Google Cloud and solution for industrial automation systems

University of Turku
Department of Information Technology
Software Engineering
Master's thesis
January 2020
Johan Nordman

University of Turku
Department of Information Technology

Johan Nordman: Google Cloud and solution for industrial automation systems

Master's thesis, 64 s.
Software Engineering
February 8, 2020

This master's thesis introduces a possible architecture with the Google cloud platform for storing industrial time-series data from automation programmable logic controller (PLC), and how different messages from PLC are categorized. Master's thesis introduces three cases of how stored data can be used to gain information from the system. The solution presents a way to divide software responsibilities between local and cloud entity. In the architecture, few cases as to how data is handled locally and writing messages to the cloud are explained with more detail and code examples.

The solution was developed concurrently with a new underlying system and because there was no easy to use and flexible solution that offered needed requirements. The solution includes interfaces and generalization that it is not dependent on the underlying system, and architecture is usable with different systems. The architecture was decided to be developed in a cloud platform that would enable global distribution with additional hosted tools.

Contents

1 Introduction

Cloud platforms have revolutionized how the IT industry works, and tech giants like Amazon, Google, and Microsoft have invested in providing their cloud solutions to market. Cloud platforms can be seen as services that offer standard interfaces to develop solutions more rapidly and securely. Most of the cloud platforms provide storage, networking and computing with well-defined documents and APIs. Cloud services can be seen as a considerable benefit for software development, but it is not an optimal solution for everything. There are additional costs regarding the cloud services, but it has proven to be a new standard even for a more noticeable company like Spotify[1]. With managed services, you can concentrate on developing critical components of your application and alter the focus from the technical details.

The goal of this master's thesis is to generalize the idea of how data from automation systems can be stored to a cloud platform, and provide services like visualizations and machine learning for stored data. Especially research is done to find the right software solutions to support the operation of industrial plants, and thinking new business opportunities. The end product of the master's thesis is software that can handle incoming data and how to add functionalities, e.g. visualize data to the users.

One has to keep in mind that the master's thesis tries to address a massive amount of domain knowledge. Regarding the underlying system, some topics cannot be discussed in detail and are only a brief introduction of the solution. Google Cloud Platform (GCP) consist of multiple sources and thesis will introduce computing, networking, storage and machine learning tools. There are three main points to concentrate on: data from automation systems, storing data, and services for data. Storing data is an essential part of the operation and gives the possibility to create services based on it. Services from data use cases: support systems users to gather daily information, collect historical data from the underlying system, and support maintaining data from multiple sources.

Chapter 2 introduces Google Cloud Platform solutions for storing, networking, and compute. Machine learning solutions are presented in Chapter 3 and how to implement them on the Google Cloud Platform. Chapter 4 introduces components of the underlying system and how data into divided to different classes. Chapter 5 discusses three cases for analytical and machine learning tools, e.g. anomaly detection. Chapter 6 introduces one possible architecture of a system and offers a few code snippets of the implementation

1.1 Honkajoki Oy and the underlying system

The master's thesis will be done in partnership with Honkajoki Oy. Honkajoki Oy is a Finnish meat rendering plant, and operates according to the Finnish and European Union laws to handle animal by-products. The goal for rendering plant is to process animal by-products, refine raw material, and eliminate the risk of possible animal diseases [2]. The incoming raw material is handled with heat processing accordingly to specific laws. In the rendering industry, there are few outflows from the process: fat, meat and bone meal, and evaporated liquid. There are and have been a lot of research to refine outflows to more valuable products. Fat can be refined into biodiesel and other energy products. Meat and bone meal can be refined to fertilizers, energy products, and animal feed (with some limitations). I have worked with Honkajoki Oy for several years and concentrated on refining evaporated liquids from the rendering process.

The underlying system, which software is developed for, is a modified wastewater treatment plant (WWTP). WWTP consists of different components and is designed accordingly to the influent parameters. Without concentrating too much on WWTP technology, it is possible to define that WWTP consists of pumps, measurement devices, chemical dosing, blowers, automation systems, basins, and many other instruments. Instruments are connected to automation systems and can be controlled with automation rules and by plant operators. From automation systems, you can also log information and record state of the instruments, and this master's thesis will concentrate on that.

A rough example of the process is presented in Figures 1.1 to 1.3. In Figure 1.1, there is a P&ID diagram of an underlying system with multiple instruments. The automation system can write instrument-specific data to a database, and an example of writable data is in Figure 1.2, where writable parameters are position, time, and measurement of the flow meter. From written data, you can visualize flow measurements between a given period, as presented in Figure 1.3.

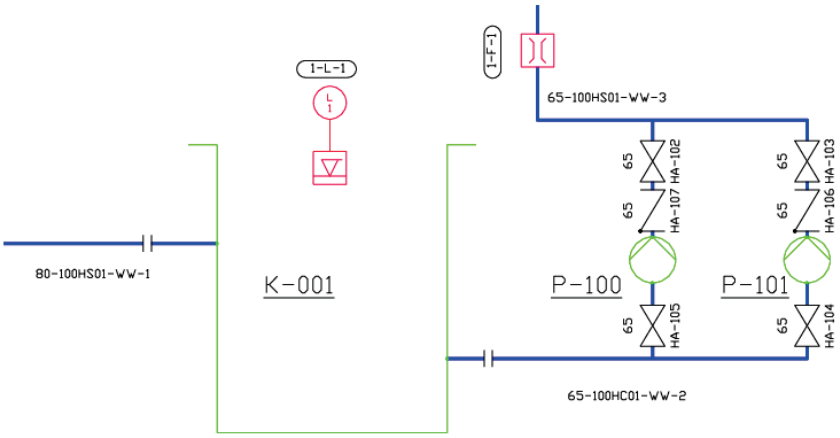

Figure 1.1: P&ID of the underlying system

Position	time	measurement
1-F-1	2018-01-01 12:00:00	0.0
1-F-1	2018-01-01 12:00:05	0.1
1-F-1	2018-01-01 12:00:10	0.4
1-F-1	2018-01-01 12:00:15	1.1
1-F-1	2018-01-01 12:00:20	1.2
1-F-1	2018-01-01 12:00:25	1.1
1-F-1	2018-01-01 12:00:30	1.2
1-F-1	2018-01-01 12:00:35	0.6
1-F-1	2018-01-01 12:00:40	0.1
1-F-1	2018-01-01 12:00:45	0.0

Figure 1.2: PLC data example

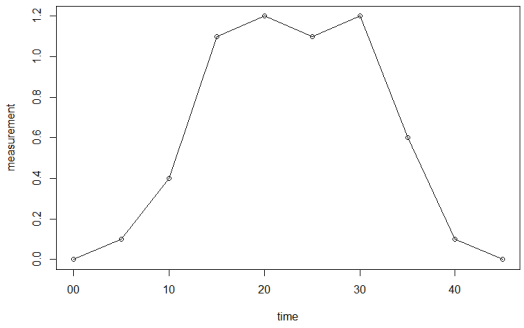

Figure 1.3: example of PLC data visualization

This master's thesis concentrates on giving the right technical solution to connection automation systems to the cloud platform and serving that data. There has been a lot of effort towards implementing more machine learning systems in multiple domains, and data from automation systems have good use cases for it. I will use machine learning algorithms for time series data, to find anomalies from instrument data.

1.2 Why cloud platform

Logging and storing data from an automation system can be done to local and cloud storage. Cloud solutions are more attractive when there are multiple systems to control, and you have to maintain these local systems. Especially if you are the provider of the industrial plant, you want to gather data for maintenance and support. Maintenance or support software, local or cloud solution, are also service sector product that creates additional business value.

Choosing a cloud platform for each solution can have multiple reasons. Abstractions of services give users more comfortable to use interface, and underlying system-specific have not to be known, e.g. cloud load balancer can internally use HAProxy, but users have a more accessible interface to manage it. Provisioning of the machines, with cloud platform there is no need for local hardware and provisioning it. Cloud platform comes with extending services, like machine learning APIs and fully managed services. In total cloud platform gives more flexibility for the development process, and each project needs less domain knowledge of the specific system (e.g. how to run and maintain a globally distributed database).

For the underlying system, the most significant benefit is to store data in an external resource where you can build services around it, and trust that data will be redundant and secure. One high-level example is presented in Figure 1.4, where cloud platform services are divided into two sections: one handling database-related tasks, and another serving these resources to users. Data is written from the automation system to the cloud, and around that services can be created. When data is stored in the cloud, there are almost limitless possibilities to extend services, e.g. detailed visualizations for different users, alert messages for alarms etc. Data from PLC is only one data source, and usually, there are other sources from the same underlying system, e.g. in WWTP there are multiple daily analyses made by operators. Combining data sources give a possibility to improve data quality and reduce information loss.

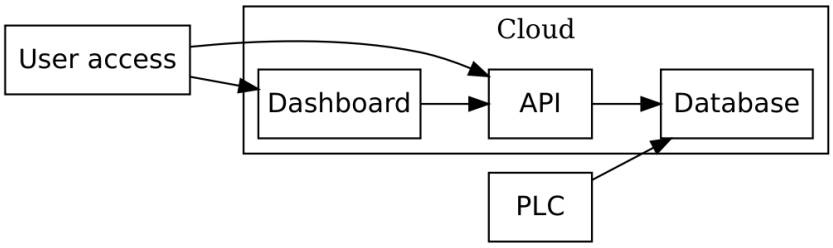

Figure 1.4: Example of GCP structure

Example of a non-optimal case for only cloud services for an automation system is where you have a weak internet connection, or any data cannot be lost. In these cases, you have to have some local buffer that can handle internet connection problems between the cloud and site. There is also a possibility that local systems are not connected to the public internet, and in these cases, the presented solution is not viable.

2 Google Cloud Platform

Google Cloud Platform (GCP), Amazon Web Services (AWS), and Microsoft Azure are few of the biggest cloud platform providers, but underlying customer needs and open-source tools make providers comparable. Each platform is differentiable, e.g. additional closed source solutions and domain specifics like Microsoft products on Azure. The solution that is described in this master's thesis is possible to implement in each of the mentioned cloud platforms, and choosing GCP is only a personal preference.

2.1 Overview

Google Cloud Platform resources are offered by region and zone basis. Regions are location specific entities and are available in North America, South America, Europe, Asia and Australia. Zones are isolated entities inside a region [3], e.g. the region us-west1 has a, b, and c zones. Used resources can have zone and region restrictions or setup, cost differences depending on regions, and not all resources are globally available.

Project in GCP is a top-level item where other resources belong. When creating new (e.g. storage) or accessing available instances, it is done on a project basis. The project itself holds more additional details, like billing and authentication.

GCP offers a variety of products toward different problems, and products are used internally by Google, so there is almost no possibility to outgrow your needs. GCP provides multiple technical innovations as a service like BigQuery for Big Data, and provides part of these innovations as open-source projects. Most products in GCP have service level agreement (SLA) and financial incentives if they are not met. Typical SLA for GCP products is 99,95 % which equals less than 5 hours of downtime in a year. Products have transparent pricing, and most of them have sustainable use discounts.

2.2 Example of GCP components

In Figure 2.1 is presented a possible architecture for a website hosting. Where website includes static and dynamic content, e.g. dynamic content as user-specific text and static content as pictures for all users. In this example architecture, dynamic content is available through a computing resource, one possibility to create this is with Compute Engine resource (Section 2.5.1) and routing domain to that resources with Cloud DNS (Section 2.3.3). Static content could be loaded straight from Storage Bucket (Section 2.4.6) where publicly available objects are hosted through Cloud CDN (Section 2.3.2), and avail-

able through google storage APIs. Linking a compute resource to a domain is done with Cloud CDN level with all compute resources. Storage Bucket can have multiple different purposes for different use cases as presented for static content storing. Storage Bucket offers more complicated options because they have massive interconnection between GCP ecosystem, e.g. notification when an object is removed from Cloud Bucket.

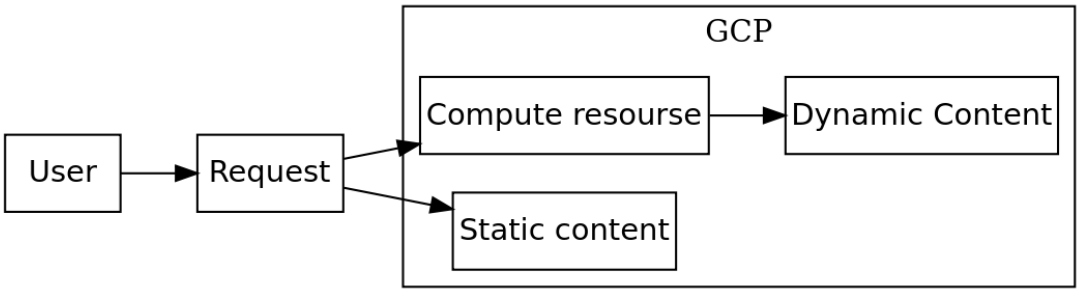

Figure 2.1: Example of GCP architecture for a website

2.3 Networking

GCP offers a stack of network services that are highly reliable and scalable. Networks services enable connecting existing enterprise networks to GCP, manage other GCP resources internally, and make them publicly available.

2.3.1 Virtual Private Cloud

Virtual Private Cloud (VPC) offers complete control of the Google Cloud Platform, e.g. tools for enabling connection to projects and resources, and vice versa to separate them and block access. VPC tools allow control connections between on-premises, GCP and other cloud platforms. Most common tools in VPC are subnetworks and firewalls. Services in VPC are scalable and managed, e.g. configuration changes do not create downtime.

VPC solutions are vital when a software solution increases in size, and there is a need for more detailed control, e.g. one subnetwork for production and another for development. VPC is a tool for enabling cross-project connection in GCP because usually, projects are isolated entities.

2.3.2 Cloud CDN and Cloud Load Balancing

Content Delivery Network (CDN) is a globally distributed system that holds data close to the user, and caches data out of normal regional bounds. Cloud CDN is offered for Compute Engine instance (Section 2.5.1) groups and Storage Buckets (Section 2.4.6).

Cloud Load Balancing (LB) is auto-scaling managed service for Compute Engine instance groups [4]. LB supports HTTP(S), TCP, SSL, and UDP traffic. LB enables internal and regional LB, and especially global LB when developing globally available applications.

An example from [5] is redrawn in Figure 2.2 and presents usage of cloud LB and DNS. In Figure 2.2 underlying LB serves a website with static and dynamic content. CDN can be used for serving Storage Bucket (static content) and Compute Engine (dynamic content) resources. LB enables serving cached content from both resources, and without it, there should be another resource for reverse proxy. Using cloud LB and CDN together, like in Figure 2.2, simplifies application structure of globally available and caching enabled solution.

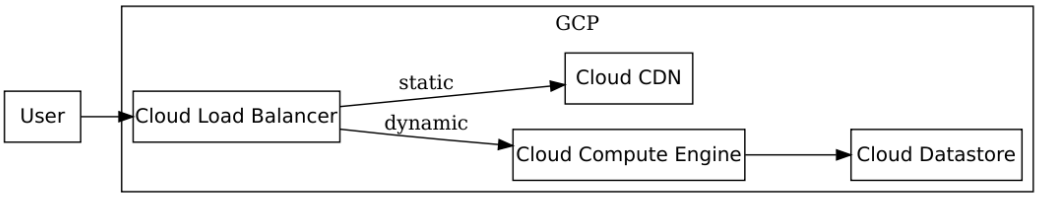

Figure 2.2: Example of CDN and Load balancer usage

2.3.3 Cloud Interconnect and Cloud DNS

Cloud Interconnect enables connecting enterprise infrastructure to Google Cloud Platform and is offered in four different ways: Dedicated Interconnect, Cloud VPN, Carrier Peering, and Direct Peering. Dedicated Interconnect, a physical connection to GCP and the connection between client and cloud will not face public internet. Cloud VPN is a secure connection with the IPsec gateway. Direct Peering is a connection between physical routers that use BGP routing. Carrier Peering offers Direct Peering like features but from assigned carriers, and gives the possibility to get benefits of Dedicated Interconnect with fewer requirements. Cloud Interconnect is intended for enterprise networks to enable low latency and high capacity needs. Cloud VPN and Carrier Peering have fewer requirements and are available for most cases.

Google Cloud domain name system (DNS) provides a fully managed global DNS. Use

cases are mainly intended for external connections, and services like Compute Engine have internal DNS services for communication. Features like 100 % SLA tells that service is highly durable, maintained and tested. Cloud DNS is excellent service to other GCP resources and serving them to the public internet, and also offers APIs e.g. possible to program a Compute Engine resource to setup DNS records at startup.

2.4 Storage and Databases

Google Cloud Platform offers multiple different storage and database solutions towards numerous different use cases. Available storage products can be divided between structured and unstructured data, and from small SQL databases to globally available ACID transaction enabled products.

2.4.1 Cloud SQL

Cloud SQL is a fully managed MySQL or PostgreSQL database, and the goal of it is to reduce the complexity of database management [6]. There is a stack of security products around Cloud SQL, e.g. all data is encrypted automatically. Automated features in Cloud SQL are replicas, backups, scaling, and updates. There are multiple connection methods with minimal configuration: IP connection with/without SSL, and proxy methods. Cloud SQL comes with some limitation, e.g. no support for user-defined functions [7]. Use cases are where you usually would use MySQL or PostgreSQL databases, and where there is a need for additional management and provisioning of the system.

2.4.2 Cloud Bigtable

Cloud Bigtable is a fully managed NoSQL database developed by Google. Cloud Bigtable was introduced in an article [8], and open-source database's like Apache HBase is based on that article [9].

Data in Bigtable is lexicographically sorted key/value map [8]. Each row in the Bigtable table has a unique row key and belongs to a column family, and data is sparsely populated, so there is no penalty for empty items. Bigtable scales to enormous size with some limitation "billions of rows and thousands of columns, allowing you to store terabytes or even petabytes of data" [10]. A game-related example of data in Bigtable is presented in Table 2.1, where row keys are the name of the player and the column family presents games between users. Each data point holds values between games. The same data can be stored as is represented in Table 2.2, where row keys are the combination of players that

have played a game and in the Column family is a score of the data. Between these two examples, there are differences with queries you can execute and how data is stored in Bigtable service. Using and designing working database schema in Cloud Bigtable needs additional work.

	Games (Column Family)		
Row Key	Mark	Hilary	John
Mark		10	15
Hilary	3		2
John	3	1	

Table 2.1: Example Schema in Bigtable

	Games (Column Family)
Row Key	Score
Mark#Hilary	10
Mark#John	15
Hilary#Mark	3
Hilary#John	2
Josh#Mark	3
John#Hilary	1

Table 2.2: Example Schema in Bigtable

Cloud Bigtable is a zonal resource, and there is regional egress cost from traffic. Non-optimal use-cases are for large objects of data, e.g. video if the total amount of data is less than 1 TB, and if there is a need for online analytical processing (OLAP). [10]

2.4.3 Cloud Datastore

Cloud Datastore is a fully managed and schema-less NoSQL database. Cloud Datastore has ACID transactions, SQL-like querying, and scales to massive datasets. One feature to enable scaling is that queries scale with the result set by limiting querying, e.g. join operations is not supported [11]. Scaling whit results set means that query execution is only dependent on the size of the result set, and not on the size of the dataset.

Data in Cloud Datastore is called entity and holds properties and keys. Properties are data values for entities and support multiple data values (e.g. Integers, Strings). Keys also have unique identifiers, can represent what data entity holds, and additional data to optimize querying. Entities can be stored in a hierarchical system that each entity has ancestors.

Example of Datastore is presented in Table 2.3, where entities store player information, and entity kind is “Player”. Entities can have multiple properties and entities can differ between properties, and this is visible between the two entities in Table 2.3.

Kind	Player
id:	1
score:	22
level:	5
id:	2
score:	21
level:	3
location:	eu

Table 2.3: Example of Datastore properties

Best uses cases for Cloud Datastore are where you need colossal scalability and ACID transactions. Non-optimal use cases are OLAP and large immutable objects, for these purposes, there are better solutions in GCP. [11]

2.4.4 Cloud Spanner

Cloud Spanner is a fully managed relational database and with high accessibility and availability. Google defines it as "globally-distributed data management system that backs hundreds of mission-critical services" [12]. Reading from Spanner can be done with ANSI SQL standard, and writing uses Spanner specific standard. Cloud Spanner has multiple features that make it differ from other storage options: a relational database that scales horizontally, automatic sharding, automatic performance optimization, ACID transactions, and made as fault-tolerant as possible (SLA is 99,999%) [12].

There are various details in an article [13] where Spanner was introduced, e.g. what enables automatic sharding, but the more presentable aspect is schema and data model of Spanner. Spanner tables consist of columns, rows, values, and primary keys, and these are familiar from common SQL-databases. Tables in Table 2.4 are an example of two tables that can be stored as well in SQL database. With Listing 1, tables `Users` and `Games` are siblings and stored in Spanner like presented in Table 2.5.

Users		
UserId	Name	Location
1	John	Europe
2	Mark	USA
Games		
UserId	GameId	Points
1	1	40
1	2	10
2	3	50

Table 2.4: Tables in Spanner

Games(1,1)			1	40
Games(1,2)			2	10
Games(2,3)			3	50
Users(1)	John	Europe		
Users(2)	Mark	USA		

Table 2.5: Multiple tables in schema

Listing 1: Greating tables syntax in Spanner

```
1  CREATE TABLE Users (
2        UserId INT64 NOT NULL,
3        Name STRING(100) NOT NULL,
4        Location STRING(100),
5     ) PRIMARY KEY(UserId)
6
7     CREATE TABLE Games (
8        UserId INT64 NOT NULL,
9        GameId INT64 NOT NULL,
10       Points INT64,
11    ) PRIMARY KEY(UserId, GameId);
```

Cloud Spanner is a more expensive product when comparing to other database solutions in GCP, and meant towards a globally distributed system that needs ACID transactions. Service is highly available, and tuning is available manually and automatically. Best use cases are mission-critical services, e.g. Google uses Spanner in Google Play and AdWords [12].

2.4.5 Persistent Disk

Persistent Disk (PD) is a storage option for unstructured data and can be seen as a fully managed network drive. PD offers multiple solutions depending on the underlying system and specific needs. There is a possibility for both hard disk drive (HDD) and solid-state drive (SSD) options. The SSD option is more expensive but higher-performing when compared to HDD, and suitable for solutions that need performance. PD is offered with

additional options for usage: Multi-Regional, Regional, Nearline, and Coldline. Multi-Regional and Regional are options for standard data usage, e.g. storage for databases. Nearline and Coldline are storage solutions for archival purposes. PDs are fully managed, and Compute Engine manages disks redundancy and performance [14]. Features like stripping disks are automatic, and resizing of disks is possible on the fly. Data in PDs are automatically encrypted with generated or provided customer keys [14].

Persistent Disks are persistent as the name implies and can be used to store persistent data, e.g. to separate data from containers, or disk space for a virtual machine. Limitations for PDs is that they are a zone-specific resource and can be accessed directly only from that zone. Few important points are that disk performance grows with the size of the disk, and you are billed by reserved space (not by used space).

2.4.6 Storage Bucket

Storage Bucket (SB) is fully managed and flexible object storage in the Google Cloud Platform. Objects consist of two parts, file itself and metadata associated with it. They are offered with the same options as Persistent Disks: Multi-Regional, Regional, Nearline, and Coldline. SBs has the same automatic data redundancy and encryption as Persistent Disk has.

SBs are not restricted to zones like Persistent Disks, and using buckets from multiple zones simultaneously is possible [14]. SBs have no folder structure, but you can mimic folder structure with backlashes in file names. SB objects are immutable, and updating object is possible only by removing and then adding a new altered object. SBs name has to be unique, which enables CNAME redirection to object in a bucket.

When comparing to Persistent Disk, Storage Bucket has higher latency and lower throughput. Storage Buckets are not an optimal solution when you need high performance, but more straightforward solution to host data in multiple regions and data accessibility is easier when comparing to Persistent Disks. SB can be used to host static elements of the website (e.g. pictures, JavaScript), and with Cloud Load Balancer, one can use Cloud CDN functionality in SBs. SB differs from Persistent Disk that pricing is based on the used space.

2.5 Compute

Google Cloud Platform offers multiple compute choices from small hosted functions to managed container orchestration. And there is an interconnection between offered com-

pute resources, that other GCP resources are using the same computing implementations inside of them. Each compute resource still offers differentiable role based on your needs.

2.5.1 Compute Engine

Compute Engine offers virtual machines (VM) instances with a layer of management tools. Available options for VM are operating system, needed CPU and RAM resources, and persistent disk, e.g. you can create Ubuntu server with six cores, four gigabytes of memory and a hundred gigabytes of a persistent disk Additional tools are shared ssh-keys, health information gathering, startup and shutdown scripts, and creating snapshots [15]. These tools are readily available only a few clicks away to implement.

Compute Engine offers external metadata servers, which holds VM, group or project-specific metadata that can be used to store key-value pairs [15]. Example usage of metadata server is project-wide ssh-keys, where ssh-keys will be downloaded from the metadata server and installed at the startup of the VM instance. A goal for metadata is to gather instance-specific information together, and that way gain easier maintenance of VM instances.

You can Group VM instances as managed and unmanaged. In managed instance, control is toward one instance, and control is replicated towards all VM in that group. Managed instances offer auto-scaling, load balancing, auto-healing and rolling updates. Compute Engine, with managed instance, offer easy to set up scaling VMs which can be combined to act as containers.

2.5.2 App Engine

App Engine (GAE) is a fully managed serverless platform, which enables an easier way to deploy, manage and scale applications. In the most simple case, you provide source code and configuration files. And based on those two items, the application can be run as managed and scaled automatically by GCP.

App Engine is divided into flexible and standard environments. In the standard environment, the code is run in a sandbox with restrictions. Few of them are limited usage of standard libraries, storing and loading of data only with Datastore, and support for a limited amount of programming languages. Restrictions in standard environments offer more affordable pricing and faster scaling [16]. The Flexible environment uses Docker images and is offered with flexible and custom runtime. The Flexible runtime uses supplied images without customer providing a Dockerfile. The custom runtime uses customer provided Docker image. Flexible runtime has support for a limited amount of programming

languages, and with custom runtime, one can run anything that runs on Docker. When using custom runtime and provided Dockerfile, you have to implement functionality that is in flexible runtime's Dockerfile. Both discussed runtimes use Container Builder service in GCP to build Docker images.

App Engine application consists of structure visible in Figure 2.3. Services are a top-level element which consists of source code and configuration files [17]. An application can have one or multiple services, and services can be structured that they function together as microservice (e.g. one for website requests, and another to handle image uploads of the website). Service versions keep track of deployment versions, they enable roll back to older releases of the deployment and routing requests to multiple different versions. Instances are runnable Docker images or sandboxed items, depending on the used environment. Service scaling is done by monitoring instance parameters, and when needed, additional instances can be created to balance the load.

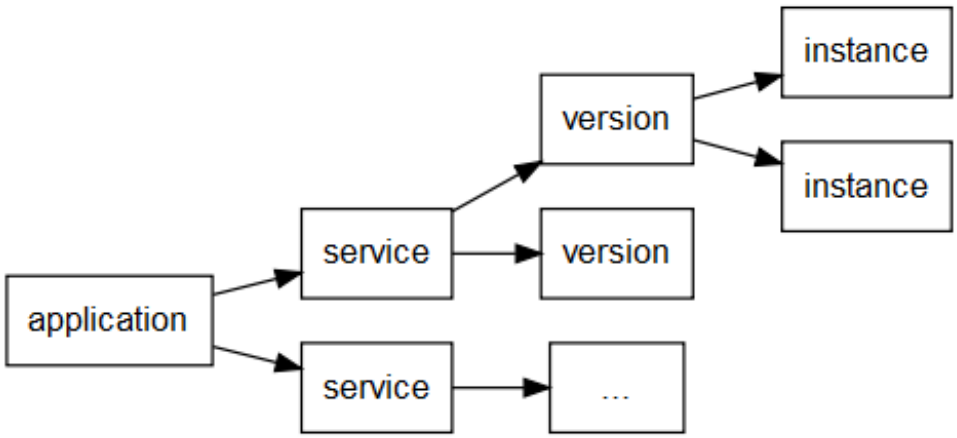

Figure 2.3: Structure of App Engine instance

Service-specific configuration files are YAML (YAML Ain't Markup Language) files. The most important file is `app.yaml` that holds data of the chosen environment, needed resources (e.g. CPU, RAM, Disk), and network rules etc. Example of `app.yaml` for the flexible environment is in Listing 2. Where used flexible environment and runtime, and automatic scaling is done accordingly of CPU load.

Listing 2: Example of app.yaml

```
runtime: go
env: flex
```

```
automatic_scaling:
  min_num_instances: 2
  max_num_instances: 10
  cool_down_period_sec: 120 # default value
  cpu_utilization:
    target_utilization: 0.5
resources:
  cpu: 2
  memory_gb: 1
  disk_size_gb: 20
```

GAE offers manual and automatic scaling of services. With manual scaling, you set a number of instances needed and this is not altered under load. With automatic scaling, one can set range of the minimum, and maximum amount of instances and load balancing is automatic under load, an example of this in Listing 2 under `automatic_scaling`.

All GAE services come with domain name `project_id.appspot.com`, and this is true for each service and instance as well `instance.version_id.project_id.appspot.com` [17]. Domains are used for user accessibility and internal routing. GAE offers custom domain routing and SSL certification support for the application.

App Engine offers a vast amount of features for implementing a serverless design that can scale to a considerable scale. GAE is a good option when you do not need all additional resources and control that Compute Engine or Kubernetes Engine offers. GAE is a regional resource that scales to zones in that region, and when writing the master's thesis, there is no tool for cross-region load balancing. The flexible environment is comparable to Compute Engine with only a few limitations [16].

2.5.3 Kubernetes Engine

Kubernetes Engine is a managed Kubernetes service on GCP. Kubernetes is an open-source container orchestrator developed internally by Google that was open-sourced. Kubernetes offers a stack of tools and services for running microservice architecture [18]. Kubernetes itself provides orchestration of compute units accordingly resources, and managed service add functionality to make it easier to use, e.g. enabling auto-updating of Kubernetes itself.

Kubernetes can be simplified to consist of a cluster, cluster master, pods and nodes, as represented in Figure 2.4. Cluster holds all entities that are inside the specific cluster. Cluster master is an entity that controls nodes inside a cluster, and all configuration to cluster is done through cluster master, e.g. updating nodes, and scheduling [18]. Node is a compute entity, VM or hardware, that holds container engine, e.g. Docker, and runs

software that connects to cluster master. More clearly in Kubernetes Engine nodes are Compute Engine (Section 2.5.1) instances [18]. The Pod is an entity that runs on node, e.g. Docker image. Pods that run on the same node can share resources, e.g. storage. By monitoring resources of a node, auto-scaling is possible, and Kubernetes can create and delete nodes and transfer pods to different nodes.

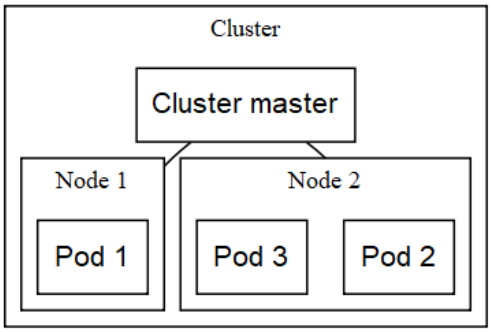

Figure 2.4: Components of Kubernetes

Pricing is based on needed Compute Engine units (nodes), and there is no additional cost from managed service. Depending on your needs Kubernetes Engine can be configured to run in one zone, multi-zone and region level [18]. Kubernetes is more complex compute choice when comparing to others, but a good choice when serving large and complex applications.

2.5.4 Google Cloud Functions

Google Cloud Functions is a serverless compute product where given function is a run as a service. Cloud Functions support Golang, Python and node runtimes at this time. Components of Cloud Functions are function, event and trigger. Where function is executable code. Events are actions that are monitored, and possible events are HTTP, Pub/Sub and Cloud Storage related [19]. Triggers are more detailed information about the event. Example of usage is to track events in Storage Bucket (Section 2.4.6), where loading successfully an object creates `OBJECT_FINALIZE` event, and Cloud Function waits for Storage Bucket event, and the trigger is `OBJECT_FINALIZE`.

Example of Cloud Function is given in Appendix C where the function is written in Golang. Appendix C also gives an example of how a Cloud Function uses the init method to initialize parameters before execution.

Possible use cases are ETL (extract, transform, load) process, listening events inside GCP and listening events from outside with HTTP-requests [19]. Google Cloud Functions is

not the best compute resource when functionality is complex, and there is a need for a lot of computing power. Cloud Functions pricing is based on usage parameters invocations, GB-seconds, GHz-seconds and networking, based on chosen hardware [19].

3 Big Data and Machine learning on Google Cloud Platform

Google has had a significant impact on how big data and machine learning is processed, stored and modeled. This impact can be seen with multiple open-source projects that were first developed internally by Google, and then became the dominant method. One example is TensorFlow machine learning framework that has enormous popularity and developer base. Google Cloud Platform offers multiple tools for big data and machine learning, and all of the tools scale to handle large sizes of data. Tools are also extensible with user needed specifics, and most of them use internal GCP resources mentioned in Section 2.

There are also prebuilt machine learning models with APIs, e.g. Cloud Natural Language for natural language processing (NLP). These models have a good use-case for multiple domains and developing own models is not needed, but these prebuilt models are not discussed in this section.

3.1 BigQuery

BigQuery is a managed data warehouse in GCP for enormous scale. BigQuery is an SQL-like query system based on Dremel [20], but as a whole consist of multiple solutions developed by Google [21]. Dremel parallelises execution of queries which scales to an enormous scale of computing power and data.

BigQuery has an internal columnar database and integrates with other GCP tools, and external ETL and BI tools. To insert data to BigQuery internal database, one can insert data with stream or batch basis. BigQuery biggest use case is for analytics, e.g. it does not compete with Cloud Bigtable but extends its usability in analytics. You can interact with GCP tools like Google Bigtable and Storage directly within BigQuery [22]. BigQuery queries are written in SQL-like syntax and offer JDBC and ODBC drivers, and REST API to access it from multiple sources. This enables that you can use BigQuery with multiple existing tools, e.g. data analysis with BigQuery within Python. An example of BigQuery is given in Listing 3, where BigQuery is used in R. In Listing 3, the size of 21.9 GB public dataset is queried in few seconds, and the result is cached for 24 hours. The cached result is offered if the same query is executed again [22] BigQuery also provides user-defined functions (UDF) written in JavaScript. UDF extends BigQuery usability further to more fine-grained analytics access.

In Listing 3 BigQuery public natality dataset is used where newborn babies in the United States is listed with few parameters. Parameters are: baby weight, baby's gender, is a

mother a smoker, and what year baby was born. Top five results by descending order by weight are presented at the bottom of the Listing.

Listing 3: BiqQuery example in R

```
1  library(bigrquery)
2
3  project <- "project-id"
4  sql <- "SELECT year, is_male, cigarette_use, weight_pounds
5          FROM [publicdata:samples.natality]
6          ORDER BY weight_pounds DESC
7  query_exec(sql, project = project)
8
9  # Result
10 year is_male cigarette_use weight_pounds
11 1 2006   FALSE          NA       18.00074
12 2 2007   FALSE          NA       18.00074
13 3 1970    TRUE          NA       18.00074
14 4 2007    TRUE          NA       18.00074
15 5 2006    TRUE       FALSE       18.00074
```

BigQuery ML is a new addition to BigQuery and enables machine learning in BigQuery. Available models are linear, binary logistic and multiclass logistic regression [22]. There is a possibility to create and predict with machine learning model and use it inside of BigQuery, exporting ML models are not yet supported.

3.2 Cloud Dataflow

Cloud Dataflow is a hosted Big Data tool for stream and batch processing. Cloud Dataflow is a data pipeline runner that takes Apache Beam pipelines as input. Apache Beam is introduced as a "unified programming model" [23], where user-defined pipelines can be implemented in multiple languages, and these pipelines can be executed in multiple different runners. Cloud Dataflow is one of these runners which has deep integration to other GCP resources when comparing to other Beam runner, e.g. SparkRunner. Apache Beam core was an internal Google Project that was open-sourced.

As presented in Figure 3.1, Beam pipeline consists of PCollections, transforms and input/output of the data. Data input and output can be a stream or data object, e.g. data from Cloud Pub/Sub (see Section 3.5), text-file or database table. PCollection is an immutable data object and transforms are operations for given PCollection. One example transform ParDo executes user code parallel for every PCollection item [24], e.g. calculates the length of each string element given as input. There can be branches in a pipeline, multiple inputs and outputs, e.g. combining data sources to a PCollection.

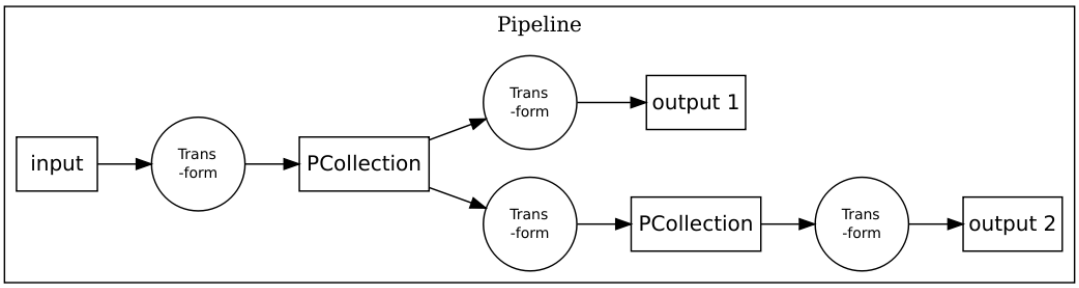

Figure 3.1: Apache Beam pipeline structure [24]

In Listing 4 example of Apache Beam with previously used public natality dataset is shown. In the Listing a local copy of CSV file is read with Beam input/output library. Input is read per row basis and Split class in the Listing split rows to needed elements. To keep code example at minimum execution of Beam pipeline is not a good representation of real use case, but execution starts at row 18. In row 19 pipeline is described, as presented in Figure 3.1, where PCollection is available on a vertical line, and element between vertical lines is a transformation, e.g. `beam.ParDo(Split())` is a transformation. The same example for natality data is given in Listing 4 and Listing 3, and when comparing these examples, global data sorting is not possible directly in Apache Beam, which is available in BigQuery.

Listing 4: Apache Beam example in Python

```
1  import apache_beam as beam
2  from apache_beam.io import ReadFromText
3
4  class Split(beam.DoFn):
5      def process(self, element):
6          all = element.split(",")
7          year = all[1]
8          is_male = all[6]
9          cigarette_use = all[19]
10         weight_pounds = all[8]
11
12
13         return [{'year': year,
14                  'is_male': is_male,
15                  'cigarette_use': cigarette_use,
16                  'weight_pounds': weight_pounds}]
17
18 with beam.Pipeline() as p:
```

```
19      p | ReadFromText("natality.csv") | beam.ParDo(Split()) | ...
```

Apache Beam pipeline is a tool for big data purposes where you need to process data on a large scale. Cloud Dataflow is a managed runner that automatically scales your Apache Beam pipeline job. Cloud Dataflow also enables serving machine learning models on a more significant scale. Cloud Dataflow is only hosted stream processing tool in GCP (when excluding Cloud Dataproc, a hosted Apache Hadoop and Spark). Apache Beam is a newer product and released in 2016, so using it will create some restrictions, but on the other hand, the built pipelines can be run with multiple runners that decrease emerged restrictions.

3.3 Data Studio

Cloud Data Studio is a hosted service where multiple different Google sources can be interactively visualized or create a dashboard. Most of the storage resources in GCP are supported, e.g. BigQuery support. Data sources are connected with connectors, and also third-party connectors are available, e.g. Facebook ad connector. Depending on the connector data query interval varies, e.g. new data fetched with 1-hour interval. Data itself can be altered at query time or after data is loaded. After data is loaded, it can be added to visualizations, there are multiple stock visualizations, and third-party solutions are also an option. Data visualization can be done manually, named as community visualization, can be done with JavaScript and CSS. In Figure 3.2 same example that was given with BigQuery is done within Data Studio and used parameters are visible at Figure 3.3. In Figure 3.4 the same data source is used and visualized with bar chart where the x-axis is year and y-axis is the total amount of babies born in the United States in that year.

	year	is_male	cigarette_use	weight_pou...
1.	1985	true	null	8.00057548797...
2.	1987	true	null	8.00057548797...
3.	1985	true	null	7.50012615324
4.	1986	true	null	8.00057548797...
5.	1985	true	null	7.7492485093
6.	1982	true	null	8.00057548797...

1 - 25 / 413306

Figure 3.2: Example data in columns from public BigQuery dataset

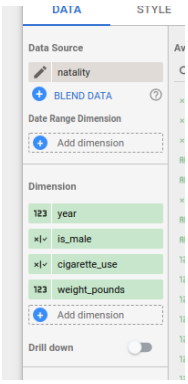

Figure 3.3: Data Studio selection tool

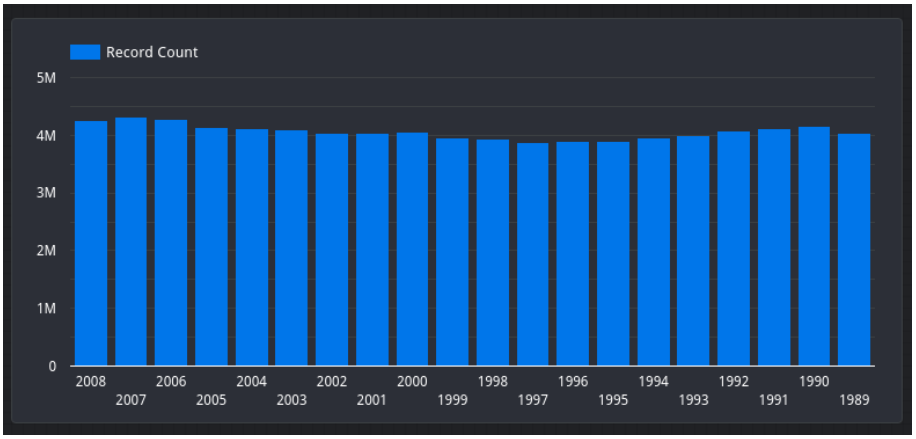

Figure 3.4: Example of Data Studio bar chart

There are multiple benefits to using Data Studio and on its core seem like business intelligent (BI) tool. Creating a dashboard with visualization and interaction is effortless, and creating more complex visualizations is possible with JavaScript. Where Data Studio lacks features is when data should be exported, a page can be exported to PDF, viewed in Data Studio or embedded to a website. When data is exported, Data Studio supports page-level presentation, where a page is an item that holds all visualizations with given layout ranges. A page belongs to a report that can hold multiple pages. When creating more visualizations that do not fit a single page, a new page has to be added. Data Studio report can be viewed only one page at a time, and this limit use cases for Data Studio if compared that you would create a larger dashboard.

3.4 Cloud Datalab

Cloud Datalab is hosted data analysis solution in Google Cloud Platform. It is built top of Jupyter notebooks with additions that integrates it to GCP, e.g. you can access BigQuery through Cloud Datalab and authorization is done internally in GCP. All the resources are inside of Jupyter notebook, e.g. as HTML, Markdown or Javascript, so sharing and easy access internally/externally is the main point of Cloud Datalab.

Cloud Datalab uses itself given Compute Engine resources and git repository inside GCP (Google Cloud Source Repository). Cloud Datalab has integrated libraries, and these base images can be extended with external libraries. In Figure 3.5 is an example of Cloud Datalab use-case, where the same BigQuery query as in Listing 3 is executed inside of Cloud Datalab.

```
import google.datalab.bigquery as bq
```

```
%%bq query
SELECT year, is_male, cigarette_use, weight_pounds
FROM `publicdata.samples.natality`
ORDER BY weight_pounds DESC
LIMIT 5
```

year	is_male	cigarette_use	weight_pounds
2,008	✗		18.001
2,004	✗		18.001
2,004	✓		18.001
2,006	✓	✗	18.001
2,004	✗		18.001

(rows: 5, time: 2.6s, 2GB processed, job: job_JIonLyju2xh0Zm2y2oYpAR--CPGt)

Figure 3.5: Example use-case of BigQuery in Cloud Datalab

Mostly use cases are quick and easily shareable data analysis. Cloud Datalab performance is limited by the underlying Python solution and runs on a single core per executable instance [25]. When more computation or general resources are needed, then other GCP tools are used inside of Cloud Datalab. As an example data cleaning can be done outside of Datalab, while testing data from BigQuery can be altered inside Datalab, and when the size of data increases, data altering can be integrated to BigQuery query. There is a lot of options to switch execution outside of Datalab that keeps resource performant even with high amounts of data.

3.5 Cloud Pub/Sub

Cloud Pub/Sub is managed, and scalable asynchronous messaging system. It consists of three parts publisher, topic and subscription. The publisher is an entity that sends messages, and messages are sent on a topic basis. The topic is an entity that publisher can publish to, and a subscriber can subscribe. The message itself consist of data and additional key values pairs. Subscription in Cloud Pub/Sub is possible on the push and pull basis, or messages are sent to a subscriber or requested by a subscriber. Messages are also stored until further acknowledge that message has been delivered to a subscriber, and resent if acknowledge is not received. In Cloud Pub/Sub subscriber can send messages to the same topic, and topic messages can be forwarded to multiple subscribers. Example of data flow in Cloud Pub/Sub system is presented in Figure 3.6.

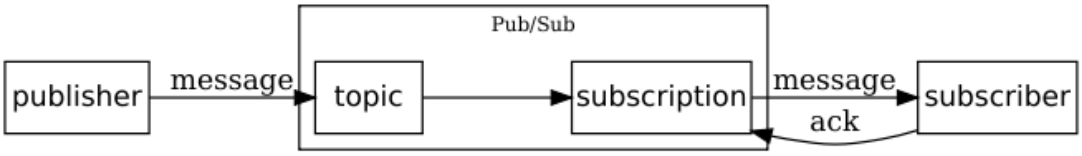

Figure 3.6: Pub/Sub flow

Cloud Pub/Sub can be used in many roles in GCP, e.g. data streaming to BiqQuery, asynchronous messages where a publisher notices that new file is loaded to Cloud Storage. A solution for easy to use asynchronous messages system in GCP that is tested and used at a massive scale internally by Google [26]. Cloud Pub/Sub is not an open-source solution. Use case limitation is that it is not meant for large objects because there is a limit on message size.

3.6 Cloud Machine Learning Engine

Cloud Machine Learning Engine (Cloud MLE) is a platform for training machine learning (ML) models and to serve ML models. Supported machine learning frameworks are TensorFlow, Scikit-learn and XGBoost. Frameworks have differences how they export models, but previously mentioned frameworks can all be trained and served on Cloud MLE. Cloud MLE functionality can be divided into two subcategories training and serving. More specifically training in GCP infrastructure offers enormous scaling potential, from one machine solution towards Google developed Tensor Processing Unit (TPU). Models can also be developed locally and only served in Cloud MLE.

In Appendix A is given an example of executable machine learning task in Cloud MLE, where linear regression is trained using Scikit-learn. Machine learning part can be ignored, and the main point is how Cloud MLE uses Storage Buckets. Data is imported, and a trained model is stored in Storage Buckets. Prediction and serving the trained model uses this exported model in Storage Buckets. Local training is possible when the exported model is according to Cloud MLE specification [27], because of hosting of the models need a specific structure that hosting service will be able to read the model.

Pricing of training is based on a training unit per hour, where used training units are computed with time spent, and after training is done, each job will output consumed training units. Prediction cost is based on time prediction nodes are used (which run trained models), and these nodes have hourly pricing. Amount of prediction nodes can be scaled automatically or manually.

Cloud MLE strengths are when compute, and data size increases to a more extensive scale. For smaller-scale machine learning purposes starting and shutting down training resources in Cloud MLE takes time from ML workflow. In a larger scale, GCP offers multiple tools for preprocessing of enormous amounts of data and accessing then easily from Cloud MLE. Serving models is a good solution because service can be scaled automatically and is cost-effective.

3.6.1 AutoML

AutoML Tables is fully hosted resources for creating machine learning models from tabular data [28]. At this moment, a data source can be imported from BigQuery, Storage Bucket or by uploading a file, where file format has to be CSV. When data is imported to the resource, it provides feedback from the data source, e.g. missing variables. When data is imported, training of the model can start. Training of the model itself is automated, and resource tries multiple different Machine Learning methods to reach the best goals, e.g. Linear and Feedforward deep neural network methods. The resource provides a lot of information on the training phase as visible on Figure 3.8. After the model is created, it can be served inside the AutoML Tables resources for batch and online predictions, and model can be exported as TensorFlow package.

In Figure 3.7 Apple stock data is used, and an example model was created with AutoML Tables where the goal was to predict the closing price by opening price and daily volume. In Figure 3.7 feature importance is presented and Figure 3.8 more details regarding the model performance is present, e.g. root-mean-square error (RMSE). The model does not hold any importance but to present that one can create a model from imported file and service handles the heavy lifting of machine learning work.

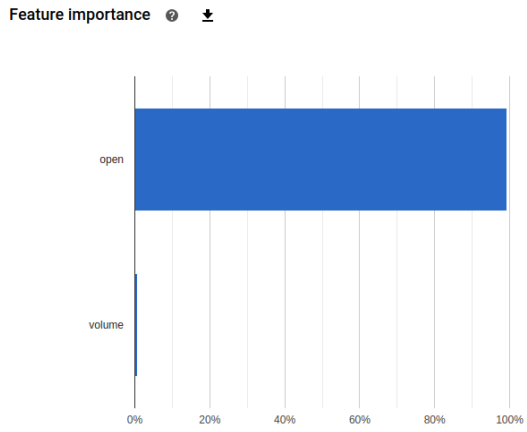

Figure 3.7: Feature importance for AutoML model

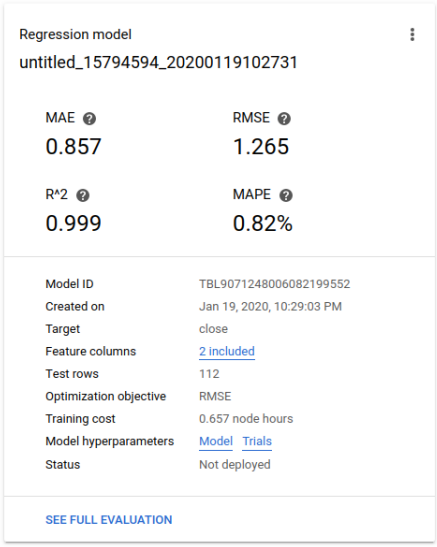

Figure 3.8: Additional parameters for AutoML model

At its core, AutoML Tables is an excellent service for a fully hosted machine learning solution. AutoML Tables is not a useful resource if more fine-grained control of the machine learning process is needed. AutoML Tables consists of an automated iterative process to train the model, and this understandably takes a long time, when faster model creation is required, there are better resources in GCP.

4 The underlying system

This chapter introduces the underlying system in more detail: what data system produces, which devices are in the system, examples of controllable devices, and flow of data to the cloud. The point is not to give an extensive description of entities but to give a brief introduction, and a reference when these entities are mentioned in other chapters.

In Figure 4.1 is shown a method of how data from the devices flow through the system. In raw form, data from devices is hard to read, and it is edited to readable form in the PLC. The PLC sends data to the local hub where data is altered accordingly specified parameters. The local hub sends data to the cloud where it goes to data pipeline, and through pipeline written to a database Previously explained steps are marked as blue lines in Figure 4.1. From stored data, PLC level parameters can be optimised and send back to the local level with a message broker, and this marked as red lines in Figure 4.1. The goal is to collect all possible data from the underlying system, and store it in the cloud platform, and have a possibility to create models from data and use the models to control the underlying system.

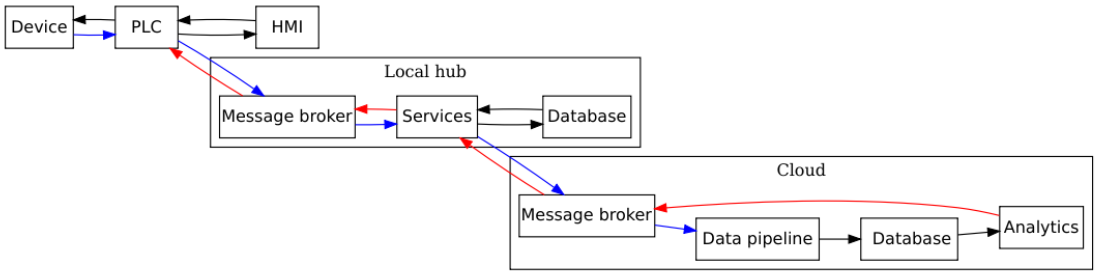

Figure 4.1: Flow of data in the underlying system

Examples of devices in the system can be pumps (P-100, P-101), flowmeter (1-F-1), basin (K-1) and level sensor (1-L-1) as given in Figure 4.2. Pumps are controlled with a frequency controller, while frequency and current data are collected. Outflow from basing is collected from the flowmeter, and the level of the basin is collected with a level sensor. From the given case, one combined data is to check historical data based on flow and pump frequency, that outflow has stayed the same, e.g. if a month ago pump outflow was one m3/h with 30 Hz and the past week pump used 40 Hz for one m3/h. Combining flow and pump/motor-related information can give information when the line is plugged, pump/motor is wearing down or multiple other reasons. The second example is to calculate inflows to basin even when there are no sensors for it based on outflow and level

sensor, e.g. at the start a hundred cubic meter basin level is 50 %, and outflow from the basin is ten m3/h, and after an hour from the start, the level is 60 %, and you can estimate the inflow as 20 m3/h. In the case of the second example, if the pump and level of the basin would be controlled with PID-controller, and the operational environment has changed enough, optimised PID parameters can be calculated and sent from the cloud to the PLC.

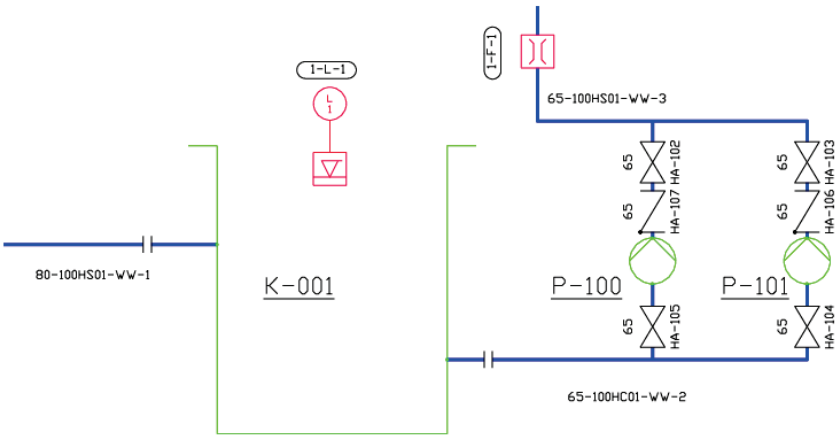

Figure 4.2: P&ID of the underlying system

An important goal is to collect data from multiple different systems and combine collected data from external sources. Analytical use cases for data are numerous, and few given examples are to provide an overview of the information that can be valuable for the system user. The big picture is to gather data to one source and provide global access to it and fade out some restrictions that globally scattered data can create.

4.1 PLC, HMI, and local hub

As shown in Figure 4.1, a local hub consist of three main entities: services, a message broker, and a database. The local hub services have few jobs: data input/output, data altering, storing data. Message brokers are used to transferring data from PLC to services that run on the local hub and give a possibility to send data from the Cloud through local service to PLC. The database provides the possibility to buffer data on the local hub when there are connection interruptions. All of the services, brokers, databases are run

in a virtual environment which provides different maintenance and scaling possibilities when comparing to other solutions. To simplify the local hub can be seen as a pipeline that alters incoming data, and sends data to the cloud if possible or writes data to a local buffer.

Control of the system happens through a Human Machine Interface (HMI) that communicates with PLC. An example of HMI is given in Figure 4.3, which uses InTouch HMI [29], and in this case, HMI is running on a standard Windows PC. To simplify, HMI can be kept as software front-end and user interface. Programmable logic controller (PLC) can be simplified as a computer which specializes in communication with multiple kinds of inputs/outputs, and connected inputs/outputs can be controlled. To simplify, PLC is a hardware and software back-end which HMI communicates with to access data. An example of what PLC looks like in the underlying system is given in Figure 4.4. In the underlying system, HMI does not hold any operational variables, and all variables regarding the system are stored in the PLC memory. When operational variables are stored in PLC, it creates one access point where data can be changed. In the underlying system, PLC does all the work: conversions, control logic, holds operational variables, writes data to the local hub. Also, to emphasize data to the local hub is sent directly from PLC, and code is written in PLC specific language and runs inside of PLC runtime. The explained simplified method is one of many ways to create a system that gives the same outputs, and several details will affect how similar systems are built, e.g. hardware used.

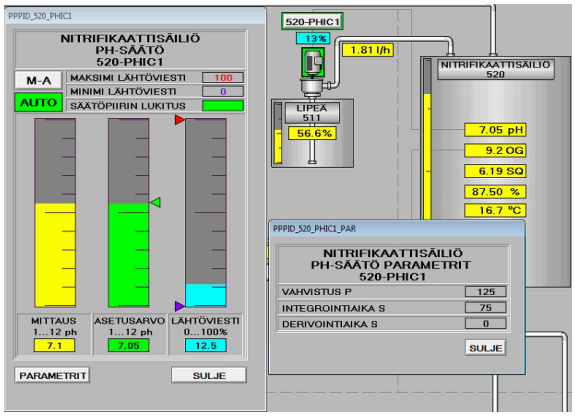

Figure 4.3: Screenshot of HMI from the underlying system

Figure 4.4: Picture of used Beckhoff PLC

4.2 Data

The underlying system will support multiple different data sources, and most important messages from PLC can be divided into six various classes. Those classes will be introduced in the next chapter. The local hub does not have all the data, and some pieces are

stored only on the cloud. E.g. status of the device is written as an identification number, and the explanation is stored only in the cloud.

Messages are presented in JSON format and a form that local hub sends to cloud. Two variables categorize the message, source identification string and type of predefined topic for different classes, e.g. `p600` as identification string and `total counter` as a class. Depending on used protocol identification and class is implemented accordingly.

All message types have two common items: timestamp and position. The timestamp is given as hyphen separated and including milliseconds The position is a unique device identification string that in this case, is the category of device and a number, e.g. FT100 is flowmeter (FT) and 100 is a unique number between all devices. As discussed previously, the local hub can alter data as needed.

4.2.1 Pulse counter

The message for class `pulse counter` is a pulse parameter from a device. The message contains one additional item value which indicates pulse amount, but not a unit of measurement which is stored only on the cloud. E.g. unit of measure for value can be a litre or a cubic meter. The message gives more fine-grained information of the device, and total counter values can be calculated from this. Listing 5 provides an example of a flowmeter with an interval of 1, and a new value is written each time a pulse interval is reached.

Listing 5: Example of pulse counter message

```
{"timestamp":"2019-01-20 16:04:40.890", "position": "FT100", "value": 1}
{"timestamp":"2019-01-20 16:25:40.000", "position": "FT100", "value": 1}
{"timestamp":"2019-01-20 16:56:40.500", "position": "FT100", "value": 1}
```

4.2.2 Total counter

The message for class `total counter` is an increasing sum of the internal counter. The message contains one additional item value which indicates an increasing total sum. This measurement is used to follow the total sum and gives easy access to data when asking the total sum between periods. The total counter can be calculated from pulse counter in the cloud and in that sense is a duplicate measurement, but useful parameter to follow. Listing 6 gives an example of a flowmeter with interval 0.1 for the total counter.

Listing 6: Example of total counter message

```
{"timestamp":"2019-01-20 16:04:40.890", "position": "FT100", "value": 100.0}
{"timestamp":"2019-01-20 16:25:40.000", "position": "FT100", "value": 100.1}
{"timestamp":"2019-01-20 16:56:40.500", "position": "FT100", "value": 100.2}
```

4.2.3 Device status

The message for class `device status` is a status of the device and written only when a device's status changes. The message contains two new items: device type and value. The device type item is a number that classifies the item, and the value is number that classifies status change, and both of these items are available only in the cloud. E.g. motor has five statuses: manual on/off, automatic on/off, and error. This data gives information about the device status and possibility to track down what has happened in a specific time interval. In Listing 7 example, a motor is used, where device type 10 specifies a motor, value of 50 is motor started in automation mode, and value of 51 means motor stopped in automation mode.

Listing 7: Example of device status message

```
{"timestamp":"2019-01-20 16:04:40.890", "position": "M100", "device_type": 10, "value": 50}
{"timestamp":"2019-01-20 16:25:40.000", "position": "M100", "device_type": 10, "value": 51}
```

4.2.4 Failure

The message for class `failure` gives more information regarding the specific failure of a device and is only generated when there is a failure. The message contains one new item, error code. The error code gives more information about the error type that has happened, and each device has multiple different error types. In Listing 8 motor with a position of M200 is given an error code of 20, which specifies overcurrent failure from the frequency controller.

Listing 8: Example of failure message

```
{"timestamp":"2019-01-20 16:04:40.890", "position": "M200", "error_code": 20}
```

4.2.5 Measurement

The message for class `measurement` contains device data with the specified interval. The message contains one new item value which is a measurement value of the value. The interval can be set to be a time-based or change of the value. From all the classes, measurement generates most of the traffic when intervals are set low. In Listing 10, a level sensor LT100 writes data from PLC with 10 second time interval.

Listing 9: Example of measurement between time interval

```
{"timestamp":"2019-01-20 16:04:10.000", "position": "LT100", "value": 10.0}
{"timestamp":"2019-01-20 16:04:20.000", "position": "LT100", "value": 10.0}
{"timestamp":"2019-01-20 16:04:30.000", "position": "LT100", "value": 10.1}
```

4.2.6 Parameter

The message for class `parameter` is a value for a specific parameter and written each time the parameter changes in the PLC memory. The parameter can be changed from HMI or external sources. The message contains one new item parameter, which is a unique string identifying the source of the parameter. In Listing 10, motor with position M100 PID-controller values are changed every ten seconds.

Listing 10: Example of measurement between time interval

```
{"timestamp":"2019-01-20 16:04:10.000", "position": "M100", "parameter": "P", "value": 60.0}
{"timestamp":"2019-01-20 16:04:20.000", "position": "M100", "parameter": "I", "value": 20.0}
{"timestamp":"2019-01-20 16:04:30.000", "position": "M100", "parameter": "D", "value": 50.0}
```

4.3 Devices in the system

The underlying system has multiple different devices that have a lot of functions and details regarding how devices work and how those can be controlled. This section gives a brief introduction to three different categories, where most of the devices fit. Also, one example is given where a data flow from a raw data signal to a local hub is presented, to give a brief introduction where the data comes from and real example of the underlying system. First category Motors gives more detailed information where given messages are generated.

4.3.1 Motors

Motors without frequency controllers in the system have few writable parameters. The motors without frequency controllers are controlled with a contactor which gives feedback to PLC if the motor is running normally or not (message to failure and status topics). PLC also keeps track of the running time of a motor (message for total and pulse counter). Few motors without frequency controllers have current monitoring devices which generate additional data (message for total counter, pulse counter and measurement). Frequency controlled motors give the same information that motor without a frequency controller, but also a lot of additional information. From a frequency controller, these parameters are available in the underlying system: voltage, current, motor moment, frequency, rpm, direction, and more failure states.

4.3.2 Sensors

The underlying system has multiple different kinds of sensors. Few mentionable devices are pH sensor, pressure sensor, flowmeter and level sensor. Depending on the device and manufacturer, there is a possibility to control sensors with external units or from PLC. In most simplistic terms, like in Section 4.3.4, devices emit 4-20 mA signal where sensor value can be calculated. Sensors have a possibility to generate messages to measurement, status, failure, and pulse counter classes.

4.3.3 Valves

The underlying system has a few different kinds of valves. Manual valves to change flow on the system which are not connected to PLC. Shutoff valves that are controlled with a signal from PLC to fully open or close, control valves that are controlled with a signal with per cent accuracy, e.g. 50% signal. Valves have a possibility to generate data to measurement, status, failure and pulse counter classes.

4.3.4 Example of raw device signal

One example of a device is a level sensor. In this example, Endress+Hausers ultrasonic level sensor FMR10 [30] is used. As Figure 4.5 shows, a level sensor is connected to PLC and provides sensors with power. Given FMR10 sensor setup is a two-wire 4-20mA and needs 18-30 VDC to the positive lead. FMR10 sensors can send 4-20mA signal on negative lead to PLC, where the signal of 4.00 mA is empty, and 20 mA is full. Level sensor's negative end is connected to PLC input. When the Beckhoff PLC's analogue input module is 12-bit, 4mA signal is measured as 0 and 20mA signal measured as 32767. When level sensor value is displayed or written to the local hub, it is converted to 0-100% scale with specified rounding. Scaling of the system has to be done on sensor level, and on the PLC level, that scale of 4-20mA signal is the same on both sources. The Local hub does not have information of scaling, but details of devices, e.g. the scale of sensors, is available on the cloud. The previous generates a risk if a new device is not scaled on both ends (sensor and PLC), the system starts to send false information. Given example in Figure 4.5 is small and straightforward, and more sophisticated devices and protocols that are not discussed. In this master's thesis, only higher-level data is presented and discussed, but in reality, more details of the system are needed when building more complex systems.

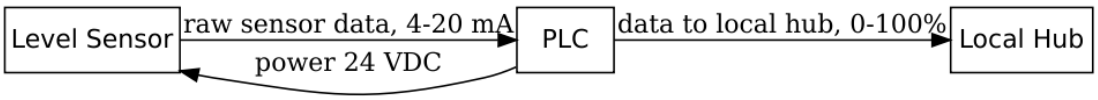

Figure 4.5: Example of level sensors data in each point

4.4 Requirements

Requirement 1 (R_1), cloud software architecture should be decoupled from local software, that all changes are done to external sources that are not run in a local hub. Data is stored only on the cloud and not accessible from the local hub. Data on a local entity does not have to be in a form that suits to the database, but it is required to make data uniform between multiple different sources. Uniforming data is done to create one source where various local sources can write without knowing how data is handled. Uniforming enables the possibility that only external source, where data is written, needs to be updated when changing architecture, which enables better scaling and maintenance possibilities. Decoupling of local and cloud entity creates a restriction that changing of data format on the local entity is hard if multiple local entities have to be updated at the same time.

Requirement 2 (R_2), writing data from the local entity to the cloud is done without or minimal downtime. Downtime is feasible only when the connection between the local hub and the cloud is down, and in these cases, the local entity will handle buffering the data until the connection is restored. The local entity is also responsible for logging connection interruptions. On the cloud end, software should not handle communication problems and only waits for data. This requirement creates a system that gives responsibilities to each part of the architecture. It should be noted that there is a possibility that some data can be lost when writing to an external source, even when the connection between a local entity and the cloud is good. When transferable data has high importance, more fine-graded control of data transfer should be used.

Requirement 3 (R_3), the local hub hosts services, a database, and a message broker in the local entity. Each local hub consists of a pipeline where multiple data sources can be written. The pipeline will alter data to a specified form. At the end of the pipeline, the local hub tries to send data to the cloud if sending is not successful data is stored in the local hub's database. The database behaves like a connection buffer and data is transferred to the cloud when sending is again successful, and in this case buffer size is multiple days. There should be no importance in which order data is received on the cloud. Local services have a responsibility to alter data to form that is suitable for use in

the cloud. The local hub message broker will send data to the cloud and receive data from the cloud. The local hub sends information regarding internal problems and operations. The goal of the local hub is to authenticate local instances, implement message broker for local systems and has single access to the cloud. Fetching data is only done through cloud instance and not served in a local entity, a local database is to ensure that no data is lost and store limited backup data. In the Local hub level time interval of syncing data to the cloud can be altered, e.g. sync data between five-second interval.

Requirement 4 (R_4), timestamp has to be written only after message broker handling, but there is a possibility to write timestamp in PLC and even doing data buffering on the PLC. Writing data from multiple PLC's would mean that each source should sync clock accordingly, each different clock should behave the same during rare events (e.g. daylight saving time adjust), and have internal PLC scan cycles the same that timing will match. Internal scan cycles of the PLC is a time interval that it reads inputs and outputs, e.g. scan cycle of 100ms means that data is only available between 100ms intervals. Mentioned issues would not be a problem when using only a few and modern PLC's, because most of modern PLC's can handle these issues internally, but when introducing many different entities and legacy systems, mentioned topics create a problem. It was decided that the local hub writes timestamp to incoming stream, but as discussed it is not a must. Adding time in the local hub creates one access point to alter time. And the local hub's time is synced with external clocks.

Requirement 5 (R_5), on the cloud one raw data source should be created that does not have any altering after the local hub. A raw data source can be used to fix data if something has happened between the processing pipeline in the cloud. All stored data should be available globally. To enforce where data is stored, data is not accessed or not available in the local hub. Data is stored and modeled in the cloud and cloud solutions can write directly to PLC, and the local hub functions only as a gateway for that data.

Requirement 6 (R_6), the software in the cloud should have access to the local hub and from that to the PLC. Local access is only used when the connection is available through a proven cloud provider solution. Without local access, the system only stores data to the cloud, where models and dashboards can be created, but controlling the underlying system is done manually. Local access gives the possibility to automate and control the underlying system.

Requirement 7 (R_7), global access and extensibility are important points for designing the system, and the importance of these goals increases when local entities are globally distributed. Global access creates a shared point that each user of the system can access data from multiple points. That global access point should also have a possibility to authenticate users and restrict access with given permissions.

Requirement 8 (R_8), no private data is stored in this project, and the architecture will not enforce extensive data security. Using best practice rules in GCP is enough for data security. Storing is planned for time-series data from various devices, and access or login is done with hosted service that most cloud platforms provide. Stored data or information itself can hold valuable knowledge, e.g. when developing new products extensive data security should be enforced.

Requirement 9 (R_9), architecture should have aggregated data source that if data is used for sources that do not need data in raw form aggregated data is used. The goal for aggregated data is to defer workload from a raw data source. Minimum aggregation interval should be minute and maximally one hour.

Requirement 10 (R_{10}), hosted services should be preferred when compared to unhosted solutions, especially where solutions cannot have any downtime. Hosted services provide few qualities that are preferable, e.g. reliability, interfaces to operate services and interconnection with other tools in the cloud platform. Especially in this architecture, all databases should be hosted services. This requirement does not create restrictions, and all hosted services can be replaced with self-hosted services if needed. Hosted services can be seen having some additional costs, but on the other hand, creating the same level of service would require more maintenance and work hours.

Requirement 11 (R_{11}), machine learning modeling should find anomalies and relations in time-series data. Anomaly detection can be narrowed down to find relations with pump, flowmeter and level measurements. Analytical algorithms should find optimised parameters based on time-series data to control the system, that outputs optimised parameters for PID controllers. Data for machine learning and analytical models are given as attributes, and the goal is not to generate a model that handles multiple different sources. The goal is to create models with given values, e.g. levels ID 1 and ID 2 are connected to pump ID 10, and not find values from raw data, e.g. find values that are correlated with pump ID 10.

5 Need for analytical and machine learning methods

This chapter introduces a few real-world problems that are present on the underlying system and gives one possible solution to how the built system can solve current issues. Older and existing solutions are local and usually work independently, and most of the optimisation work involves manual work, e.g. exporting data, manually inputting model output to the system. The goal is to create a distributed access point to develop models and analytical work and create a platform that improves efficiency and tracks modifications that are made.

The goal is not to introduce extensively why one solution was chosen and to refer solution being the best to practice for a given problem. Given problems can also lack some underlying details to present them without extensive know-how of the underlying system, and we avoid concentrating on features that are out of the scope of this master's thesis. The goal is to give an example of how a problem can be faced, and not iterative process how the chosen solution was the best performer.

5.1 Case 1, PID controller

5.1.1 Problem description

Proportional–integral–derivative (PID) controllers are the most common industrial controller withing different industries [31], and the underlying system also has multiple PID controllers. To clarify PID controller function simulated difference between on-off and PID controller can be seen between Figure 5.1 and Figure 5.1. Simulation is based on a basin with a pump and a level sensor. In Figure 5.1 simulation of an on-off controller is given where the blue line is start level and the red line is stop level. At the start level the pump starts, and on the stop level the pump stops, and start/stop levels have some hysteresis with also visible in Figure 5.1. The on-off controller can create a lot of oscillation and high peaks on the system. The on-off controller is one of the simplest but rough at some states, and with PID controller, there is a possibility to get more fine-grained control. In Figure 5.2 simulation of a PID controller is given where controller setpoint is given as the red line, and the PID controller tries to minimise the error between measurement and setpoint. Important to note, when using PID controllers, there is a need for some additional equipment that signals for the device can be controlled, e.g. a frequency controller for a pump. Possibility to use PID controller can increase total complexity and cost, e.g. more devices, more control software, but offer more fine-grade control of the process.

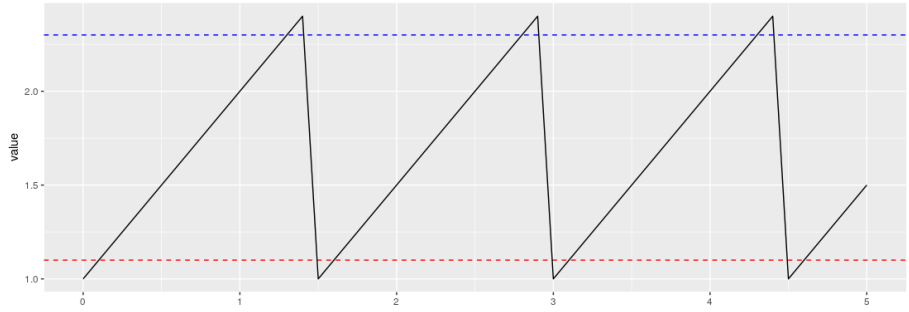

Figure 5.1: Example of on-off controller

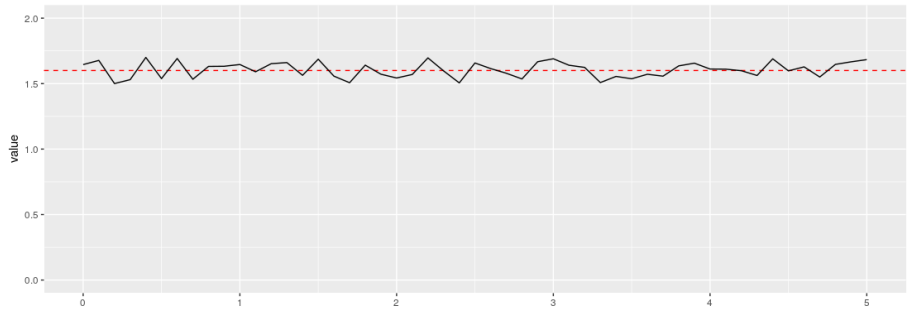

Figure 5.2: Example of PID controller

In Equation (1), PID controller equation is given as presented by Karl and Åström [31]. The first term in the equation is P-term, second I-term and third D-term, and all terms are multiplied with gain K. User given PID controller parameters are proportional gain K, integral time T_i and derivative time T_d [31]. There are setpoint and measured value: a setpoint is the desired value, a measured value is current value. With a PID controller, the error between a setpoint and measured value is tried to be minimised. P-term is a proportional error based on the difference between setpoint and measurement, and P-term increases when the error between setpoint and measurement increases. I-term is an integral error based on past data, and I-term increases when the integral increases. D-term is a derivative error based on setpoint and measurement, and D-term increases when a chance of error increases.

$$u(t) = K\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{de(t)}{dt}\right) \tag{1}$$

To clarify PID controller terms, they are graphically presented by Doren Figure 5.3. Setpoint is given as a red dotted line and measured value as a continuous black line. P-term is an error between the measured value and setpoint and marked as a blue dotted line. I-term is based on integral error and presented in the figure as integral between the green dotted

line and blue dotted line. D-term is a derivative error and presented as the purple dotted line.

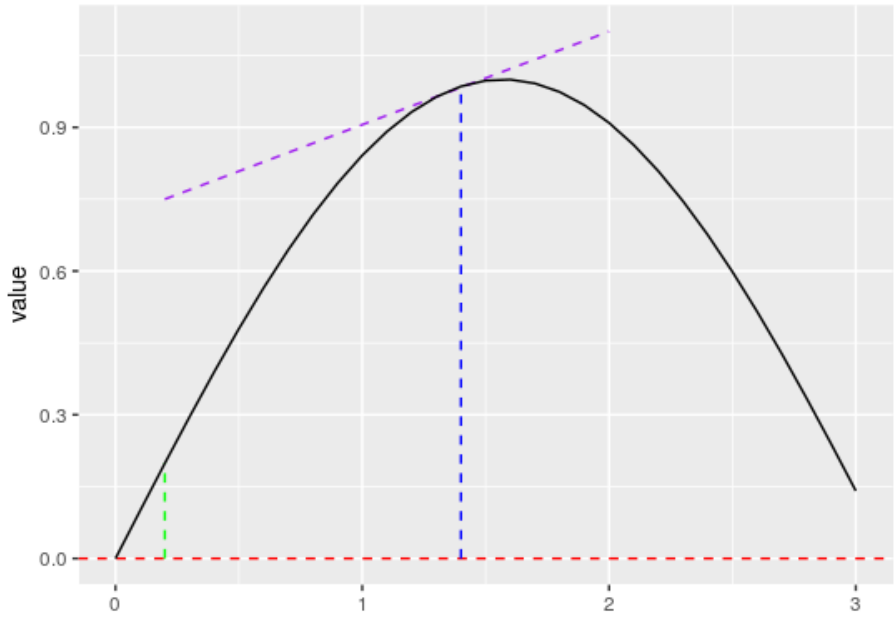

Figure 5.3: Graphical presentation of PID controller terms

5.1.2 A solution

There are multiple different tuning methods for PID controllers, and one of them is a relay method, as presented on material [32]. With the relay method, the PID controller is turned off, and the on-off controller turned on. Turning PID controller off creates oscillation on the measurement where time (ultimate period T_u) and amplitudes (a and b) of oscillation are measured [32]. When parameters are measured, one can calculate ultimate gain (P_u) as presented in Equation (2), and then calculated values can be inputted to Ziegler-Nichols tuning form [32], which outputs needed PID controller values.

$$P_u = \frac{4 \times b}{\pi \times a} \tag{2}$$

As given in Figure 5.4, there is a chemical pump, that doses chemical to the underlying system. In the system, pH decreases because the biological event and chemical pump doses caustic soda to increase pH in the basing. In this specific case, pH is kept around 7.05 (setpoint visible on Figure 5.4) and depending on a few inflow parameters rate of change differs. The measured value is pH on the basin and shows 7.05 in Figure 5.4. Signal to the chemical pump is visible at 12.5%, and the PID controller outputs this signal. The raw signal to the chemical pump is a 4-20mA signal, and an output signal of 12.5 %

converts to 1.81 l/h dosage on the chemical pump. PID controller terms are the right side of Figure 5.4: Gain 125, T_i 75, T_d as zero. When D-term from the PID controller is missing, it functions as a PI controller. If PID controller setpoint is 7.05, relay points can be set $\mp$ 0.05 from setpoint which would start (50% speed) and turn off the pump (0% speed). After the relay function, new controller terms can be calculated, and PID controlled turned back on.

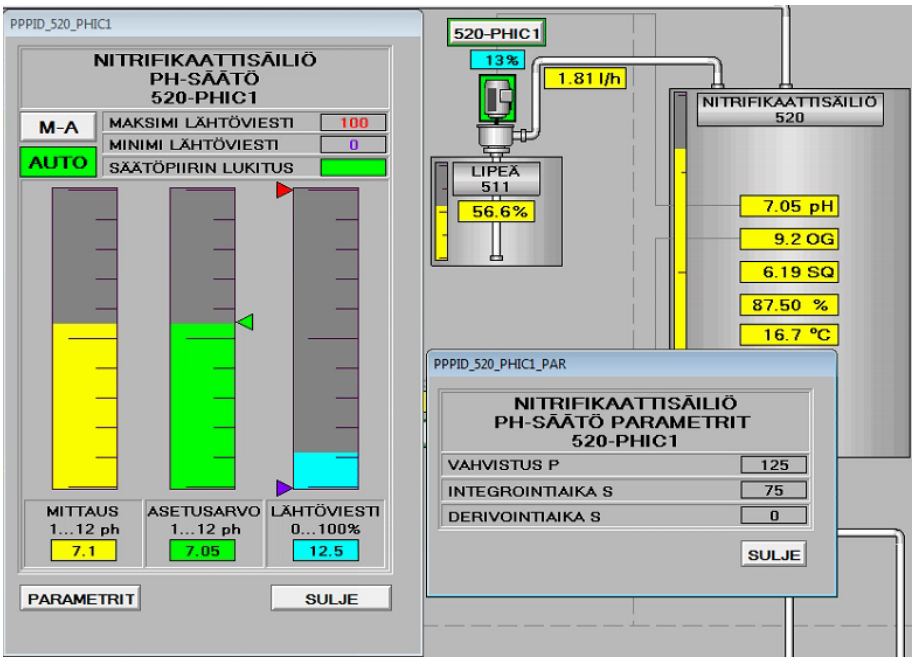

Figure 5.4: Chemical pump

The problem and solution describe a method to control values that are stored inside the PLC memory and usually manually optimised or local tuning software. PID controller values themself are changed through HMI. This section introduces an example of how parameters from the underlying system can be optimised on the cloud. Relay method also introduces some dependency because some part of the function has to be done on the PLC level, e.g. relay function runs daily for a set time interval to get accurate data. There are also more modern PID controller tuning software that are based on analytical work and tune parameters based on setpoint and measurement. Relay method was chosen because of dependency with local solution, and it introduces a point that analytical work on the cloud can be based on a local feature. Relay method was tested in the underlying system, but more modern solutions for tuning PID parameters are used.

5.2 Case 2, following pump defects

5.2.1 Problem description

The underlying system has multiple different devices, as presented in Section 4.3, and between these devices, there are numerous different possible correlations. Chosen correlation to present was to model the correlation between a frequency-controlled pump and a flowmeter because this correlation is simple and does not need any extensive domain knowledge. Pump's running frequency and outflow of the material do not always have stable correlation, because environment and parameters can change on pump suction and pressure side. One example of changing variable is the absolute level on the suction side, when potential energy increases in the basin and which should lead to easier work for the pump. These more complicated correlations are not discussed in this master's thesis, because they will need domain knowledge, specifics of each device, details of the environment, and how each parameter will affect each other.

From the underlying system, pump `500-M3` and flowmeter `500-FT3` were chosen, and the devices are visible in Figure 5.5. The flowmeter is located on the pressure side of the pump, and the pump is located between two basins to dose material from one basin to another, and the flow is not bidirectional. The absolute water level, when compared to pump, on the suction side is roughly one meter and on the pressure side roughly five meters. Water levels in the basins are roughly constant and maximal difference between two water levels is five meters. The pump is controlled with a PID controller and setpoint is suction side basin level. The PID controller is limited and can output signal of 40%-100% (setting value top left in Figure 5.5) which on this specific pump is 20-50 Hz signal range. Possible defects in this example are: the pump is wearing out, the pump is plugged, the line is plugged, measured values are wrong (e.g. signal unreadable from the flowmeter). From given information, it is not possible to classify which defects have happened, only that performance of the pump has changed in relation to the flowmeter.

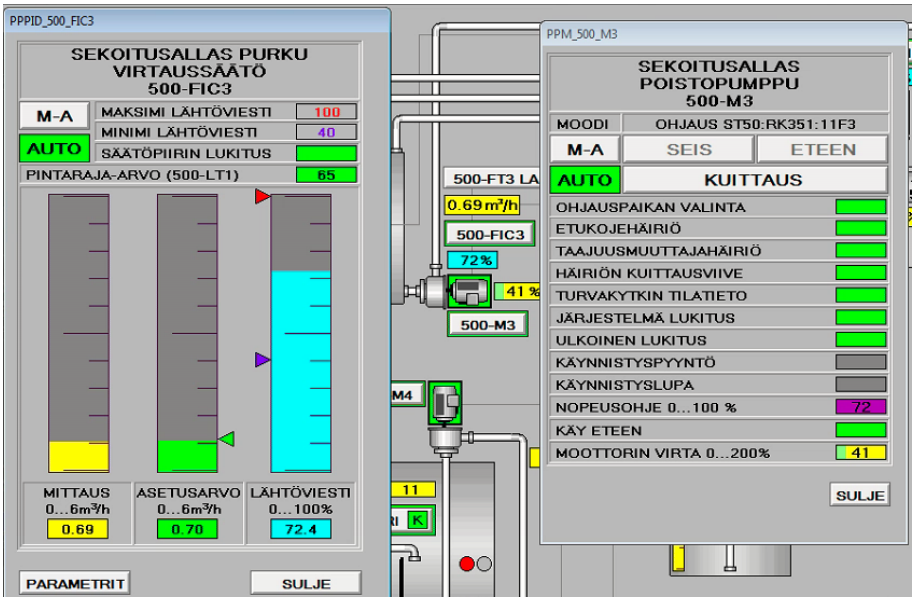

Figure 5.5: Chemical pump

5.2.2 A solution

The solution was to create a regression model. In this regression model, there are only two fields frequency of the motor and measurement from the flowmeter. In the data, the dependent variable is the frequency and independent is flow measurement. The outputted model will predict frequency based on inputted flow value. First data was cleaned, and these items were removed: pump in manual mode, PID controller not on, first ten minutes after pump starts, errors with pump, controller or flowmeter. One example of a removed item is pump start-up when this specific pump starts a controller and pump creates some oscillation (because of mechanical features of the underlying system). After some time interval, the pump controller is in a normal working condition without errors present. Cleaning the data is not a must, but if training data includes item data enough, it will affect the outputted model. After cleaning the data, it was separated to training and test dataset. GCP gives hosted service for creating regression models called AutoML Tables, and this service was used. AutoML Tables prediction resource was used to host the model. Predictions were made on daily intervals, and only selection of daily data was fed to the model. Input data to model was also cleaned with same items. When predicted and real frequency value had more than ten percent difference, timestamp of the input was logged and stored for the plant operator to check.

As a parameter correlation between frequency and flowmeter can alter fast and in this example is not a parameter that should be modeled as fast as possible. The goal is to check correlation fluctuations with more gradual perspective, to achieve goals that were

previously mentioned. In an example, the model will log that pump did run with 30 Hz frequency to achieve ten m3/h of flow, but today the pump is using 50 Hz to achieve the same flow. In this example, a simple regression model will achieve the desired goals, but when modeled entity increase in complexity details of them have to be added as independent variables. The purpose of this section is not to present the best solution for how to model the correlation between a frequency-controlled pump and a flowmeter, but to introduce an example which uses collected data to gain information of the system.

5.3 Case 3, anomaly detection

5.3.1 Problem description

The underlying system and other industrial processes have various good sources for anomaly detection. Chosen device is a frequency-controlled screw blower and a current parameter of the device. Motor current monitoring itself can be used for multiple purposes, but in this example, fast current altering was chosen to be modeled. How motor functions can be altered with frequency controller parameters, but when in normal operation conditions, e.g. controller with PID controller, current fluctuations of this device are minimal. In normal operation current of the device is around 70.00%, and when there is a problem, the current can spike fast between 0-100%. There are multiple reasons why this phenomenon occurs, but the compelling case for it is to find a case when the driving belt of the screw blower fails.

Example of this anomaly is visible in Figure 5.6. In normal operation the current is steady, and this is visible on the left side of the blue dotted line. When an anomaly happens or belt is staring to break current can jump around, and this is visible between the blue and red dotted line. After the belt is broken down, the motor starts to run freely without load, and this is visible at the right side of the red line. Depending on details, the described anomaly can take less than a second or longer than a minute.

Multiple different reasons can cause breaking of the driving belt. e.g. belt is gradually wearing down, the belt is poor quality, and extended stress. The problem is to notice when the motor current starts to oscillate, and the driving belt is going to break down, which can take several minutes. The motor oscillation occurs because the drive belt does not provide a constant load to the motor (low and high current spikes), and the PID controller tries to compensate when the error compared to the controller setpoint changes rapidly. The driving belt breaks down infrequently, and driving belts are replaced with device-specific intervals.

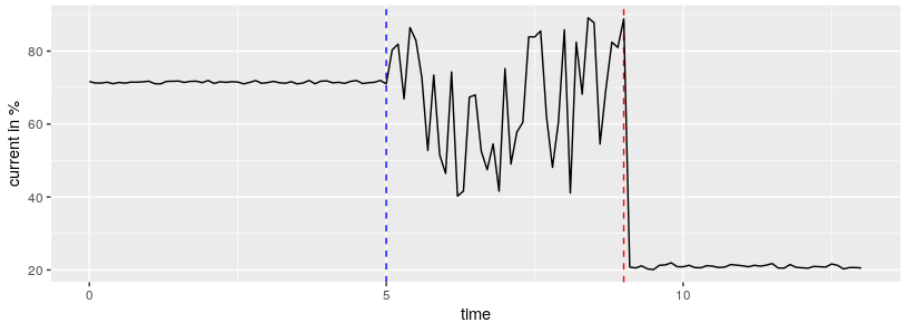

Figure 5.6: Example of current of the blower when a belt is breaking

5.3.2 A solution

Few parameters limit the solution. First, there is not enough data when this anomaly has happened, and second, real-world simulation of this given anomaly is not safe (actually breaking the driving belt). Models have to be trained and tested without anomalies or with simulation, or use algorithms that do not need training. The problem itself is simple and like visible in Figure 5.7 when adding smooth moving average, presented as green, gives a good idea of how possible anomaly is visible (oscillation generates a bigger difference between average and measured value). With some testing, this anomaly would be possible to be found based on moving averages, but this is not a good method even in this simplified case.

More sophisticated algorithm Streaming Least Squares (SLS) was used for anomaly detection. SLS work can be divided into four parts: partition data to sliding windows, calculate least squares regression on those windows, sort and filter overlapping windows, categorize output to different levels [33]. SLS algorithm functions also for batch and online predictions, because of SLS algorithm window-based calculation does not depend on past or future outputted scores. In Figure 5.8 is presented the output of the SLS algorithm with a two-second window with data from Figure 5.7. In Figure 5.8, a level and score for the anomaly are given, and the three highest anomalies are comparable when the simulated anomaly has happened.

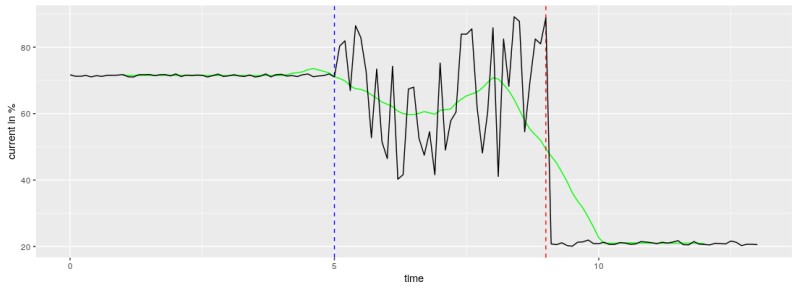

Figure 5.7: Current data with simulated drive belt breaking and moving avarage

	level	start	end	score
0	5	7.7	9.6	99.928376
1	4	5.5	7.4	60.818267
2	3	3.5	5.4	17.417605
3	2	10.8	12.7	1.740603
4	1	1.0	2.9	1.279903

Figure 5.8: Example output of SLS algorithm

First data was cleaned, and these items were removed: blower in manual mode, PID controller not on, first five minutes after blower starts and stops, errors with blower or controller. When these items were reduced, blower data will evaluate to normal working conditions. When data is cleaned, it was imported to Cloud Datalab where SLS algorithm was tested. After the used cloud solution has started gathering data given problem has not happened, and anomaly testing is based on simulations based on prior knowledge. It was decided that the problem will not have online control of the underlying system and more data has to be gathered when the error happens because the goal is to stop the blower before the belt is breaking down. False error or protecting the device without reason will not generate any harm only additional manual work and to determine that it was a false error made by anomaly detection.

SLS algorithm was used for batch processing of data, and not for online predictions. The latency between, sending data, processing data, sending it back to a local entity, and then stopping the device is too large to catch anomalies that happen under a few seconds. Some parts of the processing and predictions should be transferred to the local entity from the cloud if these fast occurring anomalies should be handled in a way that controls devices, e.g. stopping the device.

6 Architecture and specifics

Google Cloud Platform (GCP) offers multiple excellent solutions for hosted and self-maintained software architecture, and the presented system can be built with numerous different GCP tools. The architecture of it can evolve as more experience is gained and additional needs emerge. However, the main point is to keep low coupling between software components and GCP tools, that is to keep the software modular enabling changes in the software architecture, e.g. changing the database.

In Figure 6.1 components of architecture is given and divided into four components: message handling, storage, analytics, and data access. Message handling component handles incoming messages, and copy messages to each signed resource, e.g. in Figure 6.1 raw data storage and real-time analytics are a different resource. Storage component consists of used storage resources for other resources. Analytics component is used to read raw data storage and gain information about data. Data access is a component that ensures that data is readable through REST API, e.g. third party access. With these four main level components, there should be no data loss if some component has downtime, and in this section, it is discussed how this architecture achieves downtime recovery.

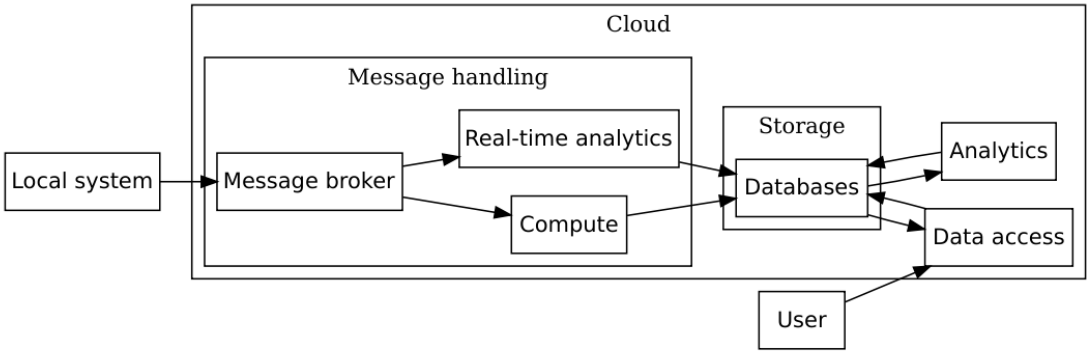

Figure 6.1: Basic Architecture of solution

Used elements of the software architecture are given in Figure 6.1. In this chapter, the architecture is divided into two sections: solution on local entity and solution on the cloud. Section 6.1 regarding a local system is to present steps between PLC and the cloud. Section 6.2 regarding cloud architecture is divided into three parts: writing data, data access and analytical and machine learning work. And most essential elements in the architecture are described and what solutions from Sections 2 and 3 were chosen. If solution implements requirements, it is referend as R_1 where the number refers to the numbering in Section 4.4. In Section 6.2.4 each case that was presented in Section 6.2.4 is discussed

and how those cases are implemented in the architecture. In Section 6.3 architecture is discussed more generally and how to improve the architecture.

6.1 The system in the local entity

A local entity like given in Figure 6.2 consists of a local hub, and data producing systems. In the figure, a local entity is an industrial plant/area with the same local network, and a local system is a production line inside of this industrial area/plant. A system can be various different things, e.g. PLC, laboratory analyser, manually filled reports, but in this master's thesis, it is limited to PLCs which writes data to a local hub. Services in a local hub handle data altering, buffering and communication between the cloud and local systems. Communication is restricted that only local hub is visible to an outside network. Cloud providers do offer a solution for a given problem like Cloud IoT Core and Cloud IoT Edge in GCP. With these tools, normal IoT solutions in GCP will work. Still, the software runs inside closed automation network with critical devices and systems, and only limited third- party software will be given access to it, this is the reason why own local hub is used.

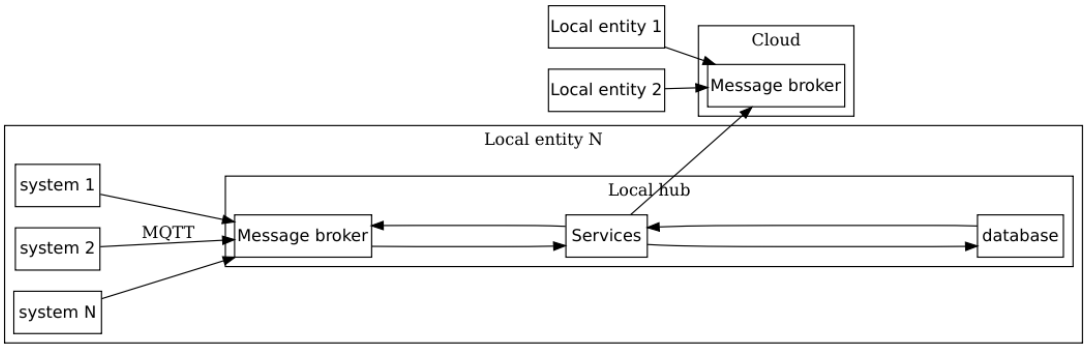

Figure 6.2: Basic architecture of a local entity

To add a new data source or a system, it is needed to define the used protocol and data model for a message broker. And how data should be handled in a data pipeline, stored in the buffering database, and written to an external source (cloud). For the underlying system message queuing telemetry transport (MQTT) protocol was chosen to be used between local hub and systems, but the chosen protocol should not be a limiting factor when selecting a message broker. MQTT is a lightweight messaging protocol that is based on a publishing and subscription model. To simplify, in MQTT, protocol topic is given where a publisher can write messages and subscriber can read those messages. Different message brokers also provide different functionality to extend functionality of each broker,

e.g. a configurable queue for messages. In the underlying system, MQTT topic was decided to be divided into the plant id and message type, e.g. /p600/measurement, which will give a possibility to filter messages at message broker level at plant and message type level. MQTT also allows two-way communication between systems, a local hub and the cloud (R_6). As a message broker RabbitMQ [34] was chosen because it supports multiple different protocols (e.g. MQTT and HTTP), multiple programming languages, and it is open-source licensed. After the message broker, a data pipeline will add a timestamp to it (R_4) and additional parameters as specified. Data pipeline makes a decision where to write data, into the cloud, back to message broker or to the local database. Data pipelines for MQTT topics were written in Golang, and no external sources were used. When throughput increases third party stream processors can be used, e.g. Apache Beam. Writing to the cloud was done with Cloud Pub/Sub SDK where it is possible to optimise writes to the cloud accordingly your needs, e.g. batching of writes with a time interval. FoundationDB [35] database was chosen to be used in the local hub. In the architecture there are few notable restrictions: the message broker has to support used protocol, the system and data pipeline have to support one of the protocols that message broker uses, and data pipeline has to have an support for the local database and the protocol that is used when writing to the external sources. Chosen components can have features that have beneficial functions for the system, but can be kept as personal preferences because they do not create a restriction, and changing the database does need only changing the interface between service and the database. Discussed points in this paragraph implement requirements R_1, R_2 and R_3.

In Figure 6.3 is an example of how a message from PLC flows through this architecture, and in Listing 11 example of data is given. First MQTT message is written from PLC to a message broker (RabbitMQ), in the Listing rows 2-3 describe data in MQTT message that was sent to topic /p600/measurement. In the data pipeline additional arguments are added to data, in the Listing rows 6-11 describe data form after data pipeline. After data pipeline entity, timestamp, system and message type items are added to categorize data. Data pipeline has options for writing data to the cloud and to a local database (FoundationDB). In the figure, the only visible service is data pipeline, but in reality, many other services are part of the total architecture, e.g. service that handles writing buffered data from the database to cloud.

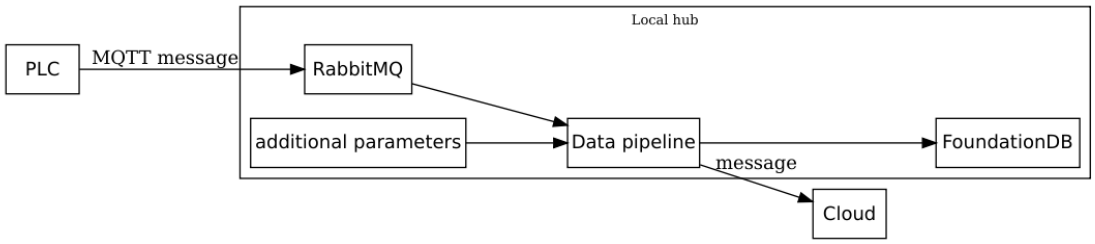

Figure 6.3: PLC writing message to local hub

Listing 11: Example of measurment messages

```
# What PLC writes to message broker
{"position": "FT100", "value": 100.0}
{"position": "FT100", "value": 100.1}

# Form that is send to the cloud
{"timestamp":"2019-01-20 16:04:40.890", "entity": "honkajoki",
    "system": "p600", "type": "measurement",
    "position": "FT100", "value": 100.0 }
{"timestamp":"2019-01-20 16:25:40.000", "entity": "honkajoki",
    "system": "p600", "type": "measurement",
    "position": "FT100", "value": 100.1}
```

In Appendix B parts of the code which handles data altering and writing to the cloud are presented. Message class Failure was chosen for this example. Failure messages are infrequent, and in the underlying system, roughly two Failure messages are generated daily. So there is no need to batch this message class and should be sent quickly to the Cloud where they can be analysed. Code is written with Golang and consist of the three functions. Failure struct is a structure for outgoing data, and incoming data have missing fields that have to added. FailureHandler function reads incoming data from MQTT topic and alters parameters as needed, and writes values to Golang channel. WriteToCloud function takes Golang channel as an argument and reads items from that channel, and function is blocked until a new item is read from the channel. WriteToCloud function writes data to Cloud Pub/Sub topic or when it fails to channel that writes data to FoundationDB. In total, Appendix B consists of less than 20 % snippet of actual code, but the three most essential functions of how data is handled before it is sent to the Cloud.

6.2 System on the cloud

6.2.1 Writing data

Data is written to the cloud from a local entity according to Figure 6.4. There a local entity is connected to a message broker on the cloud, and in this case, it is Cloud Pub/Sub (see Section 3.5). There is a limited amount of topics that local entity publishes to, and Cloud Pub/Sub is a scalable and easy to maintain solution for this use case. The Cloud Pub/Sub offers a robust way to alter the flow of data, e.g. when switching to use a new database it is possible to connect it with a new subscriber and write data to multiple subscribers. Each message class will have an own topic where compute resource or stream processing subscribes to handle incoming data. Each message to a topic will have Cloud Pub/Sub specific attributes that can be used to filter messages with computing resources. After the message broker, there is a compute resource that subscribes to a given topic, and this architecture uses Cloud Functions (see Section 2.5.4) or Cloud Dataflow (see Section 3.2). Cloud Functions provide hosted (R_{10}), and a cost-effective solution for a given solution and writing data from a local entity to storage is serverless, which provide scaling within cloud provider capabilities. Usage for Cloud functions is based on message classes that do not need a huge amount of computing resources, and throughput is lower, e.g. infrequent data that is generated only by an hourly interval. Cloud dataflow is used in cases where data is streamed, and additional stream processing is needed, e.g. machine learning for streamed data. Cloud Pub/Sub service will scale without problems but compute resources has to be chosen more gradually with a specific need.

As mentioned and presented on Figure 6.4 incoming messages are divided at message broker and which features one to many message handling model. As visible on Figure 6.4 `Message topic 1` is sent to multiple different subscribers, and even if other subscribers fail others will not be affected. This architecture provides a possibility to test and switch using different types of resource, e.g. switching database would mean to create a new subscription and routing messages to storage. Each Cloud Pub/Sub topic has an internal buffer and if something fails messages are not lost.

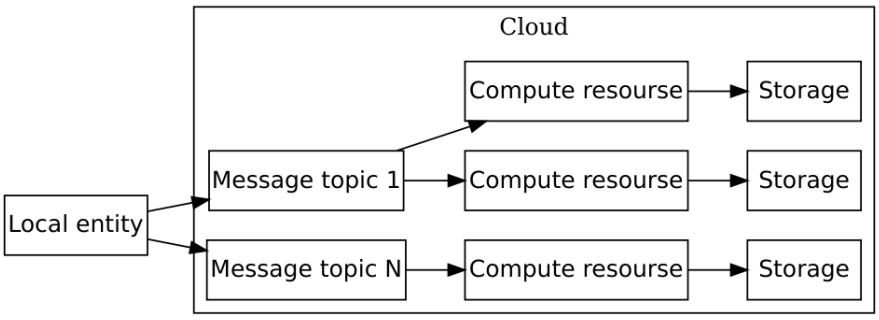

Figure 6.4: Writing data to cloud

For databases BigQuery (see Section 3.1) was chosen for raw data storage R_5, Cloud Datastore (Section 2.4.3) was chosen for aggregated data (R_{10}) and Storage bucket (see Section 2.4.6) for storing large items, e.g. database backups. There are better-hosted solutions for time-series data, especially Bigtable (see Section 2.4.2), but the underlying system does not produce enough data for a solution like Bigtable. When the amount of data increases, Bigtable is an excellent solution to use. BigQuery is used for raw data storage and Cloud Datastore for aggregated data that is computed from BigQuery raw data. The reason for aggregated data is to serve different dashboards and third-party access to data, and Cloud Datastore offers better scaling when compared to other low-cost products. Using Cloud Datastore creates a restriction that moving outside of GCP is hard because it is not an open-source product. Naming on databases are based on message classes, and which entity produces the data. In BigQuery dataset, table refers to the message class, and during solution was developed, only raw data is stored in BigQuery, and this is enforced dataset table name. As an example, message class measurement from Honkajoki entity will translate to honkajoki.measurementRaw. BigQuery table which includes data from multiple different entities. Data aggregation uses Datastore, and table name refers to a message class similar to BigQuery. All of the used resources are hosted services, and this implements requirement R_{10}.

In Appendix C, Cloud Functions code is given for handling incoming Cloud Pub/Sub messages regarding Failure message class. Given code uses Cloud Functions background functions that are invoked by events and in this case, a new Cloud Pub/Sub message. Each time Cloud Function is started, the init method is called, which creates connections between Cloud Function and BigQuery. When a new message is written to Cloud Pub/Sub topic, it is forwarded to FailureHandler function where data is written to given BigQuery dataset. If data from a specific source, e.g. failure messages, should be written to multiple different sources, e.g. BigQuery and SQL, messages are separated at Cloud Pub/Sub level. Each source should have an own subscription where messages are delivered at least once.

Code in Appendix C will work when few hard-coded variables are changed, values to BigQuery dataset and table are given, and when proper GCP IAM roles are set. Logs written in Cloud Functions are written to Stackdriver Logging service in GCP. What is not visible at Appendix C is how cloud function handles error and how data is forwarded to other subscription that handles failures for a message class.

6.2.2 Data access

Serving data to users and APIs for system data is run on Kubernetes Engine (see Section 2.5.3). It was decided that each needed service should be run in Kubernetes Engine, and service refers to software solutions like dashboard that serves generated alarms from the underlying system. Kubernetes creates one central place to run non-managed applications inside a hosted service that has complete access to services running inside the GCP project, e.g. access to databases. Kubernetes offers more extensibility and fine-grained control when compared to App Engine, Cloud Functions and Compute Engine, and functionality of other compute resources can be executed in Kubernetes Engine. Kubernetes is becoming de facto standard to run container engine, and most cloud providers serve similar hosted Kubernetes solutions, which help future proof solution if cloud provider changes. Each specified service runs on its separate cluster, e.g. client dashboard and data access API have separate containers. Each container access to GCP tools can be restricted with IAM rules, e.g. given service account for a container can only read given BigQuery dataset. So each service has limited access to the whole GCP project. As presented in Figure 6.5, centralized data access is created on Kubernetes containers, and these containers can access each other as presented with connection with dashboard reading data from API. All of the used resources are hosted services, and this implements requirement R_{10}, Kubernetes Engine offers also globally distributed point for services as mentioned in R_7.

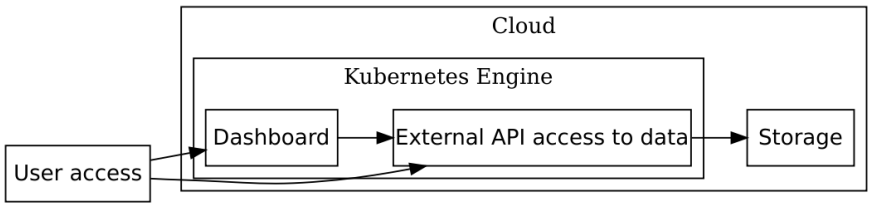

Figure 6.5: Centralized external data access for data

6.2.3 Machine learning and analytic work

Machine learning work is divided into three different steps: fetching and cleaning raw data, creating machine learning model from data and hosting machine learning model. Each step is modular with specified relations, e.g. data for linear regression model of frequency controller motor and flowmeter has to be in a specified form. Each section can be done locally, e.g. training machine learning model locally, but cloud operation should be preferred because of the ease of data shareability and model monitoring. Data is stored only on the Cloud Buckets because of high integration with GCP tools. Each step can be manual or done with an automated tool.

In Figure 6.6 steps from one machine learning job are presented. In the Figure 6.6, data is first cleaned with DataLab and python pandas library operation where raw data is loaded, and specified items are removed. After cleaning and altering, data is stored to Cloud Bucket. Cleaned data is used with the AI Platform training model and model is exported to Cloud Bucket. Hosting of the model is done with Cloud AI Platform resource that reads the model from Cloud Bucket.

Figure 6.6: Used pipeline for machine learning

For testing and of analytical work, Cloud DataLab was used. Hosted Jypeter notebooks offer a distributed environment for developing analytical and machine learning solutions in GCP. All analytical work was first tested and developed in Cloud DataLab and then moved to services where the operation is optimal, e.g. transferring data cleaning query from Cloud DataLab to BigQuery query. Architecture gives a possibility to implement globally distributed machine learning and analytical work (R_7, R_{10}, and R_{11}).

6.2.4 Cases

For case 1 (Section 5.1) which described PID-controller, and local feature that generates data where PID-values can be calculated. In described architecture, this functionality was tested with Cloud DataLab environment. The presented local feature did run with 24 intervals which generated data and described parameters were calculated manually in Cloud DataLab. For safety reasons, the parameters were manually reviewed by the operator and then inputted manually to HMI. HMI parameters are PLC variables, and when changed, new values are automatically sent to the cloud, where new parameter performance can be reviewed. Each manual step of the process can be transferred to automatic operation if

needed. To enforce outcome, an algorithm like presented in a solution for case 1 did not work, and more sophisticated Machine Learning or analytical method has to be implemented.

For case 2 (Section 5.2) which described modeling correlation between frequency controlled pump and a flowmeter. Needed parameters for the case are flowmeter value and motor frequency. At the testing phase data cleaning and creating machine learning model were both generated at the Cloud Datalab, after testing machine learning model creation was transferred to AutoML Tables. Hosting of the model was done with AutoML Tables service. Presented solution had two goals which were to find gradual wearing out, and if anomaly happens, e.g. line got plugged. These goals were achieved, but generated model or using case-specific devices will work only with trained devices. Next step is to create a more general model that will work with multiple different pumps and flowmeters, but machine learning method will get more complicated, and not discussed in this master's thesis.

Case 3 (Section 5.3) described anomaly detection for high-frequency belt breaking. Like previous cases, SLS algorithm was tasted in Cloud DataLab because it works for streaming and batch anomaly detection. The algorithm found anomalies when anomalies were inserted to data manually. The algorithm could be run with Cloud Dataflow in the cloud, but if the goal is sub-second reaction time, then execution of the model has to be transferred to the local hub, or generate a faster route to the cloud, e.g. own Cloud Pub/Sub topic without data pipeline. The anomaly itself is rare, and it was decided that only when a real anomaly is gathered for the model, it can start to work and stop motors.

6.3 Discussion

Given architecture is one of many ways to build the system in GCP. The main goal was to emphasise how to build a system where each component, e.g. writing data, is designed with low coupling and where components are hosted services. Each component in the architecture is a hosted service and scalable, so hosting this architecture for one or hundred underlying systems is possible. Reading and writing to or from the cloud are separate components and scale independently.

Generally speaking, the goal was to develop a system where the local source can write different messages with low or high frequency and small message duplication or loss. And give users globally distributed access to data and machine learning or analytical tools. Security was not extensively discussed in this master's thesis, and most of the security policies rely on GCP IAM settings. Some of the components are in use as presented, e.g. Cloud Functions in Appendix C, and few components with additional mod-

ifications because gained information and know-how. Presented goals and requirements were achieved, and the next goal is to improve created models and make them part of the underlying system.

There are multiple components to improve, but most important cases are where data should be stored, and improve machine learning and analytic data work. It was decided that data was streamed to BigQuery because it offers data warehousing and scalability for querying data. BigQuery does not provide the right solution if items have to be edited or updated, because rows in BigQuery do not have a unique identifier. There is going to be a need for a database where device information is updated, and both Cloud Spanner and Bigtable were tested, and at some point, one of them will be chosen to write data. For machine learning and analytical work Cloud DataLab was a key component for testing and developing models, but the scaling of that component is harder, e.g. parsing data for one pump and flowmeter is easy manually but how to do it for a hundred different devices. The Solution is to create more general models that work with multiple devices. Cleaning data and job orchestration were not extensively discussed. Still, it is possible with Cloud Composer, which in its core is Apache Airflow, the tool was only briefly tested and not taken apart in this master's thesis

7 Conclusions

The goal for this master's thesis was to develop a solution in the cloud where a local entity can send data, and on the other hand, reduce the complexity of sharing of data when developing new models. The critical point was to build a solution to a new underlying system that is developed with a partnership of Honkajoki Oy, and keep the design general as the solution offers groundwork to other systems. And not go in details how each component works, e.g. aspect how local software components are containerized.

The question of why the system was built is hard to answer. From designing perspective of a new type of plant, the system was used to store data from a new system and use this data as a reference point when possible new plants are built. From an analytical perspective, it gives easy to use historical reference point of what has happened. The solution gives limitless machine learning possibilities to control the underlying system and to develop more general models. When the master's thesis was done, there was no easy to use and flexible solution that could offer all the mentioned requirements.

Personally, I see this bit differently because the built solution has helped a lot and reduced manual work that is related to operating the underlying system. If the underlying system is commercially viable, new systems will have this solution included because it increases drastically process control and monitoring. In my mind, most of the future systems will have an external control system or analytical layer, where some operational work is transferred.

Cloud platform software is developing rapidly, and new components and features are added with monthly phase, e.g. Google Cloud Platform was announced in 2008 [36]. This rapidly evolving system will make some solutions presented in this master's thesis seem dated even in the near future. The naming of resources in GCP will also change, which can create some confusion when referring to recourses, e.g. during master's thesis was done Machine Learning Engine was transferred under AI Platform name.

What can be done better is to have two important cases, how data is stored and machine learning related. When data throughput increases and more expensive database products are an option, resources as Bigtable will be part of the solution. Next step for machine learning and analytic work is to gain more data from the underlying system, and generate more general models that can be part of the operating of the underlying system. In my mind, machine learning will create additional business value for the whole project.

References

[1] G. C. Platform, *Spotify chooses google cloud platform to power data infrastructure*. [Online]. Available: https://cloud.google.com/blog/products/gcp/spotify-chooses-google-cloud-platform-to-power-data-infrastructure (visited on Jan. 12, 2018).

[2] D. L. Meeker, *Essential rendering all about the animal by-products industry*.

[3] S. T. Krishnan and J. U. Gonzalez, *Building Your Next Big Thing with Google Cloud Platform: A Guide for Developers and Enterprise Architects*, 1st. Berkely, CA, USA: Apress, 2015, ISBN: 9781484210055.

[4] G. C. Platform, *Load balancing and scaling*. [Online]. Available: https://cloud.google.com/compute/docs/load-balancing-and-autoscaling (visited on Jan. 6, 2018).

[5] C. McAnlis, *Removing the need for caching servers, with gcp's load balancers*. [Online]. Available: https://medium.com/@duhroach/removing-the-need-for-caching-servers-with-gcps-load-balancers-ae516497c7fb (visited on Jan. 6, 2018).

[6] G. C. Platform, *Cloud sql*. [Online]. Available: https://cloud.google.com/sql/ (visited on Nov. 20, 2017).

[7] ——, *Cloud sql*. [Online]. Available: https://cloud.google.com/sql/ (visited on Nov. 20, 2017).

[8] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, ser. OSDI '06, Seattle, WA: USENIX Association, 2006, pp. 15–15. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267308.1267323.

[9] A. HBASE, *Load balancing and scaling*. [Online]. Available: https://hbase.apache.org/ (visited on Apr. 10, 2018).

[10] G. C. Platform, *Overview of cloud bigtable*. [Online]. Available: https://cloud.google.com/bigtable/docs (visited on Jan. 10, 2018).

[11] ——, *Cloud datastore overview*. [Online]. Available: https://cloud.google.com/datastore/docs/concepts/overview (visited on Jan. 10, 2018).

[12] D. F. Bacon, N. Bales, N. Bruno, B. F. Cooper, A. Dickinson, A. Fikes, C. Fraser, A. Gubarev, M. Joshi, E. Kogan, A. Lloyd, S. Melnik, R. Rao, D. Shue, C. Taylor, M. van der Holst, and D. Woodford, "Spanner: Becoming a sql system," in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD '17, Chicago, Illinois, USA: ACM, 2017, pp. 331–343, ISBN: 978-

1-4503-4197-4. DOI: 10.1145/3035918.3056103. [Online]. Available: http://doi.acm.org/10.1145/3035918.3056103.

[13] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford, "Spanner: Google’s globally distributed database," *ACM Trans. Comput. Syst.*, vol. 31, no. 3, 8:1–8:22, Aug. 2013, ISSN: 0734-2071. DOI: 10.1145/2491245. [Online]. Available: http://doi.acm.org/10.1145/2491245.

[14] G. C. Platform, *Cloud sql*. [Online]. Available: https://cloud.google.com/compute/docs/disks/ (visited on Nov. 20, 2017).

[15] ——, *Google compute engine documentation*. [Online]. Available: https://cloud.google.com/compute/docs/ (visited on Apr. 16, 2018).

[16] ——, *Cloud datastore overview*. [Online]. Available: https://cloud.google.com/appengine/docs/the-appengine-environments (visited on Dec. 10, 2017).

[17] ——, *Cloud datastore overview*. [Online]. Available: https://cloud.google.com/appengine/docs/flexible/go/an-overview-of-app-engine (visited on Dec. 10, 2017).

[18] ——, *Kubernetes engine documentation*. [Online]. Available: https://cloud.google.com/kubernetes-engine/docs/ (visited on Apr. 16, 2018).

[19] ——, *Google cloud functions documentation*. [Online]. Available: https://cloud.google.com/functions/docs/ (visited on Apr. 10, 2018).

[20] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, "Dremel: Interactive analysis of web-scale datasets," in *Proc. of the 36th Int'l Conf on Very Large Data Bases*, 2010, pp. 330–339. [Online]. Available: http://www.vldb2010.org/accept.htm.

[21] G. C. Platform, *Bigquery under the hood*. [Online]. Available: https://cloud.google.com/blog/products/gcp/bigquery-under-the-hood (visited on Nov. 6, 2018).

[22] ——, *Google bigquery documentation*. [Online]. Available: https://cloud.google.com/bigquery/docs/ (visited on Nov. 6, 2018).

[23] A. Beam, *Apache beam: An advanced unified programming model*. [Online]. Available: https://cloud.google.com/compute/docs/ (visited on Nov. 4, 2018).

[24] A. Beam, *Apache beam documentation*. [Online]. Available: https://beam.apache.org/documentation/ (visited on Nov. 4, 2018).

[25] G. C. Platform, *Choosing a vm machine type*. [Online]. Available: https://cloud.google.com/datalab/docs/how-to/machine-type (visited on Dec. 20, 2017).

[26] ——, *What is google cloud pub/sub*. [Online]. Available: https://cloud.google.com/pubsub/docs/overview (visited on Dec. 20, 2017).

[27] ——, *Cloud machine learning engine documentation*. [Online]. Available: https://cloud.google.com/ml-engine/docs/tensorflow/ (visited on Nov. 6, 2018).

[28] A. T. documentation, *Spotify chooses google cloud platform to power data infrastructure*. [Online]. Available: https://cloud.google.com/automl-tables/docs/ (visited on Jan. 12, 2020).

[29] A. G. plc, *Intouch hmi*. [Online]. Available: https://sw.aveva.com/monitor-and-control/hmi-supervisory-and-control/intouch-hmi-standard-edition.

[30] E. M. AG, *Radar measurement time-of-flight micropilot fmr10*. [Online]. Available: https://www.fi.endress.com/en/field-instruments-overview/level-measurement/Radar-Micropilot-FMR10.

[31] T. H. KARL J. ÅSTRÖM, *PID Controllers: Theory, Design, and Tuning*, 2st. ISA: The Instrumentation, Systems, and Automation Society, 1995, ISBN: 978-1556175169.

[32] V. Doren, "Relay method automates pid loop tuning," vol. 56, Sep. 2009.

[33] K. Tran, *Streaming least squares algorithm for time series anomaly detection*, https://github.com/MSRDL/TSA, 2015.

[34] RabbitMQ, *Rabbitmq home page*. [Online]. Available: https://www.rabbitmq.com/ (visited on Jan. 12, 2019).

[35] FoundationDB, *Foundationdb home page*. [Online]. Available: https://www.foundationdb.org/ (visited on Jan. 12, 2019).

[36] Wikipedia, *Google cloud platform*. [Online]. Available: https://en.wikipedia.org/wiki/Google_Cloud_Platform (visited on Jan. 12, 2020).

Appendices

A Scikit-learn example

```
1  import pandas as pd
2  import datetime
3  import subprocess
4  import os
5  import sys
6
7  from sklearn.externals import joblib
8  from sklearn.linear_model import LinearRegression
9
10 # Google Cloud storage buckets gs://..
11 output_dir = ""
12 data_dir = ""
13
14 data_filename = "train_data.csv"
15 target_filename = "train_target.csv"
16
17 data_full = "{}/{}".format(data_dir, data_filename)
18 target_full = "{}/{}".format(data_dir, target_filename)
19
20 subprocess.check_call(['gsutil',
21                        'cp',
22                        data_full,
23                        data_filename],
24                        stderr=sys.stdout)
25
26 subprocess.check_call(['gsutil',
27                        'cp',
28                        target_full,
29                        target_filename],
30                        stderr=sys.stdout)
31
32 data = pd.read_csv(data_filename).values
33 target = pd.read_csv(target_filename).values
34
35 target = target.reshape((target.size,))
36
37 lin = LinearRegression()
38 lin.fit(data, target)
39
40 model = 'model.joblib'
```

```
joblib.dump(lin, model)

model_path = "{}/{}/{}".format(
    output_dir, datetime.datetime.now().strftime(
        'example_%Y%m%d_%H%M%S'), model)

subprocess.check_call(['gsutil',
                       'cp',
                       model,
                       model_path],
                       stderr=sys.stdout)
```

B Local datapipeline for infrequent data

```
var toCloud chan []byte
var toLocal chan []byte

func WriteToCloud(c <-chan []byte) {
        for {
                item := <-c
                jep := gtopic.Publish(ctx, &pubsub.Message{Data: item})
                _, err := jep.Get(ctx)
                if err != nil {
                        // writing to cloud failed
                        log.Info("writing to cloud failed")
                        toLocal <- item
                }
        }
}

type Failure struct {
        Ts          string `json:"ts"`
        Position    string `json:"position"`
        Error_code  string `json:"error_code"`
        Entity      string `json:"entity"`
        System      string `json:"system"`
        Type        string `json:"type"`
}

var FailureHandler MQTT.MessageHandler =
        func(client MQTT.Client, msg MQTT.Message) {
                var failure Failure
                err := json.Unmarshal(msg.Payload(), &failure)
                if err != nil {
                        log.Info("Message unmarshalling failed")
```

```
32                  }
33                  // in this case time format that supports BigQuery
34                  m.Ts = time.Now().Format("2006-01-02 15:04:05.999")
35
36                  // additional parameters from environment variables
37                  m.Entity = ENTITY
38                  m.Type = MESSAGETYPE
39
40                  // additional parameters from MQTT topic
41                  m.System = strings.Split(msg.Topic(), "/")[1]
42
43                  out, err := json.Marshal(m)
44                  if err != nil {
45                          log.Info("Message marshalling failed")
46                  }
47
48                  toCloud <- out
49                  msg.Ack()
50  }
```

C Cloud Functions example

```
1   package testfunc
2
3   import (
4           "context"
5           "encoding/json"
6           "log"
7
8           "cloud.google.com/go/bigquery"
9   )
10
11  type Failure struct {
12          Ts          string `json:"ts"`
13          Position    string `json:"position"`
14          Error_code  string `json:"error_code"`
15          Entity      string `json:"entity"`
16          System      string `json:"system"`
17          Type        string `json:"type"`
18  }
19
20  type PubSubMessage struct {
21          Data []byte `json:"data"`
22  }
23
```

```
24 var client *bigquery.Client
25 var projectID = "projectID"
26
27 func init() {
28         var err error
29         client, err = bigquery.NewClient(context.Background(), projectID)
30         if err != nil {
31                 log.Fatal(err)
32         }
33 }
34
35 func FailureHandler(ctx context.Context, m PubSubMessage) error {
36         var items []Failure
37         err := json.Unmarshal(input, &items)
38         if err != nil {
39                 return err
40         }
41         inserter := client.Dataset("Dataset").Table("table").Inserter()
42
43         if err = inserter.Put(ctx, items); err != nil {
44                 if multiError, ok := err.(bigquery.PutMultiError); ok {
45                         for _, err1 := range multiError {
46                                 for _, err2 := range err1.Errors {
47                                         return err2
48                                 }
49                         }
50                 } else {
51                         return err
52                 }
53         }
54         return nil
55 }
56
57 // local run command
58 // gcloud functions deploy testfunc --entry-point FailureHandler \
59 // --runtime go111 --trigger-topic topicName
```