Tuomas Kiviranta

META CLOUD SERVICE

Bachelor's thesis

Bachelor of Engineering

Degree Programme in Information Technology

2022

South-Eastern Finland
University of Applied Sciences

South-Eastern Finland University of Applied Sciences

Author (authors)	Degree title	Time
Tuomas Kiviranta	Bachelor of Engineering	December 2022

Thesis title

Meta Cloud Service

19 pages
1 page of appendices

Commissioned by

Metatavu Oy

Supervisor

Timo Hynninen

Abstract

The thesis was concluded to comprehend the complexity of a multifactorial system that was provided for a customer of Metatavu. The thesis work was an important section for a larger project produced by them. The scientific goal for the thesis was to investigate the behavior of a cloud-based packet builder collaborating with a data pipeline while constantly collecting and monitoring the flowing information. The obtained knowledge was designed to be applied for the creation of a cloud system attached to the pipeline.

The theoretical section introduces technicalities extensively regarding the employed objects. They were specifically chosen for the project for their unique capabilities or benefits. The beginning will present an objective which is to familiarize cloud computing in addition to a generic serverless architecture. The two of them were connected in a cyclical manner in which one of them holds for the other's action before continuing, and vice versa. The final part was reserved for the supposed 'control center' that hosted the sole user interface. It managed the entire system with commands that onset the fundamental process.

The aftermath of the research work was studied ultimately. A few themes such as the conveniency of serverless over physical servers and CI/CD versus other development processes, were presented to justify the choices in the planning phase. It summarized the project, discussing whether or not it succeeded as predicted and particularly in what aspects. Furthermore, the profound research conclusion was discussed in further detail.

Keywords

cloud, pipeline, trigger, serverless, development

CONTENTS

1 INTRODUCTION

This bachelor's thesis is a part of a degree in South-Eastern University of Applied Sciences. It is intended to be utilized by a business in South Savo called Metatavu, which is a reasonably sized software company that offers a wide scale of corporate and consumer programming solutions. The majority of the customers originate from Finland. The company has served multiple local businesses and coincidentally a university that conducts this thesis. Knowing the fact that both companies have a history of academic relations combined with matching interests in the IT-field, gathered a found basis for Metatavu's hosting of the thesis project.

The work was initiated by one of Metatavu's clients who wished for a large-scale software development project. The project's groundwork had begun before the involvement of the thesis that was planned to be a part of the whole concept. The work conducted for them was designed to be versatile and flexible, having the ability to choose the work environment in addition to the equipment for the work. Any software licences and other subscription services are provided by the hosting company in order to support the student and facilitate their work. The official task was to implement the services of a cloud platform, which was decided to be provided by Google, to a larger system. That system would handle the controller and connections between cloud applications and data collectors. There was a need for a cloud platform with the ability to instigate a build-operation for an application package with a specific requisition, collect information from the passing data stream, and handle tasks automatically at a continuous phase.

The tasks of the project are designed to be completed with Google's cloud platform by first grasping its features and functionality, and then conducting a testing period with a replica of a smaller scale. The period would give testers an understanding of its limitations and places of an improvement. Once the proper modifications were made, it would be connected to the real project by changing only its configurations which does not necessitate a significant effort. Metatavu would then apply the project and the gathered research material to its business and internal use. The success of the project upon completion can be measured

by its functional performance and overall operating latency. Both of which are highly depended on Google's cloud infrastructure that cannot be governed by the users. However, users can optimize the developed cloud applications with an efficient programming style and high-speed internet connection.

2 CONTEMPLATION OF THEORY

It necessary to acquire the technical knowledge of required systems before the implementation, which is discussed later in the thesis in-depth. This part introduces the group of systems in a specific and detailed manner. It begins with the cloud services to give their overall perception alongside showing their ease of use and modularity. The next step will be confronting the structure on top which the whole project would be planted. Lastly, the management system that would be utilized for a controlment of the entire process. This complete arrangement is defined as a CI/CD (Continuous integration/Continuous delivery) service which is a serverless implementation that can be described as an interrupted development cycle that in most cases can modify and update the desired instance or instances accordingly without interference.

2.1 Google Cloud Platform

The multinational technology company Google offers a vast number of products, both in virtual and physical form. Its cloud platform was founded in 2008 which is slightly later than cloud solutions offered by other technology companies such as Amazon, presenting the first modern cloud service platform as early in 2006 (Miller 2016). Nevertheless, Google's own cloud services in today's times are employing the modern technology competitively amongst Amazon and Microsoft's cloud computing service Azure.

Cloud Build is one of Google's many cloud services which was chosen for this project for its beneficial key features. It is particularly focused on a rapid employment, automated functions, and strong security. The platform offers similar options as any other major cloud computing company. However, the only required services consist of a small part of the full scale. These services are

Cloud Build triggers, serverless functions, authentications, and monitoring. When combined, they form a secure and reactive deployment system.

2.1.1 Cloud Triggers

Google Cloud Build platforms offer varying services as previously mentioned, but perhaps the principal key feature in the platform is triggering. As the name suggests, triggers can initiate a set of functions when being activated. However, it requires a pre-set condition/s to be met before ‘triggering’ the effect. Typically, the conditions involve structural modifications in the use of a remote cloud repository.

The triggers are demanded to contain settings which define their intended behaviour. At a minimum, these would include an event, a source, and a configuration file. An event indicates the action that would ignite a trigger’s response. A source defines the cloud repository that is monitored by Google Cloud Build for events. The configuration file is purposed to command a build process of a project.

XML	JSON	YAML

<Servers>
<Server>
<name>Server1</name>
<owner>John</owner>
<created>123456</created>
<status>active</status>
</Server>
</Servers>
	{
	Servers: [
	{
	name: Server1,
	owner: John,
	created: 123456,
	status: active
	}
	]
	}
		Servers:
		- name: Server1
		owner: John
		created: 123456
		status: active

Figure 1. Differences between data formats (Deshpande 2019).

In Figure 1, there are the typical formats of configuration file. A trigger’s build-configurations are fundamentally stored in a YAML or JSON file. The general use case for them is information storing, precisely configuration information which remains unedited during a process.

It was decided that the project would construct its build-configuration file by using the YAML syntax. That was due to the fact it is a recommended and supported default form of Cloud Build. The file itself contains supposed steps for the trigger to comply. It executes the steps sequentially. The first step is for specifying a certain build function. It includes setting the builder, its arguments, and the environment. The builder is located in the cloud as a utility for users. The arguments vary from chosen builders, mainly involving the management of project files. The environment is a zone set in the cloud platform which represents an actual region, for example Eastern Europe. Thus, setting it properly can improve latencies noticeably. The practicality of triggers manifests itself in the best manner once being compared to non-cloud techniques. As discussed in the previous paragraph, the basic action plan is to clone a repository and build it when an event (trigger) is invoked. This process can be implemented by hand, but it can be time-consuming and monotonous. A one option for improvement is to include a third-party programming engine, which has the ability to compile a script along with the functioning code. Although that method is an autonomous solution, it lacks effortless scalability and configuration options.

Triggers can be created via Google Cloud platform, in addition to their own control line interface (gcloud CLI). It enables the option for a faster and more independent deployment without naturally occurring human errors and delays. The chosen method for producing triggers in this project is with the help of Google Cloud Functions, which will be introduced shortly. A function can execute a code with an appropriate authenticity to create a trigger with programmed behavioural instances. For example, when it will be invoked and what actions will then be instigated.

2.1.2 Cloud Functions

Google Cloud Functions is a serverless developer area in which Google Cloud manages the infrastructure around a coding platform to a remarkably high point. Its scope is to minimally concern a user beyond the code’s execution environment. Cloud Functions support multiple runtime languages that provides users with a choice of preference for their optimal developing experience.

Its typical purposes concern automated tasks and microservices, those possessing substandard needs for a processing power, storage space, and cache memory. These tasks would range from HTTP requests to simple user authentications. However, it is scalable in terms of memory usage and the number of individual instances created under heavy data traffic. The function is automatically executed once a user visits a certain website.

Google Cloud supports an application programming interface (API) for internal duties on the platform, for example in a case of creating a new trigger to the Cloud Build. Cloud functions can call the API with a POST request to a Google hosted website, thus initiating an automatic trigger construction to a specified user's cloud platform. Cloud Functions can itself request and respond to data either from a user, or from any platform with an ability to send POST data to a website, such as Apache Kafka. The ability to execute serverless functions is featured also in Cloud Run, that is a newer service from Google with other objectives in the perspective. Cloud Run has a broader scale of control for its computing platform. The restriction limits surpass Cloud Functions in deeper management operations concerning surrounding file system, and when comparing to the chosen methods of interaction with the build instance.

2.1.3 Permissions

Google Cloud platform handles permissions in the same matter as any modern physical server establishment. It is required in almost every case to ensure the proper confidentiality and integrity of the resources. Google's own method of controlling permissions is Identity and Access Management (IAM) protocols. It gives administrators the power of authorizing each person's rights for resources and conductible tasks. The most convenient aspect of the permission distribution system is the use of service accounts. The non-human accounts that are linked to a user's own Google account. Their design purpose is to be used when executing the platform's internal tasks, for which a cloud application could demand the access right for specific resources. Service account permissions are excessively limited by default in order to avoid hazardous conditions, for instance

in a case of a deletion of a cloud resource. Google IAM set's preconfigured permissions depending on the type of an account. For example, a user's own Google account which ends with gmail.com, is automatically granted the owner role that brings near all available authority to the platform. The number of roles is vast with the potential permission count that can reach to over 5000, which manual enabling and disabling can be overly time consuming. The roles are designed for generalizing the standard access rights according to a one position.

2.1.4 Data gathering

The information handling in cloud projects is an essential necessity which must be conducted accordingly to ensure project's monitoring and developing perceptions. The gathering of data is essentially effortless since its origin, path, and destination are known when the communications travel through the same cloud platform. In addition to Google's normal services there is a cloud service that can monitor activities, called Monitoring. The sole purpose of it is to collect metrics, usages, etc, the data that commonly is not perceivable by users.

The monitoring service in the cloud platform is automatically enabled by Google, so it does not require similar configurating as other Google cloud services. Cloud Monitoring provides data that is generated from them without a user's involvement. However, it can be configured specifically to monitor or record one component with its special characteristics. The monitoring supports alerts (incidents) by default which inform users if services emit abnormal behaviour or unexpected events. The incidents can be activated conditionally to cope with tougher risk management policies or specialised situations.

The gathered statistics can be set to follow a specified regime that is compliant with the set period in which the rate of service errors are intensively monitored. Google Cloud offers two concepts for this purpose. They are SLIs (Service-level indicators) and SLOs (Service-level objectives) which respectfully act as the measurement and indication of aimed service-level performance. SLIs utilize general and automatically collected metrics from cloud services to determine the ratio between different aspects, such as the percentage of successful requests

among all of requests. SLOs use the value of SLIs to define whether a desired performance objective has been met.

2.2 Apache Kafka

Kafka is an open-source data streaming platform developed by Apache Software Foundation in 2011. It's popularity and userbase started to grow exponentially in 2014 as Google trends and programming platforms ratings suggested (Fintan 2016). A couple of major involving companies include Netflix and LinkedIn. Being an open source, a great number of companies have benefitted from Kafka along its existence. Its key features propose the most favourable use cases to be in real-time services such as messaging and data analytics. The services with requirement for a rapid communication of widely divided data messages.

The architecture of Kafka is designed to streamline the communication highway between channels and partitions. The main process for distributing the information is to use producers, topics, and consumers that share a common channel. A one point can stream data on that channel which is read by only those who familiarise the specific topic.

Figure 2. Data delivery in Apache Kafka (Carter 2020).

As the Figure 2 demonstrates, the consumer is subscribed to a topic which is tagged to the data by the producer.

The operation of Kafka in the project is limited to dispatching messages for a cloud platform which in turn continues the automated process. The messages

from Kafka are delivered in a form of JavaScript Object Notation (JSON). Its sole purpose is to structurally represent the needed information onward.

2.2.1 Platform

Kafka does not possess a typical application interface or a graphical user interface (GUI), but rather a command-line-interface (CLI) as do numerous other conventional server-side applications alike. It's an event driven platform which means that the responses almost always require an input of real-time data before continuation. Apache Kafka then breaks the date down and distributes parts of it to different components i.e., Microservices.

The chosen hardware has to be considered when utilizing it for a such real-time streaming system as Kafka. Solely relying on hard disk drives (HDD) rather than solid state drives (SDD) affects the combined performance noticeably, but they aren't essential "due to Kafka's sequential disk I/O paradigm and Kafka writes to disk are asynchronous" (Akash, 2020). Thus, the core components need to favourably use other means of storing data besides HDDs. Another considerable hardware recommendation is to host a large amount of random-access-memory (RAM) which is used as a temporary data storage for the passing information. The recommended amount is 5-10 times the amount in a regular computer.

2.2.2 Supervision

Kafka offers various advantages over traditional methods of information processing and generation. Its components give the ability to create a scalable and durable system that can be closely monitored with the option to diagnose complications and performance issues. The supervision is conducted in the upper management level of the project with other features concerning performance or troubleshooting factors. A one of Kafka's APIs is focused on the management of its other resources. It is called the Admin API. Its purpose is to control and review Kafka objects that are integrated to the system itself. The API brings a configurable admin client to the system which gives a developer the ability to set SSL (Secure Socket Layer) variables such as key password, its logical location,

and certificates for it. According to the Apache Kafka documentation, these are all highly important factors on the base of secure communications for example in an authentication phase.

Since Kafka consists of multiple nodes, partitions, etc., there is a problem of managing their configuration data in a synchronized manner. That is why a controlling software called Zookeeper is beneficial for the project's interest. It interconnects singular clusters to share resources among them. In addition to this, Zookeeper offers an ability to effectively replace a weak node near its failure by fulfilling that node's duties. When that occurs Zookeeper creates a replica which will take over the responsibilities in order to disrupt the system as little as possible.

As Kafka swiftly transfers incoming and outgoing data typically in small collections there is a basis for a tool that can import larger amounts without disrupting the general streaming pattern or its rate of speed. A Connect tool is meant to be assessed if systems existing outside of Kafka's internal production circle have wanted data collections. Examples of these could be a website, database, or cloud server such as Google Cloud. Accessing information from an external source is possible through software level from a port that has been configured in the connector that matches the port of an outside destination. The data can flow in the form of JSON.

3 IMPLEMENTATION

The project's development journey was designed to start from a functioning system operating in a local (non-cloud) environment, which would be later deployed to the cloud in its entirety. The plan was estimated to optimize the time expenditure in a development phase, because cloud development would have appended unnecessary delays between code executions and modifications. A Google account would be provided by Metatavu to be used for Google Cloud Platform in which the entire project would be heavily depended. The account would also be the recipient for the monthly billings. In addition to a Google

account, it was necessary to also obtain a linked cloud repository e.g., GitHub which the cloud trigger would build upon activating.

The project's main purpose is to act as a cloud system that stays dormant until a HTTP request is sent to a public address where the system's entry point exists. The entry point initiates the creation/replacing of a trigger with the provided information. Cloud Functions is utilized for the programming code and other data files while Cloud Build handles storing of the triggers. Both cloud sections share a common service account that is specifically configured to be used for internal authentications on the cloud platform. Multiple authentication roles exist in a cloud platform to ease the labor of assigning permissions to different cloud actions. The service account requires an access comparable to a project owner for being able to create, update, and delete cloud resources.

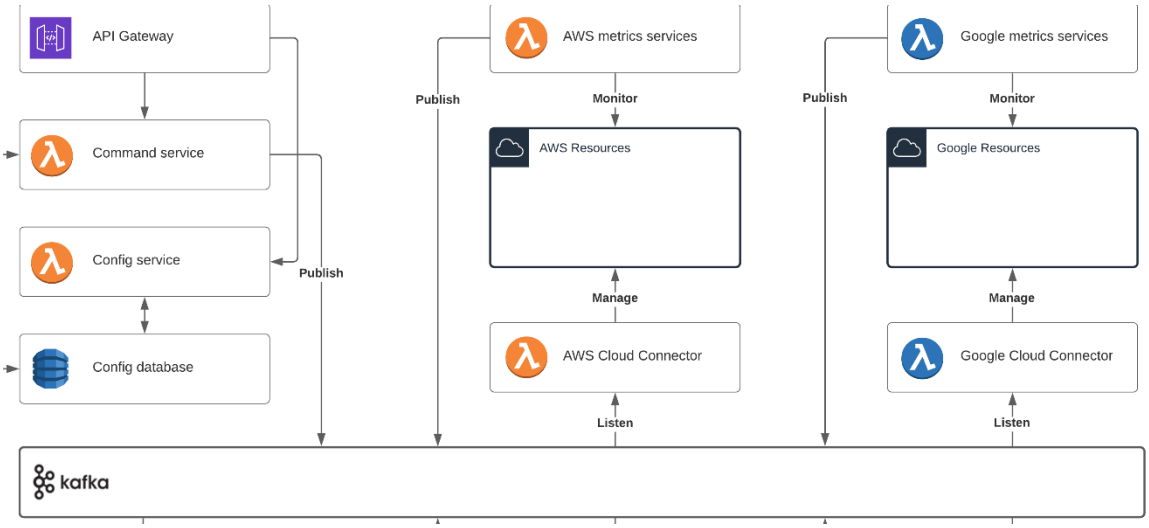

Figure 3. Visualized operating cycle (Metatavu 2022).

The system model provided by Metatavu shows Kafka's major involvement in the project. It connects the management and execution aspects without directly being linked to each other. As Figure 3 shows, the Google Cloud Connector is a web service hosted on a geographically closest Google web server that responds to invokes that are send by Kafka through one of its communication channels.

A free of charge application that was used in the testing period of the project is Postman. It is specifically designed for overall HTTP related development as it is

stated on their website, “Postman simplifies each step of the API lifecycle and streamlines collaboration”. Postman can be used as the replacement of Kafka for development purposes to send and receive requests sent to the system. The application supports the editing of HTTP header data which normally consists of fundamental information concerning the origin of the sender and sender’s device. A request would also have a body that would be used as the primary placement of the interchangeable data. In this case, the body contains multiple variables packed in a JSON format.

3.1 Cloud Platform

Google Cloud Platform can be operated with an existing Google account. The account can employ Google APIs (Application Programming Interface) such as Cloud Functions and Cloud Build by enabling them from platform settings. The account must present valid billing information to handle the compensation demanded by Google’s payment system. After any computing power is provided, the resource usage tracking will record it accordingly for Google’s own management purposes. A major benefit of Google Functions is its ability to function without a constantly working virtual machine that would consume resources far beyond of the demanded level. Functions does not require computing power in idle nor outside of code executions. In addition to them, the project consists of files hosted on the Google Cloud which is not a complimentary service.

Google Cloud Functions is utilized to host a globally accessible API that consists of multiple files and a configuration packet. The code in the project in written in JavaScript programming language which functions on Node.js 16 runtime (executable environment) offered by the cloud platform. The project imports Google’s own asset library for gaining the access to invokable functions. They are used in the creation, modification, or deletion of any cloud resources. The project uses asynchronous functions to secure that the code execution will halt until their lifecycle has been terminated. It is implemented since the latency of actions on the cloud platform cannot compensate for the speed of a programming

code's runtime. After an asynchronous function is finished, the code execution continues regularly.

```
{
    "namespace": "name","project": "Example-project-name",
    "platform": "platformio","type": "master","deployment": "deployment",
    "awsAccessKey":"1234", "awsSecretAccessKey":"4321"
}
```

Figure 4. An example body of an HTTP request.

As previously stated, the API is hosted on Google Cloud Functions. When it detects an incoming HTTP request it will validate the information in the request by examining the body section as is seen in the Figure 4. In addition, the body has an extra variable which states whether the existing trigger should be deleted.

```
const createTrigger = async (cloudBuild, type, project, triggerName, platformConfiguration) => {
  const options = {
    branchName: type,
    githubRepo: project,
    name: triggerName,
    steps: platformConfiguration.steps,
    substitutions: SubValues || platformConfiguration.substitutions,
    timeout: platformConfiguration.timeout,
    options: platformConfiguration.options
  };

  const trigger = await cloudBuild.findBuildTriggerByName(triggerName);
  if (trigger !== false) {
    if (DeleteTrigger) {
      return await cloudBuild.deleteTrigger(trigger);
    }
    console.log('Similar trigger already exists, it will be replaced');
    return await cloudBuild.replaceTrigger(trigger, options);
  }
  else {
    if (type == "pr") {
      return await cloudBuild.createPrBuildTrigger(options);
    } else {
      return await cloudBuild.createBranchBuildTrigger(options);
    }
  }
}
```

Figure 5. Trigger creation process.

Figure 5 demonstrates that once a sufficient amount of information is received, the API decides whether to create a new build trigger, update an existing one, or to delete it. An examination is conducted to ensure if a trigger containing the same information already exists. In an event where a similar trigger is not found, a new instance is created. Otherwise, the current will be updated with the provided information. The body can have an optional variable which confirms the deletion of a trigger. The HTTP body can distribute two unvalidated variables for the created Build Trigger which are “awsAccessKey” and “awsSecretAccessKey”. A trigger can access the variables in its YAML configuration file, for example in an event of data gathering from another cloud service where authentication credentials are necessary.

In order to authorize an API to conduct changes in the cloud platform an authentication must be validated. A service account key is used which can be acquired from the cloud platform. It is a JSON file containing information concerning the cloud project, the service account, and Google’s authentication procedures. The unencrypted file is located with the rest of the project on Cloud Functions. The security aspect of the method is not a key factor due to the fact that the cloud platform itself is locked behind a Google account. A breach of a user’s Google account is highly unlikely when considering the publicly available modern security policies, such as two-factor authentication.

3.2 Production

The project’s evolution is controlled and managed in a free of charge online cloud repository named GitHub. A command line interface (CLI) is operated to manipulate the actions of a project’s source code. The CLI must support GIT commands which are a part of an open-source version control application that investigates project files for modifications. The changes can be confirmed/saved for the upcoming ‘commit’ which is an individual patch that is forwarded to GitHub with a push command. The commits or “pushes” can be reviewed and separately confirmed to ensure that a code modification is legitimate. The online repository exists on a Metatavu owned account that is shared with a developer’s personal

account. A policy such as this protects the organization's other intellectual assets from any unauthorized access.

The generated Build Triggers do not activate builds without provocations. The default method for a one is a GitHub commit being singularly pushed to a repository. After a change have been noticed, Cloud Build will retrieve configured files from a development branch of a repository. The reason for variating branches is to have multiple versions of a same project present. Project changes are committed to a temporary location until a testing phase has been completed. After a trigger has activated, Cloud Build executes the commands in a trigger configuration file. The data to the file originates from a collection of steps assigned by Metatavu. These steps include commands in a JSON file ranging from repository management to cloud asset transformations.

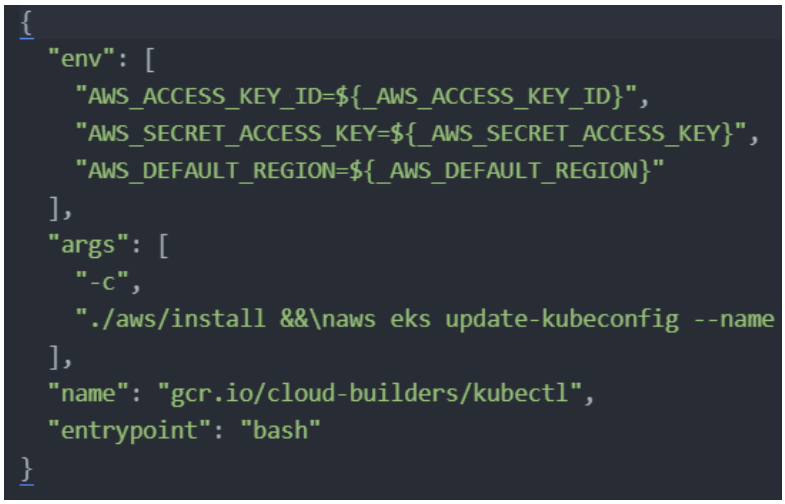

Figure 5. A set of commands in one Cloud Build file (Metatavu 2022).

The generated Build Triggers do not activate builds without provocations. The default method for a one is a GitHub commit being singularly pushed to a repository. After a change have been noticed, Cloud Build will retrieve configured files from a development branch of a repository. The reason for variating branches is to have multiple versions of a same project present. Project changes are committed to a temporary location until a testing phase has been completed. After a trigger has activated, Cloud Build executes the commands in a trigger configuration file. The data to the file originates from a collection of steps

assigned by Metatavu as seen in the Figure 5. These steps include commands in a JSON file ranging from repository management to cloud asset transformations.

The Google cloud platform offers by default a comprehensive scale of information that corresponds with resource usage. The data is not limited to a singular feature section on the platform, yet it includes all of them that were used in the thesis. The data representation is similar between Cloud Functions and Cloud Build, containing a provided activity timeline. Google Cloud Platform supports an alerting system for errors and warnings. They are especially convenient if the cloud development environment is not manually monitored or programmed to notify of such occasions. The cloud monitoring system can be set to dispatch an email to a provided address, log a detailed message to a file, or issue it on a Pub/Sub channel which Kafka would host.

4 CONCLUSION

The thesis questions the prospect of a Google Cloud Platform's involvement in a continuous development cycle as a connector between Apache Kafka and cloud resources from Google. Would it be technically possible to implement an automated structure that could communicate with a local data streaming platform such as Apache Kafka while being hosted on an online web server. When the essential access rights are provided to Google's services, GitHub repositories, and transmitted messages, the sole responsibility of the project is invested on the coding proportion. As the research stated, it is achievable to connect two virtually separated and independently operated entities together that form an elaborated production cycle of virtual objects. It also reflects on how the system gradually functions in the manageable and extensively monitored cloud environment.

The technologies discussed in the thesis are relatively matured inventions in the frame of rapid information technology development, the most recent of them being Google Cloud Functions that was released in 2017. The current trend for cloud systems such as this project started to gain popularity after the release. It can be stated that a matching system could be build or that the current project could be enhanced with newer/different methods to reduce operational latencies

and virtual file sizes. For example, changing the programming language and the structure of the code can have the capability to increase the overall performance. Yet in a system of this scale, the advantages would probably not yield any major improvements.

The research that was conducted for this project originated from previous studies as well as from material that were discovered during the development phase of the project. The new subjects were those which required further investigating of their logical functioning and applicability. Those topics regarded Google Cloud and Apache Kafka that were both independent and large entities. For being able to connect the two in a proper manner, insists a thorough knowledgebase. Lastly, a few protocols of the Google Cloud Build policies were unavoidably encountered since Google does not share the source code for their applications. Although a broad online community support can assist on such occasions.

REFERENCES

Akash, G. 2020. Kafka Hardware Requirements. Medium. Available at: https://medium.com/@akash.d.goel/kafka-hardware-requirements-9328886fe88f [Accessed 5 April 2022].

Carter, M. 2020. Apache Kafka Architecture: A Complete Guide. Instaclustr. Available at: https://www.instaclustr.com/blog/apache-kafka-architecture/ [Accessed 10 March 2022].

Deshpande, R. 2019. YAML basics in Kubernetes. IBM. Available at: https://developer.ibm.com/tutorials/yaml-basics-and-usage-in-kubernetes/ [Accessed 15 February 2022].

Fintan, R. 2016. The Rise and Rise of Apache Kafka. Redmonk. Available at: https://redmonk.com/fryan/2016/02/04/the-rise-and-rise-of-apache-kafka/ [Accessed 11 February 2022].

Kafka 3.1 Documentation. 2017. Kafka. Available at: https://kafka.apache.org/documentation/ [Accessed 9 May 2022].

Miller, R. 2016. How AWS came to be. TechCrunch. Available at: https://techcrunch.com/2016/07/02/andy-jassys-brief-history-of-the-genesis-of-aws/ [Accessed 4 February 2022].

Steven, J. 2018. What's the difference between agile, CI/CD, and DevOps?. Synopsys. Available at: https://www.synopsys.com/blogs/software-security/agile-cicd-devops-difference/ [Accessed 10 February 2022].

What is CI/CD?. 2018. Red Hat. Available at: https://www.redhat.com/en/topics/devops/what-is-ci-cd [Accessed 4 February 2022].

What is Postman. Postman Inc. Available at: https://www.postman.com/product/what-is-postman/ [Accessed 22 August 20].