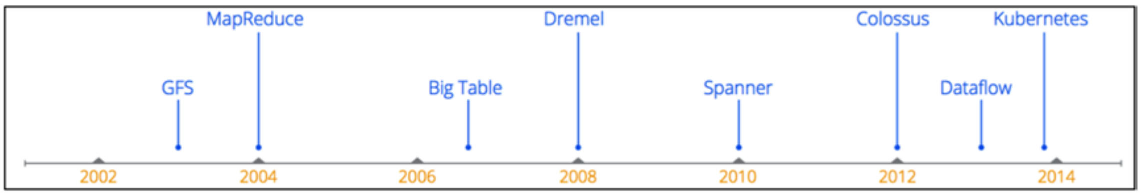

Figure 4: Timetable of cloud products. Copied from Gonsales 2015 [12]

Google Cloud Platform makes all those Big Data products available to companies. It is possible to use them from Google Compute Engine, Google Container Engine and App Engine. Google Compute Engine is an IaaS, user can upload any Linux image and run their code in the virtual machine on Google Cloud Platform. Google Container Engine is an IaaS too, major difference to IaaS that it is run in a Docker container. With use of those tools it is possible to use all needed products without limitations through external Application Programming Interface (API). On the contrary App Engine is a PaaS and with it there is no need to configure operating system or make some network configurations, but it has limitations of platform. [12] [13]

2.3.1 App Engine

App Engine is PaaS that hosts applications, that can be accessed through the web. App Engine process each request in the separate “sandbox”, which means that two requests from the same computer can be directed and handled on different servers. It reduces time and resources spent on a single request plus different applications can run on the same server and there will be no performance penalties. Comparing with the competitors it has very flexible price calculation, which is based only on the resources that were used, no fixed monthly payments. [15]

App Engine offers two environments: flexible and standard. They provide infrastructure to deploy, serve and scale, but the difference is in how it runs, scales and access external services are different in both environments. In the standard environment the application tends to be stateless. This gives performance benefits but introduces some limitations of the sandbox, such as restriction to use libraries that are not in the white list or disability to write to the disk. Also memory and Central Processing Unit (CPU) usage for one instance is limited in the standard one, but can be configured in the flexible environment. Moreover, it is possible to write to the disk and use any libraries or programming languages without limitations, because the application runs in the Docker container. One more difference is in accessing external services: the standard one uses built-in API, while the flexible external. Also scaling is different: standard scales much faster and has no limitation of the minimum running instances. [16] [17]

On the top level the application consists of the services. Services are components of the application and help to split it into logical parts that can share App Engine features. It is done like in the microservice architecture. Each service can be configured differently and have its own runtime and it consists of the configuration file and the source code itself. One of the important things in the configuration is version of the service, because it is possible to deploy different versions of the same service. It helps to switch back to the old versions in case of critical errors, redirect traffic from one version to another one without uptime or even to increase traffic step by step to a new version and see how it works with higher load. [18]

One of the advantages of App Engine is that application scales when there is a greater demand for it. It means that more computation power, network traffic, etc is allocated for this app, while app itself knows nothing about it. App Engine creates a new instance of the required version when more resources are needed and destroys extra instances after a traffic peak to reduce costs. The module hierarchy is illustrated in figure 5. [18]

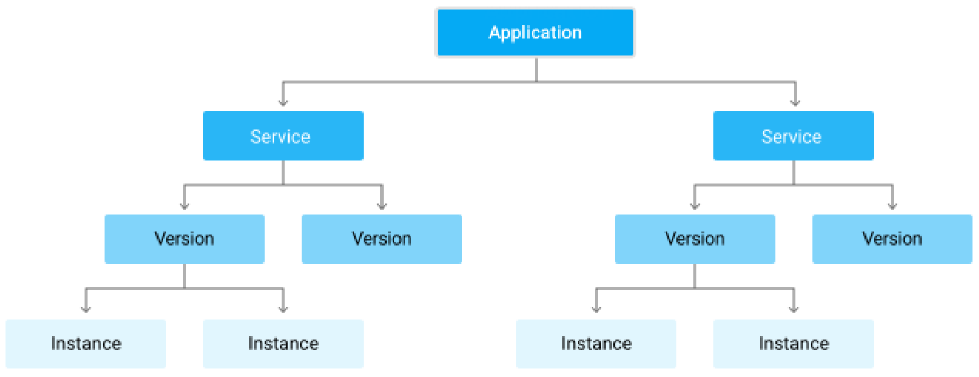

Figure 5: Modules hierarchy copied from [18]

Scaling can be configured and there are three types of it. Every instance will be crated depending on the scaling type. App Engine offers three types of scaling: automatic, manual and basic. When developers use manual scaling the service will run continuously. This type makes it possible to do complex initialisation. The basic one creates the instance of the service on the user request and turns down it when it is idle. Automatic scaling creates and destroys instances depending on the number of requests, size of queues and other data. [18]

3 Methods and Materials

This study was done by analysing the previous versions of the reporting backends, older versions of the survey tools done by Zef and the products that were done by the competitors, alongside with the technologies that were used during their development. Also the current development process was analysed and it was a compared with the recommendations to the agile methodology and software development in general. Besides the best practices in general and in processing big amounts of data were used.

The data was collected from Google's official documentation, books about cloud computing, bid data and specific products on the Google Cloud Platform and the platform itself. Interviews with the company employees were conducted and evaluated to clarify specifications and to gain more knowledge about the previous problems and mistakes. The test data was collected with the use of the previous version of the tool. During the development process, manual testing was done and unit tests were created to verify the product quality.

3.1 Research Approach and Design

While choosing among the different research approaches, the most suitable one was chosen. In this case it is the case study approach, because of specific context and company requirements.

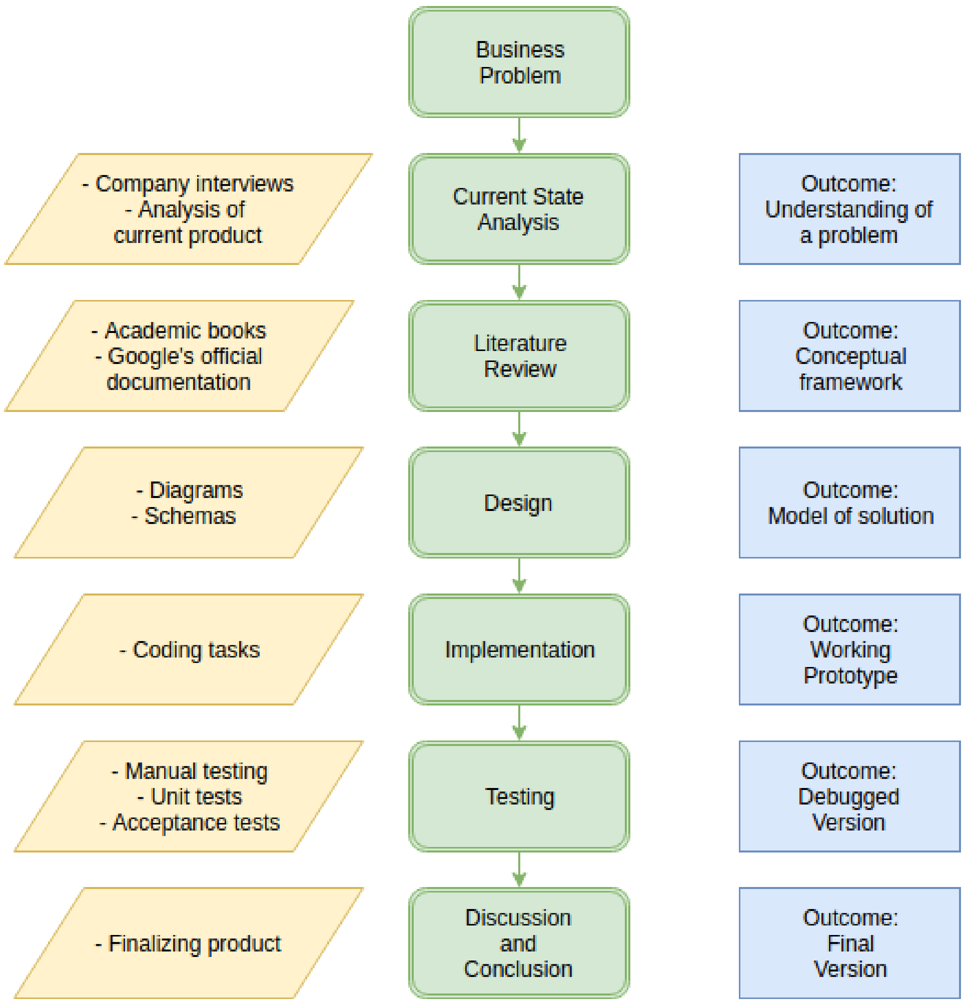
Business Problem
- Company interviews - Analysis of current product
Current State Analysis
Outcome: Understanding of a problem
- Academic books - Google's official documentation
Literature Review
Outcome: Conceptual framework
- Diagrams - Schemas
Design
Outcome: Model of solution
- Coding tasks
Implementation
Outcome: Working Prototype
- Manual testing - Unit tests - Acceptance tests
Testing
Outcome: Debugged Version
- Finalizing product
Discussion and Conclusion
Outcome: Final Version

Figure 6: Research design

The steps of the research design are shown in figure 6. It illustrates that solving the business problem will start from the interviews of employees, the analysis of the current situation, previous solutions that were implemented and cannot be extended to fulfill the new requirements that Zef is facing. After the problem is clear, a review of the latest documentation from Google Cloud Platform and search for the best practices in literature will be done. Based on that knowledge, design of the reporting backend will be created. The design will be validated and implemented. The result will be tested and verified.

3.2 Tools and Technologies

For my project I have used a lot of various tools. In this subsection I will describe the most important of them.

3.2.1 General Tools

Java is one of the most important tools. At the same time Java is a programming language and a platform where Java and the languages like Scala, Groovy and others are compiled and run. As a language it is a class-based object oriented language that has much in common with other languages. As a platform it executes the compiled programs that are written in the Java-based languages. It has has a virtual machine and an execution environment in comparison with the other platforms that might have a physical processor or an operating system. Java Virtual Machine (JVM) is an associated execution environment, it has a lot of prebuild functionalities known as a standard class library. [19]

Apache Maven is a tool for the understanding and management of the different Java based software projects. It based on the concept of the project as an object and it can easily handle the build and the documentation from one place. Maven allows to create the changelogs directly from the code that was put under the source control, it keeps the source code tree and the unit test tree separately, but it has a convenient name convention that helps to find and run tests. [20]

Cloud computing brings to the modern companies a lot of benefits. First of all, there is no need to worry about the hardware: setup a server, install an operating system, configure a deployed project, make an acceptable runtime and also the staff should be able to maintain the servers and in case of some emergencies be able to fix them. The different cloud platforms provide 99.9% uptime on an average, which means the services are online and available for usage. The uptime percentage is legally bound in the Service Layer Agreement (SLA) and for the most of the Google Cloud Platform's products is 99.95% [21]. In time values it is like a couple of hours per year, what is hard to achieve for small companies. [12]

The public cloud provides flexibility. The amount of resources such as the networks and disc traffic, Random Access Memory (RAM), CPU that are needed for a company's applications vary depending on the industry, the different system components or even on the time of the day. So it is hard to achieve a high resource utilization even for the middle sized company with its own equipment, but it is easy on the public cloud. Another benefit from the usage of the public cloud is cost efficiency. The amount of money spent on all hardware components, deployment, maintains, salaries and training provided to the staff is lower with use of the cloud. Moreover, after some period of time upgrading of the components or a full system itself will make ownership of servers even more expensive than monthly billing from the cloud provider. That is why the return on investments and the profit margin will be higher especially in the case of small companies. [12]

With Google Cloud Platform it is possible to utilise the benefits of the usage of cloud computing and one of its cases, the public cloud. Also there is a possibility to use extra Google services. In Zef's case Big Data products of Google Cloud Platform are the most useful. These products are Cloud Pub/Sub, DataFlow and BigQuery.

3.2.2 Cloud Pub/Sub

Pub/Sub is shorthand from publish and subscribe. According to Chockler (2007) it is a paradigm for supporting many to many communication in a distributed system. In such a system messages are sent in the abstract channels that are called topics. The users get only the needed information from the topics where they have subscribed to. The system guarantees that all users that are subscribed on a certain topic will receive all new messages. Pub/Sub is used in different types of applications because of its reliability and simplicity, especially in enterprise applications. [22] Pub/Sub model is shown in figure 7.

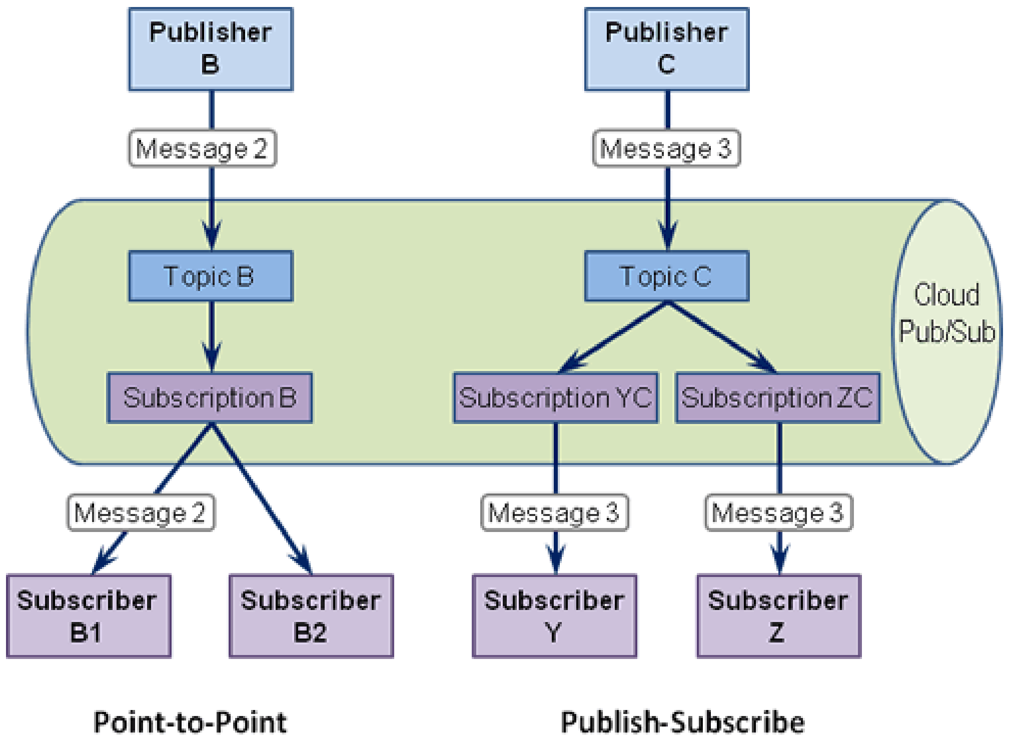

Figure 7: Google Cloud Pub/Sub model. Copied from [23]

Google has implemented this pattern as Cloud Pub/Sub on its platform. With Cloud Pub-/Sub it is possible to sent and receive the messages from any component hosted on the Internet with a guaranteed delivery. It can send up to 100 million messages per second. The system scales horizontally. It means when the number of the topics, subscriptions or messages will increase Google Cloud Platform will create new instances. Cloud Pub/Sub consists of a control plane and a data plane. The data plane transfers messages between publishers and subscribers. The control plane assigns publishers and subscribers to physical machines. The integration of Cloud Pub/Sub with the other products on Google Cloud Platform is shown in figure 8. [24] [25]

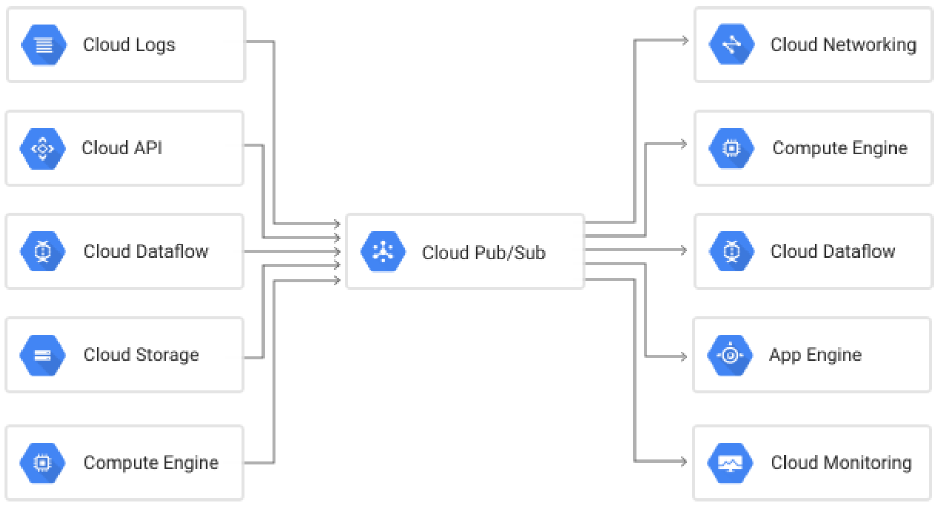

Figure 8: Google Cloud Pub/Sub among products on platform. Copied from [24]

3.2.3 DataFlow

Accarding to Akidau (2015) DataFlow is a programming model that has abstract semantics. It makes it possible to operate with batch, micro-batch, different streaming models and Lambda Architecture. It was designed to proceed big amounts of the data in real time without penalties on a correctness of the results. There is no need to think about the performance issues because this tool is fully managed. Scalability of DataFlow is almost limitless. No matter how big the request is, Google will allocate the needed resources. DataFlow is designed in the way that it is very suitable to connect Cloud Pub/Sub subscription to it. [26]

With the usage of DataFlow developers can concentrate on the logical flow of the information, while low level details are handled by a data processing jobs. These jobs are executed by DataFlow runner service. DataFlow consists of four major concepts: Input/Output (I/O) sources and sinks, pipelines, transforms and “PCollections”. [27] Pipeline is a representation of the data processing job. It consists of the data and its transformations. The pipeline can have a simple linear flow or it can consist of the loops and if/else statements. A program can have multiple pipelines, but the data and the transforms are exclusive for a certain pipeline and cannot be shared between pipelines. [28] The pipeline I/O provides interfaces to read and write the data from different sources. It can be files in the Google Cloud storage or a BigQuery table. The read and write operations transform data to or from PCollection, so it can be processed in a pipeline. [29]

All data in DataFlow is wrapped into a PCollection. It is a container that includes big amounts of data without upper limits of the elements or memory used. Almost any type of the data can be uploaded to it, but there are some limitations. PCollection is binded to the pipeline and cannot be shared with other pipelines. It is immutable, so it cannot be changed. There is no way to get a random element from the collection. Depending on the size of the received data, PCollection can be bounded or unbounded. The bounded has a fixed size, while the unbounded is a a stream of data. Each PCollection has a timestamp that is used for windowing. [30]

Windowing splits stream of the data to the elements that can be proceed individually. In other words it makes bounded data from the unbounded one. DataFlow can proceed both with batch and streaming data. Batch processing assumes the data to have the end of the input, DataFlow slice unbounded data to the finite pieces. There are three windowing strategies: fixed, sliding and sessions. The fixed one slices the data with a fixed time sized window. Sliding makes windows that also have a fixed size, but windows overlap because of the sliding period. Sessions strategy generates dynamic windows that are based on the inactivity period between the data sets. [31] [32] [33] Windowing strategies are shown in figure 9.

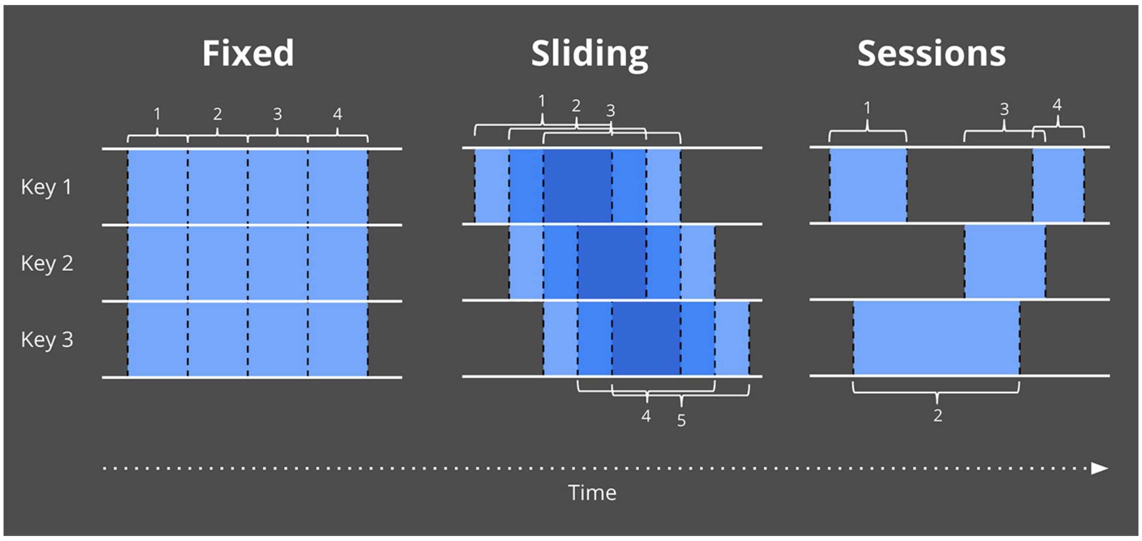

Figure 9: Windowing strategies. Copied from [32]

The transforms are the stages of the data in a DataFlow pipeline. They implement the logic of the pipeline. Different operations can be done on the data in the transforms:

computation, grouping, filtering or combining. It is possible to nest transforms in each other. Every transform has one or multiple inputs and produces one or multiple outputs. PCollections represent inputs and outputs. [34]

3.2.4 BigQuery

BigQuery is a serverless tool for quick access to the needed data in the data warehouse. A typical database is limited by the disk speed even if a query is run in parallel. Dremel is the backend of BigQuery. It executes the queries in parallel on a hundreds or even thousands of servers. The goal is to execute a query on 50 Gb of data on an average of 1 second, but because of the complex queries that use joins or a high load on the system time could be up to 5 seconds. BigQuery stores the data in the cloud in the different data centers in the world. It also replicates existing clusters to different locations, so in the case of an emergency in one of the data centers, users will always have access to the data. [13]

BigQuery is neither an Structured Query Language (SQL) or a NoSQL database. That's why it cannot be used to substitute relational database with the data that needs to update certain rows or perform subqueries on them. One more reason not to use BigQuery in, for example saving customers in database, is because Dremel limits the number of queries from one user to proceed them fast. In comparison with NoSQL it has better scalability, but it is harder to update the data in rows. [13]

3.2.5 Testing Tools

In this project various testing tools were used to ensure the code quality. For the unit testing JUnit was used. It is a framework for automation of test runs. It consists of test suits that include test classes with test methods. The developers can run JUnit tests in sequences or individually. Because of long initialisation of large projects, it is better to have sequences of tests for big projects. [35] [36]

Another framework for testing that was used in this project is Robot Framework. This generic framework is used for the acceptance testing and acceptance test-driven development (ATDD). Its syntax is simple in use, because of the tabular format and the use of the keywords. Moreover, it is an open-source project and it can be easily extended with the use of two popular languages: Java and Python. Test cases can be easily structured with the use of built-in keywords and modular structure. [37]

To run Robot Framework tests automatically Jenkins was used. It is a self-contained server, which is used for the automation of the different processes that include building projects, testing them and deploy to the remote server. Jenkins can be installed on a machine with the Java Runtime Environment, it can be a standalone machine, a Docker container or a remote server on native system. The automated builds are done in a pipeline. It is a suite of plugins, that provides the implementation of the continuous delivery and integrated it to the Jenkins. [38]

For the comfortable testing of the application Google provides Cloud Pub/Sub emulator. With the use of it developers can emulate locally environment that is configured in production with the Cloud Pub/Sub. Emulator uses “gcloud”, it is a command line based tool. When starting the emulator it is possible to set options like directory with the data. Also the environmental variables are needed to be configured before the first use. [39]

4 Results

The outcome of the study is a new reporting backend and a guideline. Guideline analyses how backend was implemented, deployed and tested. Also it includes proposals for caching and other improvements. The scope of my research includes the full cycle of data from it being generated by a user till it is available in the analyze section. The workflow of the old reporting server was not covered by this study nor other parts of backend and the whole front end, because Big Data products or Google Cloud Platform and their use in this work are so wide that they were left outside of the scope.

4.1 General Workflow

First, the client company representatives create a survey with the help of the Zef survey tool from scratch or with the help of ready-made templates. After all needed adjustments are done and the survey is ready, it will be published so end users can use it to make their responses. It is possible for end users to answer in a browser or on a mobile. The mobile view of the survey is shown in figure 10. After each answer the front end client sends the data to the backend.

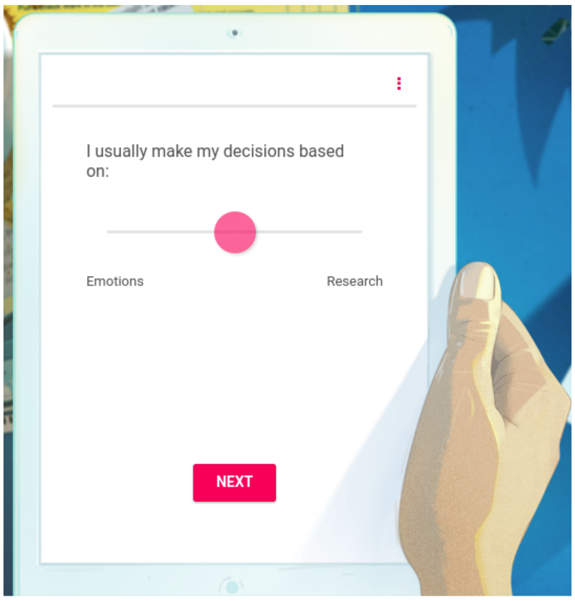

Figure 10: One questions of Zef survey on mobile. Copied from [40]

The backend is deployed on the Goggle App Engine. For this project we have used a standard environment, because a flexible environment is not available in the EU. Dynamic scaling of the project instances was chosen as an optimal solution for the company needs. The App Engine takes care of scaling of the application in case if current instances receive more requests than they can proceed, a load balancing for the newly created instances is done automatically too. It means that no matter how many requests per second the backend will receive there will be as few queues as possible and no lost data because of the timeouts. After user peak App Engine will shutdown the extra instances that are not needed anymore. [41]

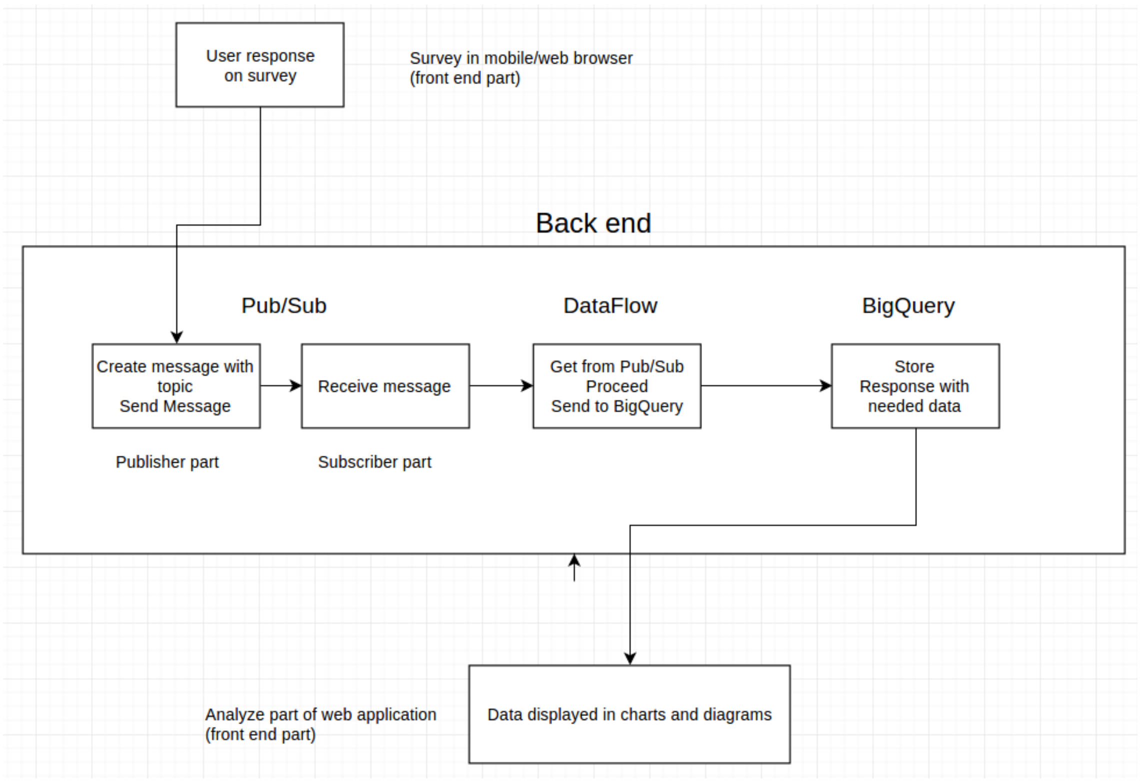
User response on survey
Survey in mobile/web browser (front end part)
Back end
Pub/Sub
DataFlow
BigQuery
Create message with topic Send Message
Receive message
Get from Pub/Sub Proceed Send to BigQuery
Store Response with needed data
Publisher part
Subscriber part
Analyze part of web application (front end part)
Data displayed in charts and diagrams

Figure 11: General flow of the data with Zef survey tool.

After the backend has received a request, it checks that all needed fields are present and all verifications are passed. If a request has passed the checks, Cloud Pub/Sub publishes it in a message with a “save” topic. Then a subscriber part with the correct subscription receives it. The subscriber is connected to DataFlow. DataFlow reads data from Cloud Pub/Sub, filters, combines it and writes it to the BigQuery. BigQuery stores the information and gives needed data to the analyze part of the front end on the requests. The general workflow schema of the data processing is illustrated in figure 11.

4.2 Cloud Pub/Sub

The Big Data products on the backend begins with publisher part of Cloud Pub/Sub. Before messages can be published an initialisation is needed: the topic needs to be defined and a request to the system is needed to be sent to activate it. After that a new topic is visible in a Cloud console and it is possible to send the actual data. After the data is verified, backend creates a new message and set data and optional attributes to it. Then the message is published in the “save” topic. Publishing of the new message is shown in listing 1. [42]

```java
Pubsub client = PubsubUtils.getClient();
PubsubMessage pubsubMessage = new PubsubMessage();
pubsubMessage.encodeData(receivedData.getBytes("UTF-8"));
PublishRequest publishRequest = new PublishRequest();
publishRequest.setMessages(ImmutableList.of(pubsubMessage));
client.projects().topics()
    .publish("save_topic", publishRequest)
    .execute();
```
Listing 1: Cloud Pub/Sub publish event

The subscription is needed to receive a sent message from Cloud Pub/Sub. To create it a developer needs to specify the topic to subscribe to. For the push deliveries an extra step is required: to set a push endpoint. There are two types of message deliveries: push and pull. In the push one the publisher initiates a message delivery by sending a request to the subscriber. The subscriber has been predefined beforehand. If the reply to a message contains HTTP success code, it will be deleted from the subscription; otherwise the publisher will resend it. Depending on the responses Cloud Pub/Sub dynamically adjusts the flow of the requests. In a pull subscription, the subscriber makes requests to the API pull method. Then it calls for a message in the subscription queue. If there is a message Cloud Pub/Sub will send it an Acknowledgement Identifier (ack ID), otherwise it will send an error. Then the subscriber sends the acknowledgement with ack ID. The pull subscriptions are recommended for a large number of subscribers or for subscribers on Google Compute Engine. Push - for the subscribers that have low traffic or even can be offline and also for subscribers that need data in almost real time. Cloud Pub/Sub pull and push schemes are shown in figure 12. [43]

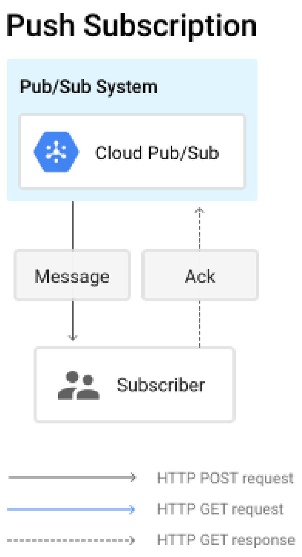

Figure 12: Cloud Pub/Sub delivery flows, copied from [43]

In this project the pull subscription is used. After the subscription is created, the system provides a sync point. It means that the subscriber has now a guaranteed delivery of all messages, but only after this point. After the message is received by a subscriber, it will notify the publisher by sending an acknowledgement for the pull subscription or a success response for the pull one. The deadline for the ack by default is ten seconds, but it is possible to create a custom deadline for the acknowledgements. Moreover, for the pull subscriptions it is possible to modify a deadline for each message individually. After the basic setup is done, the incoming messages will be retrieved and handled. The subscription is shown with the part of DataFlow pipeline in listing 2. [43]

4.3 DataFlow

First, the pipeline was designed according to the design principles. The pipeline begins when DataFlow receives the data from the Cloud Pub/Sub subscription. In this case it is an unbounded stream of data that can contain information about the answers, the top results or the responders. Because of the unbounded data there is a need for windowing. Session strategy was chosen as the best suitable among the windowing strategies. After a batch of the data is formed, DataFlow will convert JSON to suitable Java object. Depending on a type of data, it proceeds it with a different flow of pipeline. The results are filtered, combined and written to BigQuery. This project DataFlow pipeline is shown in figure 13. [44]

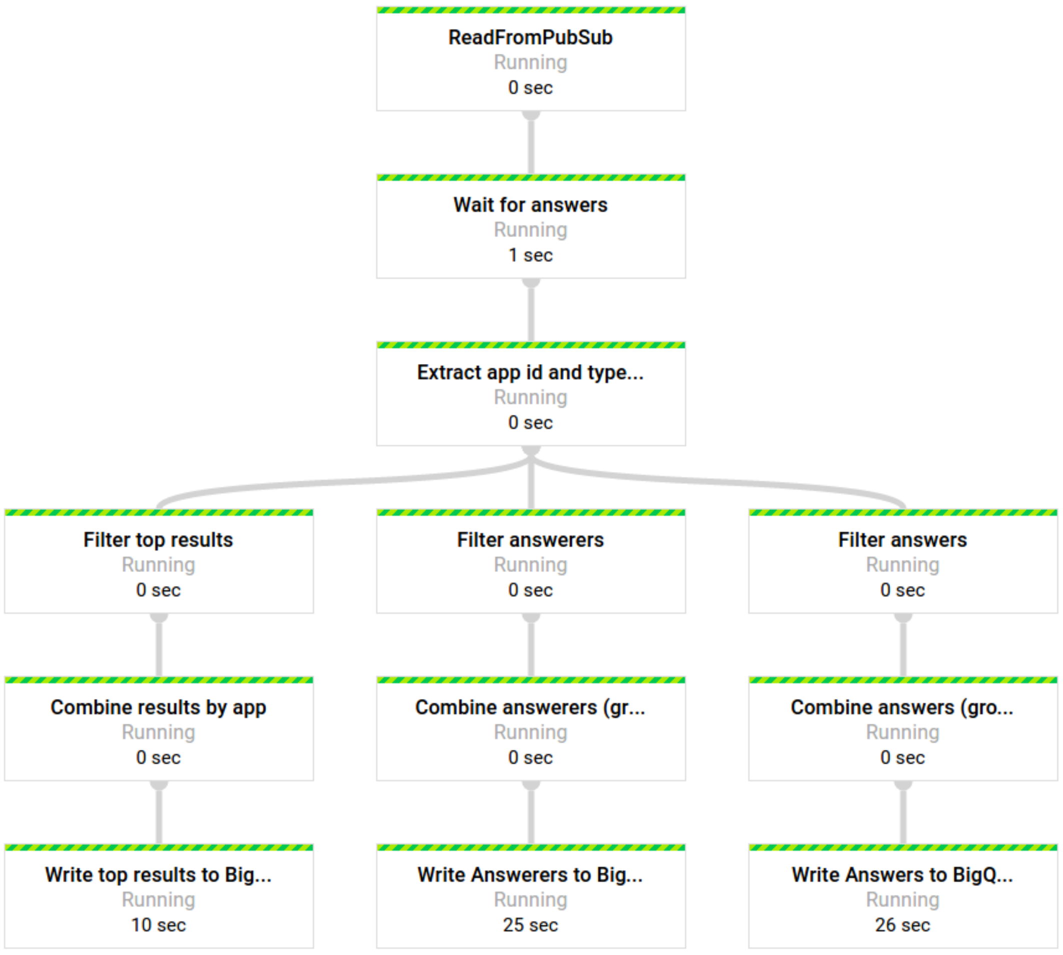

Figure 13: Pipeline of DataFlow, screenshot from Google Cloud Platform web interface

The implementation of the DataFlow starts from creation of the pipeline object and defining its options. The options can contain information about the number of the instances of Google Compute Engine, project and place of staging files. Then the source of the input is defined as a stream from the Cloud Pub/Sub with the “save” topic. The windowing is defined a the 60 second timeout. Then the transforms are applied to the pipeline. The output was written to the BigQuery. As the next step, the pipeline is launched. Some transforms, options creation and writing the output were omitted in the code snippet. Part of the pipeline is demonstrated in listing 2. [45]

```
PTransform<? super PBegin, PCollection<String>> readSource =
      PubsubIO.Read.topic(options.getPubsubTopic());
```

```
WindowFn<Object, ?> windowFn = Sessions.withGapDuration(
    Duration.standardMinutes(1));
Pipeline p = Pipeline.create(options);
PCollection<KV<Long,String>> messages = p.apply("
    ReadFromPubSub", readSource)
     .apply("Wait for answers", Window.<String>into(windowFn)
        .triggering(AfterFirst.of(
         AfterProcessingTime.pastFirstElementInPane().
             plusDelayOf(Duration.standardSeconds(60)),
         AfterPane.elementCountAtLeast(100))
     ).withAllowedLateness(Duration.standardDays(2)).
        discardingFiredPanes())
     .apply("Extract app id and type from json", ParDo.of(new
         ConvertFromJsonWithKeyFn()));
p.run();
```

Listing 2: Part of DataFlow pipeline

4.4 BigQuery

Before the beginning of writing the data to the BigQuery the billing has been enabled, the quotas have been configured and the table for the data has been created. There are several ways to write the data to the BigQuery from the Cloud Datastorage, CLoud Datastore, DataFlow, from CSV, Avro, SQL and JSON files. Streaming from the DataFlow was chosen as an optimal solution. After the insertion into a table, the data is available for analysis in a couple of seconds and for the copy and export operations it might take up to 90 minutes. The data appends to the end of the table after request. The part of streaming to the BigQuery is shown in listing 3. [46]

```
List<Rows> tableDataInsertAllRequestRows = new ArrayList<
    Rows>();
TableDataInsertAllRequest.Rows bigQueryRows = new
    TableDataInsertAllRequest.Rows();
bigQueryRows.setInsertId(id);
bigQueryRows.setJson(jsonFromDataFlow);
tableDataInsertAllRequestRows.add(bigQueryRows);
TableDataInsertAllRequest request = new
    TableDataInsertAllRequest().setRows(
    tableDataInsertAllRequestRows);
TableDataInsertAllResponse response = bigquery.tabledata().
    insertAll(project, dataset, table, request).execute();
```

Listing 3: Writing data to BigQuery

There are several ways to get the saved data from BigQuery: a synchronous query, an asynchronous query, a batch query and an interactive query. In this project the synchronous ones are used. The synchronous query waits till BigQuery will respond with a result. While waiting a temporary table is created and after the response is received it will be filled with the data. An example of the query that is used to read data from BigQuery is shown in listing 4. When the table is formed, it will be returned to the analysis part of front end. [47]

```
BigQuery bigquery = new BigQueryOptions.
    DefaultBigqueryFactory().create(BigQueryOptions.
    getDefaultInstance());
QueryRequest queryRequest = QueryRequest.newBuilder(
    queryString)
        .setMaxWaitTime(waitTime)
        .setUseLegacySql(useLegacySql)
        .build();
QueryResponse response = bigquery.query(queryRequest);

DataTable dataTable = new DataTable();
//only one row from dataTable is shown
dataTable.addColumn(new ColumnDescription("amount",
    ValueType.NUMBER, "Amount"));

  QueryResult result = response.getResult();
  Iterator<List<FieldValue>> iterator = result.iterateAll();
  while (iterator.hasNext()) {
    List<FieldValue> row = iterator.next();
    TableRow tableRow = new TableRow();
    tableRow.addCell(BigqueryUtil.getAsDouble(row.get(0)));
    dataTable.addRow(tableRow);
  }
}
//wrire dataTable to response
```

Listing 4: Reading data from BigQuery

4.5 Testing

During the development process manual tests were performed all the time. The unit tests were written after each step to ensure correctness of work. JUnit framework was used for unit tests. Also they were integrated into the build process. Cloud Pub/Sub message delivery was tested locally with the use of the Google Cloud Pub/Sub emulator. When both the backend the front end were ready, end-to-end tests with the Robot Framework were created and deployed to the Jenkins server to track that everything worked smoothly.

The indirect way of building DataFlow pipeline on the server makes it one of the most interesting and important parts of testing. It is not a trivial task to debug a pipeline on the server. There are three ways to test DataFlow locally. One of them is to test individual objects. Any DoFn[5] object can be tested with its own tester object. On a higher level it is possible to test the composite transforms. The whole pipeline can be tested with end to end tests. In this project individual tests of DoFn objects and of the whole pipeline are made. [48]

Individual DoFn objects can be tested with a tester object “DoFnTester”. The test object can be run inside of the JUnit framework. To use it, first, the tester object is needed to be created. Then the collections of the test data are prepared. Data is inserted into the main input and for side inputs, if they are present. After the inputs are ready a call to the “proseccBatch” method is done. In the end one need to verify received data with the expected one. One of the tests for individual objects is shown in listing 5. [48]

```
@Test
public void testOneOfFilters() {
  Gson gson = GsonUtil.getGson();
  SaverPipeline.Filter filter = new SaverPipeline.Filter();
  DoFnTester<KV<Long, String>, KV<Long, String>> fnTester =
     DoFnTester.of(filter);
  List<KV<Long,String>> result = fnTester.processBatch(KV.of
     (6L, jsonInputString));
  SavePubSubMsg msg = gson.fromJson(result, SavePubSubMsg.
     class);
  assertEquals(expResults.getResult(), msg.getResult().
     longValue());
  assertEquals(expResults.getAppId(), msg.applicationId.
     longValue());
  assertEquals(expResults.getKey(), result.get(0).getKey().
     longValue());
}
```

Listing 5: Unit test of one of DoFn objects

[5]argument that provides code to use to process elements of PCollection https://cloud.google.com/dataflow/java-sdk/JavaDoc/com/google/cloud/dataflow/sdk/transforms/DoFn

5 Discussion

The result of this project is deployed, fully tested and documented reporting backend. From a technical point of view, the result of this work has significant implications on Zef. Previously, it was challenging to process big amounts of data with old tools when servers would fail to perform due to the high load. At present, it is possible to extract, load and transform data almost without limitations. Based on SLA and stability of the system after launch, it is certain that backend is reliable, fast and stable.

From a business point of view, new backend offers clear market advantages in processing more data where a company can expand to new markets without the risk of an overload. This project is fully scalable and can be easily extended. Cloud Pub/Sub can be extended by adding new topics for messages. DataFlow can be enhanced with new pipelines and new functions to existing ones. BigQuery can be widened by adding new tables or rows to current ones. Moreover, the backend can be used for new versions of front end almost without any changes.

Despite the smooth flow of the project, some problems occurred during the development process. A few APIs were in the beta version where several minor changes were implemented during backend development. Moreover, Google's documentation was being updated and improved constantly without a clear presentation of the important details of the changes. Thus, a limited visibility of the following steps existed.

A few interesting decisions were made during the project development. From the DataFlow windowing strategies session one was chosen. The fixed-sized and sliding strategies were not suitable for this project because of the nature of the received data. In these strategies data is divided just based on time and brings no extra value. The sessions strategy slices the data based on the inactivity gap, so it is possible to analyze user behaviour over time during data generation.

Streaming from DataFlow was chosen as a most suitable way of loading data into the BigQuery. First of all, the solutions that allow accessing data from the BigQuery without uploading, for example public or shared datasets or external data sources, were rejected because mostly the customers wanted to keep the data privately. In addition the data is generated in real time, so there is no way to have it predefined. For the same reason files or Cloud DataStore were rejected. Writing to them as intermediate steps makes the system more complex and brings no benefits.

BigQuery synchronous interactive queries were chosen as best suitable. The difference between batch and interactive queries is that the interactive one is executed as soon as possible. The batch query, on the opposite, will wait until the resource is available and only then executes the query. The synchronous query returns the result after all the needed data was extracted from the BigQuery. The asynchronous query makes extra calls to check the updated data after the first response was returned. Synchronous interactive queries best suit the client server model, because requests for the analyze section are received mostly from web browsers.

The objective of this project was to implement a new reporting backend to analyze big amounts of data with the use of Big Data products of Google Cloud Platform and it is considered to be met. I tried to keep as close as possible to objective of the project, so the result, to develop a working and fully tested new reporting backend, meets all the requirements of the objective. The research approach and design were stated clearly: case study was used in this project. The correct tools were used to achieve the best results: among different cloud services Google Cloud Platform was chosen as the most suitable one. A literature review and study of the official documentation of Google were done sufficiently to present a clear understanding of the project background. Future study can utilize the outcome of this work for further evaluation of the usability of Big Data products on Google Cloud Platform where the results of this work can be compared to the needs of a new study.

6 Conclusion

The objective of this study was to implement a new reporting backend to analyze large amounts of the information with the use of the Big Data products of the Google Cloud Platform. Alongside with the development of the new system, maintenance of the old one was provided. In order to implement a transition from an old backend to a new backend, both backends should be available for users during the transition phase. In this way, providing a smooth migration would allow avoiding loss of data.

The outcome of this study is a new reporting backend. Its design is based on a literature review, official Google documentation and best practices. After development, it was tested manually and covered with unit tests and end-to-end tests. The ready tested version is deployed in production environment. The objective of this study was to develop a new reporting backend to analyze large amounts of information with the use of the Big Data products of the Google Cloud Platform and it is considered to be met.

This study shows how Big Data changed the production environment of companies. It also gives a clear example of how Big Data products can be implemented in modern companies using Google Cloud Platform. During this project Google released machine learning and video recognition on its platform. It might be really interesting to generate metadata from the video files or analyze data with machine learning products in future projects.

References

1 Frampton M. Big Data Made Easy: A Working Guide to the Complete Hadoop Toolset. Apress; 2014.

2 Lohr S. The origins of ‘Big Data’: An etymological detective story. New York Times. 2013;1.

3 Martikainen T. ZEF | Our story. Zef Oy; 2016. [Online; accessed 14-January-2017]. http://zef.fi/en/story/.

4 Soubra D. The 3Vs that define Big Data. Data Science Central. 2012;.

5 Feinleib D. Big data bootcamp: What managers need to know to profit from the big data revolution. Apress; 2014.

6 Mell P, Grance T. The NIST definition of cloud computing. 2011;.

7 Srinivasan S. Cloud computing basics. Springer; 2014.

8 Li A, Yang X, Kandula S, Zhang M. Comparing public-cloud providers. IEEE Internet Computing. 2011;15(2):50.

9 Doelitzscher F, Sulistio A, Reich C, Kuijs H, Wolf D. Private cloud for collaboration and e-Learning services: from IaaS to SaaS. Computing. 2011;91(1):23–42.

10 Sotomayor B, Montero RS, Llorente IM, Foster I. Virtual infrastructure management in private and hybrid clouds. IEEE Internet computing. 2009;13(5).

11 Marinos A, Briscoe G. Community cloud computing. In: IEEE International Conference on Cloud Computing. Springer; 2009. p. 472–484.

12 Gonzalez JU, Krishnan S. Building Your Next Big Thing with Google Cloud Platform: A Guide for Developers and Enterprise Architects. Apress; 2015.

13 Tigani J, Naidu S. Google BigQuery Analytics. John Wiley & Sons; 2014.

14 Google Developer. Google Cloud Platform for Big Data. Google Inc.; 2016. [Online; accessed 21-January-2017]. https://cloud.google.com/products/big-data/.

15 Sanderson D. Programming google app engine: build and run scalable web apps on google’s infrastructure. ” O’Reilly Media, Inc.”; 2009.

16 Google Developer. App Engine Flexible Environment for Users of App Engine Standard Environment. Google Inc.; 2017. [Online; accessed 1-March-2017]. https://cloud.google.com/appengine/docs/flexible/python/flexible-for-standard-users.

17 Google Developer. Choosing an App Engine Environment. Google Inc.; 2017. [Online; accessed 1-March-2017]. https://cloud.google.com/appengine/docs/the-appengine-environments.

18 Google Developer. An Overview of App Engine. Google Inc.; 2017. [Online; accessed 2-March-2017]. https: //cloud.google.com/appengine/docs/standard/java/an-overview-of-app-engine.

19 Friesen J. Beginning Java 7. Apress; 2012.

20 Apache Maven. Maven official documentation. The Apache Software Foundation; 2016. [Online; accessed 4-January-2017]. https://maven.apache.org.

21 Google Developer. App Engine Service Level Agreement (SLA) - App Engine Documentation — Google Cloud Platform. Google Inc.; 2016. [Online; accessed 4-January-2017]. https://cloud.google.com/appengine/sla.

22 Chockler G, Melamed R, Tock Y, Vitenberg R. Spidercast: a scalable interest-aware overlay for topic-based pub/sub communication. In: Proceedings of the 2007 inaugural international conference on Distributed event-based systems. ACM; 2007. p. 14–25.

23 Hohpe G. Google Cloud Pub/Sub. Google Inc.; 2015. [Online; accessed 18-February-2017]. http: //www.enterpriseintegrationpatterns.com/ramblings/82_googlepubsub.html.

24 Google Developer. What is Google Cloud Pub/Sub? Google Inc.; 2017. [Online; accessed 7-March-2017]. https://cloud.google.com/pubsub/docs/overview.

25 Google Developer. Google Cloud Pub/Sub: A Google-Scale Messaging Service. Google Inc.; 2017. [Online; accessed 7-March-2017]. https://cloud.google.com/pubsub/architecture.

26 Akidau T, Bradshaw R, Chambers C, Chernyak S, Fernández-Moctezuma RJ, Lax R, et al. The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. Proceedings of the VLDB Endowment. 2015;8(12):1792–1803.

27 Google Developer. Dataflow Programming Model. Google Inc.; 2017. [Online; accessed 9-March-2017]. https://cloud.google.com/dataflow/model/programming-model.

28 Google Developer. Pipelines. Google Inc.; 2017. [Online; accessed 9-March-2017]. https://cloud.google.com/dataflow/model/pipelines.

29 Google Developer. Pipeline I/O. Google Inc.; 2017. [Online; accessed 9-March-2017]. https://cloud.google.com/dataflow/model/pipeline-io.

30 Google Developer. PCollection. Google Inc.; 2017. [Online; accessed 10-March-2017]. https://cloud.google.com/dataflow/model/pcollection.

31 Akidau T. The world beyond batch: Streaming 101. O’Reilly; 2015. [Online; accessed 10-March-2017]. https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101.

32 Akidau T. The world beyond batch: Streaming 102. O’Reilly; 2016. [Online; accessed 10-March-2017]. https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-102.

33 Google Developer. Windowing. Google Inc.; 2017. [Online; accessed 10-March-2017]. https://cloud.google.com/dataflow/model/windowing.

34 Google Developer. Transforms. Google Inc.; 2017. [Online; accessed 10-March-2017]. https://cloud.google.com/dataflow/model/transforms.

35 Cheon Y, Leavens GT. A simple and practical approach to unit testing: The JML and JUnit way. In: European Conference on Object-Oriented Programming. Springer; 2002. p. 231–255.

36 Do H, Rothermel G, Kinneer A. Prioritizing JUnit test cases: An empirical assessment and cost-benefits analysis. Empirical Software Engineering. 2006;11(1):33–70.

37 Robot Framework foundation. Robot Framework home page; 2017. [Online; accessed 26-March-2017]. http://robotframework.org/.

38 Jenkins foundation. Jenkins Documentation; 2017. [Online; accessed 26-March-2017]. https://jenkins.io/doc/.

39 Google Developer. Google Cloud Pub/Sub Emulator. Google Inc.; 2017. [Online; accessed 11-March-2017]. https://cloud.google.com/pubsub/docs/emulator.

40 Martikainen T. ZEF | Home page. Zef Oy; 2016. [Online; accessed 9-March-2017]. http://zef.fi/en/frontpage/.

41 Google Developer. How instances are managed. Google Inc.; 2017. [Online; accessed 28-February-2017]. https://cloud.google.com/appengine/docs/standard/java/how-instances-are-managed.

42 Google Developer. Publisher Guide. Google Inc.; 2017. [Online; accessed 9-March-2017]. https://cloud.google.com/pubsub/docs/publisher.

43 Google Developer. Subscriber Guide. Google Inc.; 2017. [Online; accessed 9-March-2017]. https://cloud.google.com/pubsub/docs/subscriber.

44 Google Developer. Pipeline Design Principles. Google Inc.; 2017. [Online; accessed 10-March-2017]. https://cloud.google.com/dataflow/pipelines/design-principles.

45 Google Developer. Constructing Your Pipeline. Google Inc.; 2017. [Online; accessed 10-March-2017]. https://cloud.google.com/dataflow/pipelines/constructing-your-pipeline.

46 Google Developer. Streaming Data into BigQuery. Google Inc.; 2017. [Online; accessed 11-March-2017]. https://cloud.google.com/bigquery/streaming-data-into-bigquery.

47 Google Developer. Querying Data. Google Inc.; 2017. [Online; accessed 11-March-2017]. https://cloud.google.com/bigquery/querying-data.

48 Google Developer. Testing Your Pipeline. Google Inc.; 2017. [Online; accessed 9-March-2017]. https://cloud.google.com/dataflow/pipelines/testing-your-pipeline.