Feature and Performance Based Comparative Study on Serverless Framework among AWS, GCP, Azure and Fission

by

Sabiha Nasrin
20141043
T I M Fahim Sahryer
20141044
Partha Pratim Mazumder
16301141

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
Brac University
June 2021

Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.

2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.

3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.

4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Sabiha

Sabiha Nasrin
20141043

Fahim Sahryer

T.I.M. Fahim Sahryer
20141044

Partha Pratim Mazumder

Partha Pratim Mazumder
16301141

Approval

The thesis/project titled "Feature and Performance Based Comparative Study on Serverless Framework among AWS, GCP, Azure and Fission" submitted by

1. Sabiha Nasrin (20141043)
2. T.I.M. Fahim Sahryer (20141044)
3. Partha Pratim Mazumder (16301141)

Of Spring, 2021 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on June 06, 2021.

Examining Committee:

Supervisor:
(Member)

[illegible]

Jannatun Noor
Senior Lecturer
Department of Computer Science and Engineering
Brac University

Program Coordinator:
(Member)

Md. Golam Rabiul Alam
Associate Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

[illegible]

Sadia Hamid Kazi
Chairperson and Associate Professor
Department of Computer Science and Engineering
Brac University

Abstract

Cloud computing is one of the most flourishing technologies in today's internet based world and the upcoming trend for the future. It has roughly been a decade of the development on this field. The first initiative had been taken by the world class companies like Amazon, IBM, Microsoft, Google etc. And in a matter of time the average companies, brands and enterprises has adopted this revolutionary technology by building their own cloud platform. One of the very recent technology of cloud is the serverless technology. Here the server side management is conducted by the cloud providers and the clients' only need to deploy their codes once in the server. This system prevents a great deal of unessential consumption of power and is a Pay-as-you-go service. This technology has added a great impact on the software and application development. But the major obstacle to this development field is that there are not enough documentation on how the big companies provide this facility and how their architecture is build. However many developers can not decide suitable platform for their required application. Also documentation on privately built serverless architecture is not available. This can be done through massive documentation on comparison and evaluation on the existing cloud platforms on which the companies can run their own serverless applications and embrace the hybrid world. Therefore, our research purpose is to focus on the serverless architecture of the existing platforms. A comparative study with necessary measures can be effective and efficient to use for further serverless implementation. So, we and others can follow our research for understanding the technical complexity. We will emphasize on the actual characteristics of serverless throughout our work. Furthermore, our goal is to come up with effective analogy on how different serverless platforms behaves in different Use cases.

Keywords: Serverless, Docker, Virtual Machine, AWS Lambda, GCP, Azure, Kubernetes, Fission, Comparison analysis, Load balancing, elasticity.

Dedication

We want to dedicate our thesis work to our parents, without whom, it wouldn't have been possible to come to this verge of graduation neither could we achieve anything. We also want to dedicate our work to the doctors and people who have fought tremendously in this time of pandemic to safe the life of others.

Acknowledgement

Firstly, all praise to the Great Allah for whom our thesis have been completed without any major interruption.
Secondly, to our Advisor Jannatun Noor miss for her kind support and advice in our work. She helped us whenever we needed help.
And finally to our parents and families without their throughout sup-port it may not be possible. With their kind support and prayer we are now on the verge of our graduation.

Table of Contents

List of Figures

List of Tables

Nomenclature

The next list describes several symbols & abbreviation that will be later used within the body of the document

AWS	Amazon AWS Cloud Service
Azure	Microsoft Azure
FaaS	Function as a Service
GAE	Google App Engine
GCE	Google Compute Engine
GCF	Google Cloud Function
GCP	Google Cloud Platform
GCR	Google Cloud Run
GKE	Google Kubernetes Engine
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
SaaS	Software as a Service

Chapter 1

Introduction

1.1 Background Study

In the advancement of modern age we are now more driven in the data centric technology in our day to day activities. We prefer more of a predictive world and expect more accuracy from the technologies than ever before.The more we dive into a data centric technology the more we require space, compatibility of hardware and security for our data. This act of demand has given birth to a new era of technology, that is the Cloud. Storage, server, compatibility, network, database, intelligence, sophisticated economy, scalability putting all this together creates a Cloud platform. The more we are advancing towards web based applications and online on demand services, the more small, private and independent companies or organizations are entering into the techno industry. In order to suffice the on demand need of the audiences, the companies need to provide a spontaneous service through their platform. For small or medium companies, it is unreasonable to own a Cloud server. Vendors can rent cloud properties from simple storage to database, networking, computation, processing power for machine learning, artificial intelligence to natural language processing and everything that does not require any hardware or physical presence close to the server. From the client side, they only need to subscribe to a Cloud provider for renting access and using the instances provided by the cloud providers and paying a usage or subscription charge for the service.
With the growth of the cloud service industry, there has been a lot of variety and new opportunities provided by the leading cloud providers. But for the lack of guidance and documentations of the versatile cloud dimension, potential customers often fails to see the optimal option for their desired application. As a result, the metaphor of the Cloud service remains hidden behind closed doors. Hence, we tried to deliver a comparison based study on the most demanding Public cloud provider namely Amazon Web Service, Google Cloud Platform and Microsoft Azure. Additionally we introduced a new comparison with the open source Fission serverless platform to enhance the paradigm of the study [17]. According to our study, we also have designed a serverless architecture. This study will guide the potential users to help choosing the right Cloud provider for their work.

1.2 Problem Description

Today's Cloud platforms are providing numerous facilities and advancement in terms of compute, storage, mechanism, capabilities, pricing method, security system, data storage advancement and so on. The cloud computing features are upgrading and updating time to time, for which there are not much documentation on the updated facilities. Moreover, the advancement is too fast that potential users can not keep track with the growing industry [27]. For which, they are not able to understand which will be the better solution to choose. Even in cases, there is one provider who might be well efficient in Machine Learning computation but not well enough in back-end API management. There are such cases, which varies over providers to providers.
Most of the recent studies are focused on the theoretical comparison of the leading cloud servers but very few actually showed any practical implementation. For which, there is no way to understand which platform will perform better in what environment and data. Again, no research conducted any implementation on any kind of open source cloud platform. Therefore opportunities for development in open source platforms are not highlighted in most cases.

1.3 Aim of the study

Our work is to help developers to choose among the popular three cloud providers based on feature analysis. Our work will also guide them to choose between given two serverless use cases that we have showed in the practical experiment. This study also opens up new development opportunity for the new developers in an open source platform. Our serverless architecture can provide a powerful and event driven efficiency in terms of serverless computing.

1.4 Motivation

Cloud Platform is now a days the new fashion in techno industry. Companies are diving in this new wave of technology. But unfortunately, there are not much research done in this field. Big and small companies around the world have already shifted to this new platform. Our work will open up opportunities for new developers to choose their work space among the three chosen platforms and will also help them make new development.
Most of the new developers like us are mostly unaware of the technological advancement of serverless cloud computing. But as we studied through the features, components and applications that can be deployed through serverless, we have come to the conclusion that, serverless can be the next revolution in the technological industry. The young developers like ourselves should not lag behind from this revolution.

1.5 Contribution

In brief, following are our contribution with this paper:

1. We have provided a fair feature based comparison study between top three popular cloud provider.

2. We made comparison with this new open source platform Fission which has never been mentioned in any research work before. This will open a new paradigm in the serverless cloud computing.

3. We have designed a new serverless architecture prototype which is a futuristic architecture and will add a new illustration in serverless cloud computing.

1.6 Limitation of the study

We have conducted our work on only three public cloud provides and practically implemented only on two of them. Although there are some other public cloud providers who are very popular in the industry. There are a bunch of use cases for serverless, but we could only execute for two. That is because, the other services are paid ones and this is not favorable for students like us. As we are not provided any fund by any organizations.
For the world wide pandemic situation, we had to work remotely. We did not get access of our lab and server, where we needed a lot of resource. For which we could not implement some features of our design. Specially the proposed model of serverless, which needed access to our university server.

1.7 Overview of the Thesis

Our thesis work consists of seven chapters. Chapter One is for Introduction of our thesis work, Chapter Two is the literature review of previous work and related research, Chapter Thee consists of the necessary theoretical framework and their details, Chapter Four outlines the methodologies that we used for comparison analysis, Chapter Five show all the experimental setups that we needed to conduct out experiment, Chapter Fix consists of all the results and discussion of our experiment and finally Chapter Seven concludes our thesis work with the future opportunities that this study hold.

Chapter 2

Related Work

The study [22] presents the newest envision of cloud computing, the Serverless architecture based on Amazon Lambda and discussed a working principle of serverless computing. In this paper cloud computing is presented as an on demand compute power consumption process for both the web applications and IT resources. Serverless has the ability in enhancing the dispatch on provisioning resources while minimizing the client size management. This paper noted the increment of hyper scale data centers from 338 to 628 by 2021 which is about 53 percent of all existing data centers these days. Hyper scaling the data centers for faster, more reliable and automated systems is the current goal. Serverless computing has been signified as a pool of virtual machines which bestows resources such as hardware, software, OS, runtime environment. It provides the function as a service (FaaS), where developers can trigger their events through API and record actions. The author emphasized on lighgh waited applications which are basically stateless applications that are trigger based. The paper discussed some use cases that are now shifting towards serverless for better performance. Such as multimedia processing which are now using event trigger computing for doing watermarking, transcoding, data fetch etc. The types of files that are processed here are text file, image and video. Then comes the broadcasting of live video, which harmonizes audio and video streams received from one end to another. The computation is done through function and delivered through Content Delivery Network (CDN). Thirdly the Iot data, received form various digital devices, processed through event based computer architecture and finally carried to other nodes of data storages. Lastly, the shared delivery process, an event based proclamation advertises notifications of the owner company to the nearest outlet. One Lambda function appended with an AWS EC2 instance is triggered that creates a snapshot, proclamation of the success of the instance creation. Lastly they talked about the Deviceless Edge Computing, that is the non cloud leading serverless system.

The research [25] presented an architecture in their paper that conducts an experiment that identifies the problems of stateful microservices of serverless in terms of kubernetes and proposed a solution that will use a state controller to enrich kubernetes for the state replication and automation of the application towards a healthy node or entity. This operation undergoes secondary label management. Their proposed solution has increased the recovery time upto 55-99%. Here, microservices basically defines the serverless functions while kubernetes is a local host platform and an orchestration tool for deploying private serverless functions through containers. Stateless microservices are often considered the most suitable for deployment and replication as they do not preserve state and can be interchanged among the pods or instances. But this is not the case for stateful microservices as the state can not be elastrated without synchronization. This research work has illustrated their survey on finding the limit of resource in kubernetes for stateful microservice and the effect of state controller for stateful microservice. They have deployed a stateful microservice with stateful set controller and deployment controller. They have achieved automatic redirection of applications towards healthy instances for stateful microservices with a significantly lesser amount of time and they have figured that the redirection of pods takes lesser time then restarting of a pod.

The author in his paper[1] evaluated AWS and Microsoft Azure, which are two of the most popular cloud systems. They concentrated their research on three key cloud features: storage, cloud type, and computing service supplied and they compared these features in depth. They presented their findings in a table containing all of the elements listed, as well as some user case studies and recommendations. They came to the conclusion that Amazon RDS is paid on a per-use basis. The deployment technique used in here is standard.

Author conducted a study in which they compared AWS, Microsoft Azure, and Rackspace [3]. They concentrated their comparison on the price and performance of these platforms. They showed cost-benefit analysis on small, medium, and large computations using memory and CPU. From their analysis they revealed that AWS had the superior cost plans in case of small and medium scale computing, while Microsoft is less costly in case of big scale computing. Rackspace was determined to be more expensive across all three situations. They considered AWS as more cost effective solution.

Author Aljamal conducted a survey on the high performance computing (HPC) of four prominent cloud platforms, including Microsoft Azure, Amazon, Google, and Oracle [12]. To assist consumers of HPC apps, they analyzed the services on offer as well as the comparative benefits of each of the selected cloud platforms. Cloud-based high-performance computing (HPC) is one of most popular study topic. It's a low-cost solution for performing high performance computing and eliminates the need for a dedicated cluster. They included feature like migration tools, developer tools, and management tools for comparison. This research revealed that while not all cloud platforms may meet consumers' requirements, the majority of them offer a wide range of services and amenities to maintain and entice new and existing consumers. According to popularity and market share, Microsoft and Microsoft has more popularity and user rather than any other provider.

Author Dutta [19] investigated the top three most popular cloud platforms, which were AWS, Microsoft, and Google. Their research concentrated on the storage, computation, and management features that these platforms offered. Before picking the finest Cloud Service Provider, a firm should define its particular requirements with context to IaaS, SaaS & PaaS. According to their findings, even if AWS has a larger market share, it would be incorrect to believe it provides the better services because

Microsoft and Google are increasing their facilities.

In this study author Dordevic compared two popular cloud computing systems, Microsoft Azure and Amazon Web Services based on the computing performance and service of both platforms. Ubuntu Linux Server 14.04 LTS 64-bit on same virtual machine is used for performing the test based on researcher's specified parameters which included price per hour, CPU cores, disk space, and RAM [2]. The findings of their compression indicated that both systems are relatively equivalent in terms of performance. If only CPU and storage space are considered, Microsoft Azure gives tiny advantage over AWS.

Serverless technology is quite a new practice in the modern era and the development of the work has not yet spread with the developers. And the topic we have chosen to work with is a completely new serverless platform. First off, containers were introduced to intact the necessary requirement to run an application and bundle them up for easy portability among domains. Then to give it a form in the cloud sector and to overcome its shortcomings, developers introduced serverless architecture which took part in the wider range of business administrations. It has been only about five years in this field and there is very little research so far recorded. Therefore, we lack resources while doing research in this field and more importantly container based serverless systems have not been introduced widely before. Therefore our research is comparatively new. Yet we have analyzed some documents in the serverless and container basics to understand more details of the architecture.

Serverless computing is rising as a new and incredible paradigm for the arrangement of cloud applications because of the recent shift of enterprise application architectures to containers and micro services. More precisely, the State of the Cloud 2018 report shows a very impressive growth rate of serverless, picking up to 667 percent. This means that the expanding consideration that serverless architecture has gathered in industry expos, meetups, web journals, and the developer community. Containerization evolved resource-sharing further through OS-level virtualization [14]. Containerization is a standard unit of software that packages up code and all dependencies. Containers hold all the components that configure the actual code, the runtime engine and dependencies packaged in a software unit which is called docker and this is necessary to run a specific software program and a minimal subset of an OS.

Virtualization and Containerization introduced provisioning speeds and efficiencies but further improvements are limited by the management of underlying infrastructural components, in this case, servers. Reference [6] defines serverless computing as "a software architecture where an application is decomposed into 'triggers' (events) and 'actions' (functions), and there is a platform that provides a seamless hosting and execution environment." Serverless processing visualizes a model of figuring whereby successfully all assets are pooled including hardware, operating systems and runtime environments.In serverless architecture simplified programing models are used by engineers for building cloud applications that abstract away most. It brings down the expense of sending cloud code by charging for execution time as opposed to asset assignment and a stage for quickly conveying little bits of cloud code that reacts to events.

From the point of view of a cloud supplier, there is an extra chance to control the whole advancement stack, diminish operational expenses by efficient improvement and the board of cloud assets by using serverless architecture. This architecture supports the utilization of extra administrations in their environment, and lowers the exertion required to create and oversee cloud-scale applications.

Characteristics: There are a number of characteristics of serverless architecture:

- Cost: Typically, in serverless functions users are allowed to pay for the time and resources used by them. How much every time you are utilizing that much amount of money you have to pay that is why people thought that serverless architecture was the best fit. This is a key benefit of serverless architecture.
- Programming languages: Serverless services support a so many programming languages including Javascript, Java, Python, Go, C# and Swift. Most platforms support more than one programming language.
- Deployment: Developers just need to provide a file with the function source code.
- Security and accounting: This is multi-occupant and must hide the execution of capacities among clients. Serverless function give detail about what user have to pay
- Performance and limits: In serverless architecture limits can be set. Limit depends on the runtime resource requirements of serverless code, maximum memory and available CPU resources .Limits and performance are increased or decreased depending on the user's need.

Benefits: Application developers need to think about the cost of codes in Serverless architecture. User and service provider both are benefited by the serverless paradigm .Developers no need to think about managing servers and VMs and no need to pay more. For providers it's easy to get control over the software stack by using a stateless programming model .Provider can transparently deliver security patches and optimize the platform.

In another paper of microservice based architecture, the authors tried to make a new architecture which assures the availability of state-full microservices and identifies problems. Here they mentioned that micro-service availability is high in Stateless microservices but in State-full it is different [22]. That is why they use Kubernetes with a State Controller which replicated and automatic service redirection to the healthy entries through the management of secondary labels. Here they also compared the Default configuration of Kubernetes with the other architectures for availability perspective such as OpenSAF. In the containerization concept the micro-services mainly focuses on the IOT based systems which are Stateless, whereas the containers are isolated. That means the containers are separated from each other. So for this process it needs to deploy and manage by orchestration platform. Which handle the deployment, scaling and management of the applications. In this containerization concept availability is a non-functional and very important things. But in Docker it is not that much reliable as for cloud computing availability is a must needed thing. For this problem Kubernetes make it better by increasing the availability 55% to 99%. For this here OpenSAF middleware is used for implementation Availability Management Framework. They also observed that the node failure handling is improved by changing the Kubernetes configuration parameters to 22% better.

In the paper, there is analysis on a container based serverless architecture and how docker can be best suited for the system. Serverless features are nothing but wholesome of some IaaS (Infrastructure as a Service) and this can be provisioned through the virtual machines in the traditional cloud system. But for the easy portability and compatibility of the Docker images, Dockers are more suited for this application. Amazon also offers their CMP as a service and this makes it easier to explore [15]. The AWS Lambda also provisions the trigger functions executed in the runtime through the container like lambdas depending on the requests. The AWS Lambda process is less complex processing and a large sum of parallelism is being used in the processing. But the execution of a general application is a little difficult in this case. But with the image processing and Docker Hub support the difficulty can be minimized in the serverless platform. The cost reduction and response time of AWS Lambda has been recorded to be 70% better than the other approaches in the serverless and by introducing the high throughput computing programming model the parallel paradigm can be emerged to a larger scale. Docker image can bring this new paradigm.

Chapter 3

Theoretical Framework

3.1 Basics of Cloud Computing

Our field of research is based on cloud computing. Thus, we present the study of Cloud Computing from the very scratch.

3.1.1 Definition

Cloud is a kind of infrastructure containing heavy high end servers for storage which requires a heavy cooling system. Also for operating the inner system and hardware maintenance, it requires experienced developers and engineers in both hardware and software. Leading cloud vendors in present time are Amazon web service (AWS), Google cloud, Microsoft Azure, IBM and the rest large organizations have their own cloud server such as facebook, youtube, tweeter etc. Cloud platforms have become so popular in our daily life that people in general don't even know when they are using it and for how long they are spending time on it. Our regular activities for instance email, dropbox and drive, social activity, online applications starting from grocery to share market, business and every other possible things are all created based on some kind of cloud platform. With this advancement, the small startups and personal web applications become more handy. More developers are motivated to work on new business ideas and design new forms of service applications. The cloud platform therefore was getting more popular until it reached a new prototype. This has become so handy for the advancement of some big cloud companies to lease their servers and services to the other seeking companies. Cloud is an open source paradigm which is available for anyone who wants to build their own private or public cloud and use it. In this way a company can ensure more security and reduce any third-party involvement in their sensitive data and the process has become more secure.

3.1.2 Cloud Services

The traditional cloud services falls upon two categories of cloud services: Infrastructure as a service (IaaS), Software as a service (SaaS), Platform as a service (PaaS). After many developments in cloud service, there came this Function as a service (FaaS) which basically is the base service provided in the serverless architecture. This is the upgraded version of Platform as a service and provides a whole new dimension to cloud computing. Here are the cloud services described below:

Infrastructure as a service (IaaS) : The most basic type of cloud service is infrastructure as a service. This is the idea of virtualized compute space where clients need not think about hardware resources as the cloud provider will provide a virtual work environment which will act as a physical machine. Companies or organizations only need to subscribe for the number of Virtual cloud workspaces with their specification and thus the cost is reduced to minimum. Security, scalability, reliability, database management all these features are managed by the cloud providers.

Software as a service (SaaS): This is another type of cloud service that provides access to software that is used in a wider range but needs to be maintained centrally or some that need a lot of access over time. Cloud providers offer this kind of applications using privilege with a subscription charge or pay as you go basis. This helps the organizations and specially business by no provisioning of data synchronization or software maintenance.

Platform as a service (PaaS): In this category of service, both infrastructure and software service are provided to the client, that is a working environment is provided. Clients can manipulate it by making their own applications and deploying it by using these resources. It provides a platform where clients can both enjoy the cloud services and also work their way out. Load balancing and scalability are automated, so the end users need not think about that.

Function as a service (FaaS): FaaS provides services like PaaS but instead of a wider work space, it gives a trigger based service which adds a whole new layer in the PaaS. It provides an segregated workspace where developers are unaware of the computing platform and everything except for the code. This infrastructure is built upon containers in most cases. The code or application is deployed through forms of functions which are mostly stateless and type of the functions are microservice based applications. The block events are triggered into the system and are only activated when the function triggers. The service only charges for the amount of resources used and the time of execution.

3.1.3 Cloud Deployment type

Based on the purpose of the operation, type and range of application, security patch required for the application and many other features like this, Cloud computing is divided into four major types of service. All of these have different features and are defined to achieve different purposes from the system. The types of cloud deployment and their pros and cons are described below:

Public Cloud: Public cloud has a wide range spread across general purpose computing and it provides both Software as a Service (SaaS) and Infrastructure as a Service (IaaS). It is offered by Amazon, Google Cloud and Microsoft. For software development and test cases, public clouds are often used. After verification, those software are moved into other cloud platforms. The space availability in the public cloud is unlimited.

The billing process follows pay as you go theory which includes IT and development cost as well. As public cloud spreads a vast area in the internet world, the scalability is also very simple and quick. As it can merge with other available clouds to handle outbound traffic. Security patches are not as strong in public clouds than that of public or other cloud platforms. The system management and operating systems are all under control of the service provider. Therefore it is difficult to change in the default set ups and apply personalized systems, environment and Operating system. The secret credentials and datas are often shared with one or many third parties outside of the organization. This makes the security even more vulnerable.

Private Cloud: These are pre-purchased cloud platforms owned by a single organization. The availability scale is not as huge as private cloud but it is very strict and well maintained. Only authorized customers can use the private cloud. Apart from the authorized users, access is restricted for any one else. These clouds are usually dedicated to significant companies and the company can do whatever they want within the cloud.
Security is very strictly maintained in private clouds and secret credentials and data are not accessible without authentication. As there is no outsider involvement in this platform, companies can modify, change, share or move their data however they want to. They can arrange and customize their data and application of their choice. No security risk unlike public cloud. But as the association is limited within single company ownership, the owners have to carry all the costs for hardware and software and are responsible for both maintenance. And for scalability, private cloud is not as flexible as private cloud because not having the versatility. For load balancing or scaling, more servers and hardware needs to be added. Private clouds are protected with firewalls.

Hybrid Cloud: This is the mixture of both the public and private cloud. To facilitate customers with both opportunity and to minimize the lakings of private cloud, this hybrid platform was built. It has the scalability of public cloud and security advantages of private cloud together. Hybrid clouds are broadly being used by Data mining companies, who work with sophisticated and sensitive data. The credentials are securely kept in a private cloud and are delivered secretly to the private cloud when needed. Serverless functions run more like a hybrid cloud platform. It is cost effective and also firewall protected [11]. Has versatility in data management and offers a great deal and resources in data manipulation.

Community Cloud: This type of infrastructures are not so common as these are used by companies with similar infrastructural backgrounds. It is a type of private cloud with public cloud functions. The users preferences are similar about the security and management, as the users might reside in the same field or expertise. The security management is done privately and the functions run publicly but within the community and not outside that. Thus transection with public cloud and community cloud is easy.
The cost of community cloud is carried out by the companies and it is reasonable when companies share them. But when few collaborators share one community cloud, it brings security threads for the valuable datas. And also scalability can be a problem as the cloud is not so large to accommodate big change in the traffic.

3.2 Basics of Serverless Architecture

Serverless is a service provided by the cloud platforms, which is basically Function as a Service, the newest invention in the field of cloud computing. The whole service is described in sections below.

3.2.1 About Serverless

Serverless runtime as it is called was first introduced in 2014 with some prominent significance in cloud technology. The norm of serverless is not about reducing or eliminating servers rather it comprehends that you need to think of servers less. Unlike traditional technology, serverless does not require operating systems or servers to maintain, easy and efficient scalability, high availability with no specialized infrastructure, fault tolerance, no ideal capacity needs to be specified rather you pay for the period that your code runs in the system. For these features the use of serverless architecture has widened among the developers and mostly in the small startup software application developers. Usually the execution time is being calculated in milliseconds.

3.2.2 Definition

Serverless is an event driven architecture and with its extraordinary elasticity it can serve a high performance of computing. It helps developers to focus more on the development of the application and less on the server maintenance. One of the very famous serverless architectures is Amazon Lambda, first introduced in November 2014, which helps its clients to compute services without managing or provisioning any server side management. Most of the applications these days are developed in this platform as it is offering high performance, availability and very much cost reduction. But at the same time, for highly sensitive data, having a third-party involvement can make the security more vulnerable.
The concept of serverless architecture first arrived form the ideology of Infrastructure as a service (IaaS) but with some major changes to provide the best fit system which focuses on decreasing the cost and increasing efficiency. The model servers the purpose of deployment of code in the cloud by only charging for the time the code is executed rather than the code allocation memory and so. Also, the services offer an ecosystem that makes it easier for the development of the services as well as the scalability and operational cost. Serverless optimizes the operational cost by provisioning the resources and by giving the authority of fault tolerance, monitoring, scalability, and maintenance directly to the cloud provides, which makes it easier for the cloud providers to use the service without much hassle. Although the infrastructure is quite similar to that of the Platform as a service (PaaS) but here the developers get the privilege of using the prepackaged application and are not restricted to write arbitrary codes. This use of deploying functions in serverless is the ideology of Function as a service (FaaS). This adds a new paradigm in the serverless platform by writing microservices and some famous providers of the cloud also make those microservices available as open source [9]. Thus the developers only need to worry about the deployment and execution of the codes or services and provisioning the virtual machines or hardware. The developers also have control over the codes they deploy which must be in the stateless form but they need not worry about the

scalability of fault tolerance. The users will only be charged for the time being when the codes are executed in the server.

This paradigm of serverless has been evaluated form many theories and concepts from the past to its current state. There are similarities between Mobile Backend as a Service (MBaaS) and Parse Cloud Code of Facebook with the concept of serverless. Software as a service (SaaS) may have the functionality of executing server-side applications but it has limitations over the application domain. The codes in SaaS are invoked through API calls which have a similar feature like serverless.

3.2.3 Serverless Components

To architecture the serverless some constraints need to be present there, which are the cost-efficiency, auto scalability, and fault tolerance. The services need to be directed and redirected according to the workload. Events are stored in the event queues and based on the availability of the queue. Based on the capacity of the status, other events are triggered. The status of the queue invokes the arrival of the events by scheduling their execution, assigning, and managing the events and also retrieving the resources when the functions are done with execution. Also, the profound cloud providers provide the flexibility of using any language, because they create a basic structure for the functions and it provides a framework where any code is composed to a docker and that is invoked via an API and as long as the language supports this, it is good to do go. Some programming modes are such that it executes a dictionary in terms of an input dictionary which allows the platform to trigger events. There may not be any state between the executions of the functions, which increases the scalability. The framework logically groups the functions which save more time and make the system efficient while deploying and updating. While the function workload and control over operation and cost function defines the likely use of serverless.
There are always some advantages and drawbacks to the platform. The developer side advantage is that they no longer need to take the hassle of maintaining the VMs and containers, provisioning the servers, managing the platforms, etc. and only focus on the codes and services. This is all because the cloud developers are offering the basic computational building block of a distributed system as well as the security patch along with a stateless programming model. But there are also some disadvantages for both the client and cloud providers, which is sometimes the structure is too abstract for the client developers and that may not support various new versions and may fail to achieve the proper application mechanism for implementing their function which will meet the capacity of the platform.

3.3 Container & Virtual Machine

The concept of container and virtual machine was the pre-requisite of this work. Without the knowledge of this two, the research work on this field is impossible. As we have worked on docker container and VM in our pre-thesis (II) elaborately and in this paper, we have worked in advance level with these concepts. Therefore, we have given a brief idea about container and virtual machine in this section.

Container: Containers have been shown to be a viable lightweight technique for virtualizing programs, particularly for handling cloud-based applications. A docker container is a standard unit of software that is lightweight, independent, packages up code, files, system libraries, tools and all other required things to run desired application so that the application can be moved from one computing environment to another [29] . Without the use of virtual machines, containers rely on virtual isolation to install and run programs that access a shared operating system kernel.

Virtual Machine: When a virtual environment that functions similarly to a computer within a computer this is called a virtual machine (VM). Virtual machines (VMs) are made feasible by virtualization technology. It operates on a separate partition of the host computer, with its own CPU, RAM, operating system (such as Windows, Linux, or macOS), and other resources. The real machine is referred to as the host, while the virtual machines that operate on it are referred to as guests. Hypervisor is the name of software which manages this operation from the host to guest. Hypervisor also used to virtualize and distribute host resources.

3.4 Kubernetes

While visualizing a private serverless we have encountered Kubernetes, which is an orchestration tool for containers or microservices that enhances the scope of managing the stateless and stateless serverless applications. The platform of Kubernetes is open source and is an infrastructure like any other serverless platform. Kubernetes was developed by google which consists of the property of autoscaling, elasticity, auto restart, replication of clusters with a zero downtime.

- The advantage of using Kubernetes is that it gives an easy load balancing paradigme by distributing traffic.
- Display of cluster containers is done using IP address or DNS name.
- Automatically escalates to different kinds of storage systems as per need.
- It makes it easier to do automatic switches between actions like deleting containers, cloning, creating new, and shifting data and resources to a new cluster when an existing cluster is being damaged or destroyed.
- Designing container clusters with desired RAM and CPU is done with no hassle.
- Kubernetes provides a Secret field for storing sensitive information such as password, SSH key etc. in the configuration file without exposing it.

The architecture of Kubernetes in figure: 3.1 is built with some components. We give a brief description of those components.

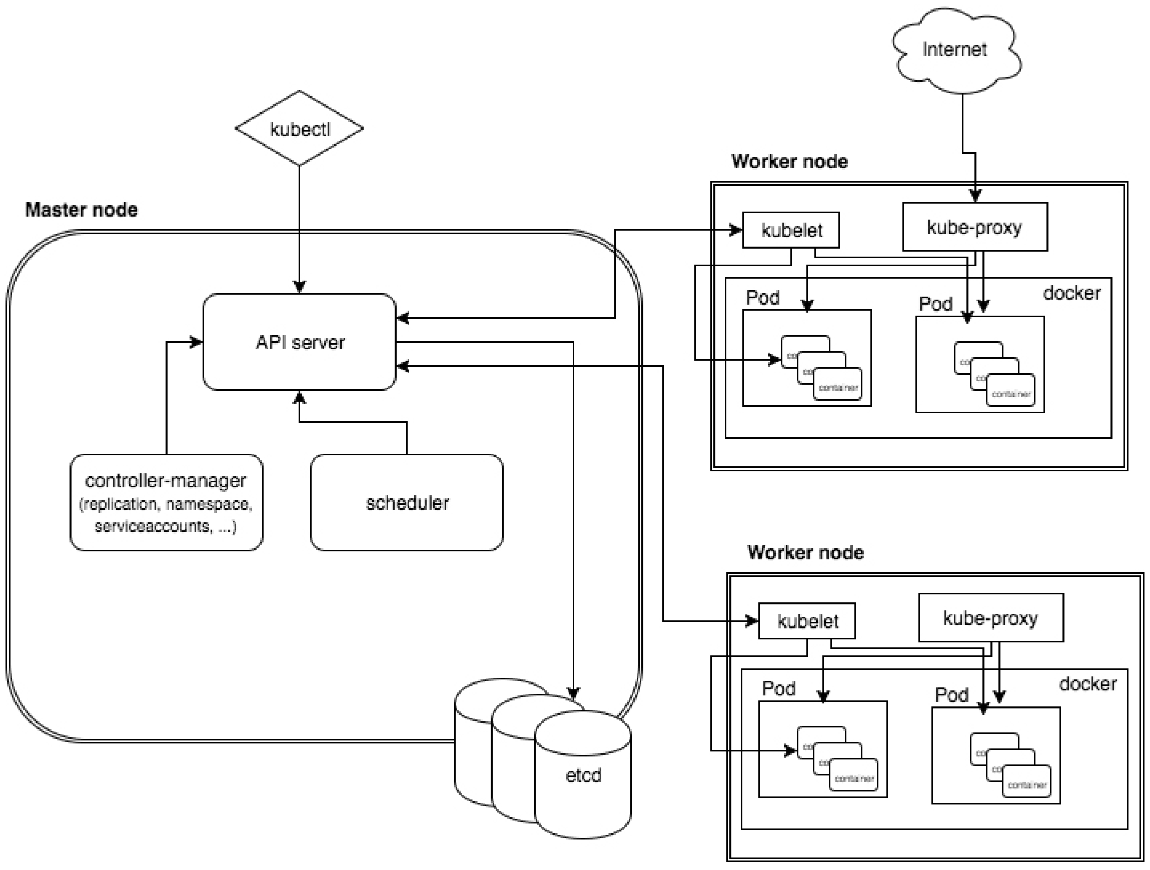


Figure 3.1: Structural Architecture of Kubernetes

Pods: A pod is one or more containers enclosed with instructions over a specific application and it's configuration such as resources, storage, variable components, status, network ID etc. Pods are the application container that basically carries out the function that needs to be deployed. Pods are no static entities, thus they may not contain physical ID and can be destroyed when the application processing is being done [24].

Service: Service serves as a set of pods that are logically set to perform as a gateway to the other pod instances so that they can communicate with the service without bothering the physical identity of the pod.

Volume: Volume preserves the data of the pods outside of the kubernetes cluster and it is an extension from the cluster. If any pod gets restarted or destroyed, the datas is being sent to the local machine or remote cluster that is the volume.

Namespace: It is a virtual grouping of the clusters which itself is another cluster. Based on different criterias and operation objectives, namespaces are being made with a number of specific pods. One namespace can not use resources from another namespace. They are anomalous to each other.

Deployment: This manages the replication of pods as per need. If there are less number of pods than required then it will create new by replication. And if the replicas are more than the required, then it will delete the extra.

The orchestration model is divided into two components : master node and slave node. This is to follow a client and server modeling. Master node is the controller node for managing the cluster while slave nodes are the working nodes. The master node components are described as follows:

Etcd: This is the cluster that stores information of all other clusters and nodes, such as namespace, changes in cluster, state of the clusters, pods created in each cluster etc. It also notifies the cluster on configuration change through API requests.

Kube-API server: This communicates with the Etcd cluster and does the central management of all the processes that requests for cluster relocation, pod modification, service discovery, change configuration etc. It monitors the data storage agreement of the pods that are being deployed.

Kube control manager: This synchronizes the controller management process with the credence of any primitive cloud service. All node management such as request query, set up route, load balancing, illustration or node transmission to new state are controlled by this component.

Kube scheduler: Scheduler allocates and schedules pods in different nodes by doing a comparison study on the operational precondition of the pods. This benefits in terms of maximum utilization of the resources. The scheduler is called every time there is a status change in the pod requirements and it needs to be relocated or if any new pod joins the cluster.

Following are the worker node components in brief:

Kubelet: This component takes care of the status of a pod and notifies the master node about it. It monitors if the pods and containers are stable and are operating in the described configuration.

Kube proxy: It works on host subnetting on each worker node so that the correct request reaches the correct pod from the vast network.

Kubectl: It is a command line tool for communicating with the Kube-API server and calls upon the master node through API call.

Helm: Helm is Kubernetes packet manager. Helm maintains the YAML mainfestes of kubernetes requests. It is equivalent to apt or yum. With Helm installed, we can install ready softwares like MySQL, MongoDB and use it in our application. Helm chart is like a packaged application for the clusters we create in kubernetes.

Minikube: One of the most common forms of Kubernetes for local hosts is Minikube. The features of Kubernetes are difficult for starter or reguler cloud developers as it requires resources, time consuming and has version variation. In order to smoothen this experience, minikube was developed which basically runs the same kubernetes cluster in a local host with more efficiency and less time complexity for a deployment is required. It helps developers to test their application first hand before deploying it globally. Like Kubernetes, minikube has the same components such as Pods, Service, Master node, Worker node and requires kubectl for cluster communication. We have worked with minikube in our thesis.

3.5 Fission

Fission is a framework of Kubernetes that created a platform for running serverless functions on a larger scale. Fission is open source and is suitable for running in any cloud platform be it Kubernetes, local host or any other public, private or hybrid cloud. It helps to manage serverless functions with any language and runtime and helps scaling the application. All the management of kubernetes are auto handled by Fission and pre-built containers are provided, so no need to worry about provisioning. It provides a function as a service for kubernetes in figure: 3.2.
The framework runs on top of kubernetes and facilitates the faster convergence of deployment of functions while also featuring all the advantages of serverless. The function calls are HTTP trigger based.

Following are some use cases of fission:
Event driven system: Fission can identify different events based on the characteristics of activity and thus can execute the required function.

Augmentation of web applications: Fission expands base web applications while keeping the original application intact. The deployment is automated.

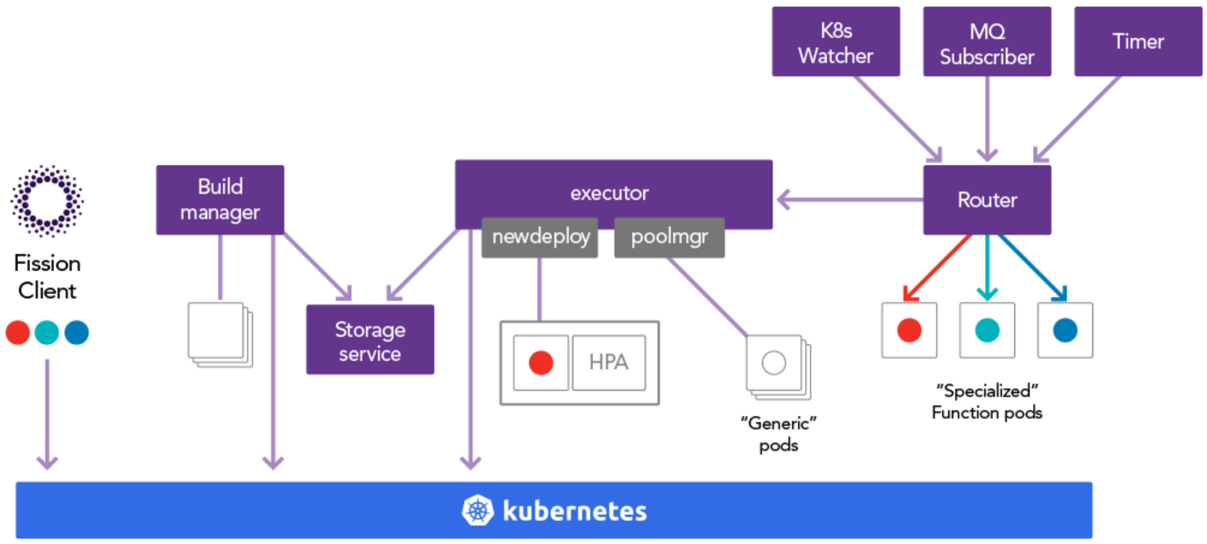


Figure 3.2: Structural Architecture of Fission

Backend management: With Fission platform, it is easier to build API backend for web applications for mobile or any operating system as there is no hassle of managing the kubernetes components. Only the functions need to be mapped in the HTTP route.

Edge data processing: With Fission it is easier to fetch and filter data. Normalizing data for various uses and storage is important which can be done easily through Fission.

Mainframe function modernize: The expansion of mainframe applications that may benefit the conversion of data or convert resource type so that the mainframe is upgraded, these all can be done easily through Fission.

3.6 Amazon AWS

Amazon AWS service is one of the leading cloud architectures available. As a serverless architecture AWS Lambda provides the opportunity to deploy stateless microservices without having to worry about managing or provisioning the code. The server side management for a microservice such as administration of server and resource, scalability, elasticity, operating system provisioning, language environment, security etc all services are managed by the cloud provider. Lambda provides a platform where clients need not to worry about the serverside and only need to focus on the part of the code. The functions will only be executed when they are triggered and will consume the state only for the time being when the execution is being proceeded. Furthermore, the billing of the service will only depend on the amount of time and resources that will be consumed during the execution, which is in other hand called Pay-as-you-go service.
Aws lambda supports multiple programming languages through a runtime environment. The supported languages and versions are Node.js10, 12, 14, Python 2.7, 3.6, 3.7, 3.8, Ruby 2.5, 2.7, Java 8, 11, Go 1.x, .Net Core2.1, 3.1 with specific runtime

environment for each language. When a version of any language reaches the end of life in any certain time while the security updates for that version are no longer available for the runtime, as per the runtime support policy of lambda, the runtime of that version is then deprecated. Which states, lambda will no longer provision nor give security support, update, technical support to that version while only the existing functions can be updated. And after about a month or so, lambda prohibits the use and creation of any function that uses the runtime. Only the existing function created earlier with that runtime can be used as events. For container image, defined as functions in Lambda can appoint any runtime environment and linux distribution while the container is being created. When a lambda function is being invoked, the system strives to use any previous runtime environment that had been used earlier for limiler kind of function. This reduces the execution cost by saving resources, connectivity with the database and execution time . In figure: 3.3 the architecture is shown.
Aws lambda is a serverless compute service which allows users to run application code without having to manage EC2 instances. AWS will thereby take the responsibility and administration of maintaining the EC2 instance for the client. AWS charges the computer power per 100 ms of use only for the compute power when a lambda function is in use in addition to the number of times any function runs.
There are four steps in deploying a lambda function. First the function code needs to be uploaded in the lambda or needs to be written within the code code editor provided by lambda. The language options are Node.js, Java, C#, Python, Go, PowerShall and Ruby. Next the lambda function needs to be configured to execute upon the right trigger from any supported event source [26]. After that, if the trigger initiates successfully, the lambda function will run and will use the compute power as mentioned. While this execution is in process, AWS will record the compute time in milliseconds and the quantity of lambda function running time which determines the cost of the service. AWS also shows a statistic of the Lambda function with respect to the running time, CPU usage, trigger, errors and success rate and other parameters in various scales.

The components of AWS lambda are as follows:
Lambda function: The lambda function is compiled of the code that the user wants to trigger. This allows the user to deploy their code through triggers without provisioning any server maintenance. It is simple to deploy functions in lambda which can be done through any language or container, that will be zipped and uploaded. And according to the resource allocation, lambda will automatically allocate the function in the compute space. Also the scaling and triggers will be done automatically.

EC2 instances: EC2 instance refers to the Elasticity feature of Amazon web service that provides scalability to AWS Lambda by resizing the compute capacity of the cloud. This whole process with computing resources and configuring capacity is provided for the users to use without having to worry about the rest. EC2 also offers a cloud platform with the availability of choosing the storage system, operating system, processor, network and other latest features. AWS provides 400 Gbps of ethernet networking with the availability of AMD, supports macOS, Intel and Arm-based processors with workload for graphics, powerful GPU instances for ML training, SAP, HPC and many more. About 400 EC2 instances are provided.

S3 Bucket: This virtual bucket is like a file container, where one can store files, remove files etc. which is going to be triggered by the lambda function. The type of files that needs input to be given for showing an output requires S3 bucket for them to carry this to the function. Such files are image and video files, database and so on. Once the file is uploaded, then the handler function triggers the function as it contains the details of the event. Then the lambda collects the input from the bucket and processes the file for an output.

Elastic Block Storage: EBS is the block storage area of the data that contains an independence and existence away from the instances. This is a virtual data storage that stores the data which can be fetched even after the instance is destroyed. This component is mainly used when there needs to be data storage, raw storage or file storage. There can be an API network connected to the EBS if necessary. Therefore it can be attached and used with an ongoing running lambda EC2 instance [28].
IAM role: IAM role provides an IAM identity that administers an instance with

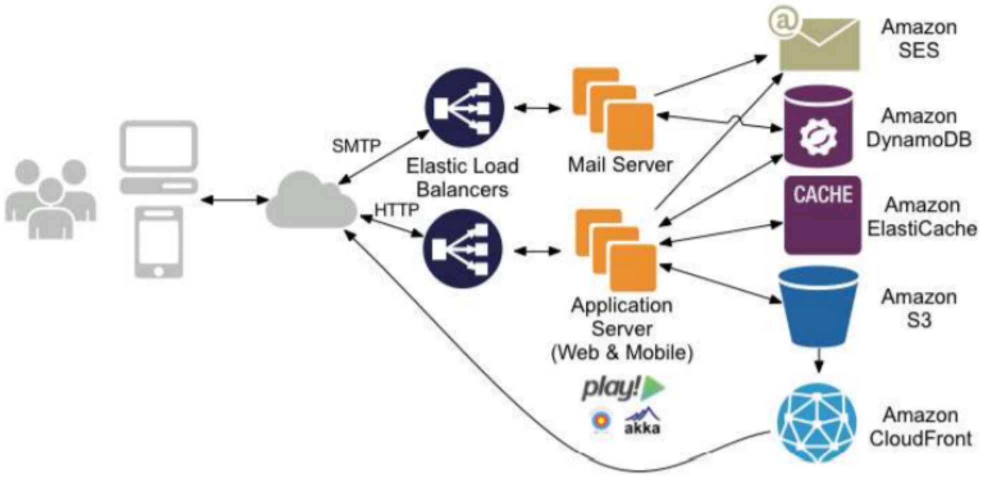


Figure 3.3: Structural Architecture of Amazon AWS

an AWS identity if attached to it. But instead of giving a permanent and unique security credential, it conducts with a temporary security credential for managing role sessions. Therefore, anyone with the role key can use it when needed and so the sessions are conducted in such. The roles can be created when there is a demand and can be designed to however users are allowed to use it. Users may need to access the resources and sharing keys within the instance will bring leakage to security. With the IAM role, one can create, manage, change and delete roles within the application.

Event source: Event sources are AWS lambda services that are used to invoke the lambda function when the event occurs. This is kind of a configuration mapping for the lambda function. Some regular event sources of AWS are: Amazon API gateway, DynamoDB streams, CloudWatch event, CloudWatch log, Event bridge, S3, SNS, Amazon Kinesis data streams, SQS and Amazon Step function.

Downstream resources: These are the resources that are required during the execution of the code.

Log streams: This helps in the troubleshooting and identity issues with the sequence of lambda events when triggered. Now we know that in AWS the event source is a service that produces the event that the lambda function responds to by invoking. The event sources can be of 2 types, poll or push based. Poll based services are the ones that look for specific events associated with the function and invoke them when it can find a match. The Push based services are the ones that push a model event that is required to invoke the lambda function. In order to link the event source with the lambda function, the source mapping configuration is used.

3.7 Google Cloud Platform

Google introduced Google cloud platform (GCP) in 2011. GCP designed and developed by Google To provide cloud service on the same world class infrastructure and that is used by google for its product. Google is a tech giant which has most advanced computer technology . GCP is becoming more popular to developers for its infrastructure that is highly scalable and dependable .Developer can easily build, test, deploy, and monitor application by using these infrastructure in google cloud platform. Google provide highly efficient server energy efficient data center, industry level efficiency. Google Cloud gives unlimited access to user of data from any computer with a web browser. These hardware advancement helps Google to reduce the cost on operation and which is beneficial for customer to save the cost. Customer has to pay bill for computational processes depending on the amount of time spent computing by the customer. The architecture is shown in figure: 3.4. This is the main reason of being less expensive provider rather than other cloud providers for GCP , so aBeside hardware advancement Google has added some cloud services . These services include Compute Engine, App engine, Container Engine, Kubernetes engine, Cloud Storage, Data store, BigQuery, Google API, etc..
Google Compute Engine (GCE): Compute Engine is an infrastructure as-a-service (IaaS) product [5].Developer can launce scalable and high performance virtual machine by using it. Also developer can create and run specialized architecture having feature like load balancing and auto scaling. Developer use it to deploy a Docker container host and deploy containers.

Google App Engine (GAE): App Engine is mainly a platform as a service product. Developer can easily build and run web-scale, autoscaling applications by using google framework and a comprehensive collection of libraries . So without the need for costly database acquisition and maintenance since the database is maintained by Google. That's why App engine is becoming developer friendly and also targeted to developer [20]. The objective of GAE development is to boost the online presence by allowing several users create apps for the web. It does not charge anything to get started, charges are made based on the use of storage and bandwidth by a user at an affordable price range.

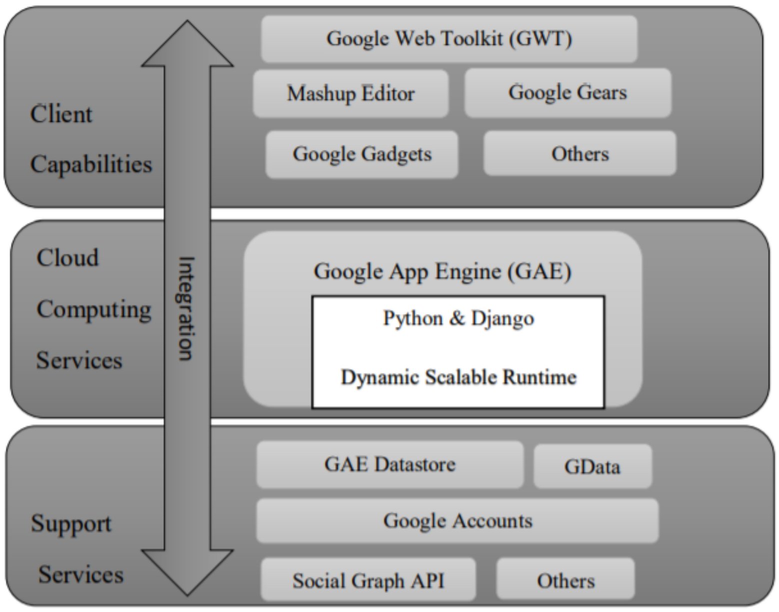


Figure 3.4: Structural Architecture of Google App Engine

Google Kubernetes Engine (GKE): Kubernetes is the industry's most popular and most advanced container orchestration platform which is developed by google .Google Kubernetes Engine (GKE) provides a managed environment for deploying, managing, and scaling your containerized applications using Google infrastructure. The GKE environment consists of multiple machines (specifically, Compute Engine instances) grouped together to form a cluster [30]. Developers uses GKE for the management of Container Orchestrator which helps to develop, manage and scaling of micro service based containerized application .It's a fully automated container orchestration which deploys , manages application and monitor the status of deployed application and cluster with features like node auto-scaling, upgrading, repairing by using the Kubernetes APIs and commands. Google Kubernetes Engine in figure: 3.5 consists of a cluster that has at least one master and multiple worker machines known as nodes. The orchestration of the Kubernetes system runs with the help of master and node machines.

Google Cloud Run (GCR): Google Cloud Run is a stateless container runtime service which helps user to run and execute highly scalable containerized applications . Its serverless, flexible, and straightforward. Auto scalling is another feature of clou run . P. To deploy our containerized applications to Cloud Run, all we need to do is point Cloud Run to our Docker image and Cloud Run will pull the image and run it When more resources are required, Cloud Run can automatically scale the execution of application. GCR is user friendly because its simple and easy to deploy stateless application. Cloud run is serverless so it don't need any management effort .Which is One of the most powerful characteristics of cloud run. No language barrier and pricing model are key features of Cloud run. Cloud run allow any language and it also offer pay per use for service. By using Cloud run vDeveloper can develop and deploy any public or private microservices.

Google Cloud Function (GCF): Cloud function is a function as a service. GCF

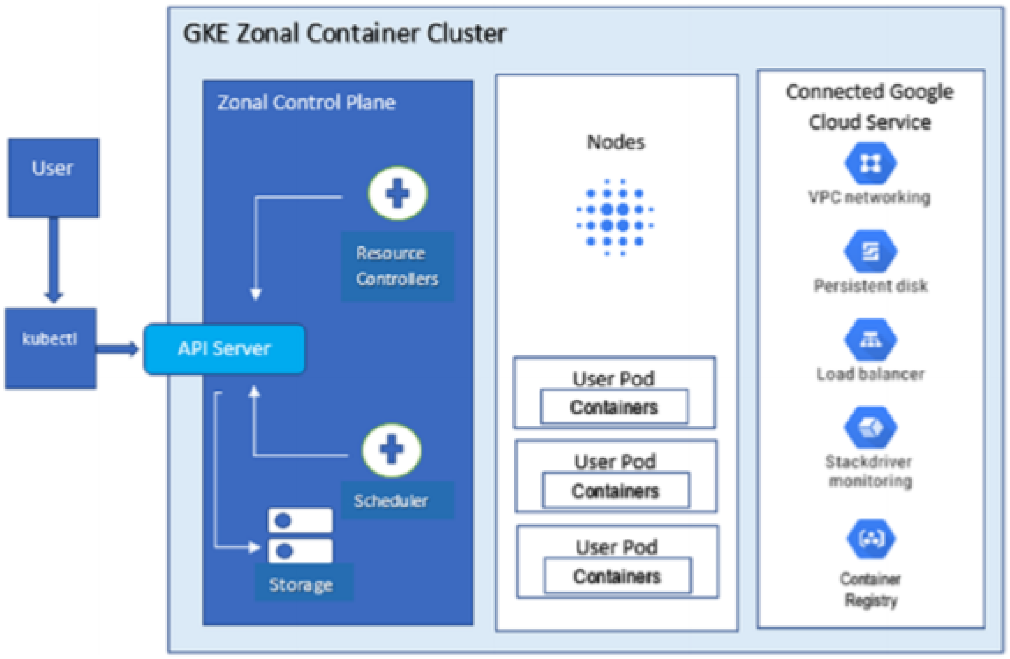


Figure 3.5: Structural Architecture of GKE

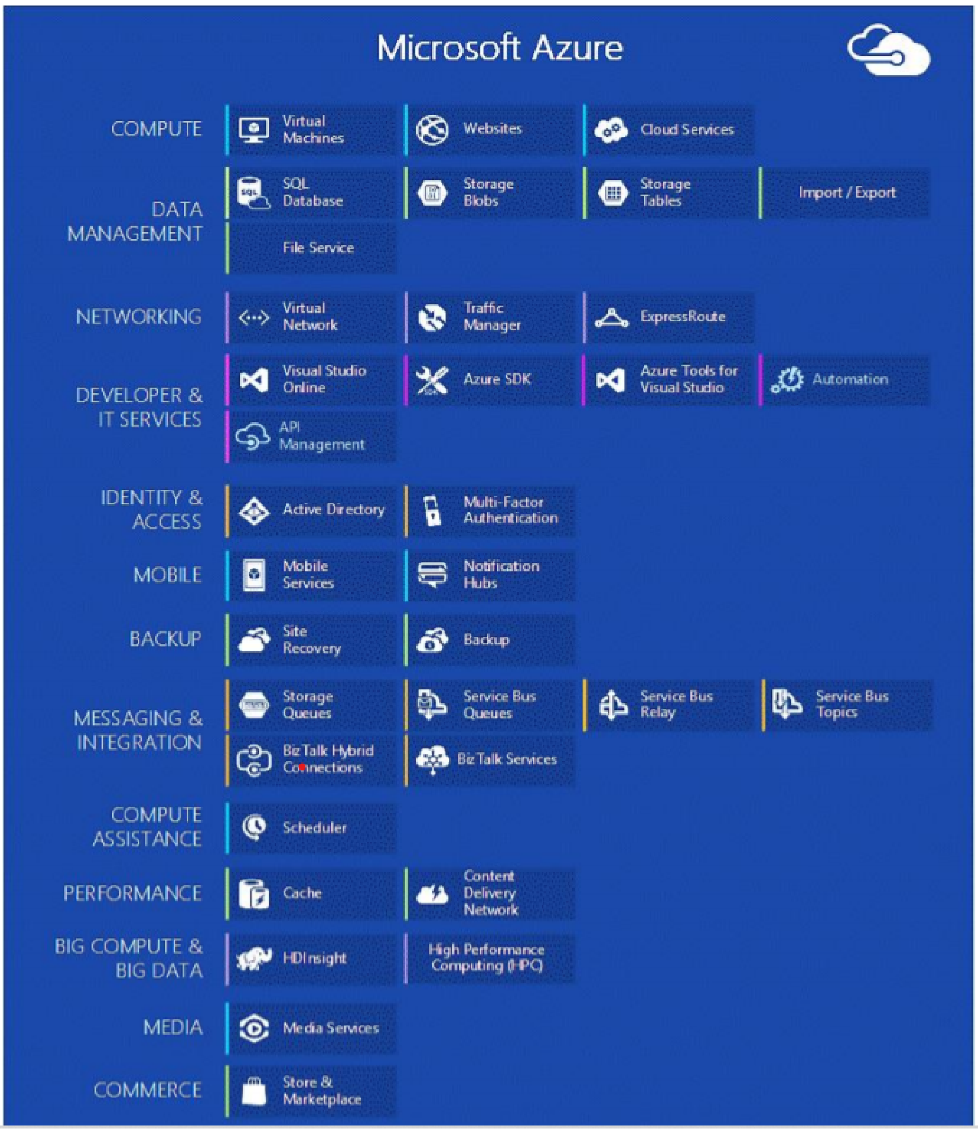
allows developer to write code in their preferred language (JavaScript, Python, and Go are supported) and deploy it in the cloud. By using GCF, Creating infrastructure as code and developing event-driven serverless cloud platforms is very much easy for its user. GCF helps developer to deploy their code with zero server management. There are no servers to manage, upgrade, or provision.Scale automatically dependent on the load. GCF has a developer friendly Environment which makes developer experience simple and their efficiency better. It's a pay as you go service so user only billed for function execution time. Hybrid and multi-cloud systems require key networking features which is one of the key feature of GCF [30].

Google APIs: Applications can consume both Cloud Platform product APIs (for example Google Storage) and Google products APIs (for example Google Maps). This book includes an example of using the Translate API to translate content among 90 pairs of human languages.

3.8 Microsoft Azure

Microsoft azure is one of the industry leading cloud provider. Microsoft launched azure in 2010 to provide cloud service. Azure provide all kinds of tools and platforms to developer for developing, operating, and executing web applications. High performance, flexibility, and low service cost are main feature of Azure. Microsoft Windows Azure is built on Windows Server and allowed to use all identical service, software and feature of platform.

Compute: This feature allows the development and implementation of applications and services on the Windows Azure platform. By using Windows Server foundation,



Microsoft Azure

COMPUTE	Virtual Machines	Websites	Cloud Services	
DATA MANAGEMENT	SQL Database	Storage Blobs	Storage Tables	Import / Export
	File Service			
NETWORKING	Virtual Network	Traffic Manager	ExpressRoute	
DEVELOPER & IT SERVICES	Visual Studio Online	Azure SDK	Azure Tools for Visual Studio	Automation
	API Management			
IDENTITY & ACCESS	Active Directory	Multi-Factor Authentication		
MOBILE	Mobile Services	Notification Hubs		
BACKUP	Site Recovery	Backup		
MESSAGING & INTEGRATION	Storage Queues	Service Bus Queues	Service Bus Relay	Service Bus Topics
	BizTalk Hybrid Connections	BizTalk Services		
COMPUTE ASSISTANCE	Scheduler			
PERFORMANCE	Cache	Content Delivery Network		
BIG COMPUTE & BIG DATA	HDInsight	High Performance Computing (HPC)		
MEDIA	Media Services			
COMMERCE	Store & Marketplace			

Figure 3.6: Components of Microsoft Azure

Azure computing service develop and executes application . Developers can utilize Visual Studio for executing application and technologies including PHP,ASP.NET are also available [16]. Web roles is used for implementing web-based programs, the Worker Role is intended to allow user to execute a wide range of code. and virtual machines for migrating programs from Windows servers to Azure shown in figure: 3.6.

Storage: To store any amounts of data for an extended period of time is allowed by the Azure Storage. Blobs, table and queues are the three form of storage. Blobs(Binary Large Object) are often used For storing the large amount of data which are not organized and binary items. Tables enable programs to operate in fine-grained order. Tables enable programs to operate in an orderly fashion. Queues enable web-based applications to communicate with user-implemented code by interacting between web role and worker role.

SQL Azure: It provides storage capabilities equivalent to Amazon S3. It allows relational queries to be performed against data in storage that can be organized, semi-organized, or disorganized, enabling users to connect in a variety of ways such as ADO.NET, PHP, and Open Database Connectivity [20].

FABRIC CONTROLLER: Fabric controller works as kernel of the Azure operating system. By provisioning, storing, delivering, monitoring, it manages servers, VMs. Fabric controller also coordinates the development of Microsoft Windows Azure application.

Chapter 4

Methodology

4.1 Goal of Analysis

We have selected some features from the analysis of different cloud computing and their complexity. Some of the features were from the facts that we faced upon doing practical work in the selected framework. And some of them were extracted from the analysis that we made of the cloud features. All of them affect the whole paradigm of a framework in terms of different technological perspectives. Our study is not about finding the best and worst of the platforms, instead, we are trying to find an analogy of different features to understand what kind of application will be suited in which platform. For differences in application and environment, different platforms of cloud will act differently. That is why we wanted to bring all three of them under a fair scale and compare the features thereby. We have done this through theoretical analysis of the features and practical implementation of the cases. Our goal of this paper is to regulate a comparison study among the Cloud service providers to have a deeper knowledge of those paradigms. This comparison will further help us with development of serverless platforms on a private scale. We have reached on the following:

- Theoretical comparison of the existing Cloud platforms in terms of serverless infrastructure, it's components and performance. Those are AWS Lambda, Google Cloud Provider (GCP) and Microsoft Azure.
- Implementation of different cloud services and its complexity measurement.
- Implementation of some serverless use cases in AWS Lambda, GCP and their performance comparison.
- Implementing the same use cases in Fission and recording the differences.

We have used the following platforms for hour experiment. Cloud platforms for feature comparison: Amazon AWS, Google Cloud Platform (GCP), Microsoft Azure. Cloud Platforms for Practical experiment: Amazon AWS, Google Cloud Platform (GCP), Fission.

4.2 Selected comparative cloud features

In this section, we have proposed the following features of cloud computing which features a cloud platform and can define the performance of the working complexity. Following are the features:

1. **Cloud Service:** Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS) are the main three service models of cloud computing. Which service mode is provided by the selected cloud platform will be shown by this feature. Some cloud platforms offer their users all three service models.

2. **Cloud Storage:** Storing users data is very important for any cloud provider. Users need easy access to their data from anywhere on any device. It plays a significant role in choosing any cloud provider.

3. **Security factor:** Security measurement of different cloud platforms provides security to the client's private credentials and data security within the cluster. Security instances of a cloud platform, also regulates the data traffic, controls the role and follows the event description.

4. **Language variation:** All programming languages are not supported in every cloud platform. This will depict the programming languages supported by cloud platforms. From many different programming languages which may be useful to developers in any platform.

5. **Virtualization Technique:** Virtual machine is virtualization of a computer system which will provide functionality like a computer in a computer. Base3d on computer architecture, virtual machines allow two or more operating systems on a single machine. It is a commonly used idea in cloud computing to save energy and power on physical hardware [23].

6. **Implementation complexity:** Different cloud platforms work differently in terms of implementation of the functions. Also specific clouds act differently in terms of different languages.

7. **User Interface:** The variety of interfaces that varies for developers and end users for different applications are noticeable for different cloud platforms. The variation can come from database, web application, API etc.

8. **Container Orchestration:** Container Orchestration is an important factor for any cloud provider. Such as Kubernetes is a container orchestration platform designed by google. It is used for developing and managing containerized applications in various cloud platforms [13].

9. **Networking service:** Different cloud providers support different types of networking service. Networking service is the one main factor for choosing a cloud platform. A lot of other aspects are directly related to the networking service.

10. **Database Support:** Different cloud providers support different types of databases . This provides support for handling different kind of database such as relational, SQL, No SQL, Data warehouse, in-built memory cache [17].

11. **Pricing methodology:** The pricing methods of all cloud platforms are pretty much the same, as all of the methods are based on pay as you go. That is the customer only needs to pay for the amount of resource they have used for what specific time frame.

12. **Load balancing and scaling:** Serverless provides an extensive scaling based on the traffic load. Clients do not need to provision load balancing and scaling. This management is done within the internal processing.

13. **Backend logic processing:** Each serverless platform works on a unit logic process that orchestrates and manages the events and requests. For different cloud platforms, the logical units varies. Based on the serverless application it creates workspace clusters, and integrates the events into it. The main process proceeds with the function invocation.

14. **Concurrent execution:** Concurrent Execution refers to the number of execution, executed over a unit period of time. This is a measuring standard in the serverless cloud platforms that defines the time of per application execution and thus defines how faster the serverless applications are executed over different platforms [7]. It is measured for different cloud platforms in standard unit.

15. **Server Distribution:** These three cloud providers have a number of servers around the world. Depending on the demand on the computing platform, servers being distributed in regions. The subdivision of regions are the zones. One noticeable thing is that, the closer to the server a client can be, the faster convergence and data traffic they will get. The cloud and networking distributions are therefore provided based on the closest region.

4.3 Comparative feature for Program Execution

Aws, GCP and Azure are the industry leading public cloud service provider. Developer can easily develop and execute the application easily. Cloud providers offers specific amount of CPU, memory for executing any program . For the infrastructure and other features, computing efficiency and memory usage can be different for different cloud platform. Practical comparison study between popular cloud platforms is very important for any developer in spite of providing various kind of features. Based on this comparison, developer can easily choose any platform for executing application. So practical comparison has great importance. The conducted practical comparison study based on the following two parameters:

1. Programme Execution Time
2. Memory Usage

1. Programme Execution Time: Computation time is very important for executing any programme. Based on computation efficiency execution time can be different. Some application needs to be executed in least execution time. Sometimes Least computation time can be developer's main requirement. So based on execution time comparison developer can choose appropriate cloud platform. For conducting the comparison study, we chose to run Recursive function and Image processing programme. We ran the Fibonacci programme on AWS, GCP and Fission successfully. After that, we executed image processing programme on AWS and GCP.

2. Memory Usage: Calculating memory for executing any programme in cloud platform is most essential feature for conducting any comparison study. For less memory usage, developer can get better productivity and system can use memory for more efficient system distribution. Ram Test can help any developer to consider any platform before choosing. So, we conduct Ram Test for Recursive function and Image processing programme on AWS and GCP.

Fission platform: Fission is an open source Kubernetes native serverless framework. Although this is an open source platform and not a public cloud, still there is a big room for development in fission. We could not find any dedicated development work in fission as the platform is very latest. No paper has been published under this framework neither any research work or comparison study has been conducted upon this platform. Fission exposes a whole new dimension for local host users, where they can provision their code without having much knowledge of Kubernetes. The function construction time in Fission are very impressive, without even having resources like the on demand cloud platforms.
Although it is offering such platform for serverless work, yet the infrastructure is still very fragile. We faced many difficulties while provisioning it and making comparison with other works. As fission works in the Command Line Interface and does not have a visual console or dashboard. Therefore, bringing out execution results were quite challenging.

4.4 Proposed Model

We design a performance-oriented serverless computing platform to study serverless implementation considerations. Our designed architecture comprises two segments: one is web services and another is a worker service. Web service will uncover the platform and finds accessible worker service via a messaging layer and worker service oversees and implements containers. We design a messaging layer to control the interaction among the web and worker service. Elasticity is one of the main factors which influences the performance of microservice deployed to serverless computing platforms. So for leveraging elastic serverless computing infrastructure for microservice hosting us use a cold queue and warm stack in messaging layers of our proposed architecture [10]. Execution request is done by the user and the webserver respond to the request and try to process a free container in the worker service to implement the execution request. Worker service can start a container when the worker has free memory for the container and the message in the cold queue represent the free memory of worker service.
In this process, some of the function containers are existing but which are not dealing with the execution request and the message of warm stack represents those existing containers. The worker service works based on the warm stack and cold queue. First of all, when a web service gets a new request from a user it lookup on the warm stack and tries to extract a message from a warm stack so that it could get

the information of those existing function containers [4]. If there is no message in a warm stack than the web service looks up on cold queue and tries to deque a message from the cold queue to know the current status of worker service to assign a new container to the function. If no message is found from the cold queue that means all the workers are occupied with running function containers. When both warm stack and cold queue is empty, that represent the unavailable container to execute the request and HTTP service unavailability will be sent by the system 4.1. Once any message is dequeued from cold queue, an HTTP request will be sent to worker service by web service to open a container which will execute the function. After the execution, the worker will return the output of that function to the web service. Container allocation management is an important fact for any microservice architecture.

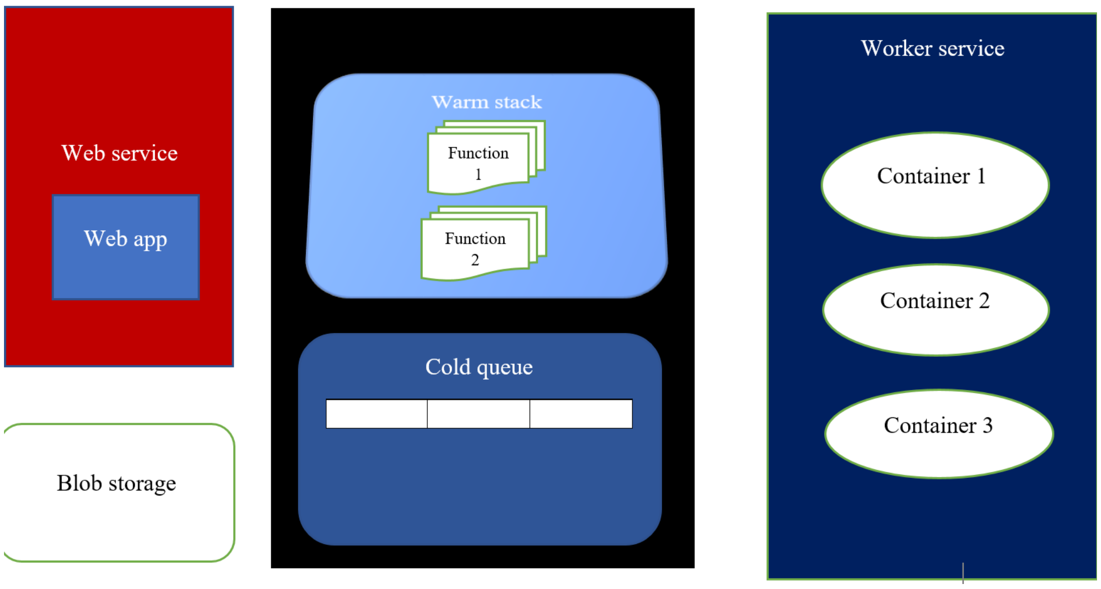


Figure 4.1: Overview of the Proposed Model of Serverless Architecture)

Here, every worker service has contained a bunch of free memory to manage the function container. Every message of the cold queue has a unique identity. Each message has a specific amount of memory F Fig: Overview of our proposed model reservation in worker service. When a message is dequeued then a container is generated with a unique name that represents its unique identity and reserved memory of that particular container in worker service. For removing containers we follow two criteria. First of all, if any function is deleted, the work service will automatically remove that warm stack of function.
As a result, worker service will also remove the container which was allocated for that function and retrieves memory. Secondly, if any container idles for a certain period of 20 minutes, that container will be removed and retrieves memory. For container orchestration we use Kubernetes which is developed by Kubernetes. Blob storage are used for storage. There is a concept of trigger based docker that we are going to use in our model such that we can extract the feature of serverless trigger based action to invoke a function just like serverless. We are going to put this whole thing in Kubernetes to orchestrate our container. This will give bring out the elasticity and scalability feature of our serverless platform. We use Linux based container because of its useful operation and support for serverless computing.

Chapter 5

Experimental Setup

For our analysis of features and practical experiment of the serverless use cases, we have worked with some renowned cloud platforms. All the necessary steps and codes from setting up this platforms and implementing the features are distributed in the Appendix: A part.Following are the short description to those platforms:

5.1 Amazon Seb Service

Amazon AWS is one of the leading cloud architectures available. AWS Lambda is the serverless platform of Amazon. Amazon launched Amazon Web Service in 2006 and it is enriched with resources. Lambda is a trigger based Function as a Service (FaaS), which is executed based on trigger events. In order to use the serverless feature of AWS we had to set up the components such as Lambda, Ec2 instance, S3 Bucket, Elastic block storage, IAM role, Event source, Downstream resources, Log streams etc.

5.2 Google Cloud Platform

Google introduced Google cloud platform (GCP)in 2011. GCP designed and developed by Google to provide cloud service on the same world class infrastructure and that is used by google for its product. Developer can easily build, test, deploy, and monitor application by using this infrastructure in google cloud platform. Google provide highly efficient server energy efficient data centre, industry level efficiency. Google Cloud gives unlimited access to user of data from any computer with a web browser. Compute Engine, App engine, Container Engine, Kubernetes engine, Cloud run, Cloud function, Storage, Data store, BigQuery, Google API, etc. were the products that we had to setup for using the serverless feature in GCP.

5.3 Microsoft Azure

Microsoft azure is one of the industry leading cloud provider. Microsoft launched Azure in 2010 to provide cloud service. Azure provides all kinds of tools and platforms to developer for developing, operating, and executing web applications. High performance, flexibility, and low service cost are main feature of Azure. We implemented some of the services like Azure AI, Azure Analytics, AZURE Internet of

Things (IoT), Azure Storage, Azure DevOps, Azure Virtual Machines (VMs), Azure Container Service, Azure Content Delivery Network (CDN) etc.

5.4 Fission

Fission is a framework of Kubernetes that created a platform for running serverless functions on a larger scale. Fission is open source and is suitable for running in any cloud platform be it Kubernetes, local host or any other public, private or hybrid cloud. It helps to manage serverless functions with any language and runtime and helps scaling the application. All the management of kubernetes are auto handled by Fission and pre-built containers are provided, so no need to worry about provisioning. It provides a function as a service for kubernetes
The framework runs on top of kubernetes and facilitates the faster convergence of deployment of functions while also featuring all the advantages of serverless. The function calls are HTTP trigger based.

Chapter 6

Experimental Results and Findings

6.1 Result of Feature Analysis

The different part of the result, feature analysis results and experimental results are shown in different sections below.

6.1.1 Comparative Study of the selected Features

Based on our implementation knowledge and research related to the selected Cloud providers, we have come up with a comparative study among the Cloud platforms. These leading Cloud platforms are vast in it's service. Therefore, in this comparison study we have only focused on the serverless related features and services. Here, we made a detailed comparison study followed by a brief table of comparison for better understanding.

1. **Cloud Service:** This shows the type of cloud service are provided by the selected cloud platform for users.

    - *AWS:* Provides all three service types IAAS,PAAS,SAAS. It is well-known throughout the world for its excellent Infrastructure as a Service solution. Elastic Beanstalk used for PAAS service.
    - *GCP:* GCP also offers IAAS,PAAS,SAAS. App engines mostly provides platform as a service This distinguishes it from others provider. Gmail,drive are SAAS.
    - *Azure:* Offers all three service IAAS ,PAAS, SAAS. Virtual machines, web hosting is IAAS. Email, calendars and Microsoft tools are SAAS

2. **Cloud Storage:** This explains the type of available storage service provided by the cloud service for the users.

    - *AWS:* Aws offers simple storage for big data storage , backup for storing data to users. Elastic block storage used for Nosql database,.Rds.Snowmobile for database migration service,Snowball for redshift,Elastic filw system for dyanamodb.Storage gateway for elastic cache.

- *GCP:* allows user with many option for storage like Cloud storage for cloud sql, persistant disk for cloud bigtable,Transfer appliance for cloud spanner,Transfer service for cloud datastore.
- **Azure:** Provides various option for simple storage Blob storage for SQL database,Queue storage for database for MYSQL,File storage for Postgre SQL ,Disk storage for data warehouse,Data lake store for cosmos DB,data factory,table storage.

3. **Security factor:** Here are the security factors, used in the cloud platform that we have selected, to secure the users credentials on private data.

    - *AWS:* For the management for securing service and resource access, amazon uses IAM Security(Amazon identity and access management). Cloud WatchDog monitors the cloud health on demand system. Unauthenticated traffic is secured by AWS security Hub. In serverless Lambda architecture, AWS GuardDuty is used for managing security threats. Application analysis and thread detection is done by AWS Inspector. DDos attacks are handled through AWS shields. CloudHSM for protecting key value credentials. Protection Firewall for the resources management is done by AWS Firewall management. Private key protection is done by AWS Key management service. Apartform all this, AWS provides security service in all aspects for security client data and privacy [21]. Some other services dedicated for internal and external resources are AWS Cognito, AWS directory service, AWS network firewall etc.
    - *GCP:* Google Binary Authorization is a security product that manages the verification of container deployment in the kubernetes engine. Cloud Asset Inventory does all the inventory management, monitors and analyzes inventory and produces real time notifications on changes. Cloud Data Loss Prevention prevents unauthorized data from trafficking, insects sensitive data and covers them from any harm. Cloud Key Management manages the encrypted keys and exports them securely to the requirement. Firewalls are used to safeguard the entire google platform. Some of the other security products are Secret manager, Shield VMs, Identity aware proxy, Identity platform and so on.
    - *Azure:* Azur has a number of security service products such as Azure Information Protection for secure sensitive data which are exposed outside the organization. Azure DDoS Protection to manage security threads. Application Gateway to protect from common application threads by building firewalls. Key Vault to protect cryptographic security keys that are used in the cloud platform. Apart from all this, there are Security Center, Microsoft Azure Attestation and Microsoft Azure Confidential Ledger.

4. **Language Variation:** This depicts the programming languages that are supported by the cloud platforms that have been chosen for a developer.

    - *AWS:* It works with any language. This distinguishes AWS as a standout for developers.
    - *GCP:* It supports .Net,Go,Java,Nodejs,Php,Python,Ruby.

- *Azure:* It supports C#, Java, Python, PHP, JavaScript.

5. **Virtualization Technique:** This depicts the virtualization process possible in the cloud platforms chosen.
    - *AWS:* Amazon Elastic compute uses Kernel- based Virtual machine (KVM).
    - *GCP:* Google compute engine uses Kernel- based Virtual machine (KVM).
    - *Azure:* To construct a virtual computer, the Hyper-V hypervisor is used.

6. **Implementation complexity:** There are different implementation techniques for different platforms with different complexity levels.
    - *AWS:* Lambda acts differently in terms of environment and language. Serverless functions are needed to be deployed directly in the console and can not be edited on the web. In the case of java, deployment of functions needs to be done within a project. The project will consist of dependencies, and the serverless code. The project will be a jar file of a zip file. The project will be uploaded as a serverless function. Also the deployment can be done through container images. AWS lambda supports a variety of languages and environments.
    - *GCP:* Google Cloud supports a number of languages but less than AWS lambda. All the language code can be deployed in the console and also codes can be edited in the console. GCP also supports container images [18].
    - *Azure:* Serverless function deployment in Azure is very different from the other two platforms. Although Azure supports a number of languages, any serverless function in any language needs to be deployed through an app in Azure. An app consists of dependencies, code and is edited with microsoft visual studio, microsoft visual studio code or any other IDE. In microsoft visual codes, we need to install some plugins for instance: Azure functions. Then the app is directly uploaded in the console.

7. **User Interface:** Different Cloud platform offering different Graphical User Interface (GUI) for customer flexibility. Customer comfort and flexibility solely depends on the customer demand.
    - *AWS:* Provides a web console, which shows all the variety of Amazon applications, customers can choose among them. Accessible from any smart device. Serverless functions have different sections, that is Lambda, where there are different kinds of deployment methods provided in different environments and languages. The GUI is pretty much used friendly specially for beginners. Amazon has a lot of official documentation, although the development is going through a constant change. Lambda UI is faster than GCP as the deployment and processing does not take a longer time [2].
    - *GCP:* It provides a web based user interface through application program interface (API) console. Google provides the Google function for deploying serverless functions. The UI is a bit more scattered than AWS and Azure as one needs to discover a lot of variants before in the console.

Documentations are provided in the console but the overall GUI is not user friendly for beginners. Deployment and processing in GCP takes comparatively longer time.

- *Azure:* The console is a web based interface through application program interface (API) console. For serverless functions the name of the service is called Function App. The GUI is very much user friendly but deployment of serverless functions is very different from AWS and GCP.

8. **Container Orchestration:** This shows vsrious container Orchestration service provided by different cloud platform.

    - *AWS:* Amazon lambda platform is built upon containers as a serverless platform. EC2 container service and Kubernetes(EKS) provides container sevice.
    - *GCP:* Google cloud platform is structured from Kubernetes platform, this gives an orchestration platform with the docker container. Kubernetes engine, Container engine used for container service.
    - *Azure:* Azure infrastructure is built with basic containerization. Uses Container Service.

9. **Networking Service:** It shows the different networking service provided by the different cloud platform.

    - *AWS:* uses Virtual private cloud used for isolated and private cloud networking, Route 53 for manage DNS, Cloud front for content delivery,API Gateway for cross premises connectivity,Elastic load balancing for Load balancing configuration.
    - *GCP:* uses Virtual private cloud used for isolated and private cloud networking, Google cloud DNS for manage DNS ,Cloud interconnect and Cloud CDN for content delivery, Cloud VPN for cross premises connectivity ,Cloud load balancing for Load balancing configuration.
    - *Azure:* uses Virtual network used for isolated and private cloud networking, Azure DNS, Traffic manager for manage DNS ,Content delivery network for content delivery, VPN gateway for cross premises connectivity, Load balancer and application Gateway for Load balancing configuration.

10. **Database Support:** This describes the types of database choices made available to a user by the specified cloud platforms. It is typically divided into two types: relational databases and non-relational databases.

    - *AWS:* Users may access a variety of databases, including Aurora, RDS, Elastic Cache, Dynamo DB, Neptune, Redshift and Database migration system.
    - *GCP:* GCP provides Cloud SQL, Cloud Spanner, Cloud Datastore and Cloud Bigtable which supports SQL and NOSQL.
    - *Azure:* Users may access databases that are mainly SQL-compatible, such as SQL, MySQL, PostreSQL, Data Warehouse, Table, and CosmoDB.

11. **Pricing methodology:** This describes the pricing technique of the different platforms. Pricing is based on subscription charge, usage charge, software and storage charge etc.

    - *AWS:* AWS provides a number of services for free. One can use an account from any email and the 1st trial of one year is free for new users but not for all instances. For example: Only Ubuntu EC2 instances are free but other EC2 instances will be charged. Their pricing methodology is based on pay per use. The application is ideal until there is a trigger calling the event. The billing process is based on per millisecond.
    - *GCP:* GCP pricing methodology is also pay as you go. GCP provides a 300$ credit for 90days trial for new users. Within this credit, one can use any feature. GCP charges for use per millisecond.
    - *Azure:* Microsoft Azure provides a free trial version for students only if they hold a microsoft student account or email id. For free trial they provide a 100$ credit for year time, and any service can be credited from there. The pricing is pay as you go per millisecond.

12. **Load balancing and scaling:** Load balancing and scaling is a major component in serverless architecture. The methodology is described here.

    - *AWS:* Lambda has an elastic load balancing with an automatic load balancing methodology. Application Load Balancer (ALBs) supports the incoming and outgoing traffic and function configuration management of AWS lambda. The management of requests from clients are managed with an HTTPS request by ALBs.
    - *GCP:* GCP uses instance grouping for load balancing.. Google Cloud Load Balancer (GCLB) is the distributed system of google cloud which manages the cloud load balancers. It does not require warm ups and can handle upto 1million cloud requests in each second. There is another distributer system for Google Cloud network balancing and distribution that is Maglev.
    - *Azure:* Azure follows an automatic load balancing system for it's traffic. For traffic balancing and management Azure uses Traffic Manager. There are other two types of load balancer in Azure that are Azure Load Balancer (ALB) and Internal Load Balancer (ILB) for managing internal loads for both inbound and outbound traffic. Azure load balancer is a layer4, TCP and UDP application with lower latency and higher throughput [8].

13. **Backend logic processing:** This section states what logical base are there platforms structured upon.

    - *AWS:* The backend logic that integrates events in AWS lambda is the Lambda function. The backend logic is hosted in lambda when the function is being invoked.
    - *GCP:* Cloud Function is the backend logical process for Google cloud services as it interacts with the necessary components like real time processing, database etc. for function deployment.

- *Azure:* Azure Event Grid initiates invocation for serverless function data processing and event handling. It manages the logical workflow for deploying a serverless function.

14. **Concurrent Execution:** The different execution time of the cloud platforms are stated here.

    - *AWS:* For AWS lambda the concurrent execution limit is 1000 parallel execution. Also the maximum execution time for a lambda function is 15 minutes/execution.
    - *GCP:* Google Cloud can handle as many parallel executions for a cloud function. The concurrency of requests can be handled through Google Cloud Run by selecting the maximum scaling of concurrency in Google function.
    - *Azure:* The concurrency on Azure is based on the app. The number is unlimited but downscaling or limiting concurrency is possible in Azure through Event Hub service.

15. **Server Distribution:** The geographical distribution of the regions for datacenter distributions are as follows.

    - *AWS:* Aws is the 1st cloud provider among these three platforms, therefore it is spread in a wide range of the graphica region. There are 25 regions with 80 zones throughout the globe. Later in 2020 they announced 25 more regions and 18 zones available in Asia, United Arab Emirates and some parts of the United Kingdom.
    - textitGCP: Although Google Cloud is the latest, it's spread across the world is nothing less than Amazon cloud. The branches of Google cloud have reached 25 regions with over 76 zones. In 2020 they have proposed 7 more regions in North America, Doha and Las Vegas.
    - *Azure:* The distribution of Azure is not well documented but they have a huge number of data centers. Azure more than 54 regions by 2018 and 16 more available.

6.1.2 Tabular form of comparison study

Here in table: 6.1 we have shown the tabular form of the comparison study conducted above:

6.2 Result of Practical Experiment

The results of practical implementations of the use cases, Backend API and Data processing are described in different sections below:

6.2.1 Recursive Function execution comparison

We have considered one of the most commonly used use cases for complexity measurement of function that is recursive function. We have seen that the widely used

Feature	AWS	GCP	Azure
1.Cloud Service	IAAS, SAAS, PAAS	IAAS, SAAS, PAAS	IAAS, SAAS, PAAS
2.Cloud Storage	Elastic Block storage, simple storage, Elastic File System, Storage Gateway, Snowmobile, Snowball	Cloud storage, Persistent disk, Transfer appliance, Transfer service	Simple storage, Blob storage, Queue storage, File storage, Disk storage, Data lake storage
3.Security Factor	IAM, Cloud watch-Dog, AWS Security Hub, AWS Guard Duty,, AWS shields, AWS key management, CloudHSM, Protection Firewall, AWS Firewall management, AWS Cognito, AWS Directory service, AWS network firewall	Google Binary Authorization, Cloud Asset Inventory, Cloud Data Loss Prevention, Cloud Key Management, Firewalls, Secret manager, Shield VMs, Identity aware proxy,Identity platform	Azure Information Protection, Azure DDoS Protection, Application Gateway, Key Vault, Security Center, Microsoft Azure Attestation, Microsoft Azure Confidential Ledger
4.Language variation	All possible programming languages	.Net, go, Java, Python, Ruby, PHP	C#, Java, PHP, Python, JS
5.Virtualiza–tion Technique	KVM	KVM	Hyper-V
6.Implementation complexity	Serverless function code, project, container image	Serverless function Code(Can be edited in console)	App, Project
7.User Interface	Web based console	Web based API console	Web based API console
8.Container orchestration	EC2, EKS	Kubernetes engine, Container engine	Container service
9.Network Service	Virtual private cloud, API gateway, Route 53, Cloud front, Elastic Cloud balancing	Virtual Private Cloud, Google Cloud DNS, Cloud Interconnect, Cloud CDN, Cloud VPN, Cloud load balancing	Virtual network, Azure DNS, Content delivery network, VPN gateway, Load balancer

Feature	AWS	GCP	Azure
10.Database Support	Aurora, RDS, Elastic cache, Dynamo DB, Neptune, Redshift, Database migration system	Cloud SQL, Cloud Spanner, Cloud Bigtable	SQL, MySQL, PostreSQL, Data warehouse, CosmoDB, Table
11.Pricing methodology	Pay per use, free trial 12months for some instances	Pay per use, Free trial 3months with a credit of 300$	Pay per use, free trial for 12months with a credit of 100$
12.Load balancing and scaling	Application Load Balancer (ALBs)	Google Cloud Load Balancer (GCLB), Maglev	Traffic Manager, Azure Load Balancer (ALB), Internal Load Balancer (ILB)
13.Backend logic processing	Lambda Function	Cloud Function	Azure Event Grid
14.Concurrent execution:	1000 parallel execution	Unlimited parallel execution	Unlimited parallel execution per app
15.Server distribution	25 regions 6 new available announced, 80 zones and 18 new available announced	25 regions, 76 zones	54 regions and 16 available

Table 6.1: Feature comparison of Cloud platforms

use cases of serverless functions are backend API which requires a lot of code complexity. For having a better idea on how these cloud platforms behave with the gradually increasing operational complexity, we have run a recursive function in AWS, GCP, Fission and collected data on execution time and memory usage.
As a recursive function, we have used Fibonacci for this experiment, as this is one of the strongest recursive functions which has greater deviation with each increment of number.

Fibonacci Execution:
To have a fair comparison, we have taken all three platforms with the same configuration. For the Fission cluster that we have created by using minikube in our pc, it was configured with 2GB RAM, 2 core CPU from our host machine. Same configuration was customized in AWS and GCP for this experiment.

Execution Time Comparison
AWS: We run the Fibonacci programme on AWS. For numbers 30,35,38 and 40 we execute the programme successfully. Here we show the execution time for AWS in the Table 6.2.
GCP: We run the Fibonacci programme on GCP. For numbers 30,35,38 and 40 we execute the programme successfully. Here we show the execution time for GCP in the Table 6.3.
Fission: We run the Fibonacci programme on Fission. For numbers 30,35,38 we

Fibonacci Number	Execution time(ms)
30	324.29ms
35	3539.54ms
38	15096ms
40	39576ms

Table 6.2: (AWS) Execution time for different Fibonacci numbers in

Fibonacci Number	Execution time(ms)
30	695.39ms
35	4240ms
38	18300ms
40	46300ms

Table 6.3: (GCP) Execution time for different Fibonacci numbers in

execute the programme successfully. But for Number 40 it couldn't execute the programme. Here we show the execution time for Fission in the Table 6.4.

Fibonacci Number	Execution time(ms)
30	405.5ms
35	4372.63ms
38	18404.68ms
40	Could not run

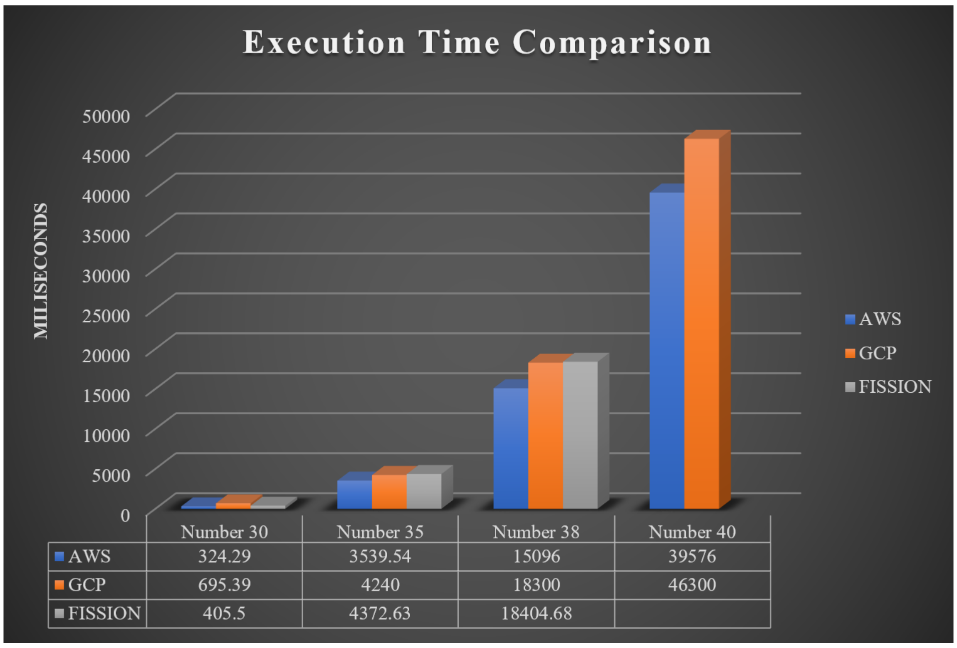
Table 6.4: (Fission) Execution time for different Fibonacci numbers in

Memory usage Comparison
AWS: We also run the Ram test on AWS for Fibonacci Programme. For AWS, we got Smallest Memory Request is 47.6837 MB, Max Memory Request is 48.6374 MB and Average Memory Used is 47.781 MB.
GCP: We also run the Ram test on GCP for Fibonacci Programme. For GCP, we got Smallest Memory Request is 59.8533 MB, Max Memory Request is 63.8973 MB and Average Memory Used is 61.8755 MB.

6.2.2 Image Processing Execution comparison

Another popular use case of serverless function is data processing, which is widely used in functions. To test this case, we have taken image processing function, as it is a perfect example of data pre-processing. Image processing deals with a lots of data at a time and affects both space and time complexity of a function. Therefore, we have chosen this as our test case.
We have run an image processing function that is converting color image to gray scale image in AWS and GCP platform. We have recorded significant differences in the execution time and memory utilization of this two platform.

Color to gray scale image:
For both platform, we have selected South Asia as regional server and Mumbai as zone. This is the closest server from our geographical region. As we now, this has



	Number 30	Number 35	Number 38	Number 40
AWS	324.29	3539.54	15096	39576
GCP	695.39	4240	18300	46300
FISSION	405.5	4372.63	18404.68	

Figure 6.1: Execution Time Comparison between AWS, GCP, Fission

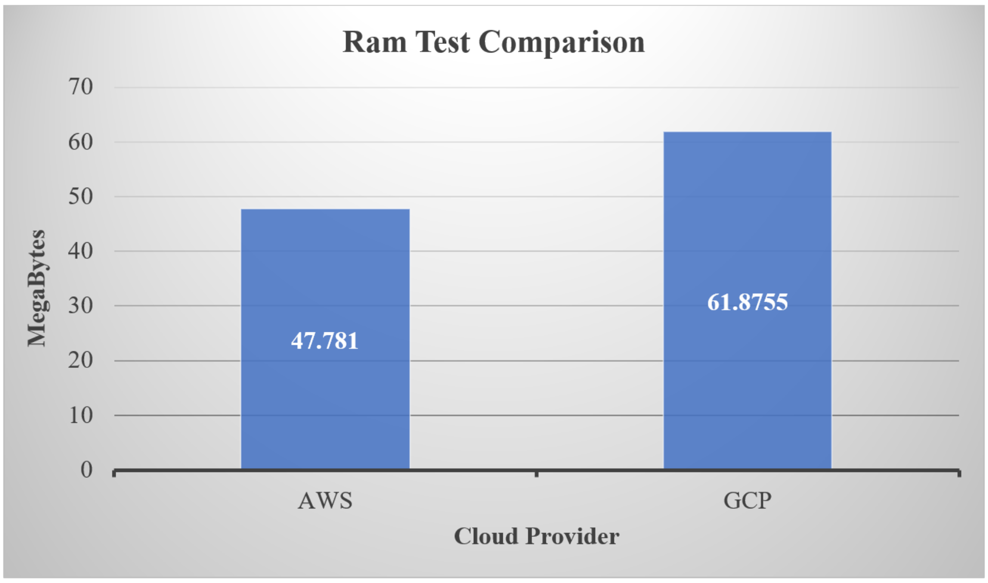


Figure 6.2: Memory usage comparison between AWS and GCP

a greater impact on the latency of accessing the function. The more server is away from the source, the less the latency is.
For this experiment for both the platform, we have chosen the configuration of 128mb of RAM.

Execution Time Comparison
AWS: We execute Image Processing using Numpy and OpenCV library .We also used BOTO3 library to get the image from Bucket. For executing these image processing, Maximum Duration is 3005 milliseconds, Minimum Duration is 2724 Milliseconds and Average duration for AWS is 2890 milliseconds.
GCP: We execute Image Processing using Numpy and OpenCV library. We also used Wand for reading the image, after reading the image we use Google cloud storage for accessing the image from Google bucket and we also used Google Cloud Version Library for completing the process. For completing 50% of process execution GCP took 16030milliseconds and for 95% it took 16570 milliseconds. For Total processing GCP took 16610 milliseconds.

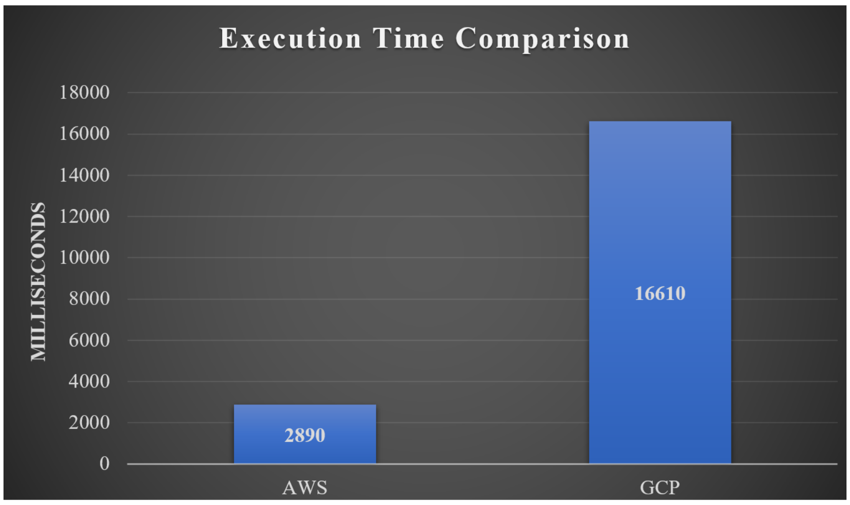


Figure 6.3: Comparison Between AWS and GCP for Image Processing Execution Time

Memory usage Comparison
AWS: For executing this image processing Programme AWS needs average 113 MB of memory. The iterative memory execution is shown in Table: 6.5.

Iteration No.	RAM usage(mb)
1	111
2	114
3	115
4	112

Table 6.5: (AWS) RAM usage for each iteration

GCP: For Total processing GCP took 16610 milliseconds. For executing this image processing Programme GCP needs average 120 MB of memory.The iterative memory execution is shown in Table: 6.6.

Iteration No.	RAM usage(mb)
1	123
2	118
3	120
4	119

Table 6.6: (GCP) RAM usage for each iteration

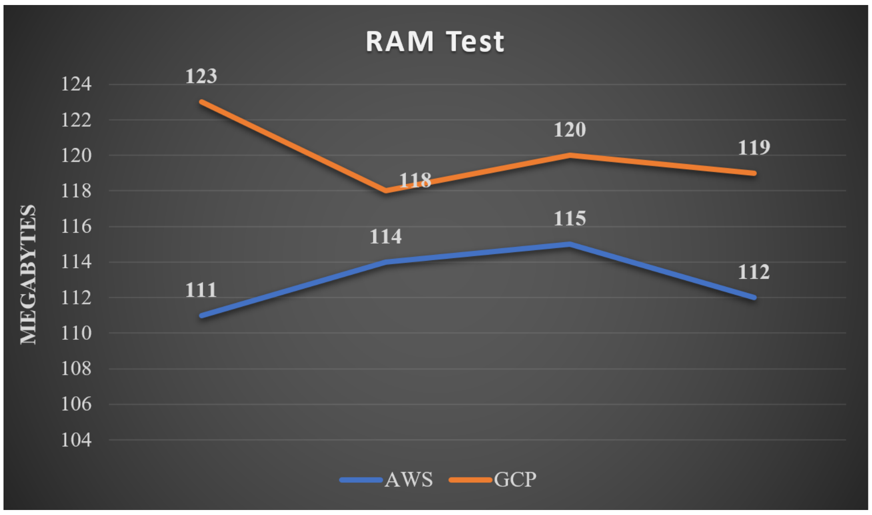


Figure 6.4: Memory usage comparison between AWS and GCP

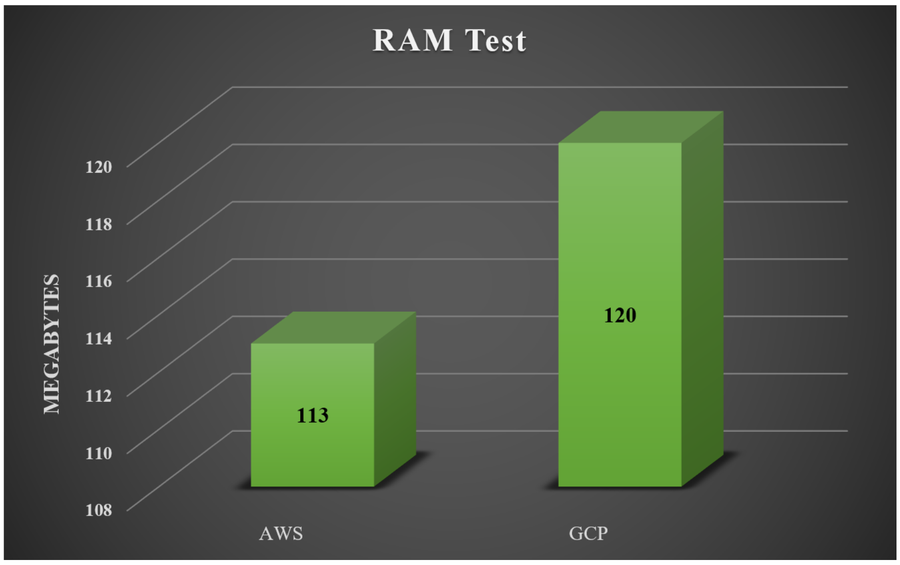


Figure 6.5: Average memory usage comparison between AWS and GCP

6.3 Findings

The analytical findings from the feature analysis and practical experimental section are described in following sections:

6.3.1 Findings on Feature Comparison

After the analysis of most of the features, the result shows that three of the cloud platform provides all three cloud services IaaS, PaaS, SaaS. For database storage they use their own cloud storage service. Among the three, Google security factor is the most strongest. But other provides are also providing better security service. All three providers are giving many language support but Amazon AWS is giving support for all possible programming language. In terms of virtualization AWS and GCP are the best as they are using KVM. The most user friendly implementation environment was given in GCP. Based on user interface, Azure has the top notch UI design. GCP is workable for any container based orchestration. This is why GCP is most efficient.
Google has plenty of their own networking service, for which their network service is rich among all. For database support, all three platforms are provides SQL based database system. Among them, Azure has the most variety of database but AWS is the most reliable one. GCP follows most simple pricing methodology, and low cost service. For load balancing, AWS performs best among all three. For back-end logic processing, Google Cloud Functions provide best performance. Concurrent execution is more flexible in GCP and Azure rather than AWS. AWS has the most distributed server among all three. Then there is Azure which has so many server in so many different regions.

6.3.2 Findings on practical Implementation

Recursive Function
In recursive function experiment, AWS performed better in both execution time and memory usage. GCP performance was slight lower than AWS but the overall performance was good. Execution time complexity of Fission was almost close to GCP.
We can see that in Fission, firstly it is performing worse as the Fibonacci number increased from 30 and secondly it was unable to run for Fibonacci number 40. The first case can happen as fission is running in our local host machine and host machine processor clock speed is lower then that of the AWS and GCP server processor, therefore it's execution time is increasing. Although the exicution time is quite remarkable in-spite of running from local host. The second possible case is that, although we have prepared all the three platform with same configuration and run the same code, Fission was unable to execute when the number increased to 40. The possible reason for this is, Fission was unable to make memory utilization as AWS or GCP.
We could not calculate memory usage in Fission because, it does not have any profound visual dashboard like AWS or GCP. For using fission, we installed Fission CLI, a commend line interface for editing the functions and deploying from command prompt, as Fission does not have any Graphical User Interface and log functions of the execution. The Fission CLI works as an API. So the features like

memory utilization and processor are not available like other cloud platforms. Only execution time is returned in the CLI.

Image Processing
We can see a noticeable difference between the execution time on GCP compared to the execution time of AWS in terms of the image processing. We have come up with some possible conclusions to this matter. Firstly, GCP fatches the data or in this case image from the bucket, deploy it to the Numpy library which then processes it to a Numpy array, return it to openCV then after processing openCV returns the image or data to the bucket and bucket again delivers the data. This process of fetching and delivering in GCP takes significantly a larger amount of time compared to AWS. Secondly, GCP works in such a way that, for any program execution, the dependent libraries are given in the Requirement.txt file which executes at the time of function execution. For AWS, the libraries are added in the layers. Upon deployment of the function, AWS layers install the libraries from some accessible nearby cash which is very much time efficient. But for GCP, the libraries in requiremen.txt file are directly installed from the internet which requires nearly manual library installation time which is very much greater than the AWS. Therefore the overall time complexity of GCP function is much greater than AWS.
Usually in AWS, an entire serverless function execution takes a few seconds, or we can scale it as less than 1 minute of time. On the other hand, for GCP, an entire function deployment takes about 3 minutes or more, like a range from 3 to 7 minutes. But there is an advantage of GCP, that is, it can take unlimited library function input in the requirement.txt file which may or may not be used in the serverless function execution. But in the AWS layer, the addition of layers has been limited to 5. So, in terms of complex functions, we will need to add multiple libraries in one layer which will perform slowly or may become more complex. But in GCP, unlimited dependency can be added all together. Therefore we predict form our experiment that, even though AWS performance is better for small and less complex functions than that of GCP, for more complex functions, GCP might perform better than AWS.
Here fission was not used for image processing comparison. As fission is not a complete serverless platform but an open source Kubernetes native serverless framework. Unlike AWS and GCP, Fission does not have any storage system. For AWS we have S3 bucket and for GCP we have google bucket to carry our data, specially in this case, carrying the image we want to process. Therefore, taking input image for function is impossible in fission.
Necessary libraries for image processing just like numpy and openCV were not possible to import in fission. For GCP there is a file called Requirement.txt, a text file which has mentions of all necessary libraries. And for AWS we have layers to add necessary libraries for a function. But in fission, there are no such clusters for adding libraries or we can say no proper documentation on how to add such things. Again in Fission, adding versions of different environments are mismatched. As there are specific versions of the openCv library for specific versions of language. Even though the AWS and GCP works fine with it, Fission does not work accordingly. For all this above, we did not add fission to the image processing experiment.

Chapter 7

Future work and Conclusion

7.1 Conclusion

As the revolution of the paperless world is the new progression and necessity, where we collectively gather our data in a more organized manner. Hence the urge of this field of technology cannot be denied. In our work we have compared three leading customer based public services which are Amazon AWS, Google Cloud Platform and Microsoft Azure. From hour research in the practical and technological field, we have come to state the point that, Amazon AWS is enriched with more resources then the other. Developers are more into Amazon than any other. Therefore, it has a large developer community. As this was the first Cloud based platform, they have vast resources and documentation in almost all aspects.On the other hand, Google has a lot more open source service and a huge development prospect. GCP is the most cost effective platform rather than AWS and Azure. With this platform, user have the advantage of accessing all Google products. For Azure, they are providing a huge opportunity for windows user and windows user community is the largest in the globe. So, Azure is more startup friendly. We hope our research will come as a little contribution in the serverless field and also will help us learn the related fields more practically. Hopefully our research work will help others to distinguish among the public cloud platforms. Our designed serverless architecture can hopefully make a significant impact in serverless computing. We have tried to bring all related resources under one roof so that we can have a more precise look on this topic and utilize the other studies.

7.2 Future work

For our theoretical comparisons, we only selected three cloud providers. In future, we want to bring out the other cloud providers in this comparison. For our practical work, we showed two use cases of serverless function. Serverless function can be used in few more different cases like Integration of 3rd party services, internal tooling, chatbot and IoT. We want to work in all these use cases and bring those work in our comparison and experiment. Fission is very new in this platform. As a prior future work, want to install Fission in a real server and develop this to take it to the comparison with other cloud platforms. We will implement our designed prototype serverless architecture and make more development on that design, so that that we can have a working serverless like the other Public serverless platforms.

Bibliography

[1] B. Rajeev, V. Baliga, *et al.*, "A comparative study of amazon web service and windows azure," *International Journal of Advanced Computer Research*, vol. 3, no. 3, p. 80, 2013.

[2] B. S. Đorđević, S. P. Jovanović, and V. V. Timčenko, "Cloud computing in amazon and microsoft azure platforms: Performance and service comparison," in *2014 22nd Telecommunications Forum Telfor (TELFOR)*, IEEE, 2014, pp. 931–934.

[3] I. Bari, S. Babu, M. M. Iqbal, Y. Saleem, and Z. A. Masood, "Cost and performance based comparative study of top cloud service providers," *International Journal of Computer Science and Information Security (IJCSIS)*, vol. 13, no. 12, 2015.

[4] M. Copeland, J. Soh, A. Puca, M. Manning, and D. Gollob, "Microsoft azure," *Apress: New York, NY, USA*, 2015.

[5] S. Krishnan and J. L. U. Gonzalez, *Building your next big thing with google cloud platform: A guide for developers and enterprise architects*. Springer, 2015.

[6] A. Glikson, S. Nastic, and S. Dustdar, "Deviceless edge computing: Extending serverless computing to the edge of the network," in *Proceedings of the 10th ACM International Systems and Storage Conference*, 2017, pp. 1–1.

[7] L. N. Hyseni and A. Ibrahimi, "Comparison of the cloud computing platforms provided by amazon and google," in *2017 Computing Conference*, IEEE, 2017, pp. 236–243.

[8] T. Islam and M. S. Hasan, "A performance comparison of load balancing algorithms for cloud computing," in *2017 International Conference on the Frontiers and Advances in Data Science (FADS)*, IEEE, 2017, pp. 130–135.

[9] T. Lynn, P. Rosati, A. Lejeune, and V. Emeakaroha, "A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, IEEE, 2017, pp. 162–169.

[10] G. McGrath and P. R. Brenner, "Serverless computing: Design, implementation, and performance," in *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, IEEE, 2017, pp. 405–410.

[11] T. I. Tanni and M. S. Hasan, "A performance analysis of a typical server running on a cloud," in *2017 20th International Conference of Computer and Information Technology (ICCIT)*, IEEE, 2017, pp. 1–6.

[12] R. Aljamal, A. El-Mousa, and F. Jubair, "A comparative review of high-performance computing major cloud service providers," in *2018 9th International Conference on Information and Communication Systems (ICICS)*, IEEE, 2018, pp. 181–186.

[13] C. Kotas, T. Naughton, and N. Imam, "A comparison of amazon web services and microsoft azure cloud platforms for high performance computing," in *2018 IEEE International Conference on Consumer Electronics (ICCE)*, IEEE, 2018, pp. 1–4.

[14] W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly, and S. Pallickara, "Serverless computing: An investigation of factors influencing microservice performance," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*, IEEE, 2018, pp. 159–169.

[15] A. Pérez, G. Moltó, M. Caballer, and A. Calatrava, "Serverless computing for container-based architectures," *Future Generation Computer Systems*, vol. 83, pp. 50–59, 2018.

[16] K. Tajane, S. Dave, P. Jahagirdar, A. Ghadge, and A. Musale, "Ai based chatbot using azure cognitive services," in *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, IEEE, 2018, pp. 1–4.

[17] A. Wahid and M. T. Banday, "Machine type comparative of leading cloud players based on performance & pricing," in *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, IEEE, 2018, pp. 2364–2368.

[18] E. Bisong, *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*. Apress, 2019.

[19] P. Dutta and P. Dutta, "Comparative study of cloud services offered by amazon, microsoft & google," *International Journal of Trend in Scientific Research and Development*, vol. 3, no. 3, pp. 981–985, 2019.

[20] D. M. Laxmaiah, D. Sharma, Y. Kumar, *et al.*, "A comparative study on google app engine amazon web services and microsoft windows azure," 2019.

[21] P. Pierleoni, R. Concetti, A. Belli, and L. Palma, "Amazon, google and microsoft solutions for iot: Architectures and a performance comparison," *IEEE Access*, vol. 8, pp. 5455–5470, 2019.

[22] L. A. Vayghan, M. A. Saied, M. Toeroe, and F. Khendek, "Microservice based architecture: Towards high-availability for stateful applications with kubernetes," in *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*, IEEE, 2019, pp. 176–185.

[23] H. Martins, F. Araujo, and P. R. da Cunha, "Benchmarking serverless computing platforms," *Journal of Grid Computing*, vol. 18, no. 4, pp. 691–709, 2020.

[24] K. Nakanishi, F. Suzuki, S. Ohzahata, R. Yamamoto, and T. Kato, "A container-based content delivery method for edge cloud over wide area network," in *2020 International Conference on Information Networking (ICOIN)*, IEEE, 2020, pp. 568–573.

[25] R. A. P. Rajan, “A review on serverless architectures-function as a service (faas) in cloud computing,” *TELKOMNIKA*, vol. 18, no. 1, pp. 530–537, 2020.

[26] V. Sharma, V. Nigam, and A. K. Sharma, “Cognitive analysis of deploying web applications on microsoft windows azure and amazon web services in global scenario,” *Materials Today: Proceedings*, 2020.

[27] B. Gupta, P. Mittal, and T. Mufti, “A review on amazon web service (aws), microsoft azure & google cloud platform (gcp) services,” 2021.

[28] L. Amazon, *Aws lambda-serverless compute-amazon web services.*

[29] M. Deploy, S. Containerized, and S. Ifrah, “Getting started with containers in google cloud platform,”

[30] N. Sabharwal and P. Pandey, “Pro google kubernetes engine,”

Appendix A

Setting up Cloud Platform and Practical Implementation Procedures

Our analysis is based on the comparison of different leading cloud platforms in terms of their working principle and complexity.We tried to capture Microsoft Azure, Google cloud, Amazon AWS and fission in the same scale and record their behavior and complexity.

A.1 Microsoft Azure

For the azure account, we needed a student account of windows but our university provides us g-suite email, with that g-suite email Microsoft does not give us the free trial of azure. So we get a Microsoft student email from other university friends. Using that account we got a 100 USD for 12months. Form that credit within the time we can spend all the money in any service of Azure and we did not need any VISA or Master Card for activating the trial. Here we wanted to run two kinds of serverless functions. First of all, we wanted to familiarize ourselves with the AWS console how to use and run a serverless function. Shown in figure: A.1 and figure: A.2 respectively.
But here in Azure, there is no way to write a function with a built-in editor. We need to install Microsoft studio or Microsoft studio code to build an app then we have to upload that app with all the dependencies and others in Azure. Shown in figure: A.3.
That means we need to upload an app directly with serverless. Shown in figure: A.4.
We were able to run the function withs test and also with the HTTP trigger. Shown in figure: A.5 and figure: A.6 respectively.
But when we tried to change any of this code and upload it then the HTTP trigger did not work, we tried different ways to make it work but there was no solution to make it works for us. So this platform is not user friendly for beginners like students but is more likely for experienced full-stack developers. But the Azure console is more organized compared to the other cloud providers. Here we can also select our host operating systems from Windows or Linux. Then we move forward with other cloud providers.

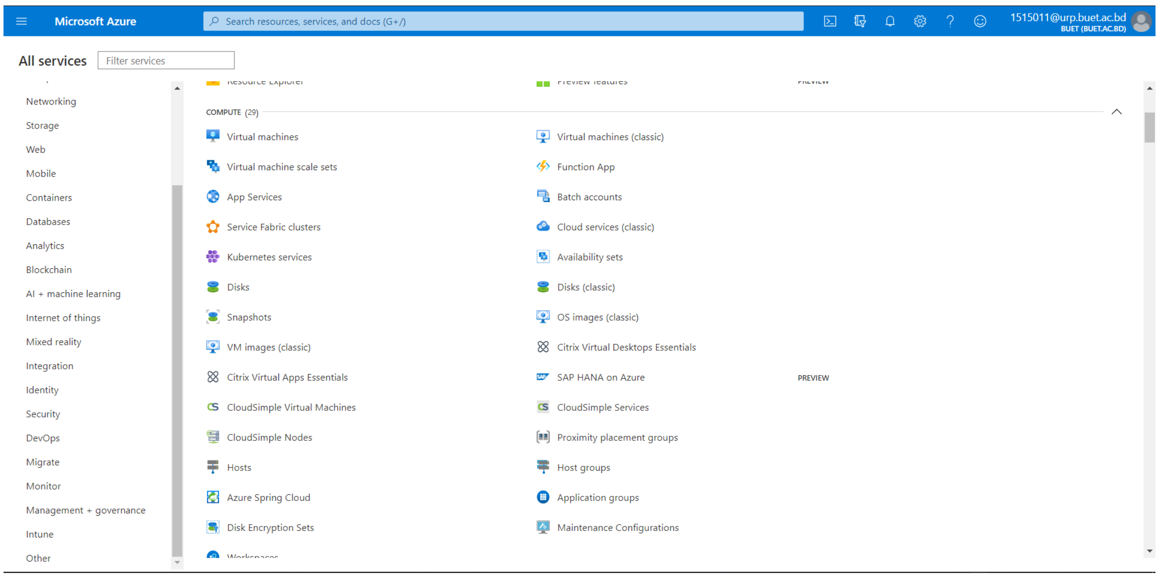
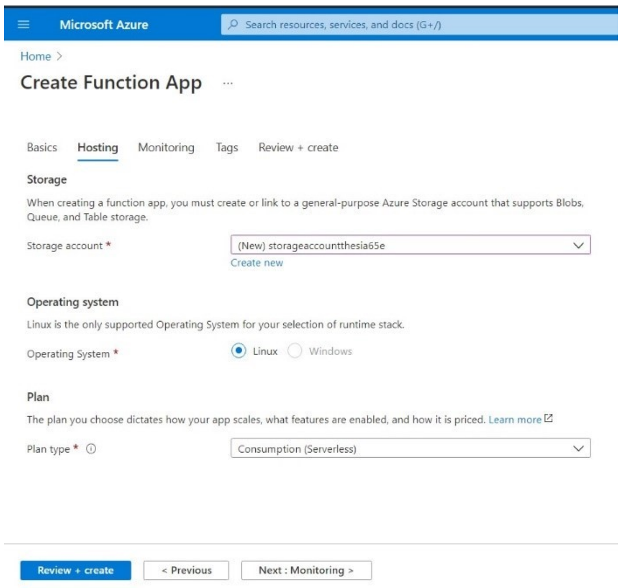


Figure A.1: Microsoft Azure Portal (Dashboard)



Storage

When creating a function app, you must create or link to a general-purpose Azure Storage account that supports Blobs, Queue, and Table storage.

Storage account * (New) storageaccountthesia65e

Operating system

Linux is the only supported Operating System for your selection of runtime stack.

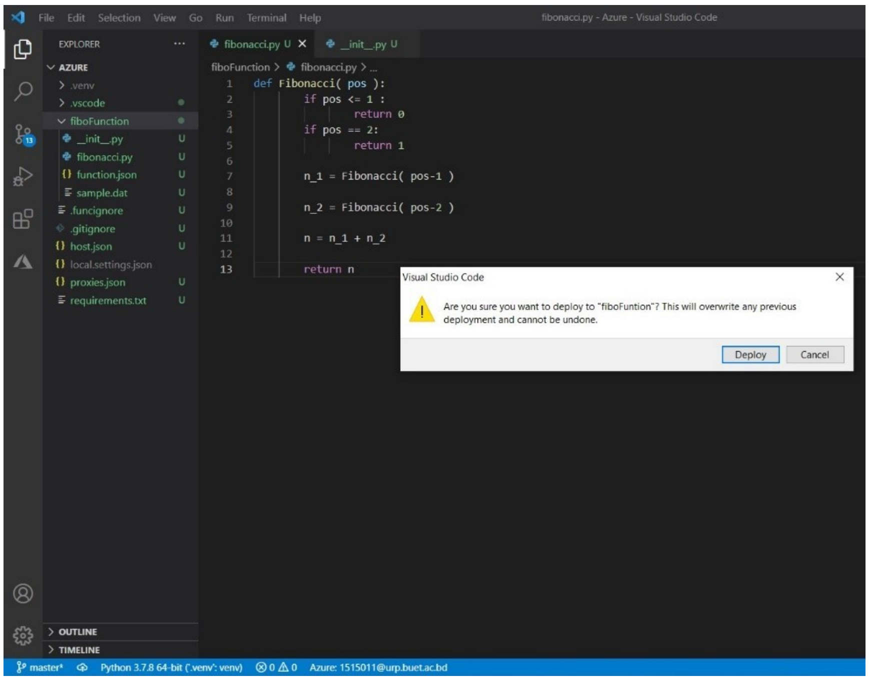
Operating System * [✓ Linux] Windows

Plan

The plan you choose dictates how your app scales, what features are enabled, and how it is priced. Learn more

Plan type * Consumption (Serverless)

Figure A.2: Azure Serverless Function creation

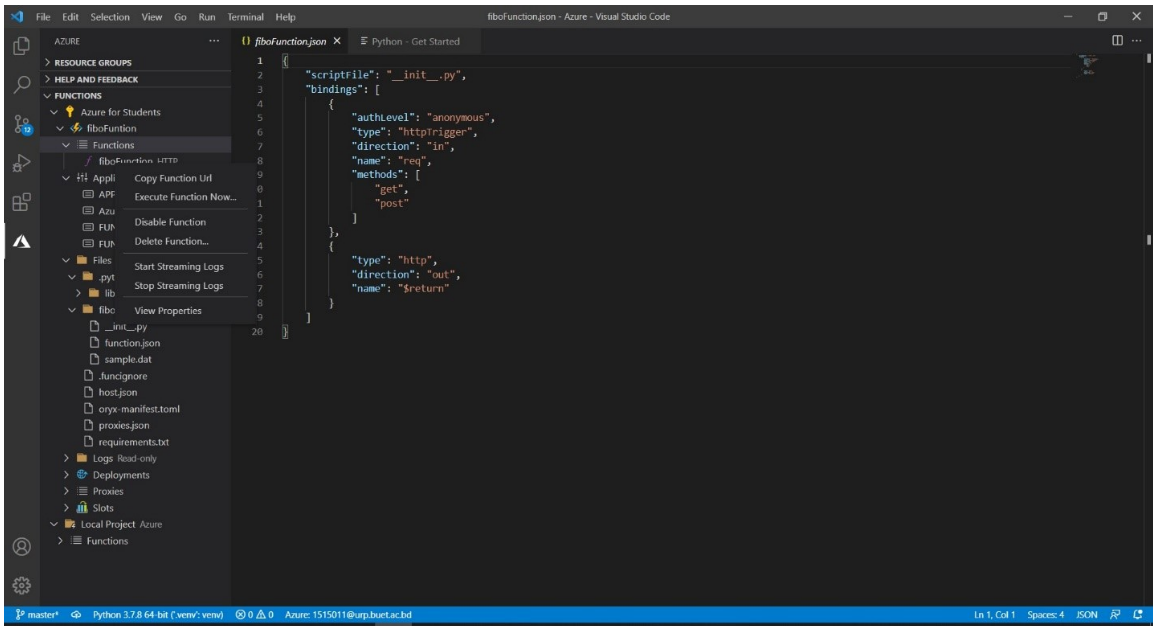


```python
def Fibonacci( pos ):
    if pos <= 1 :
        return 0
    if pos == 2:
        return 1

    n_1 = Fibonacci( pos-1 )

    n_2 = Fibonacci( pos-2 )

    n = n_1 + n_2

    return n
```

Visual Studio Code

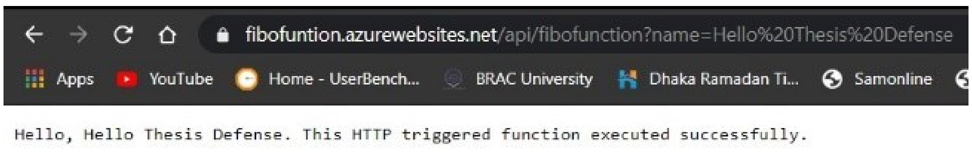
Are you sure you want to deploy to "fiboFuntion"? This will overwrite any previous deployment and cannot be undone.

Figure A.3: Deploying Serverless Function app using Microsoft code studio



```json
{
    "scriptFile": "__init__.py",
    "bindings": [
        {
            "authLevel": "anonymous",
            "type": "httpTrigger",
            "direction": "in",
            "name": "req",
            "methods": [
                "get",
                "post"
            ]
        },
        {
            "type": "http",
            "direction": "out",
            "name": "$return"
        }
    ]
}
```

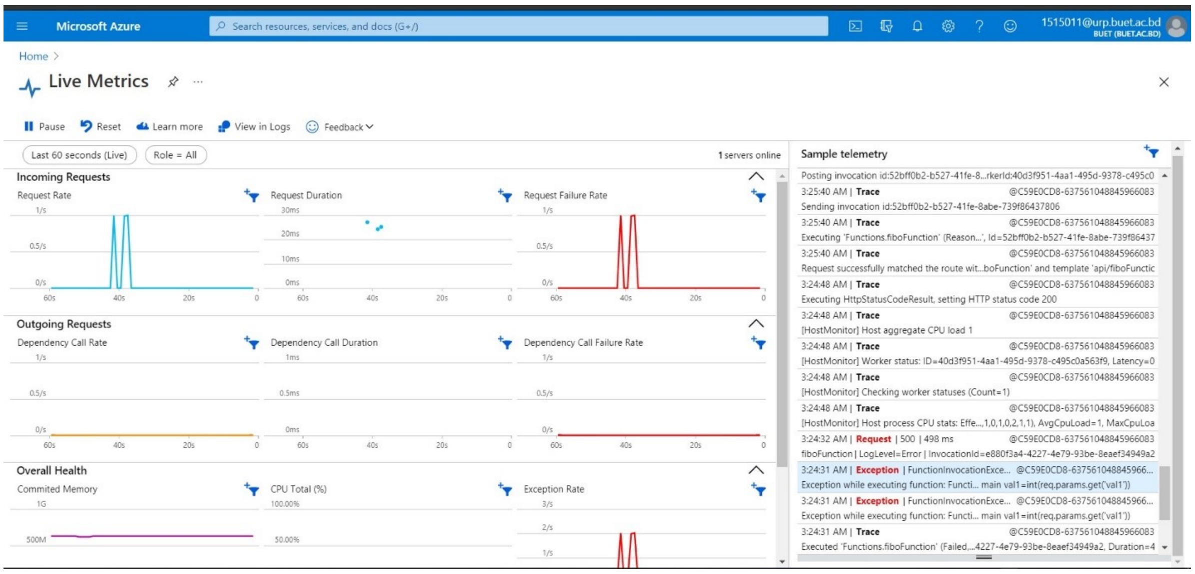
Figure A.4: Detailed Function testing in Azure



fibofuntion.azurewebsites.net/api/fibofunction?name=Hello%20Thesis%20Defense

```
Hello, Hello Thesis Defense. This HTTP triggered function executed successfully.
```

Figure A.5: Function app public HTTP trigger in Azure



```
Posting invocation id:52bff0b2-b527-41fe-8...rkerId:40d3f951-4aa1-495d-9378-c495c0
3:25:40 AM | Trace                                   @C59E0CD8-637561048845966083
Sending invocation id:52bff0b2-b527-41fe-8abe-739f86437806
3:25:40 AM | Trace                                   @C59E0CD8-637561048845966083
Executing 'Functions.fiboFunction' (Reason...', Id=52bff0b2-b527-41fe-8abe-739f86437
3:25:40 AM | Trace                                   @C59E0CD8-637561048845966083
Request successfully matched the route wit...boFunction' and template 'api/fiboFunctic
3:24:48 AM | Trace                                   @C59E0CD8-637561048845966083
Executing HttpStatusCodeResult, setting HTTP status code 200
3:24:48 AM | Trace                                   @C59E0CD8-637561048845966083
[HostMonitor] Host aggregate CPU load 1
3:24:48 AM | Trace                                   @C59E0CD8-637561048845966083
[HostMonitor] Worker status: ID=40d3f951-4aa1-495d-9378-c495c0a563f9, Latency=0
3:24:48 AM | Trace                                   @C59E0CD8-637561048845966083
[HostMonitor] Checking worker statuses (Count=1)
3:24:48 AM | Trace                                   @C59E0CD8-637561048845966083
[HostMonitor] Host process CPU stats: Effe...,1,0,1,0,2,1,1), AvgCpuLoad=1, MaxCpuLoa
3:24:32 AM | Request | 500 | 498 ms                  @C59E0CD8-637561048845966083
fiboFunction | LogLevel=Error | InvocationId=e880f3a4-4227-4e79-93be-8eaef34949a2
3:24:31 AM | Exception | FunctionInvocationExce...  @C59E0CD8-637561048845966...
Exception while executing function: Functi... main val1=int(req.params.get('val1'))
3:24:31 AM | Exception | FunctionInvocationExce...  @C59E0CD8-637561048845966...
Exception while executing function: Functi... main val1=int(req.params.get('val1'))
3:24:31 AM | Trace                                   @C59E0CD8-637561048845966083
Executed 'Functions.fiboFunction' (Failed,...4227-4e79-93be-8eaef34949a2, Duration=4
```

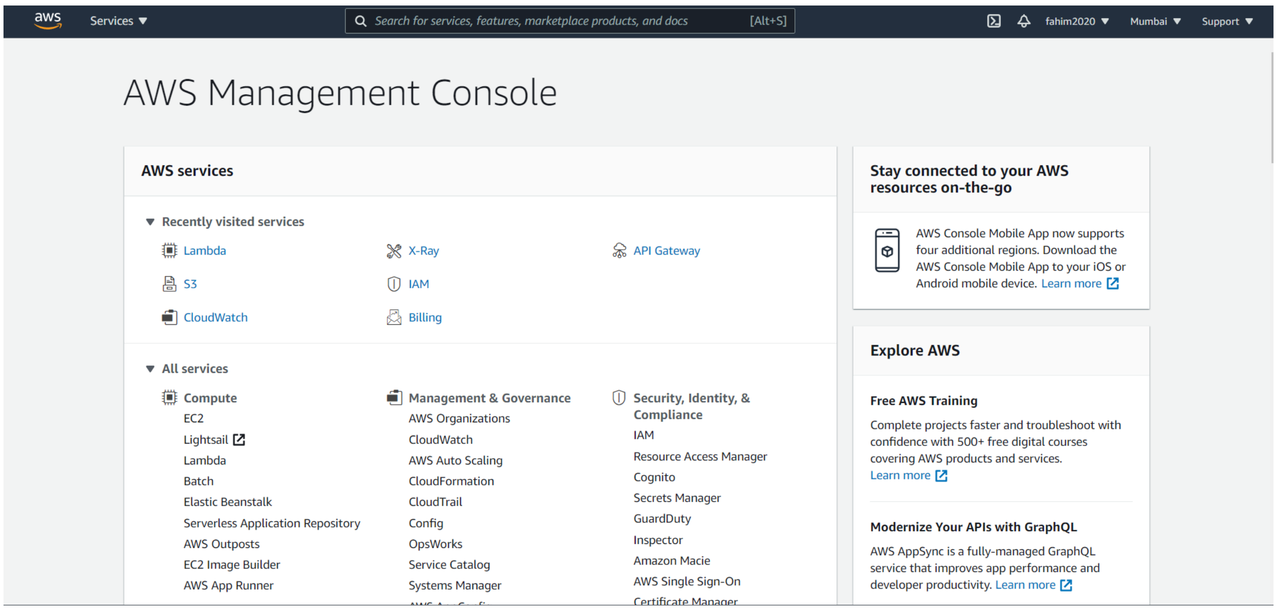
Figure A.6: Live function log of Azure function

A.2 Amazon AWS:

In AWS they provide free 12 months of tier account. For the free tier account, we need to give them a VISA or Master Card number so that if we exceed the limit of the specific service then we will charge us from that card. In that free services, the serverless function which is called Lambda function in AWS was included with a four million execution number and four million seconds. After that, we familiar with the AWS console. We noticed that here in AWS there are more features comparative to others but we only needed the Lamda, EC2 instance(Linux), and S3 bucket for our work.
In 2021 we can create functions from the four following options.

- Author from scratch: Here we get a simple hello world code to run.
- Use a blueprint: Here we can build functions with simple code and configurations preset for common use cases like Microservice HTTP endpoint, Dynamodb process stram, slack eco command python etc.
- Container image: This feature is very useful for the developers using this we can deploy a container image directly into Lamda so we don't need to worry about dependencies or library import problem and version mismatched problem.
- Browse serverless app repository: In AWS Lamda Application Repository there are 684 apps we can choose any of the Lamda applications from there.

First of all, we tried to write a Lambda function using java but in AWS there is no way to write code directly in their editor or console for JAVA. Shown in figure: A.7 and figure: A.8.
So for writing any Lambda function in java we need to use JAVA IDE. We need to make a Maven project so that the Lamda function dependencies and plugins can be installed in the zip or JDK file. Shown in figure: A.9.


aws
Services ▼
Search for services, features, marketplace products, and docs [Alt+S]
fahim2020 ▼
Mumbai ▼
Support ▼

AWS Management Console

AWS services

▼ Recently visited services

Lambda | X-Ray | API Gateway
S3 | IAM
CloudWatch | Billing

▼ All services

Compute
EC2
Lightsail
Lambda
Batch
Elastic Beanstalk
Serverless Application Repository
AWS Outposts
EC2 Image Builder
AWS App Runner

Management & Governance
AWS Organizations
CloudWatch
AWS Auto Scaling
CloudFormation
CloudTrail
Config
OpsWorks
Service Catalog
Systems Manager

Security, Identity, & Compliance
IAM
Resource Access Manager
Cognito
Secrets Manager
GuardDuty
Inspector
Amazon Macie
AWS Single Sign-On
Certificate Manager

Stay connected to your AWS resources on-the-go

AWS Console Mobile App now supports four additional regions. Download the AWS Console Mobile App to your iOS or Android mobile device. Learn more

Explore AWS

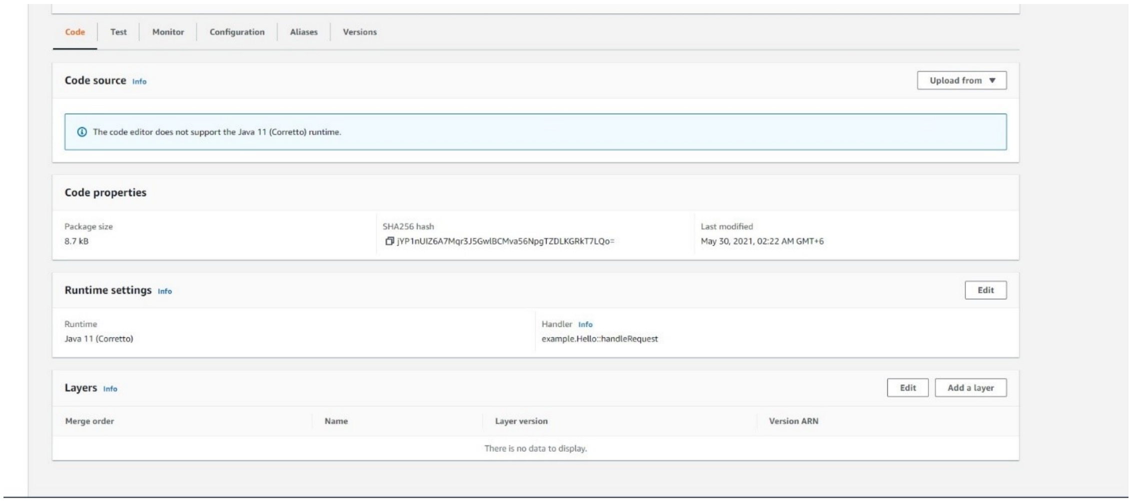
Free AWS Training

Complete projects faster and troubleshoot with confidence with 500+ free digital courses covering AWS products and services. Learn more

Modernize Your APIs with GraphQL

AWS AppSync is a fully-managed GraphQL service that improves app performance and developer productivity. Learn more

Figure A.7: Amazon AWS Management Console



Code source Info

The code editor does not support the Java 11 (Corretto) runtime.

Code properties

Package size	SHA256 hash	Last modified
8.7 kB	jYP1nUIZ6A7Mqr3J5GwlBCMva56NpgTZDLKGRkT7LQo=	May 30, 2021, 02:22 AM GMT+6

Runtime settings Info

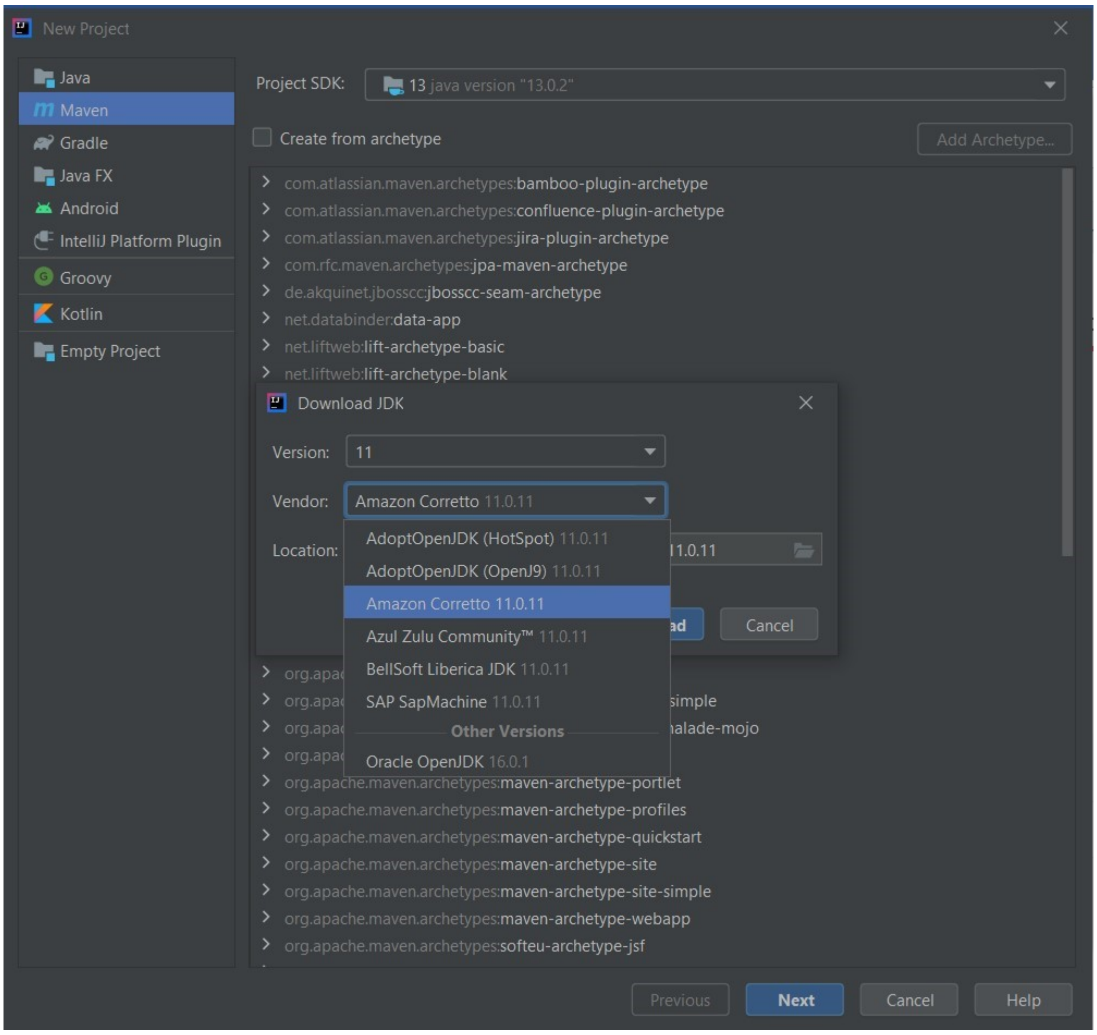
Runtime	Handler Info
Java 11 (Corretto)	example.Hello::handleRequest

Layers Info

Merge order	Name	Layer version	Version ARN

There is no data to display.

Figure A.8: Lambda function for Java runtime and editor


New Project

Project SDK: 13 java version "13.0.2"

Create from archetype

com.atlassian.maven.archetypes:bamboo-plugin-archetype
com.atlassian.maven.archetypes:confluence-plugin-archetype
com.atlassian.maven.archetypes:jira-plugin-archetype
com.rfc.maven.archetypes:jpa-maven-archetype
de.akquinet.jbosscc:jbosscc-seam-archetype
net.databinder:data-app
net.liftweb:lift-archetype-basic
net.liftweb:lift-archetype-blank
org.apache.maven.archetypes:maven-archetype-portlet
org.apache.maven.archetypes:maven-archetype-profiles
org.apache.maven.archetypes:maven-archetype-quickstart
org.apache.maven.archetypes:maven-archetype-site
org.apache.maven.archetypes:maven-archetype-site-simple
org.apache.maven.archetypes:maven-archetype-webapp
org.apache.maven.archetypes:softeu-archetype-jsf

Download JDK

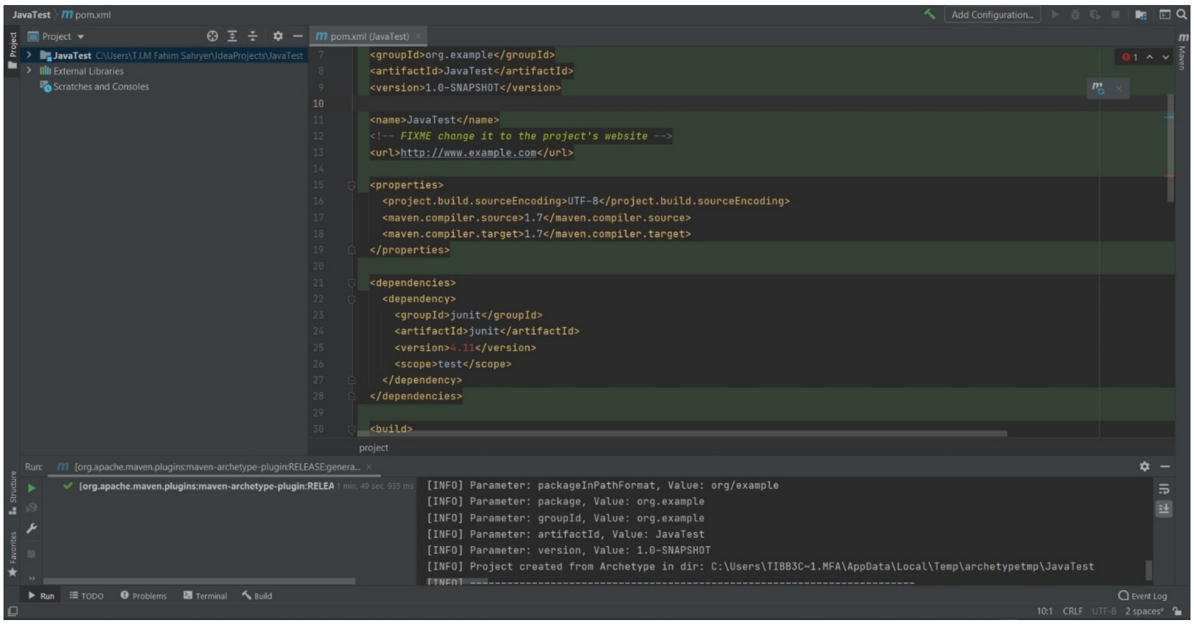
Version: 11

Vendor: Amazon Corretto 11.0.11

Location: 11.0.11

Figure A.9: Java Maven project for Lambda function in IDE

Here in the Maven project is a **pom.xml** file where we need to add dependencies and plugins from AWS websites then the IDE will download the dependencies and plugins from the internet. Shown in figure: A.10.



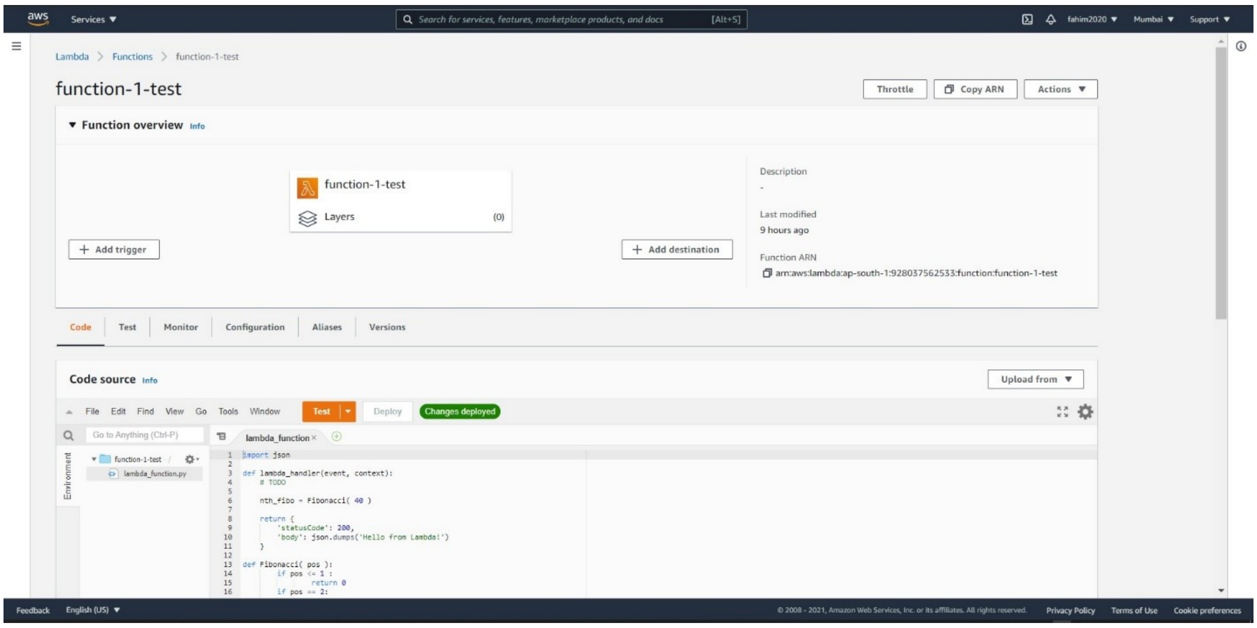
```
<groupId>org.example</groupId>
<artifactId>JavaTest</artifactId>
<version>1.0-SNAPSHOT</version>

<name>JavaTest</name>
<!-- FIXME change it to the project's website -->
<url>http://www.example.com</url>

<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>1.7</maven.compiler.source>
  <maven.compiler.target>1.7</maven.compiler.target>
</properties>

<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
  </dependency>
</dependencies>

<build>
```

```
[INFO] Parameter: packageInPathFormat, Value: org/example
[INFO] Parameter: package, Value: org.example
[INFO] Parameter: groupId, Value: org.example
[INFO] Parameter: artifactId, Value: JavaTest
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Project created from Archetype in dir: C:\Users\TIBB3C~1.MFA\AppData\Local\Temp\archetypetmp\JavaTest
[INFO] ------------------------------------------------------------------------
```

JavaTest › pom.xml
Add Configuration...
Project
JavaTest C:\Users\T.I.M Fahim Sahryer\IdeaProjects\JavaTest
External Libraries
Scratches and Consoles
pom.xml (JavaTest)
project
Run: [org.apache.maven.plugins:maven-archetype-plugin:RELEASE:genera...
[org.apache.maven.plugins:maven-archetype-plugin:RELEA 1 min, 49 sec, 935 ms
Run
TODO
Problems
Terminal
Build
Event Log
10:1 CRLF UTF-8 2 spaces*

Figure A.10: pom.xml file for Lambda dependencies and plugings

After downloading the plugins and dependencies we could import the necessary library for the Lamda function. At first, we were getting many errors importing the libraries but after downloading the error gone. After writing a simple code in the main Java file we run the project as a Maven project. After running it makes a zip file name of the project. Then upload it in AWS but after 1st test run, we got errors. To solve the error we need to change of codes in our IDE and build the project again and upload it to the AWS and there is also a limitation of the zip file size of 10MB. For that, we tried to build a new project and changed our codes and tried several times but did not work. So we moved forward for python. The good thing about python in Lambda is we can edit our code in the AWS console's editor. Shown in figure: A.11.
We can select python 2.7, 3.6, 3.7 and 3.8 for the python runtime. After creating a function from scratch we first able to run our hello world code as a Lambda function. Then we tried to implement a simple calculator there and able to run it our calculator can only do the basic operations. After analyzing the code we understand that for any serverless function it first looks for the handler method to run **def lambda_handler(event, context):** and it takes event and context as parameters. So to confirm we changed the handler function and also removed that but after doing that the function did not work at all. And the handler function returns a not other than string only. If we tried to return other data types it will give an error. We wanted to test two types of serverless functions here which is simple recursive function and data processing. For the recursive function, we chose to calculate a Fibonacci number recursively. Our logic for choosing this function is to push the limit or test the AWS hosting servers capability to efficiently handle the function. And in website or app backend API there are many recursive functions is used and we know recursive functions are CPU and ram intensive.


Services
Search for services, features, marketplace products, and docs [Alt+S]
fahim2020
Mumbai
Support
Lambda > Functions > function-1-test
function-1-test
Throttle
Copy ARN
Actions
Function overview Info
function-1-test
Layers (0)
+ Add trigger
+ Add destination
Code Test Monitor Configuration Aliases Versions
Code source Info
Upload from
File Edit Find View Go Tools Window
Test
Deploy
Changes deployed
Go to Anything (Ctrl-P)
lambda_function
Environment
function-1-test
lambda_function.py
Feedback English (US)
© 2008 - 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy Terms of Use Cookie preferences

Description
-

Last modified
9 hours ago

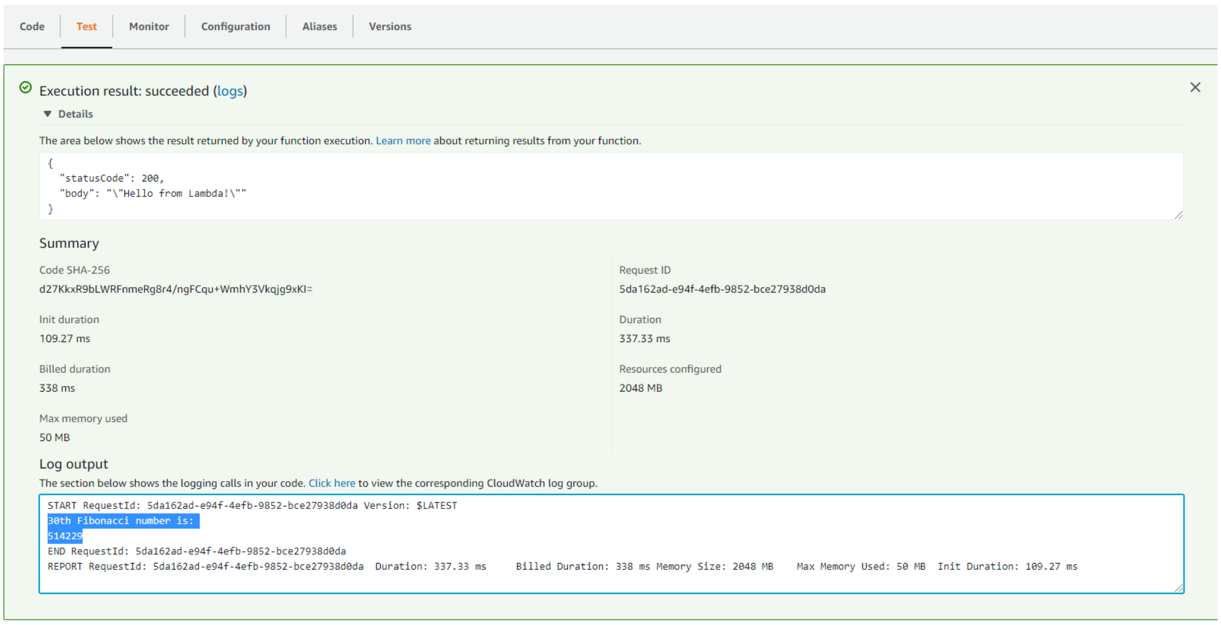
Function ARN
arn:aws:lambda:ap-south-1:928037562533:function:function-1-test

```
import json

def lambda_handler(event, context):
    # TODO

    nth_fibo = Fibonacci( 40 )

    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }

def Fibonacci( pos ):
    if pos <= 1 :
        return 0
    if pos == 2:
```

Figure A.11: Lambda inline code editor for Python environment

```
import json
def lambda_handler(event, context):
    # TODO
    nth_fibo = Fibonacci( 40 )
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
def Fibonacci( pos ):
        if pos <= 1 :
                return 0
        if pos == 2:
                return 1
        n_1 = Fibonacci( pos-1 )
        n_2 = Fibonacci( pos-2 )
        n = n_1 + n_2
        return n
```

We use this code for calculating the Fibonacci numbers and took the value of finding the time to calculate the various Fibonacci number to compare with another cloud platform. Shown in figure: A.12 and figure: A.13.
After doing this we moved for data processing using the Lambda function. Shown in figure: A.14.
For this, we get into many problems with the external library importation in the lambda function. Like our python code editor, we cannot import any kind of library in Lambda. For image processing, we must need NumPy and OpenCV library to do that. For the libraries, Lamda provided Layer options. Layer gives us the ability to make a library for use in lambda functions. We can add a maximum of 250MB of total layer size and a maximum of five layers in a single function. But the same layer can be used in an unlimited number of functions. For making a layer AWS provide



Execution result: succeeded (logs)

Details

The area below shows the result returned by your function execution. Learn more about returning results from your function.

```
{
  "statusCode": 200,
  "body": "\"Hello from Lambda!\""
}
```

Summary

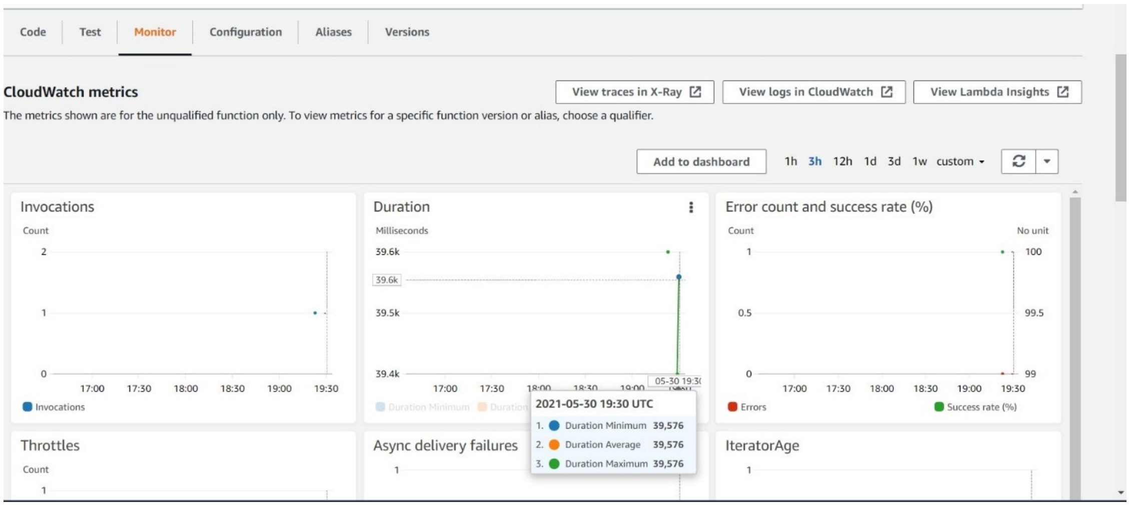
Code SHA-256	Request ID
d27KkxR9bLWRFnmeRg8r4/ngFCqu+WmhY3Vkqjg9xKI=	5da162ad-e94f-4efb-9852-bce27938d0da
Init duration	Duration
109.27 ms	337.33 ms
Billed duration	Resources configured
338 ms	2048 MB
Max memory used	
50 MB	

Log output

The section below shows the logging calls in your code. Click here to view the corresponding CloudWatch log group.

```
START RequestId: 5da162ad-e94f-4efb-9852-bce27938d0da Version: $LATEST
30th Fibonacci number is:
514229
END RequestId: 5da162ad-e94f-4efb-9852-bce27938d0da
REPORT RequestId: 5da162ad-e94f-4efb-9852-bce27938d0da  Duration: 337.33 ms     Billed Duration: 338 ms Memory Size: 2048 MB    Max Memory Used: 50 MB  Init Duration: 109.27 ms
```

Figure A.12: Lambda function logs with details



CloudWatch metrics

The metrics shown are for the unqualified function only. To view metrics for a specific function version or alias, choose a qualifier.

Figure A.13: Graphical view of Lambda function execution

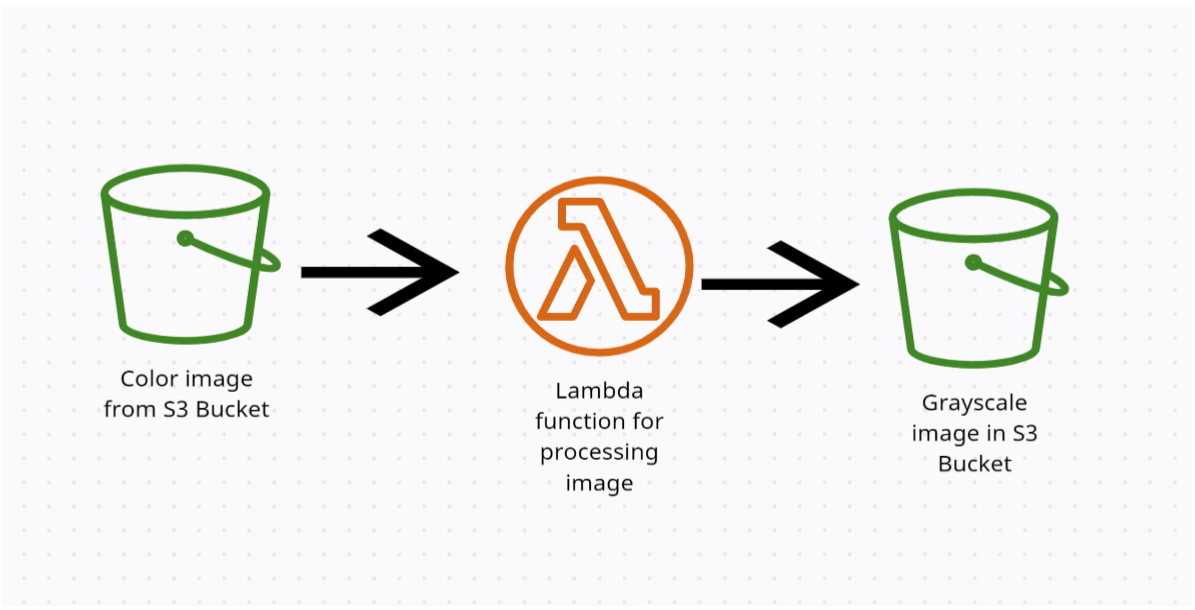
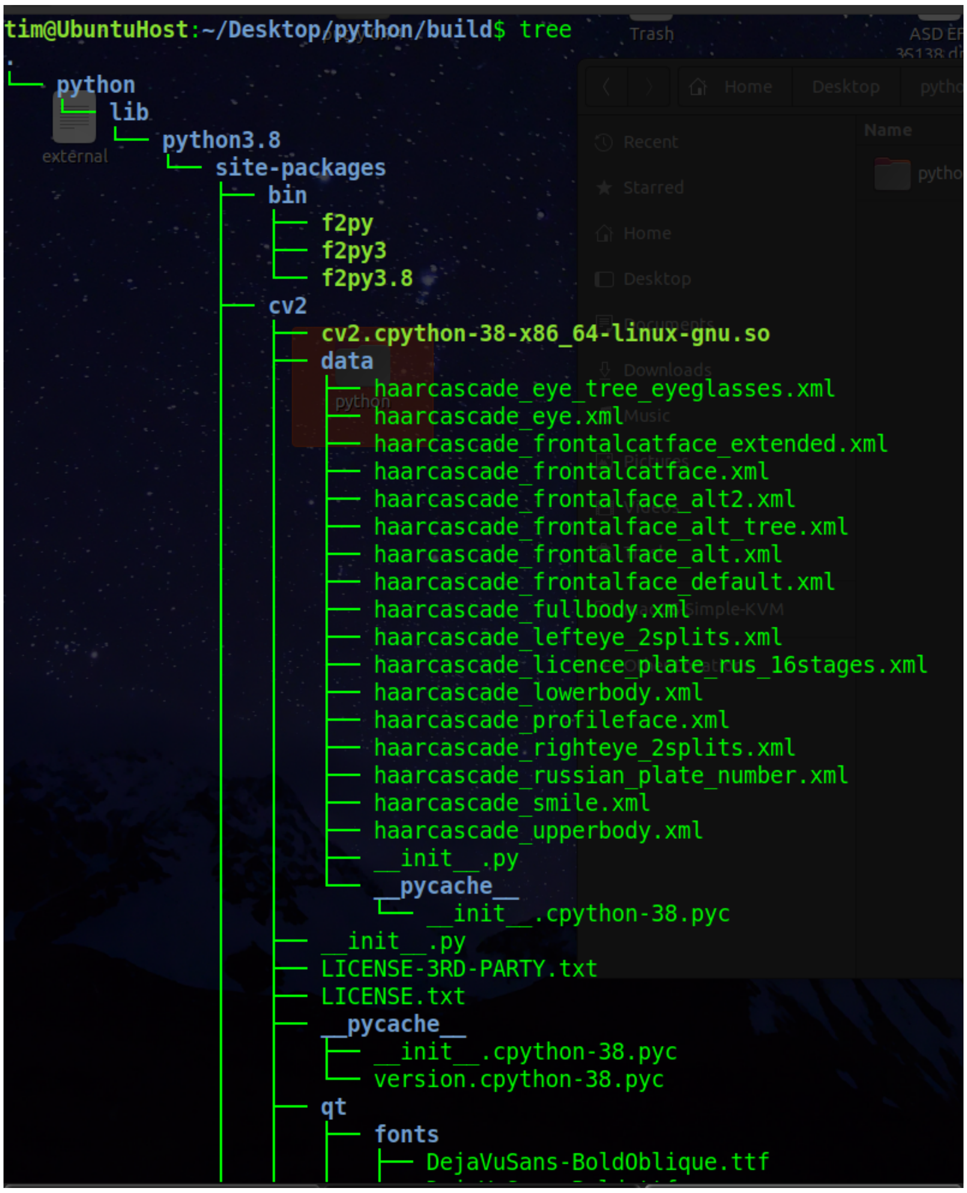

Color image from S3 Bucket
Lambda function for processing image
Grayscale image in S3 Bucket

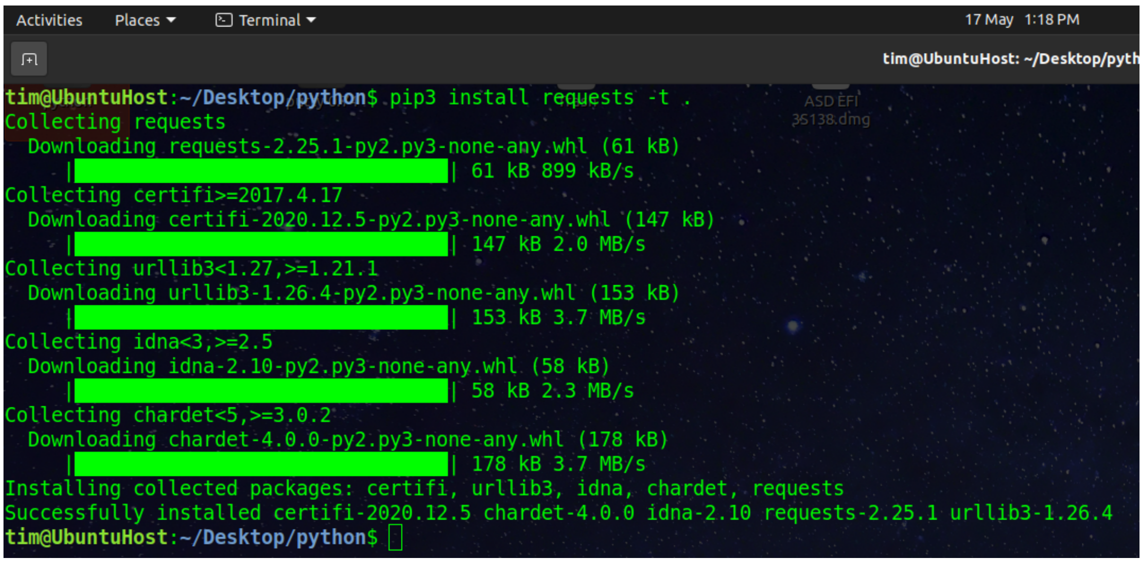
Figure A.14: Workflow diagram of Image Processing in Lambda function

the directory for making a layer. For layer, we have to use a specific directory and name for the layer and zip it and upload it in the layer. In AWS documentation they give us this directory **python/lib/python3.8/site-packages(site directories)**. Then we tried this in our Linux machine Ubuntu 20.04. Shown in figure: A.15.
Before all else, we installed "pip3" for downloading the libraries of Python3.++. Shown in figure: A.16.
After that, we needed AWS CLI to directly upload or deploy anything in our computer to AWS. Now, we need the directory where the libraries will be downloaded. Shown in figure: A.17.
After downloading we zipped the folder and tried to upload the zip file in a layer but in the layer, we can upload a maximum of 50MB size of the library. Shown in figure: A.18.
But our zipped folder size was more than 60MB, so we uploaded this in an S3 Bucket and then made the layer with that bucket object file. Shown in figure: A.19 and figure: A.20.
Then we created a function and add that layer made by us and import cv2 but we got the error that says "no modules named cv2". Shown in figure: A.21.
We thought this might be a version mismatch with the Linux version so we created an EC2 instance using 15GB of disk space and 2GB of ram which was included in the free tier of AWS. Shown in figure: A.22.
They price Ubuntu 20.04 LTS server to access that instance we have to take that instance terminal using any of our computers we cloud to do that with Linux terminal, MacOS terminal and windows command prompt also. For accessing that instance AWS provide a public IP of that instance and while creating the instance they provide a key which extension in " .pem". The key is to login into the instance without providing the AWS email and password. After accessing the instance we did the same thing as our ubuntu 20.04. Shown in figure: A.23.
After making the zip we can upload that from our instance to the S3 bucket for that we have to give permissions for both S3 and EC2 instance to do it otherwise



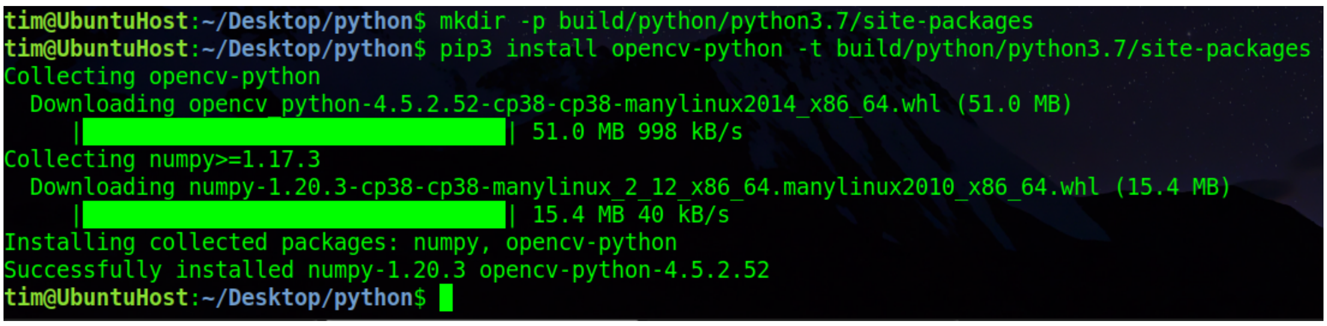
```
tim@UbuntuHost:~/Desktop/python/build$ tree
.
└── python
    └── lib
        └── python3.8
            └── site-packages
                ├── bin
                │   ├── f2py
                │   ├── f2py3
                │   └── f2py3.8
                ├── cv2
                │   ├── cv2.cpython-38-x86_64-linux-gnu.so
                │   ├── data
                │   │   ├── haarcascade_eye_tree_eyeglasses.xml
                │   │   ├── haarcascade_eye.xml
                │   │   ├── haarcascade_frontalcatface_extended.xml
                │   │   ├── haarcascade_frontalcatface.xml
                │   │   ├── haarcascade_frontalface_alt2.xml
                │   │   ├── haarcascade_frontalface_alt_tree.xml
                │   │   ├── haarcascade_frontalface_alt.xml
                │   │   ├── haarcascade_frontalface_default.xml
                │   │   ├── haarcascade_fullbody.xml
                │   │   ├── haarcascade_lefteye_2splits.xml
                │   │   ├── haarcascade_licence_plate_rus_16stages.xml
                │   │   ├── haarcascade_lowerbody.xml
                │   │   ├── haarcascade_profileface.xml
                │   │   ├── haarcascade_righteye_2splits.xml
                │   │   ├── haarcascade_russian_plate_number.xml
                │   │   ├── haarcascade_smile.xml
                │   │   ├── haarcascade_upperbody.xml
                │   │   ├── __init__.py
                │   │   └── __pycache__
                │   │       └── __init__.cpython-38.pyc
                │   ├── __init__.py
                │   ├── LICENSE-3RD-PARTY.txt
                │   ├── LICENSE.txt
                │   ├── __pycache__
                │   │   ├── __init__.cpython-38.pyc
                │   │   └── version.cpython-38.pyc
                │   ├── qt
                │   │   ├── fonts
                │   │   │   ├── DejaVuSans-BoldOblique.ttf
```

Figure A.15: Layer directory for openCV



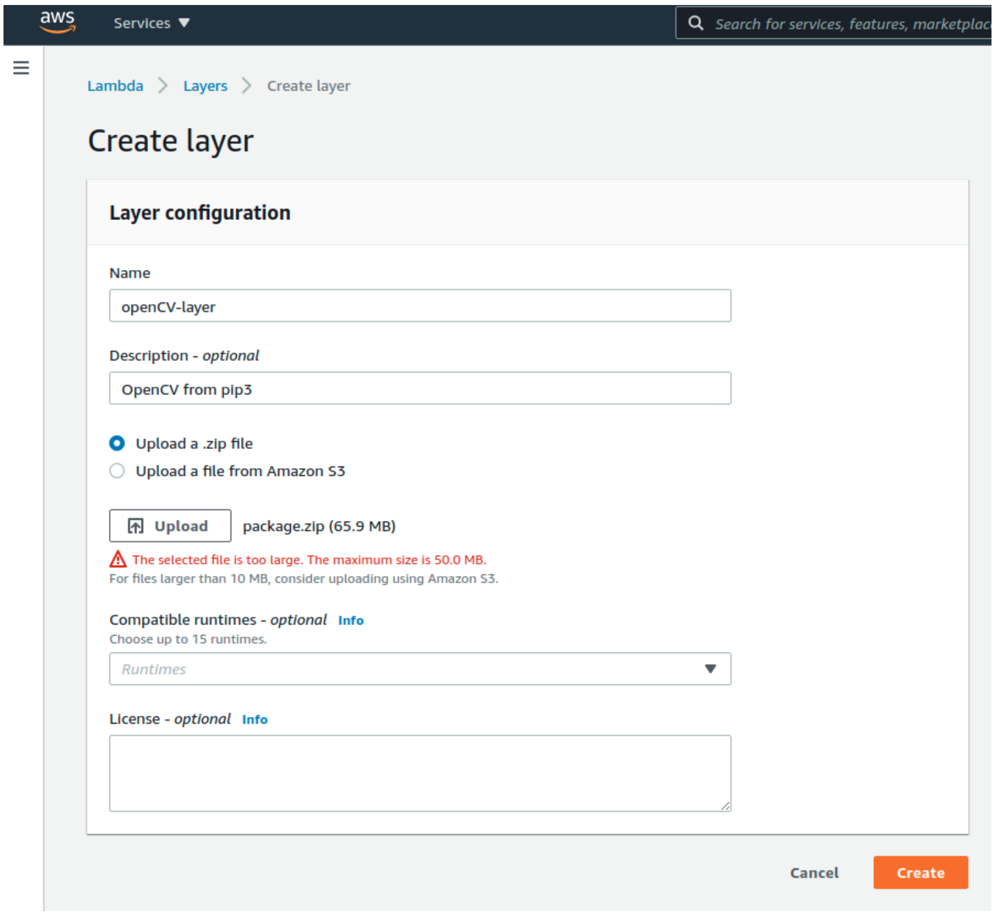
```
tim@UbuntuHost:~/Desktop/python$ pip3 install requests -t .
Collecting requests
  Downloading requests-2.25.1-py2.py3-none-any.whl (61 kB)
     |████████████████████████████████| 61 kB 899 kB/s
Collecting certifi>=2017.4.17
  Downloading certifi-2020.12.5-py2.py3-none-any.whl (147 kB)
     |████████████████████████████████| 147 kB 2.0 MB/s
Collecting urllib3<1.27,>=1.21.1
  Downloading urllib3-1.26.4-py2.py3-none-any.whl (153 kB)
     |████████████████████████████████| 153 kB 3.7 MB/s
Collecting idna<3,>=2.5
  Downloading idna-2.10-py2.py3-none-any.whl (58 kB)
     |████████████████████████████████| 58 kB 2.3 MB/s
Collecting chardet<5,>=3.0.2
  Downloading chardet-4.0.0-py2.py3-none-any.whl (178 kB)
     |████████████████████████████████| 178 kB 3.7 MB/s
Installing collected packages: certifi, urllib3, idna, chardet, requests
Successfully installed certifi-2020.12.5 chardet-4.0.0 idna-2.10 requests-2.25.1 urllib3-1.26.4
tim@UbuntuHost:~/Desktop/python$ 
```

Figure A.16: Installation of pip3



```
tim@UbuntuHost:~/Desktop/python$ mkdir -p build/python/python3.7/site-packages
tim@UbuntuHost:~/Desktop/python$ pip3 install opencv-python -t build/python/python3.7/site-packages
Collecting opencv-python
  Downloading opencv_python-4.5.2.52-cp38-cp38-manylinux2014_x86_64.whl (51.0 MB)
     |████████████████████████████████| 51.0 MB 998 kB/s
Collecting numpy>=1.17.3
  Downloading numpy-1.20.3-cp38-cp38-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (15.4 MB)
     |████████████████████████████████| 15.4 MB 40 kB/s
Installing collected packages: numpy, opencv-python
Successfully installed numpy-1.20.3 opencv-python-4.5.2.52
tim@UbuntuHost:~/Desktop/python$ 
```

Figure A.17: Installing openCV in directory



Create layer

Layer configuration

Name

openCV-layer

Description - *optional*

OpenCV from pip3

Upload a .zip file [✓]

Upload a file from Amazon S3

Upload package.zip (65.9 MB)

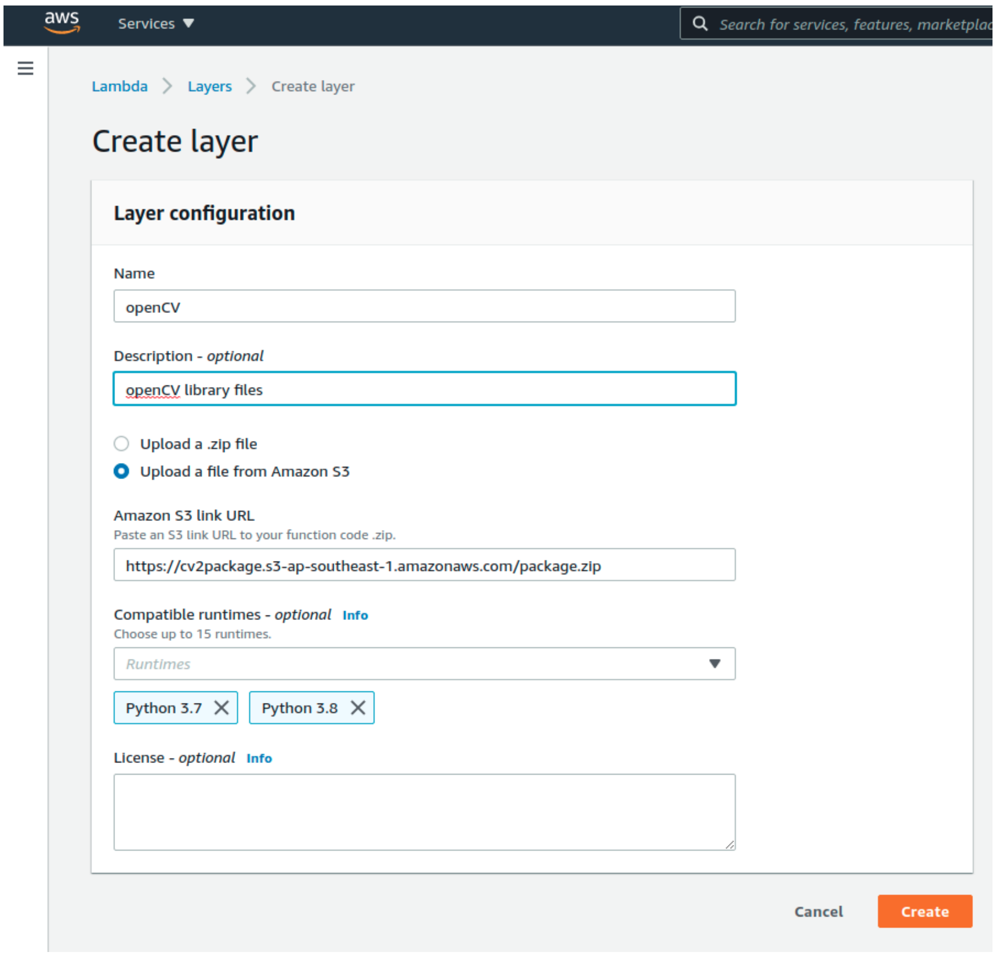
The selected file is too large. The maximum size is 50.0 MB.
For files larger than 10 MB, consider uploading using Amazon S3.

Compatible runtimes - *optional* Info
Choose up to 15 runtimes.

Runtimes

License - *optional* Info

Figure A.18: Lambda layer size limitation


aws
Services
Search for services, features, marketplac
Lambda > Layers > Create layer

Create layer

Layer configuration

Name
openCV

Description - *optional*
openCV library files

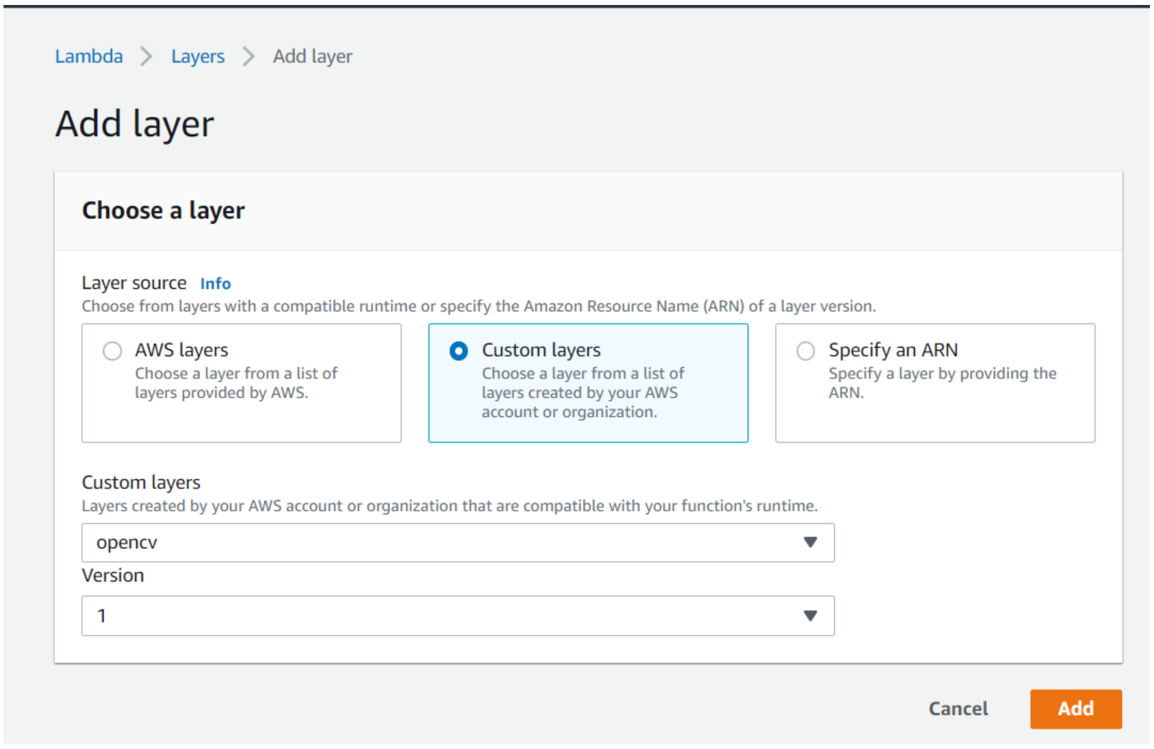
Upload a .zip file
[✓ Upload a file from Amazon S3]

Amazon S3 link URL
Paste an S3 link URL to your function code .zip.
https://cv2package.s3-ap-southeast-1.amazonaws.com/package.zip

Compatible runtimes - *optional* Info
Choose up to 15 runtimes.
Runtimes
Python 3.7 Python 3.8

License - *optional* Info

Figure A.19: Lambda layer from S3 bucket object



Add layer

Choose a layer

Layer source Info
Choose from layers with a compatible runtime or specify the Amazon Resource Name (ARN) of a layer version.

AWS layers
Choose a layer from a list of layers provided by AWS.

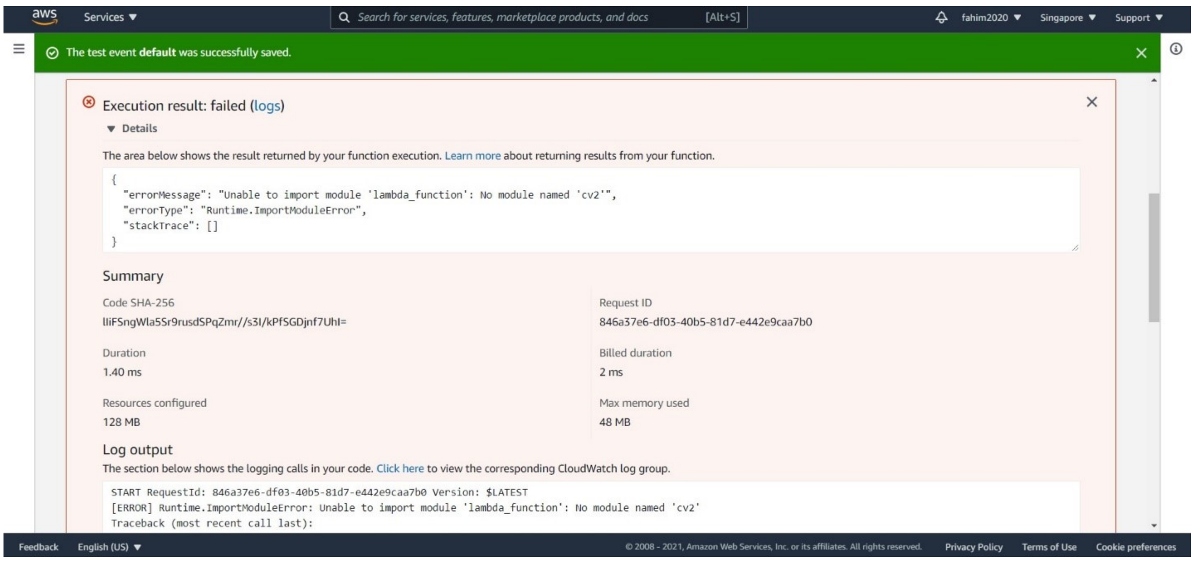
[✓ Custom layers]
Choose a layer from a list of layers created by your AWS account or organization.

Specify an ARN
Specify a layer by providing the ARN.

Custom layers
Layers created by your AWS account or organization that are compatible with your function's runtime.
opencv

Version
1

Figure A.20: Adding customized layer in the Lambda function


aws
Services
Search for services, features, marketplace products, and docs [Alt+S]
fahim2020
Singapore
Support

The test event **default** was successfully saved.

Execution result: failed (logs)

Details

The area below shows the result returned by your function execution. Learn more about returning results from your function.

```
{
  "errorMessage": "Unable to import module 'lambda_function': No module named 'cv2'",
  "errorType": "Runtime.ImportModuleError",
  "stackTrace": []
}
```

Summary

Code SHA-256	Request ID
lIiFSngWla5Sr9rusdSPqZmr//s3I/kPfSGDjnf7UhI=	846a37e6-df03-40b5-81d7-e442e9caa7b0
Duration	Billed duration
1.40 ms	2 ms
Resources configured	Max memory used
128 MB	48 MB

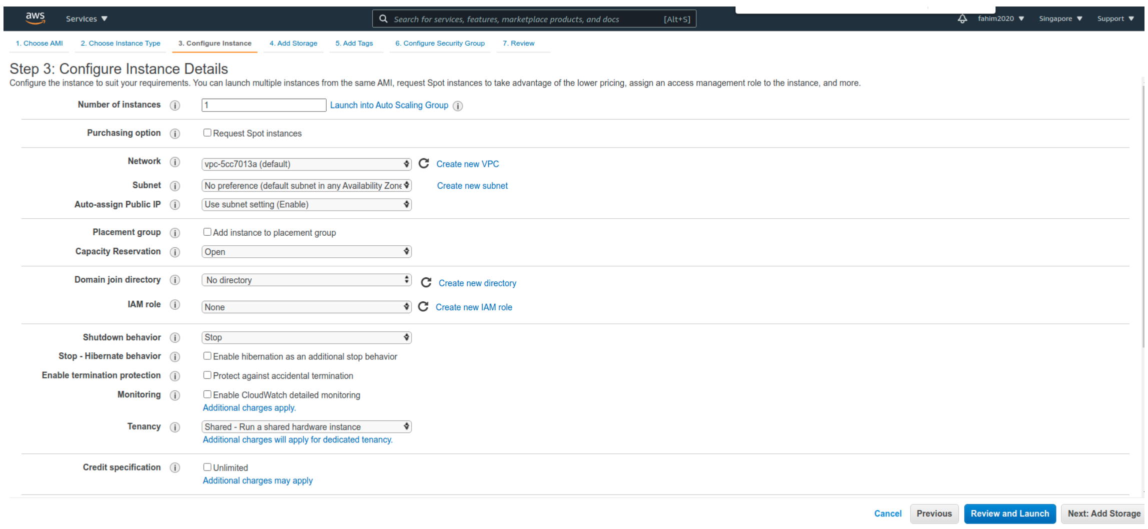
Log output

The section below shows the logging calls in your code. Click here to view the corresponding CloudWatch log group.

```
START RequestId: 846a37e6-df03-40b5-81d7-e442e9caa7b0 Version: $LATEST
[ERROR] Runtime.ImportModuleError: Unable to import module 'lambda_function': No module named 'cv2'
Traceback (most recent call last):
```

Feedback
English (US)
© 2008 - 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use
Cookie preferences

Figure A.21: Failed to import library from customized layer in Lambda

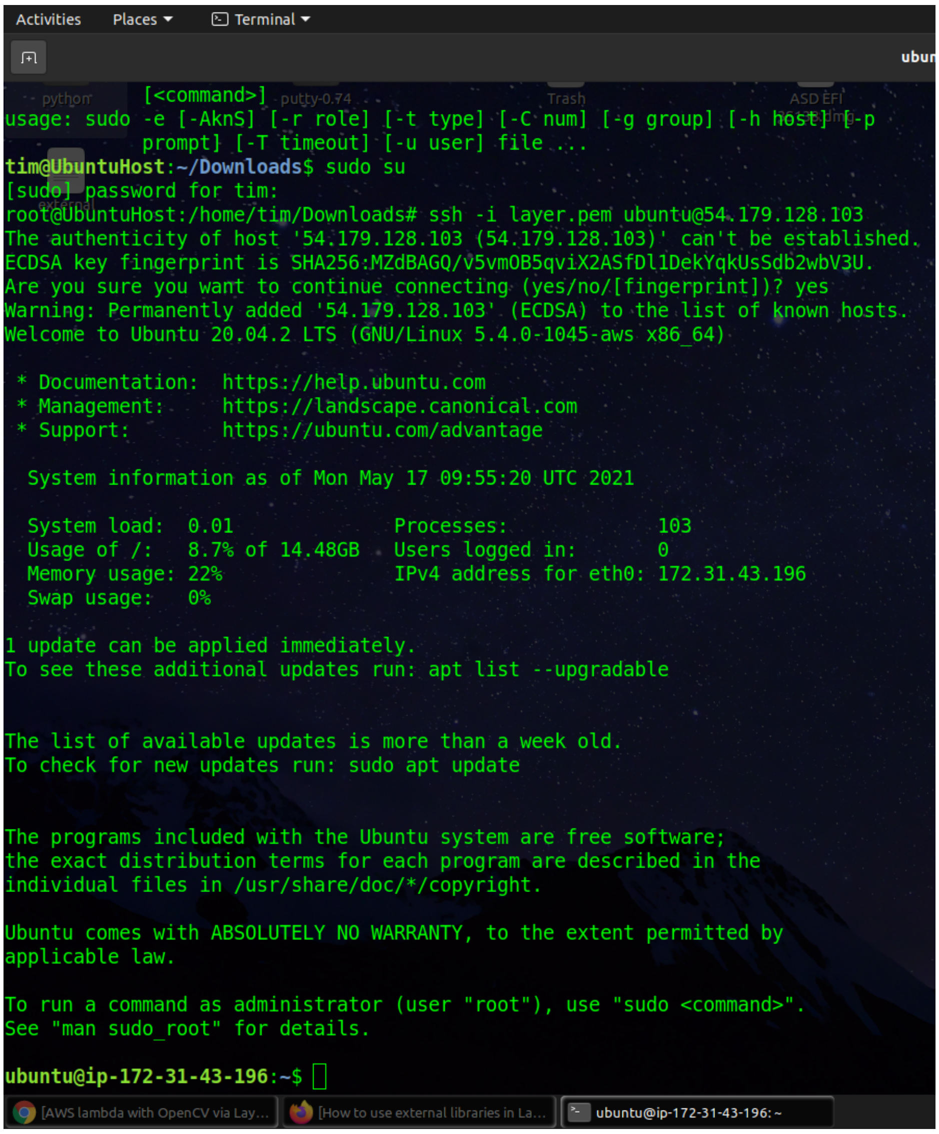


Step 3: Configure Instance Details

Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to take advantage of the lower pricing, assign an access management role to the instance, and more.

Number of instances	1 Launch into Auto Scaling Group
Purchasing option	☐ Request Spot instances
Network	vpc-5cc7013a (default) Create new VPC
Subnet	No preference (default subnet in any Availability Zone Create new subnet
Auto-assign Public IP	Use subnet setting (Enable)
Placement group	☐ Add instance to placement group
Capacity Reservation	Open
Domain join directory	No directory Create new directory
IAM role	None Create new IAM role
Shutdown behavior	Stop
Stop - Hibernate behavior	☐ Enable hibernation as an additional stop behavior
Enable termination protection	☐ Protect against accidental termination
Monitoring	☐ Enable CloudWatch detailed monitoring Additional charges apply.
Tenancy	Shared - Run a shared hardware instance Additional charges will apply for dedicated tenancy.
Credit specification	☐ Unlimited Additional charges may apply

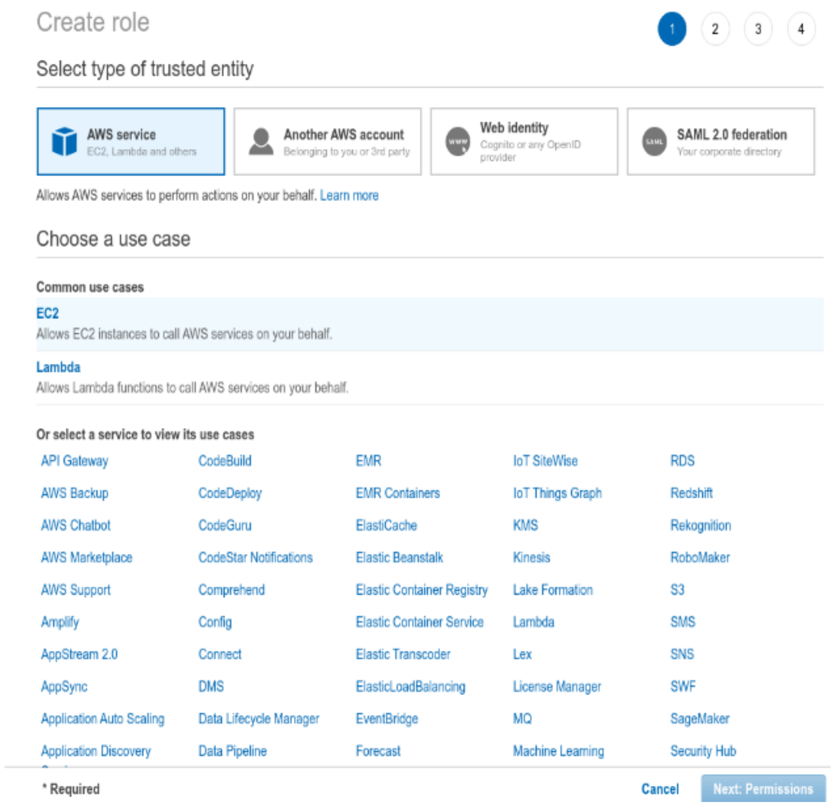
Figure A.22: Ubuntu20.04 server EC2 instance details


Activities
Places
Terminal
[AWS lambda with OpenCV via Lay...
[How to use external libraries in La...
ubuntu@ip-172-31-43-196: ~

```
            [<command>]
usage: sudo -e [-AknS] [-r role] [-t type] [-C num] [-g group] [-h host] [-p
            prompt] [-T timeout] [-u user] file ...
tim@UbuntuHost:~/Downloads$ sudo su
[sudo] password for tim:
root@UbuntuHost:/home/tim/Downloads# ssh -i layer.pem ubuntu@54.179.128.103
The authenticity of host '54.179.128.103 (54.179.128.103)' can't be established.
ECDSA key fingerprint is SHA256:MZdBAGQ/V5vmOB5qviX2ASfDl1DekYqkUsSdb2wbV3U.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '54.179.128.103' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 20.04.2 LTS (GNU/Linux 5.4.0-1045-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Mon May 17 09:55:20 UTC 2021

  System load:  0.01              Processes:             103
  Usage of /:   8.7% of 14.48GB   Users logged in:       0
  Memory usage: 22%               IPv4 address for eth0: 172.31.43.196
  Swap usage:   0%

1 update can be applied immediately.
To see these additional updates run: apt list --upgradable


The list of available updates is more than a week old.
To check for new updates run: sudo apt update


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-43-196:~$
```

Figure A.23: Taking over control of EC2 instance with IP .pem key in host machine terminal

AWS will block any kind of read or write permissions. That's why there is "I AM ROLE" with the help of that we can give permissions to read or write. Shown in figure: A.24 and figure: A.25.


Create role

1 2 3 4

Select type of trusted entity

AWS service EC2, Lambda and others | **Another AWS account** Belonging to you or 3rd party | **Web identity** Cognito or any OpenID provider | **SAML 2.0 federation** Your corporate directory

Allows AWS services to perform actions on your behalf. Learn more

Choose a use case

Common use cases

EC2
Allows EC2 instances to call AWS services on your behalf.

Lambda
Allows Lambda functions to call AWS services on your behalf.

Or select a service to view its use cases

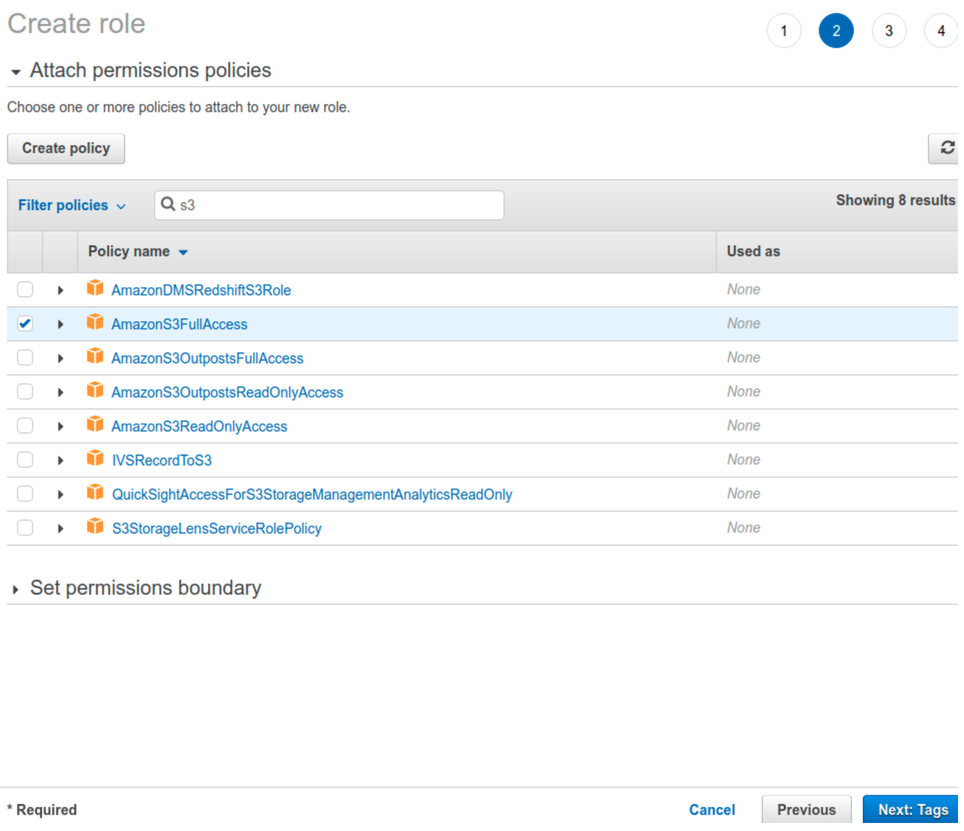
API Gateway	CodeBuild	EMR	IoT SiteWise	RDS
AWS Backup	CodeDeploy	EMR Containers	IoT Things Graph	Redshift
AWS Chatbot	CodeGuru	ElastiCache	KMS	Rekognition
AWS Marketplace	CodeStar Notifications	Elastic Beanstalk	Kinesis	RoboMaker
AWS Support	Comprehend	Elastic Container Registry	Lake Formation	S3
Amplify	Config	Elastic Container Service	Lambda	SMS
AppStream 2.0	Connect	Elastic Transcoder	Lex	SNS
AppSync	DMS	ElasticLoadBalancing	License Manager	SWF
Application Auto Scaling	Data Lifecycle Manager	EventBridge	MQ	SageMaker
Application Discovery	Data Pipeline	Forecast	Machine Learning	Security Hub

* Required

Cancel Next: Permissions

Figure A.24: Setting role in S3 bucket for permission

When the permission-giving is finished then we can upload that zip file directly to the S3 bucket using this upload command. Shown in figure: A.26.
Then we made a layer from that file but that didn't work at all and we tried for all the python versions. Finally, we moved to the containerized process. As we know that for the problem of version mismatch docker container has been invented. So one of the DevOps engineers from social media helps us to solve this layering problem. In the first place, we checked our docker version to confirm that we have Docker installed in our ubuntu Host machine. As we worked with Docker in pre-thesis 2 docker was installed. Then we made a directory **/tmp/mylayer** and in that directory, we made a requirement.txt file where we wrote OpenCV version 4.4.0.42 and wrote a bash command to download/pull the docker image **labci/lambda:build-python3.8** from docker repository. Shown in figure: A.27.
After pulling those image we run some bash commands mentioned in the screenshot. Shown in figure: A.28.
After completing all those things finally we got a zip folder named my layer than from that we made that layer and it worked.
Next, we started working on the image processing part. We wanted to make a simple image processing which is a colour image to greyscale image conversion. For this, we imported cv2 from our layer and NumPy from an already existing layer of AWS we also needed boto3 which is a library from AWS to accessing the s3 bucket. Here we imported an image from our bucket for importing that bucket we had to make an I


Create role

1 2 3 4

Attach permissions policies

Choose one or more policies to attach to your new role.

Create policy

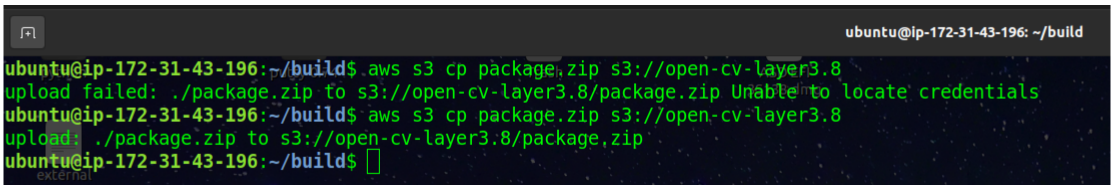
Filter policies s3 Showing 8 results

	Policy name	Used as
	AmazonDMSRedshiftS3Role	None
✓	AmazonS3FullAccess	None
	AmazonS3OutpostsFullAccess	None
	AmazonS3OutpostsReadOnlyAccess	None
	AmazonS3ReadOnlyAccess	None
	IVSRecordToS3	None
	QuickSightAccessForS3StorageManagementAnalyticsReadOnly	None
	S3StorageLensServiceRolePolicy	None

Set permissions boundary

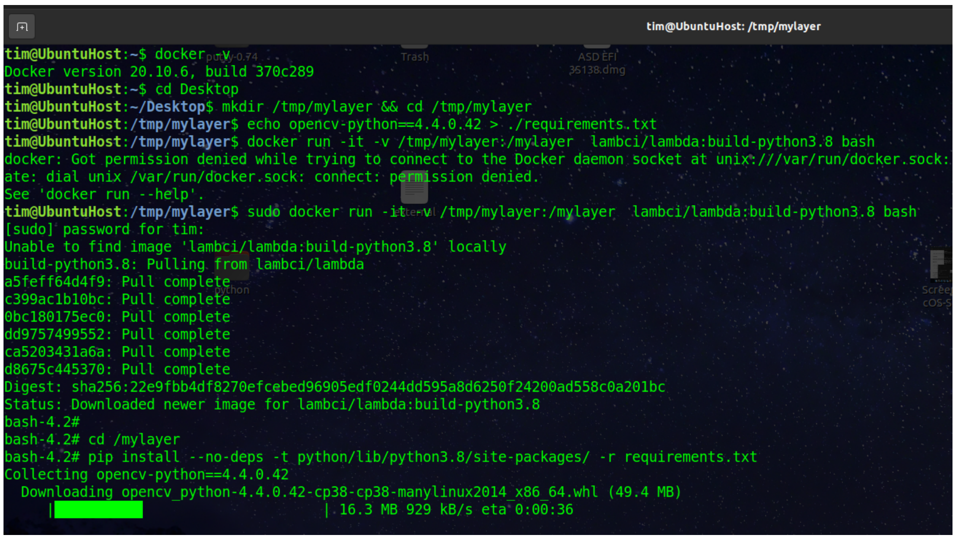
* Required

Figure A.25: Role creation for EC2 instance


ubuntu@ip-172-31-43-196: ~/build

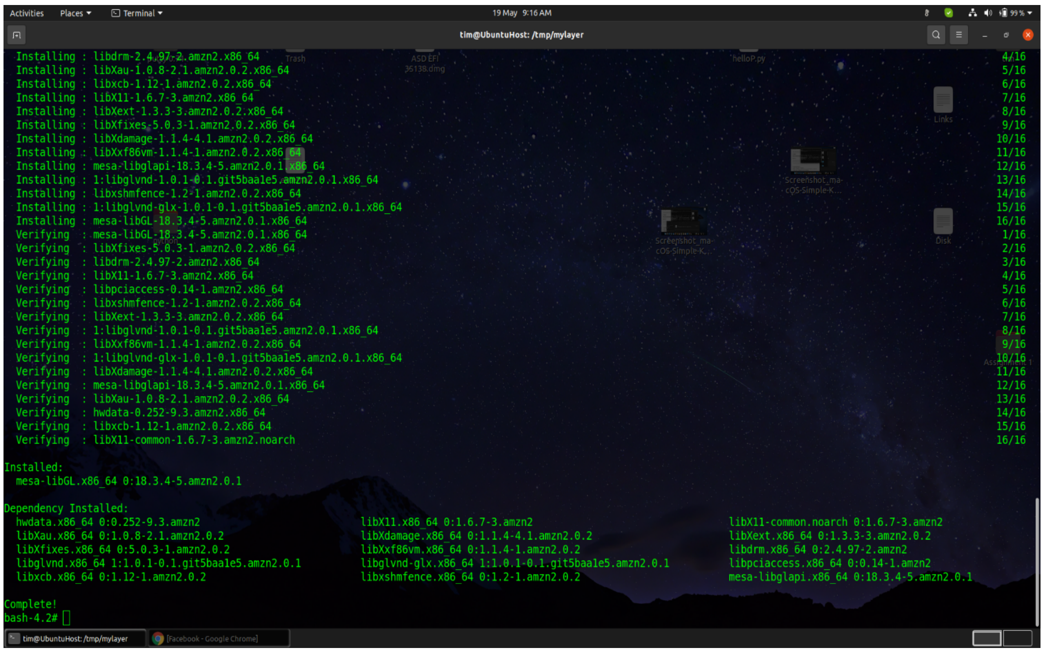
```
ubuntu@ip-172-31-43-196:~/build$ aws s3 cp package.zip s3://open-cv-layer3.8
upload failed: ./package.zip to s3://open-cv-layer3.8/package.zip Unable to locate credentials
ubuntu@ip-172-31-43-196:~/build$ aws s3 cp package.zip s3://open-cv-layer3.8
upload: ./package.zip to s3://open-cv-layer3.8/package.zip
ubuntu@ip-172-31-43-196:~/build$
```

Figure A.26: Uploading layer package from EC2 instance to S3 bucket


tim@UbuntuHost: /tmp/mylayer

```
tim@UbuntuHost:~$ docker -v
Docker version 20.10.6, build 370c289
tim@UbuntuHost:~$ cd Desktop
tim@UbuntuHost:~/Desktop$ mkdir /tmp/mylayer && cd /tmp/mylayer
tim@UbuntuHost:/tmp/mylayer$ echo opencv-python==4.4.0.42 > ./requirements.txt
tim@UbuntuHost:/tmp/mylayer$ docker run -it -v /tmp/mylayer:/mylayer  lambci/lambda:build-python3.8 bash
docker: Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock:
ate: dial unix /var/run/docker.sock: connect: permission denied.
See 'docker run --help'.
tim@UbuntuHost:/tmp/mylayer$ sudo docker run -it -v /tmp/mylayer:/mylayer  lambci/lambda:build-python3.8 bash
[sudo] password for tim:
Unable to find image 'lambci/lambda:build-python3.8' locally
build-python3.8: Pulling from lambci/lambda
a5feff64d4f9: Pull complete
c399ac1b10bc: Pull complete
0bc180175ec0: Pull complete
dd9757499552: Pull complete
ca5203431a6a: Pull complete
d8675c445370: Pull complete
Digest: sha256:22e9fbb4df8270efcebed96905edf0244dd595a8d6250f24200ad558c0a201bc
Status: Downloaded newer image for lambci/lambda:build-python3.8
bash-4.2#
bash-4.2# cd /mylayer
bash-4.2# pip install --no-deps -t python/lib/python3.8/site-packages/ -r requirements.txt
Collecting opencv-python==4.4.0.42
  Downloading opencv_python-4.4.0.42-cp38-cp38-manylinux2014_x86_64.whl (49.4 MB)
     |█████                              | 16.3 MB 929 kB/s eta 0:00:36
```

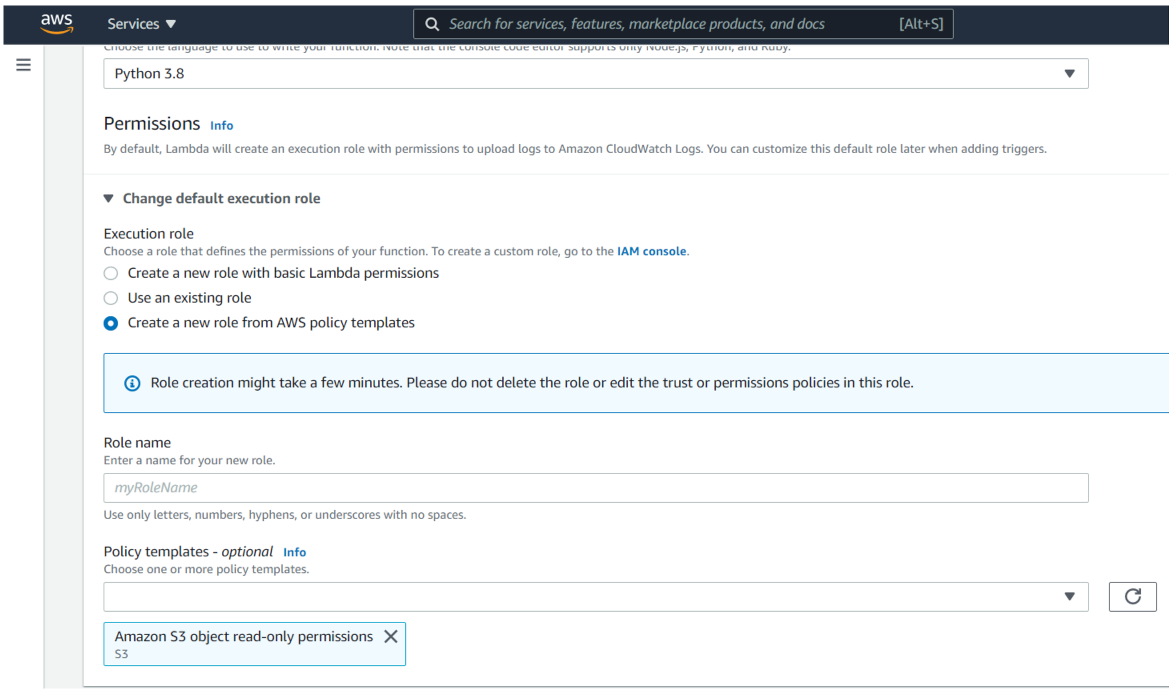
Figure A.27: Downloading library image for Python3.8 from Docker repository



```
  Installing : libdrm-2.4.97-2.amzn2.x86_64                                   4/16
  Installing : libXau-1.0.8-2.1.amzn2.0.2.x86_64                              5/16
  Installing : libxcb-1.12-1.amzn2.0.2.x86_64                                 6/16
  Installing : libX11-1.6.7-3.amzn2.x86_64                                    7/16
  Installing : libXext-1.3.3-3.amzn2.0.2.x86_64                               8/16
  Installing : libXfixes-5.0.3-1.amzn2.0.2.x86_64                             9/16
  Installing : libXdamage-1.1.4-4.1.amzn2.0.2.x86_64                         10/16
  Installing : libXxf86vm-1.1.4-1.amzn2.0.2.x86_64                           11/16
  Installing : mesa-libglapi-18.3.4-5.amzn2.0.1.x86_64                       12/16
  Installing : 1:libglvnd-1.0.1-0.1.git5baa1e5.amzn2.0.1.x86_64              13/16
  Installing : libxshmfence-1.2-1.amzn2.0.2.x86_64                           14/16
  Installing : 1:libglvnd-glx-1.0.1-0.1.git5baa1e5.amzn2.0.1.x86_64          15/16
  Installing : mesa-libGL-18.3.4-5.amzn2.0.1.x86_64                          16/16
  Verifying  : mesa-libGL-18.3.4-5.amzn2.0.1.x86_64                           1/16
  Verifying  : libXfixes-5.0.3-1.amzn2.0.2.x86_64                             2/16
  Verifying  : libdrm-2.4.97-2.amzn2.x86_64                                   3/16
  Verifying  : libX11-1.6.7-3.amzn2.x86_64                                    4/16
  Verifying  : libpciaccess-0.14-1.amzn2.x86_64                               5/16
  Verifying  : libxshmfence-1.2-1.amzn2.0.2.x86_64                            6/16
  Verifying  : libXext-1.3.3-3.amzn2.0.2.x86_64                               7/16
  Verifying  : 1:libglvnd-1.0.1-0.1.git5baa1e5.amzn2.0.1.x86_64               8/16
  Verifying  : libXxf86vm-1.1.4-1.amzn2.0.2.x86_64                            9/16
  Verifying  : 1:libglvnd-glx-1.0.1-0.1.git5baa1e5.amzn2.0.1.x86_64          10/16
  Verifying  : libXdamage-1.1.4-4.1.amzn2.0.2.x86_64                         11/16
  Verifying  : mesa-libglapi-18.3.4-5.amzn2.0.1.x86_64                       12/16
  Verifying  : libXau-1.0.8-2.1.amzn2.0.2.x86_64                             13/16
  Verifying  : hwdata-0.252-9.3.amzn2.x86_64                                 14/16
  Verifying  : libxcb-1.12-1.amzn2.0.2.x86_64                                15/16
  Verifying  : libX11-common-1.6.7-3.amzn2.noarch                            16/16

Installed:
  mesa-libGL.x86_64 0:18.3.4-5.amzn2.0.1

Dependency Installed:
  hwdata.x86_64 0:0.252-9.3.amzn2                      libX11.x86_64 0:1.6.7-3.amzn2                              libX11-common.noarch 0:1.6.7-3.amzn2
  libXau.x86_64 0:1.0.8-2.1.amzn2.0.2                  libXdamage.x86_64 0:1.1.4-4.1.amzn2.0.2                    libXext.x86_64 0:1.3.3-3.amzn2.0.2
  libXfixes.x86_64 0:5.0.3-1.amzn2.0.2                 libXxf86vm.x86_64 0:1.1.4-1.amzn2.0.2                      libdrm.x86_64 0:2.4.97-2.amzn2
  libglvnd.x86_64 1:1.0.1-0.1.git5baa1e5.amzn2.0.1     libglvnd-glx.x86_64 1:1.0.1-0.1.git5baa1e5.amzn2.0.1       libpciaccess.x86_64 0:0.14-1.amzn2
  libxcb.x86_64 0:1.12-1.amzn2.0.2                     libxshmfence.x86_64 0:1.2-1.amzn2.0.2                      mesa-libglapi.x86_64 0:18.3.4-5.amzn2.0.1

Complete!
bash-4.2#
```

Figure A.28: Installing library in our directory with bash command

am role to accessing the s3 bucket. We had to set the role in our Lambda function for accessing the s3bucket. Shown in figure: A.29.
We have to make roles for the s3 bucket also. We can add policies in the role. Shown in figure: A.30.
After adding the policies in the role and adding the role in the functions we can proceed with the code. Shown in figure: A.31.
Using boto3 we get an object from our bucket that we created and the image name was "color" with jpg extension then we read the file object and put it into a NumPy array and decoded that array in OpenCV image. Then we converted the image to grayscale using OpenCV. After conversion, we saved that image in python temporary writable directory(/tmp/). From that writeable directory, we put that image into the s3 bucket using boto3. Then we execute the function initially it failed to execute because our function had a time limit of 3 seconds which mean our function will automatically stop after 3 seconds. So all work in the code must be done within 3 seconds. On the first try, the is a init duration it takes for initializing the code for our case it takes 1129.41ms. Shown in figure: A.32.
After 1st try our function executes successfully and takes a total of 2751.43ms in the 2nd run it does not require the initialization time. So it was under 3 second. Now we can check our s3 bucket for the gray scaled image and we found that. Shown in figure: A.33 and figure: A.34.
Then we downloaded the image to see that. Shown in figure: A.35 and figure: A.36.
We execute the same function 5 times and took the values for the total execution time. Moreover, we can connect API getaway or any trigger to Lambda functions. Shown in figure: A.37.


aws
Services
Search for services, features, marketplace products, and docs [Alt+S]

Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Python 3.8

Permissions Info

By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▼ Change default execution role

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.

- ○ Create a new role with basic Lambda permissions
- ○ Use an existing role
- ● Create a new role from AWS policy templates

Role creation might take a few minutes. Please do not delete the role or edit the trust or permissions policies in this role.

Role name

Enter a name for your new role.

myRoleName

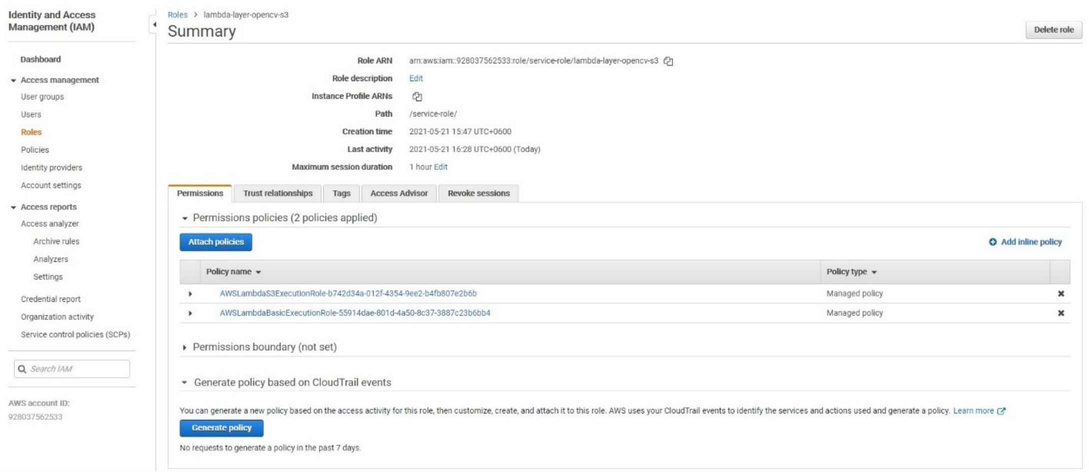
Use only letters, numbers, hyphens, or underscores with no spaces.

Policy templates - *optional* Info

Choose one or more policy templates.

Amazon S3 object read-only permissions
S3

Figure A.29: Setting S3 permission role in Lambda function



AWS account ID: 928037562533

Summary

Role ARN	arn:aws:iam::928037562533:role/service-role/lambda-layer-opencv-s3
Role description	Edit
Instance Profile ARNs	
Path	/service-role/
Creation time	2021-05-21 15:47 UTC+0600
Last activity	2021-05-21 16:28 UTC+0600 (Today)
Maximum session duration	1 hour Edit

Permissions policies (2 policies applied)

Policy name	Policy type
AWSLambdaS3ExecutionRole-b742d34a-012f-4354-9ee2-b4fb807e2b6b	Managed policy
AWSLambdaBasicExecutionRole-55914dae-801d-4a50-8c37-3887c23b6bb4	Managed policy

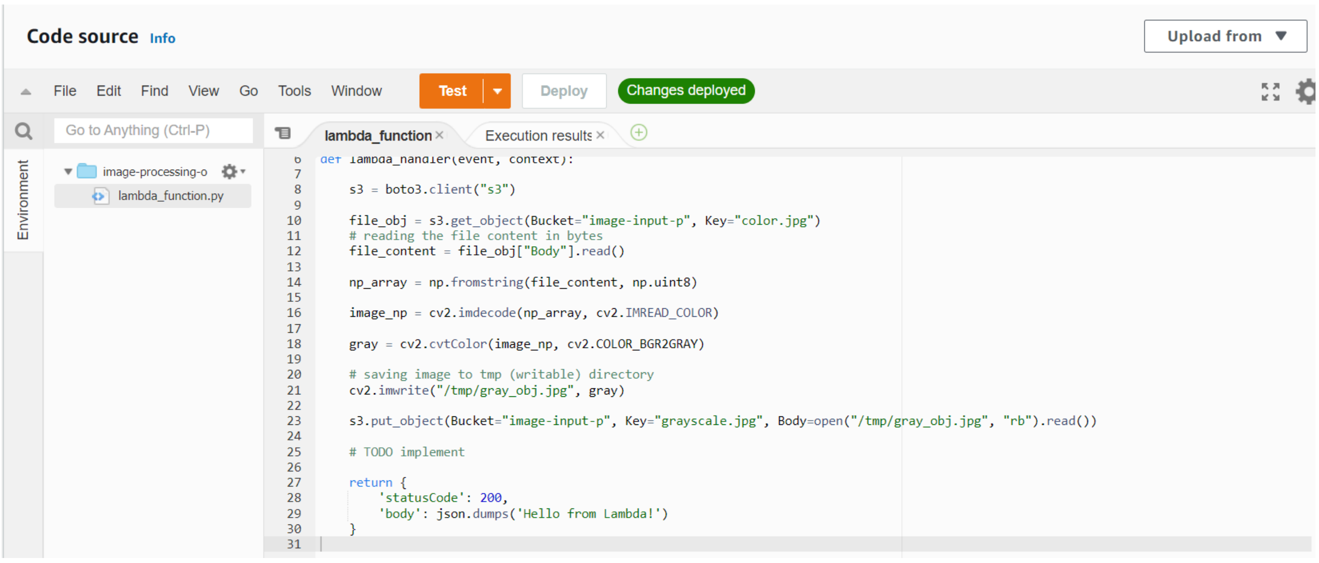
Permissions boundary (not set)

Generate policy based on CloudTrail events

You can generate a new policy based on the access activity for this role, then customize, create, and attach it to this role. AWS uses your CloudTrail events to identify the services and actions used and generate a policy.

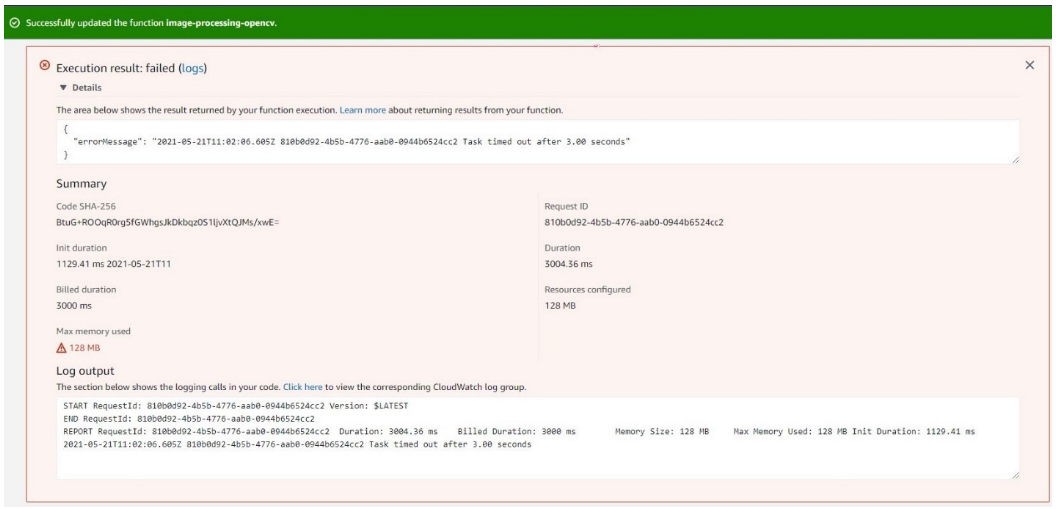
No requests to generate a policy in the past 7 days.

Figure A.30: Setting Lambda function execution policies in S3 bucket role



```python
def lambda_handler(event, context):

    s3 = boto3.client("s3")

    file_obj = s3.get_object(Bucket="image-input-p", Key="color.jpg")
    # reading the file content in bytes
    file_content = file_obj["Body"].read()

    np_array = np.fromstring(file_content, np.uint8)

    image_np = cv2.imdecode(np_array, cv2.IMREAD_COLOR)

    gray = cv2.cvtColor(image_np, cv2.COLOR_BGR2GRAY)

    # saving image to tmp (writable) directory
    cv2.imwrite("/tmp/gray_obj.jpg", gray)

    s3.put_object(Bucket="image-input-p", Key="grayscale.jpg", Body=open("/tmp/gray_obj.jpg", "rb").read())

    # TODO implement

    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }

```

Figure A.31: Source code of image processing in the Lambda function



Successfully updated the function **image-processing-opencv.**

Execution result: failed (logs)

Details

The area below shows the result returned by your function execution. Learn more about returning results from your function.

```
{
  "errorMessage": "2021-05-21T11:02:06.605Z 810b0d92-4b5b-4776-aab0-0944b6524cc2 Task timed out after 3.00 seconds"
}
```

Summary

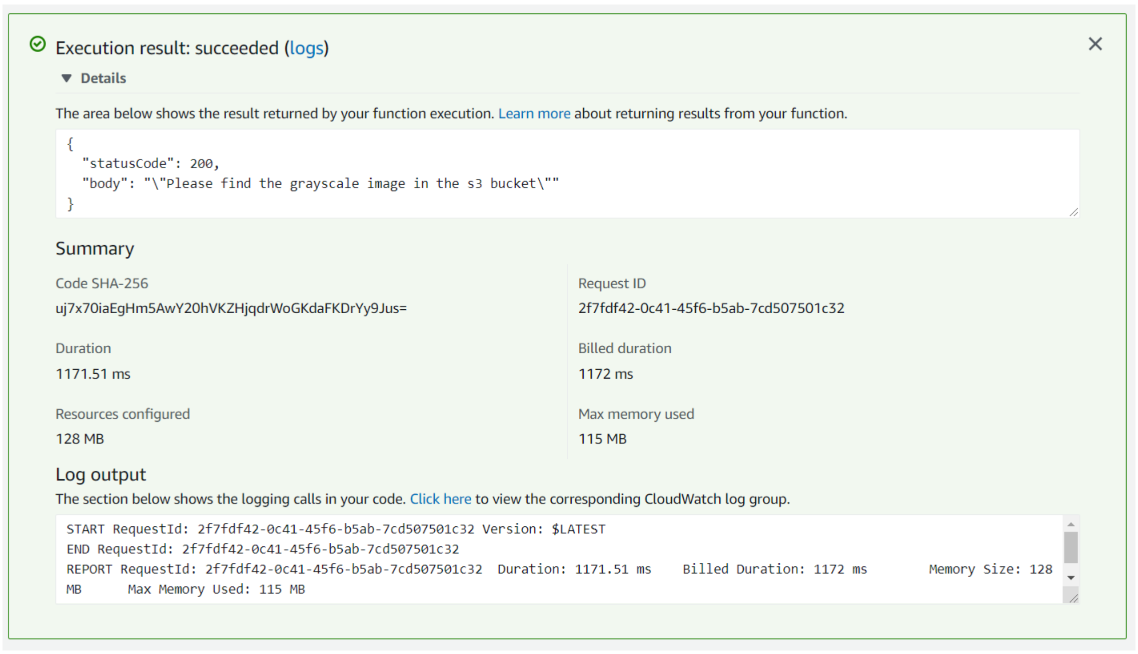
Code SHA-256	Request ID
BtuG+ROOqR0rg5fGWhgsJkDkbqz0S1ljvXtQJMs/xwE=	810b0d92-4b5b-4776-aab0-0944b6524cc2
Init duration	Duration
1129.41 ms 2021-05-21T11	3004.36 ms
Billed duration	Resources configured
3000 ms	128 MB
Max memory used	
⚠ 128 MB	

Log output

The section below shows the logging calls in your code. Click here to view the corresponding CloudWatch log group.

```
START RequestId: 810b0d92-4b5b-4776-aab0-0944b6524cc2 Version: $LATEST
END RequestId: 810b0d92-4b5b-4776-aab0-0944b6524cc2
REPORT RequestId: 810b0d92-4b5b-4776-aab0-0944b6524cc2  Duration: 3004.36 ms    Billed Duration: 3000 ms        Memory Size: 128 MB     Max Memory Used: 128 MB Init Duration: 1129.41 ms
2021-05-21T11:02:06.605Z 810b0d92-4b5b-4776-aab0-0944b6524cc2 Task timed out after 3.00 seconds
```

Figure A.32: Time limit error for initialize duration


Execution result: succeeded (logs)

Details

The area below shows the result returned by your function execution. Learn more about returning results from your function.

```
{
  "statusCode": 200,
  "body": "\"Please find the grayscale image in the s3 bucket\""
}
```

Summary

Code SHA-256	Request ID
uj7x70iaEgHm5AwY20hVKZHjqdrWoGKdaFKDrYy9Jus=	2f7fdf42-0c41-45f6-b5ab-7cd507501c32
Duration	Billed duration
1171.51 ms	1172 ms
Resources configured	Max memory used
128 MB	115 MB

Log output

The section below shows the logging calls in your code. Click here to view the corresponding CloudWatch log group.

```
START RequestId: 2f7fdf42-0c41-45f6-b5ab-7cd507501c32 Version: $LATEST
END RequestId: 2f7fdf42-0c41-45f6-b5ab-7cd507501c32
REPORT RequestId: 2f7fdf42-0c41-45f6-b5ab-7cd507501c32  Duration: 1171.51 ms    Billed Duration: 1172 ms        Memory Size: 128
MB      Max Memory Used: 115 MB
```

Figure A.33: Success log for image processing function execution

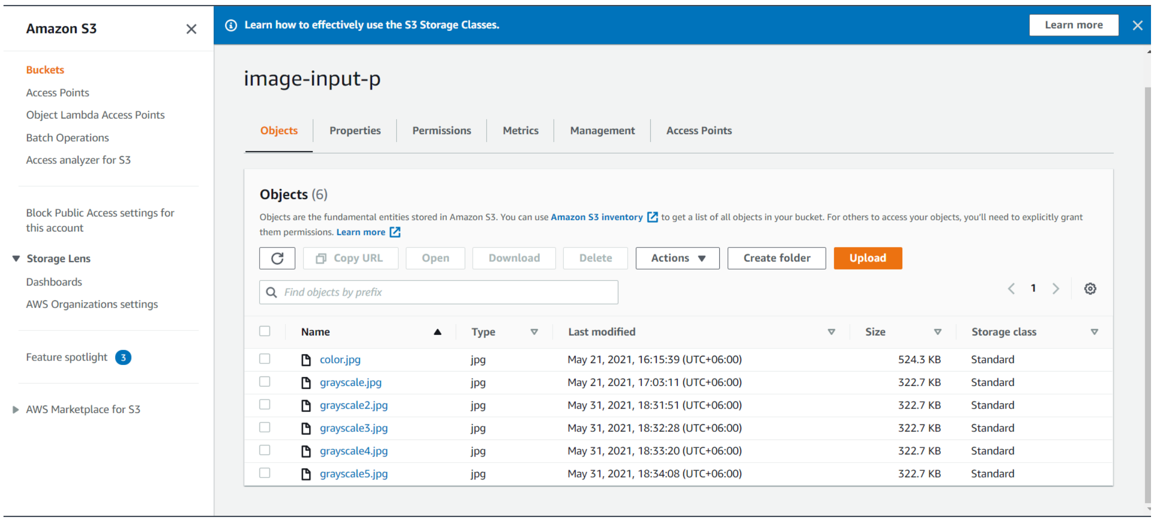


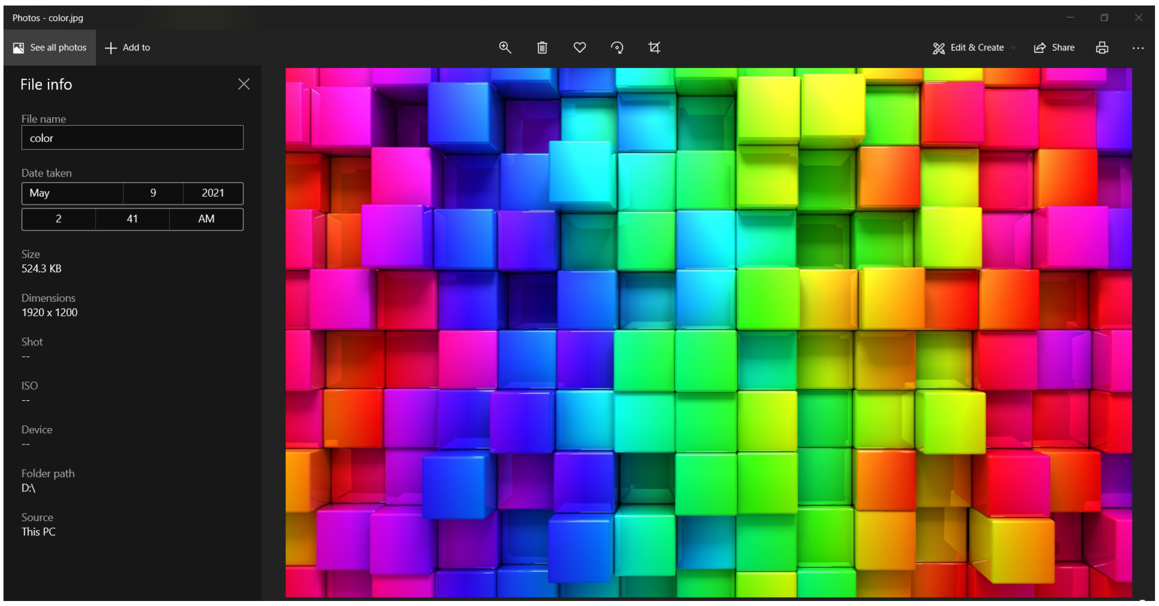
image-input-p

Objects (6)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
color.jpg	jpg	May 21, 2021, 16:15:39 (UTC+06:00)	524.3 KB	Standard
grayscale.jpg	jpg	May 21, 2021, 17:03:11 (UTC+06:00)	322.7 KB	Standard
grayscale2.jpg	jpg	May 31, 2021, 18:31:51 (UTC+06:00)	322.7 KB	Standard
grayscale3.jpg	jpg	May 31, 2021, 18:32:28 (UTC+06:00)	322.7 KB	Standard
grayscale4.jpg	jpg	May 31, 2021, 18:33:20 (UTC+06:00)	322.7 KB	Standard
grayscale5.jpg	jpg	May 31, 2021, 18:34:08 (UTC+06:00)	322.7 KB	Standard

Figure A.34: Gray-Scale images processed by the Lambda function



File info

File name
color

Date taken
May 9 2021
2 41 AM

Size
524.3 KB

Dimensions
1920 x 1200

Shot
--

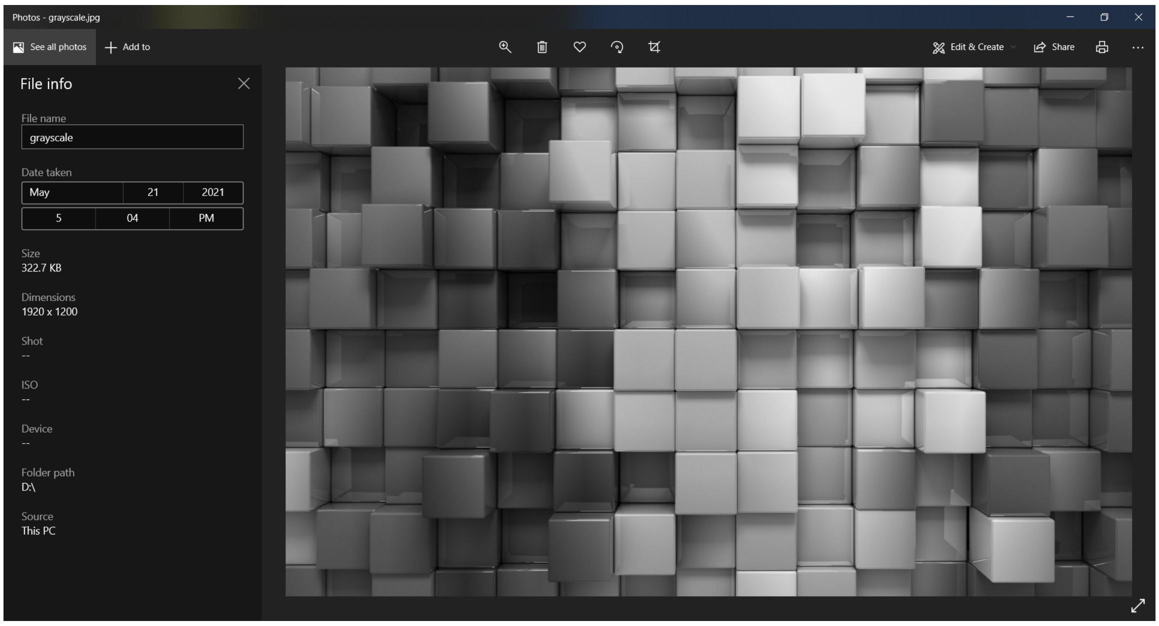
ISO
--

Device
--

Folder path
D:\

Source
This PC

Figure A.35: Input Color Image



File info

File name
grayscale

Date taken
May 21 2021
5 04 PM

Size
322.7 KB

Dimensions
1920 x 1200

Shot
--

ISO
--

Device
--

Folder path
D:\

Source
This PC

Figure A.36: Output Gray-scale Image

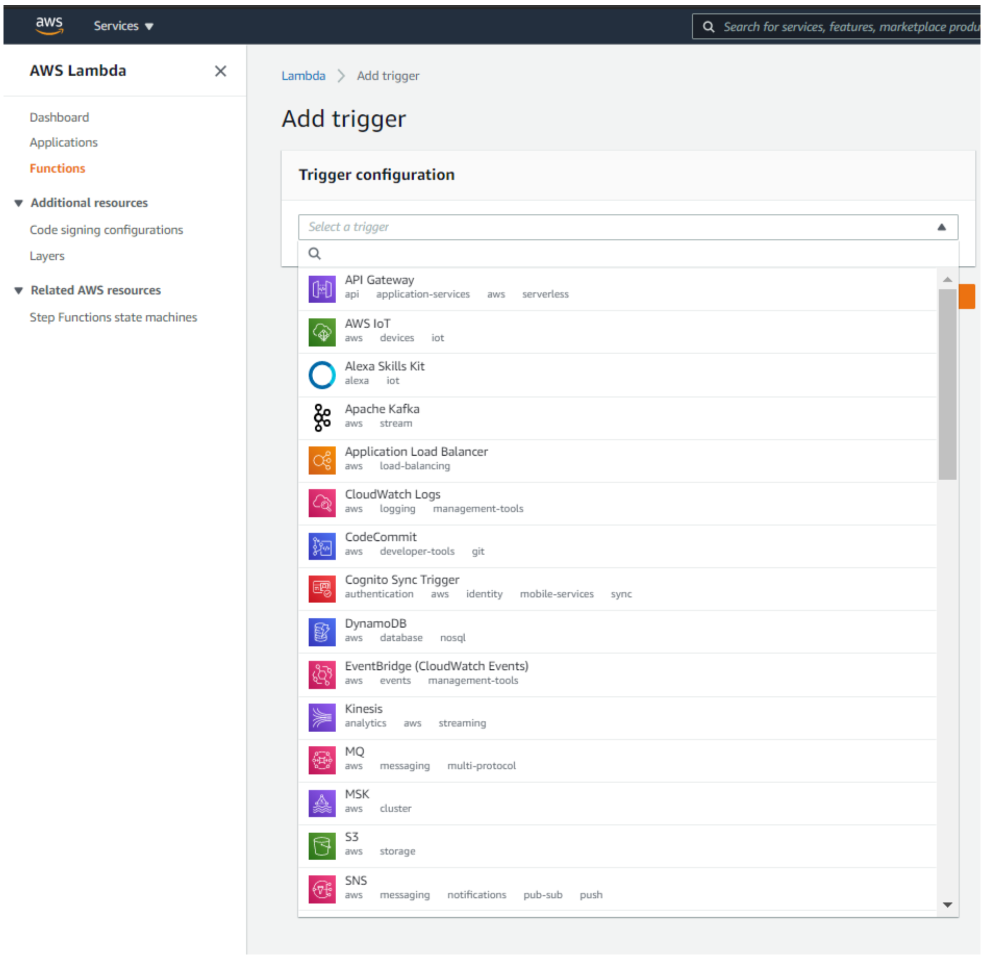
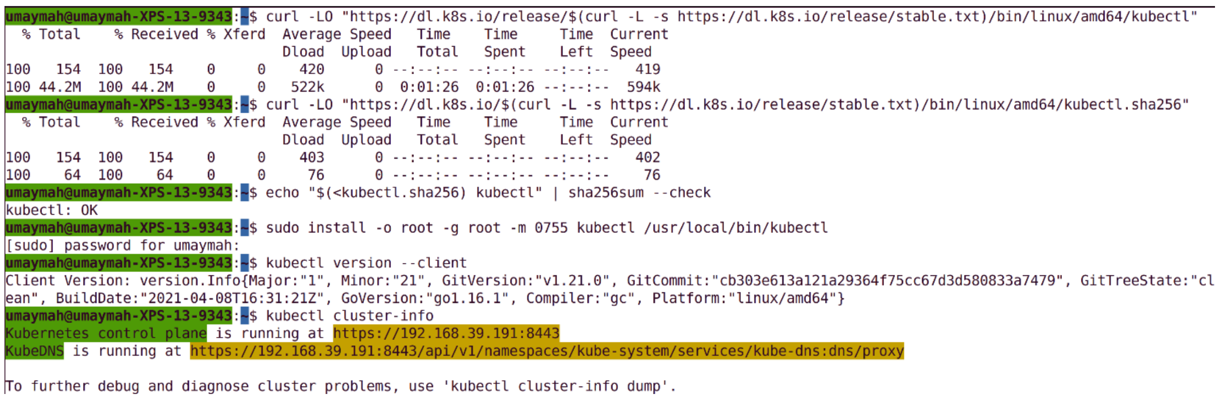


Figure A.37: Various type of trigger options in Lambda function

A.3 Fission:

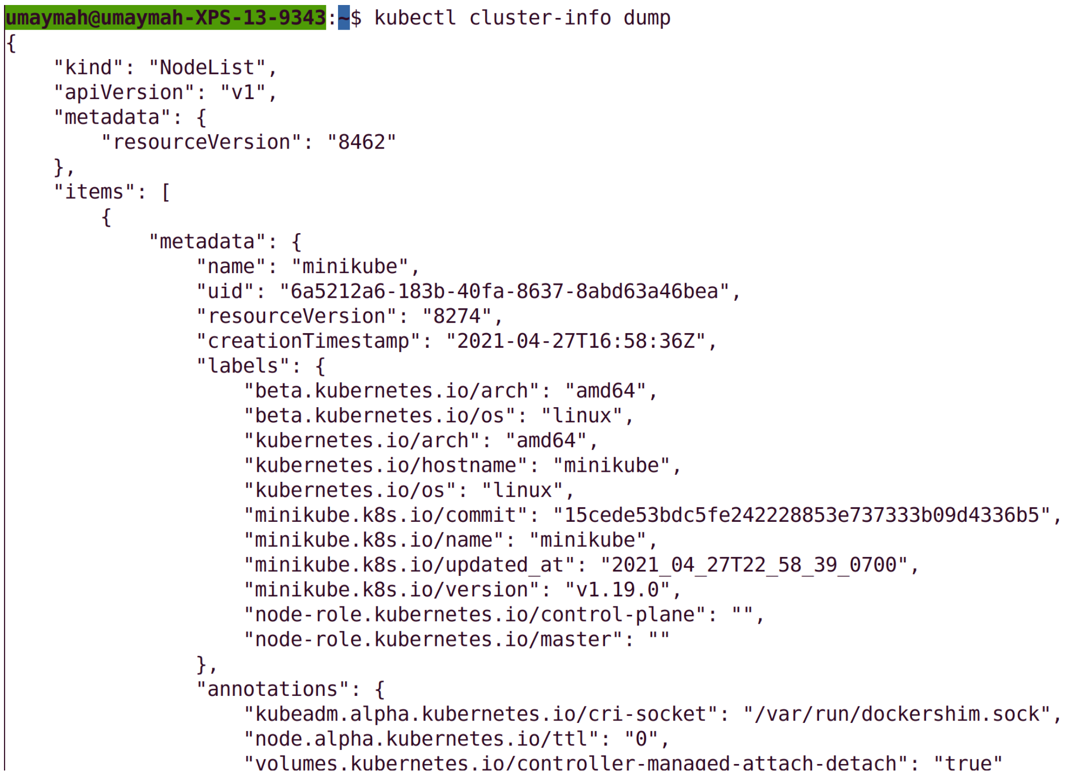
For working in fission we need to have detailed knowledge of virtualization, docker container, Kubernetes and other tools like kubectl and helm. This part is most important in our thesis work because we had to know about Kubernetes in details. In the first place, we installed Kubernetes and we knew that Kubernetes is mainly working with Docker container what Kubernetes does is Kubernetes manages the docker container automatically. To make it work in our host Linux machine we needed a command-line tool which is called Kubectl tor Kubernetes to work with. For Kubectl we add the repository from where we can download the tool with the curl command. Then we install it from that repository with root access. Shown in figure: A.38.



```
umaymah@umaymah-XPS-13-9343:~$ curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   154  100   154    0     0    420      0 --:--:-- --:--:-- --:--:--   419
100 44.2M  100 44.2M    0     0   522k      0  0:01:26  0:01:26 --:--:--  594k
umaymah@umaymah-XPS-13-9343:~$ curl -LO "https://dl.k8s.io/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   154  100   154    0     0    403      0 --:--:-- --:--:-- --:--:--   402
100    64  100    64    0     0     76      0 --:--:-- --:--:-- --:--:--    76
umaymah@umaymah-XPS-13-9343:~$ echo "$(<kubectl.sha256) kubectl" | sha256sum --check
kubectl: OK
umaymah@umaymah-XPS-13-9343:~$ sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
[sudo] password for umaymah:
umaymah@umaymah-XPS-13-9343:~$ kubectl version --client
Client Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.0", GitCommit:"cb303e613a121a29364f75cc67d3d580833a7479", GitTreeState:"cl
ean", BuildDate:"2021-04-08T16:31:21Z", GoVersion:"go1.16.1", Compiler:"gc", Platform:"linux/amd64"}
umaymah@umaymah-XPS-13-9343:~$ kubectl cluster-info
Kubernetes control plane is running at https://192.168.39.191:8443
KubeDNS is running at https://192.168.39.191:8443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

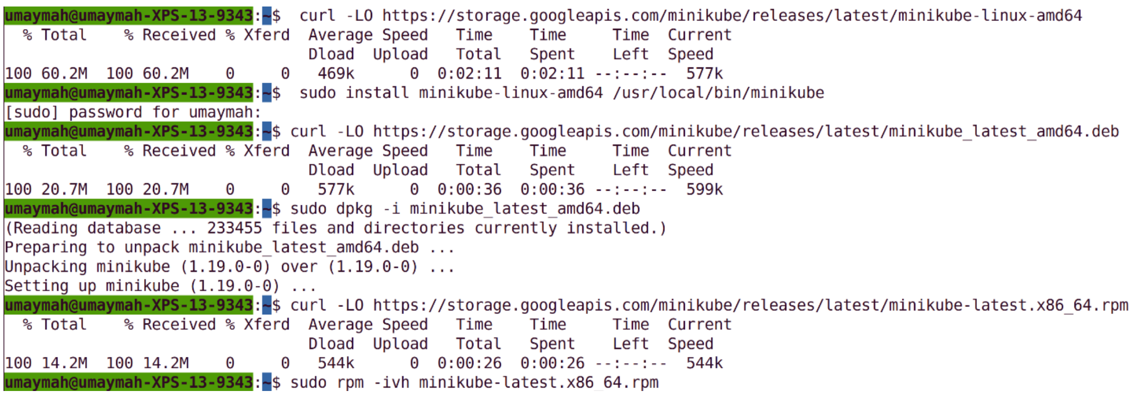
Figure A.38: Adding Kubectl repository and Installation

After installing Kubectl will make a cluster and we can see the cluster information with a simple command mention in the screenshot. Shown in figure: A.39.
For running the Kubernetes in our local host we used Minikube. And we also add the repository for Minikube in our Ubuntu app repository and installed it from there. Shown in figure: A.40.
We started our Minikube using the **minikube start** command. Then we can see the version of Minikube version and which driver it is using in which profile. Also it started contol palne node of minikube in cluser minkube. Then it updated the running kernel virtual machine 2 and its virtual machine which named "minikube". Then it prepares Kubertes(latest stable version) base on Docker (latest stable version). After those it verified Kubernetes components it enabled two addons storage-provisioner, default-storageclass. Shown in figure: A.41.
After that, we need to interact with Minikube with our cluster using two Kubectl command **kubectl get po -A** and **minikube kubectl - - get po –A**. By default, the cluster created with 2GB ram and 2cores of CPU. Shown in figure: A.42.
Now finishing interaction with the cluster finally it open into our browser. Shown in figure: A.43.
Here we can see the Kubernetes user interface. We can also start Minikube with specific Kubernetes versions. Shown in figure: A.44.
Now it's time to install fission because we had done the prerequisites. To begin with we need to check our Kubectls version. And our Kubectl seems fine with the unique gitCommit ID, compiler name, server version and clean GitTreeState which means we proceed next steps. Shown in figure: A.45.



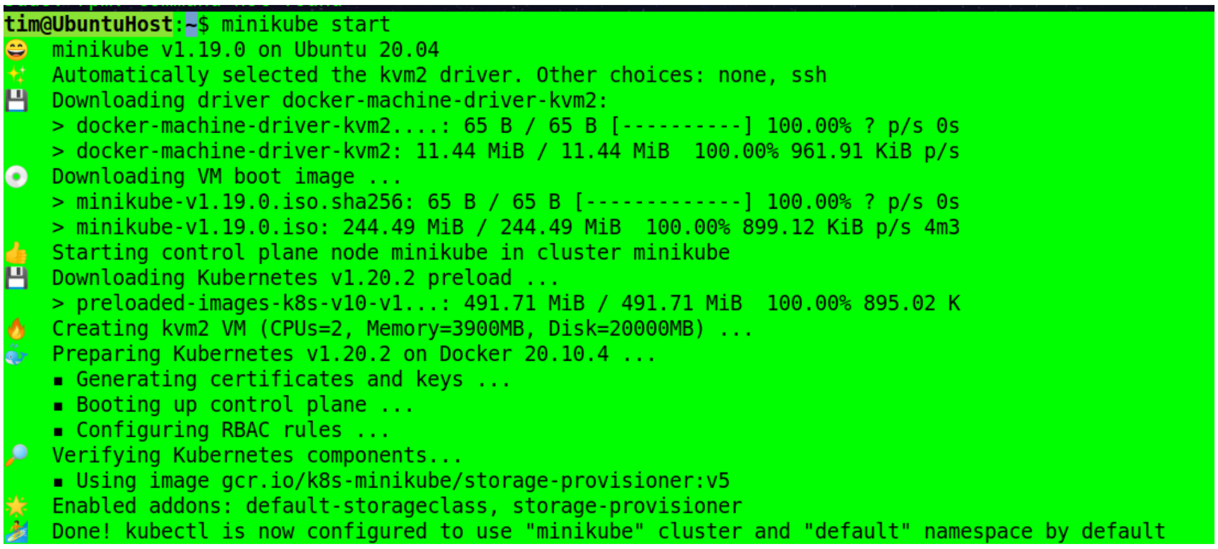
```
umaymah@umaymah-XPS-13-9343:~$ kubectl cluster-info dump
{
    "kind": "NodeList",
    "apiVersion": "v1",
    "metadata": {
        "resourceVersion": "8462"
    },
    "items": [
        {
            "metadata": {
                "name": "minikube",
                "uid": "6a5212a6-183b-40fa-8637-8abd63a46bea",
                "resourceVersion": "8274",
                "creationTimestamp": "2021-04-27T16:58:36Z",
                "labels": {
                    "beta.kubernetes.io/arch": "amd64",
                    "beta.kubernetes.io/os": "linux",
                    "kubernetes.io/arch": "amd64",
                    "kubernetes.io/hostname": "minikube",
                    "kubernetes.io/os": "linux",
                    "minikube.k8s.io/commit": "15cede53bdc5fe242228853e737333b09d4336b5",
                    "minikube.k8s.io/name": "minikube",
                    "minikube.k8s.io/updated_at": "2021_04_27T22_58_39_0700",
                    "minikube.k8s.io/version": "v1.19.0",
                    "node-role.kubernetes.io/control-plane": "",
                    "node-role.kubernetes.io/master": ""
                },
                "annotations": {
                    "kubeadm.alpha.kubernetes.io/cri-socket": "/var/run/dockershim.sock",
                    "node.alpha.kubernetes.io/ttl": "0",
                    "volumes.kubernetes.io/controller-managed-attach-detach": "true"
```

Figure A.39: Verifying Kubectl cluster information



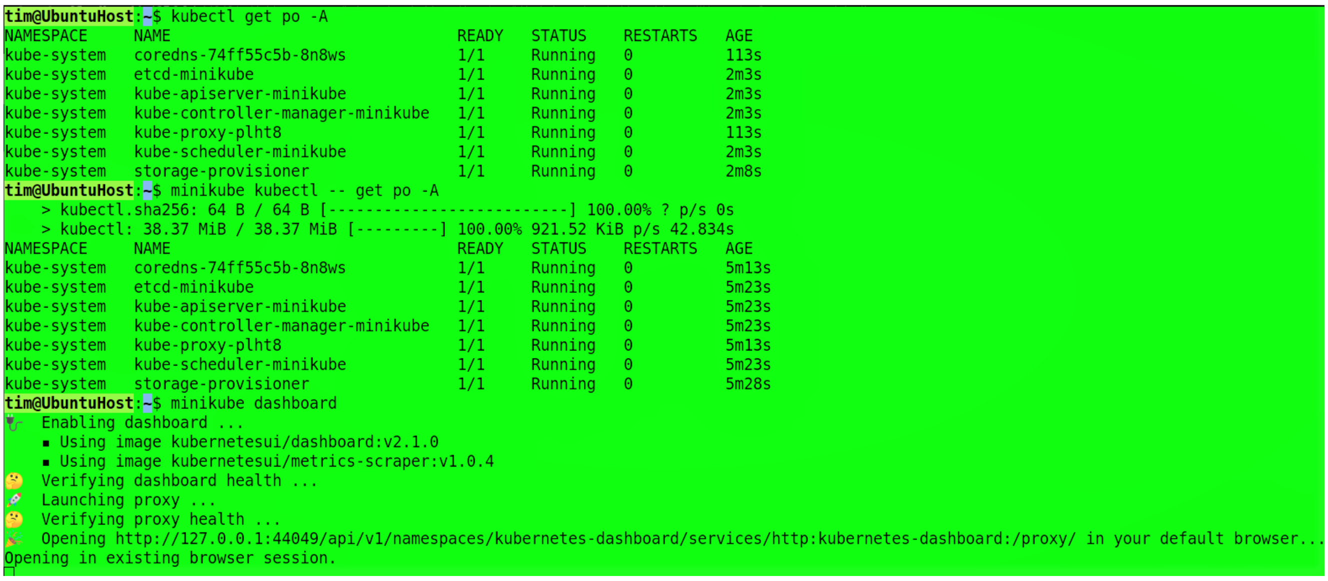
```
umaymah@umaymah-XPS-13-9343:~$  curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 60.2M  100 60.2M    0     0   469k      0  0:02:11  0:02:11 --:--:--  577k
umaymah@umaymah-XPS-13-9343:~$  sudo install minikube-linux-amd64 /usr/local/bin/minikube
[sudo] password for umaymah:
umaymah@umaymah-XPS-13-9343:~$ curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube_latest_amd64.deb
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 20.7M  100 20.7M    0     0   577k      0  0:00:36  0:00:36 --:--:--  599k
umaymah@umaymah-XPS-13-9343:~$ sudo dpkg -i minikube_latest_amd64.deb
(Reading database ... 233455 files and directories currently installed.)
Preparing to unpack minikube_latest_amd64.deb ...
Unpacking minikube (1.19.0-0) over (1.19.0-0) ...
Setting up minikube (1.19.0-0) ...
umaymah@umaymah-XPS-13-9343:~$ curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-latest.x86_64.rpm
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 14.2M  100 14.2M    0     0   544k      0  0:00:26  0:00:26 --:--:--  544k
umaymah@umaymah-XPS-13-9343:~$ sudo rpm -ivh minikube-latest.x86_64.rpm
```

Figure A.40: Adding Minikube repository and Installation



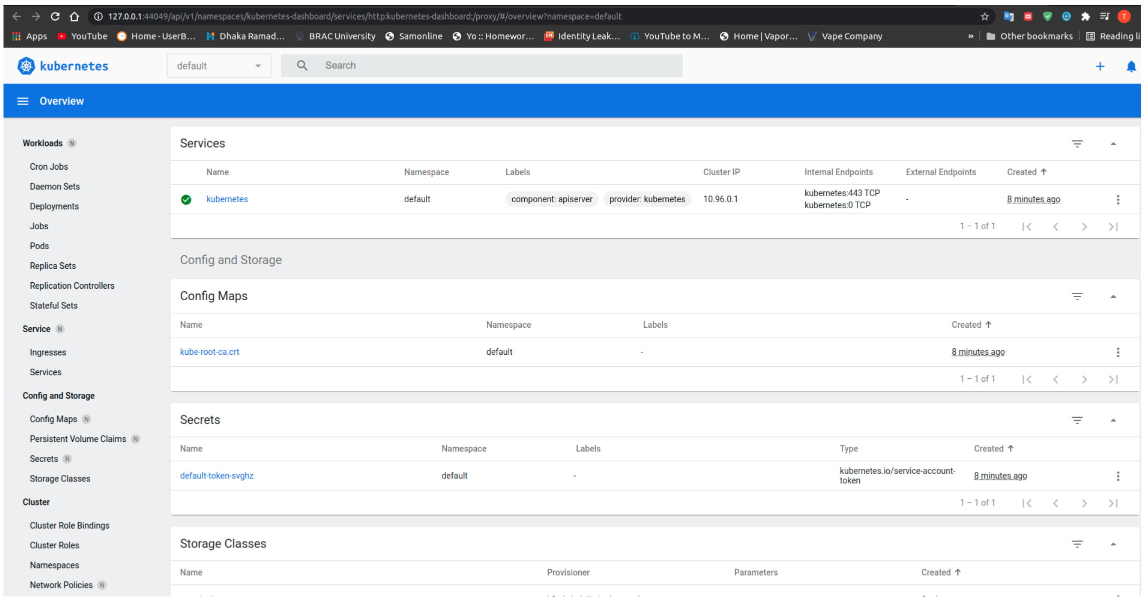
```
tim@UbuntuHost:~$ minikube start
😄  minikube v1.19.0 on Ubuntu 20.04
✨  Automatically selected the kvm2 driver. Other choices: none, ssh
💾  Downloading driver docker-machine-driver-kvm2:
    > docker-machine-driver-kvm2....: 65 B / 65 B [----------] 100.00% ? p/s 0s
    > docker-machine-driver-kvm2: 11.44 MiB / 11.44 MiB  100.00% 961.91 KiB p/s
💿  Downloading VM boot image ...
    > minikube-v1.19.0.iso.sha256: 65 B / 65 B [-------------] 100.00% ? p/s 0s
    > minikube-v1.19.0.iso: 244.49 MiB / 244.49 MiB  100.00% 899.12 KiB p/s 4m3
👍  Starting control plane node minikube in cluster minikube
💾  Downloading Kubernetes v1.20.2 preload ...
    > preloaded-images-k8s-v10-v1...: 491.71 MiB / 491.71 MiB  100.00% 895.02 K
🔥  Creating kvm2 VM (CPUs=2, Memory=3900MB, Disk=20000MB) ...
🐳  Preparing Kubernetes v1.20.2 on Docker 20.10.4 ...
    ▪ Generating certificates and keys ...
    ▪ Booting up control plane ...
    ▪ Configuring RBAC rules ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟  Enabled addons: default-storageclass, storage-provisioner
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Figure A.41: Downloading dependent components for starting Minikube



```
tim@UbuntuHost:~$ kubectl get po -A
NAMESPACE     NAME                               READY   STATUS    RESTARTS   AGE
kube-system   coredns-74ff55c5b-8n8ws            1/1     Running   0          113s
kube-system   etcd-minikube                      1/1     Running   0          2m3s
kube-system   kube-apiserver-minikube            1/1     Running   0          2m3s
kube-system   kube-controller-manager-minikube   1/1     Running   0          2m3s
kube-system   kube-proxy-plht8                   1/1     Running   0          113s
kube-system   kube-scheduler-minikube            1/1     Running   0          2m3s
kube-system   storage-provisioner                1/1     Running   0          2m8s
tim@UbuntuHost:~$ minikube kubectl -- get po -A
    > kubectl.sha256: 64 B / 64 B [--------------------------] 100.00% ? p/s 0s
    > kubectl: 38.37 MiB / 38.37 MiB [---------] 100.00% 921.52 KiB p/s 42.834s
NAMESPACE     NAME                               READY   STATUS    RESTARTS   AGE
kube-system   coredns-74ff55c5b-8n8ws            1/1     Running   0          5m13s
kube-system   etcd-minikube                      1/1     Running   0          5m23s
kube-system   kube-apiserver-minikube            1/1     Running   0          5m23s
kube-system   kube-controller-manager-minikube   1/1     Running   0          5m23s
kube-system   kube-proxy-plht8                   1/1     Running   0          5m13s
kube-system   kube-scheduler-minikube            1/1     Running   0          5m23s
kube-system   storage-provisioner                1/1     Running   0          5m28s
tim@UbuntuHost:~$ minikube dashboard
🔌  Enabling dashboard ...
    ▪ Using image kubernetesui/dashboard:v2.1.0
    ▪ Using image kubernetesui/metrics-scraper:v1.0.4
🤔  Verifying dashboard health ...
🚀  Launching proxy ...
🤔  Verifying proxy health ...
🎉  Opening http://127.0.0.1:44049/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/ in your default browser...
Opening in existing browser session.
```

Figure A.42: Interacting with the cluster



Services

Name	Namespace	Labels	Cluster IP	Internal Endpoints	External Endpoints	Created ↑
kubernetes	default	component: apiserver provider: kubernetes	10.96.0.1	kubernetes:443 TCP kubernetes:0 TCP	-	8 minutes ago

1 – 1 of 1

Config and Storage

Config Maps

Name	Namespace	Labels	Created ↑
kube-root-ca.crt	default	-	8 minutes ago

1 – 1 of 1

Secrets

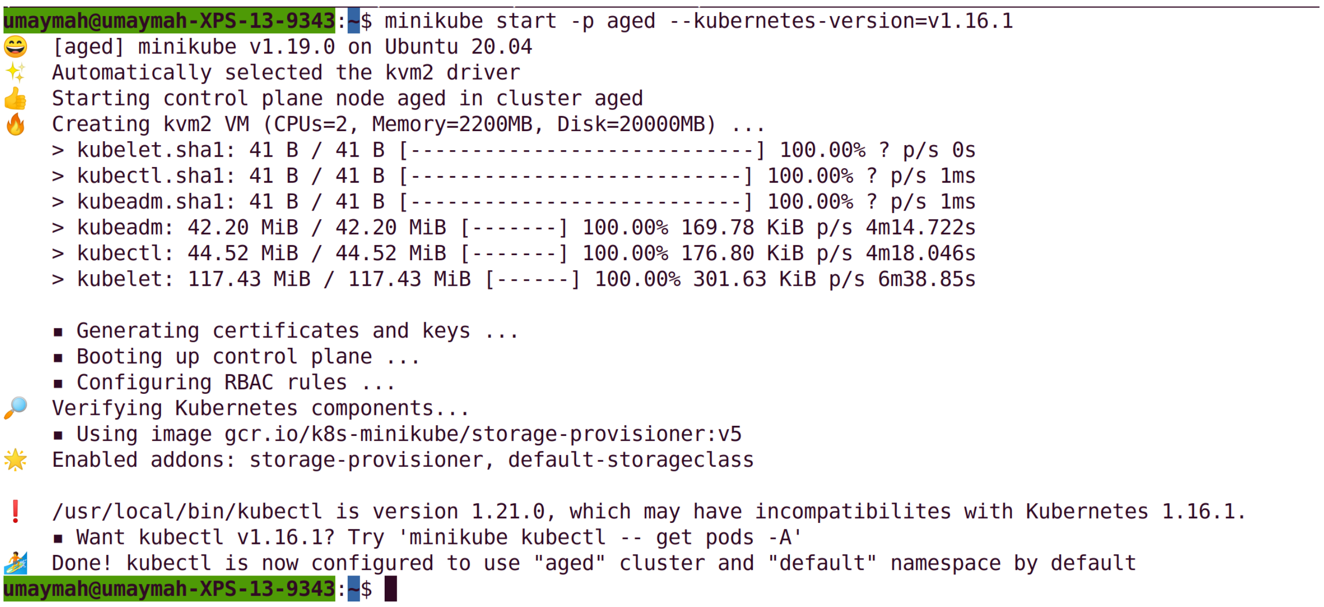
Name	Namespace	Labels	Type	Created ↑
default-token-svghz	default	-	kubernetes.io/service-account-token	8 minutes ago

1 – 1 of 1

Storage Classes

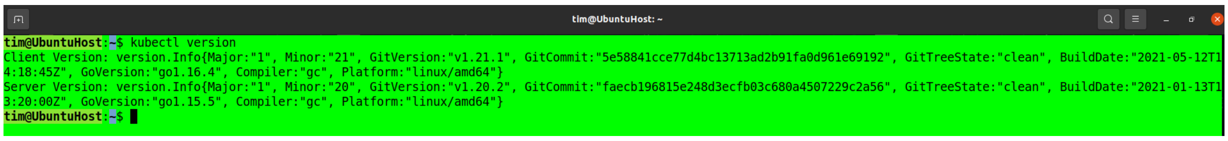
Name	Provisioner	Parameters	Created ↑

Figure A.43: Kubernetes (or Minikube) local host dashboard



```
umaymah@umaymah-XPS-13-9343:~$ minikube start -p aged --kubernetes-version=v1.16.1
😄  [aged] minikube v1.19.0 on Ubuntu 20.04
✨  Automatically selected the kvm2 driver
👍  Starting control plane node aged in cluster aged
🔥  Creating kvm2 VM (CPUs=2, Memory=2200MB, Disk=20000MB) ...
    > kubelet.sha1: 41 B / 41 B [----------------------------] 100.00% ? p/s 0s
    > kubectl.sha1: 41 B / 41 B [---------------------------] 100.00% ? p/s 1ms
    > kubeadm.sha1: 41 B / 41 B [---------------------------] 100.00% ? p/s 1ms
    > kubeadm: 42.20 MiB / 42.20 MiB [-------] 100.00% 169.78 KiB p/s 4m14.722s
    > kubectl: 44.52 MiB / 44.52 MiB [-------] 100.00% 176.80 KiB p/s 4m18.046s
    > kubelet: 117.43 MiB / 117.43 MiB [------] 100.00% 301.63 KiB p/s 6m38.85s

    ▪ Generating certificates and keys ...
    ▪ Booting up control plane ...
    ▪ Configuring RBAC rules ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟  Enabled addons: storage-provisioner, default-storageclass

❗  /usr/local/bin/kubectl is version 1.21.0, which may have incompatibilites with Kubernetes 1.16.1.
    ▪ Want kubectl v1.16.1? Try 'minikube kubectl -- get pods -A'
🏄  Done! kubectl is now configured to use "aged" cluster and "default" namespace by default
umaymah@umaymah-XPS-13-9343:~$ 
```

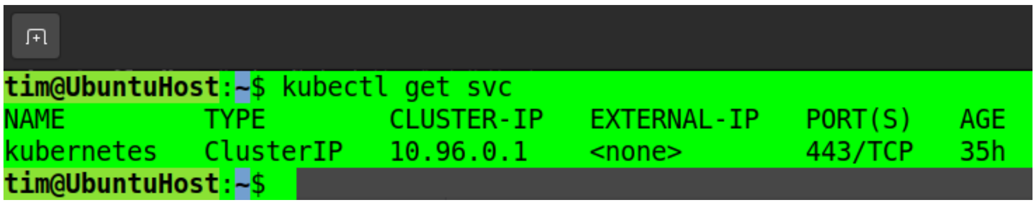
Figure A.44: Starting specific Kubernetes version



```
tim@UbuntuHost:~$ kubectl version
Client Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.1", GitCommit:"5e58841cce77d4bc13713ad2b91fa0d961e69192", GitTreeState:"clean", BuildDate:"2021-05-12T1
4:18:45Z", GoVersion:"go1.16.4", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"20", GitVersion:"v1.20.2", GitCommit:"faecb196815e248d3ecfb03c680a4507229c2a56", GitTreeState:"clean", BuildDate:"2021-01-13T1
3:20:00Z", GoVersion:"go1.15.5", Compiler:"gc", Platform:"linux/amd64"}
tim@UbuntuHost:~$ 
```

Figure A.45: Checking and verifying the Kubectl version

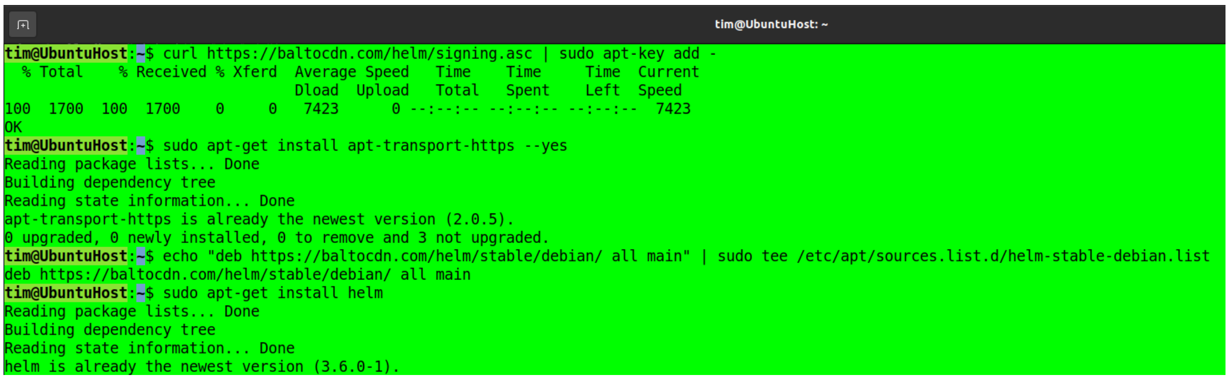
In Minikube it does not support LoadBlancer that's why instead of LoadBalancer we will be using NodePort. We checked the NodePort with this **kubectl get svc** command. Here we can see the cluster IP and port with others. Shown in figure: A.46.



```
tim@UbuntuHost:~$ kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   35h
tim@UbuntuHost:~$
```

Figure A.46: Checking the cluster and NodePort IP

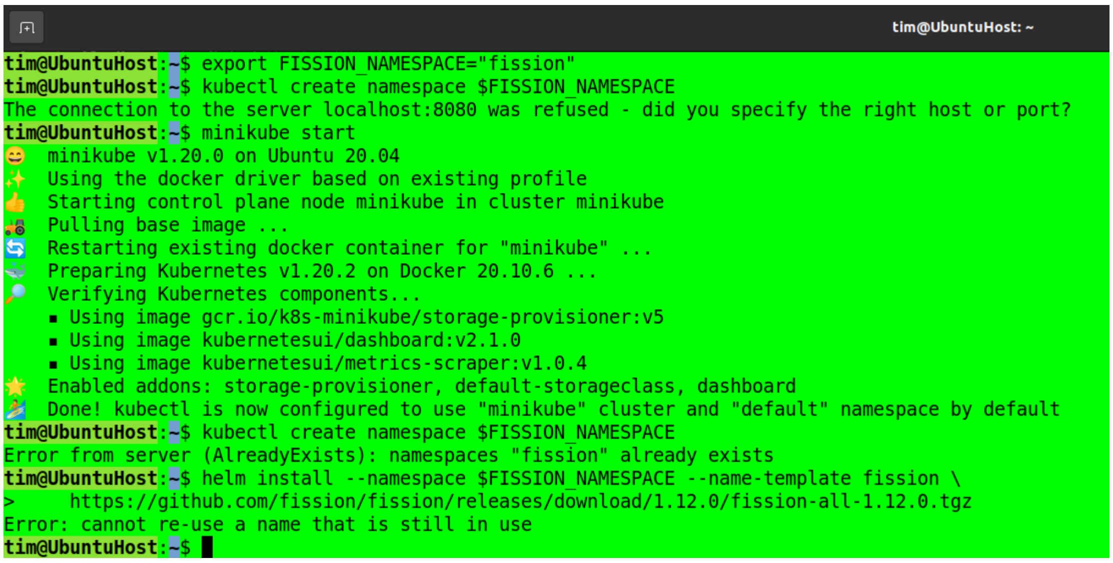
Now we need to export the Fission namespace then create that same namespace with Kubectl. For the next step, we need Helm which is a Kubernetes package manager. To install Helm like Kubectl we need to add the repository to our apt package with the curl command and add the key there. Next, we download the list of helm from there we will install like a normal software in Ubuntu with **sudo apt-get install** command. Shown in figure: A.47.



```
tim@UbuntuHost:~$ curl https://baltocdn.com/helm/signing.asc | sudo apt-key add -
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1700  100  1700    0     0   7423      0 --:--:-- --:--:-- --:--:--  7423
OK
tim@UbuntuHost:~$ sudo apt-get install apt-transport-https --yes
Reading package lists... Done
Building dependency tree
Reading state information... Done
apt-transport-https is already the newest version (2.0.5).
0 upgraded, 0 newly installed, 0 to remove and 3 not upgraded.
tim@UbuntuHost:~$ echo "deb https://baltocdn.com/helm/stable/debian/ all main" | sudo tee /etc/apt/sources.list.d/helm-stable-debian.list
deb https://baltocdn.com/helm/stable/debian/ all main
tim@UbuntuHost:~$ sudo apt-get install helm
Reading package lists... Done
Building dependency tree
Reading state information... Done
helm is already the newest version (3.6.0-1).
```

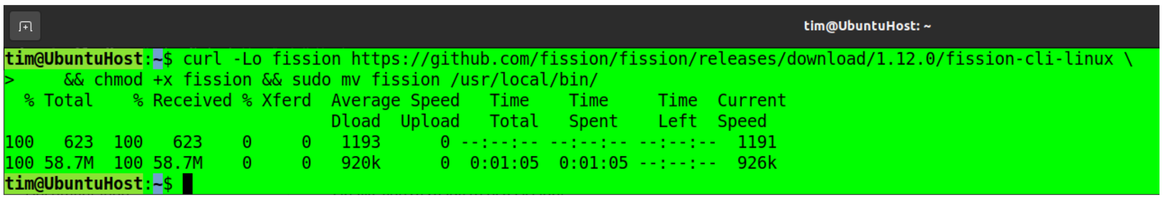
Figure A.47: Adding repository and installation of Helm

After installing we had to start our Minikube engine otherwise, it will give a connection refused error. So after starting Minikube, we created namespaces with Kibectl shown in figure: A.48 and with help of Helm we installed helm from the GitHub repository. But in our screen shoot, there is already an existing error that arrived because we did that before with our taking screenshots.
To continue we need to install Fission-CLI to make fission work with the command in the terminal. Shown in figure: A.49.
We are successfully done installing fission and make it useable in our Linux machine. To run any code first of all we need to make an environment for these languages NodeJS, Python, Ruby, Go, PHP, Bash, and any Linux executable with the function name and the function name must be all small or lowercase letter. Shown in figure: A.50 and figure: A.51.
As an example we made a python environment named "hellot" and the .py file name was helloT. After making the environment with the code. Finally, we could test the function with fission function test –name hellot. To update the code we can make changes in the .py file and update that code in the Fission function. Shown in figure: A.52.



```
tim@UbuntuHost: ~
tim@UbuntuHost:~$ export FISSION_NAMESPACE="fission"
tim@UbuntuHost:~$ kubectl create namespace $FISSION_NAMESPACE
The connection to the server localhost:8080 was refused - did you specify the right host or port?
tim@UbuntuHost:~$ minikube start
😄  minikube v1.20.0 on Ubuntu 20.04
✨  Using the docker driver based on existing profile
👍  Starting control plane node minikube in cluster minikube
🚜  Pulling base image ...
🔄  Restarting existing docker container for "minikube" ...
🐳  Preparing Kubernetes v1.20.2 on Docker 20.10.6 ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
    ▪ Using image kubernetesui/dashboard:v2.1.0
    ▪ Using image kubernetesui/metrics-scraper:v1.0.4
🌟  Enabled addons: storage-provisioner, default-storageclass, dashboard
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
tim@UbuntuHost:~$ kubectl create namespace $FISSION_NAMESPACE
Error from server (AlreadyExists): namespaces "fission" already exists
tim@UbuntuHost:~$ helm install --namespace $FISSION_NAMESPACE --name-template fission \
>     https://github.com/fission/fission/releases/download/1.12.0/fission-all-1.12.0.tgz
Error: cannot re-use a name that is still in use
tim@UbuntuHost:~$ 
```

Figure A.48: Exporting and Creating NameSpace for Fission



```
tim@UbuntuHost: ~
tim@UbuntuHost:~$ curl -Lo fission https://github.com/fission/fission/releases/download/1.12.0/fission-cli-linux \
>     && chmod +x fission && sudo mv fission /usr/local/bin/
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   623  100   623    0     0   1193      0 --:--:-- --:--:-- --:--:--  1191
100 58.7M  100 58.7M    0     0   920k      0  0:01:05  0:01:05 --:--:--  926k
tim@UbuntuHost:~$ 
```

Figure A.49: Adding repository and installation of Fission CLI

In fission, we cannot run our conventional python codes. It always looks for the main method of the code with the return type of string and it prints the return value. Our python was.

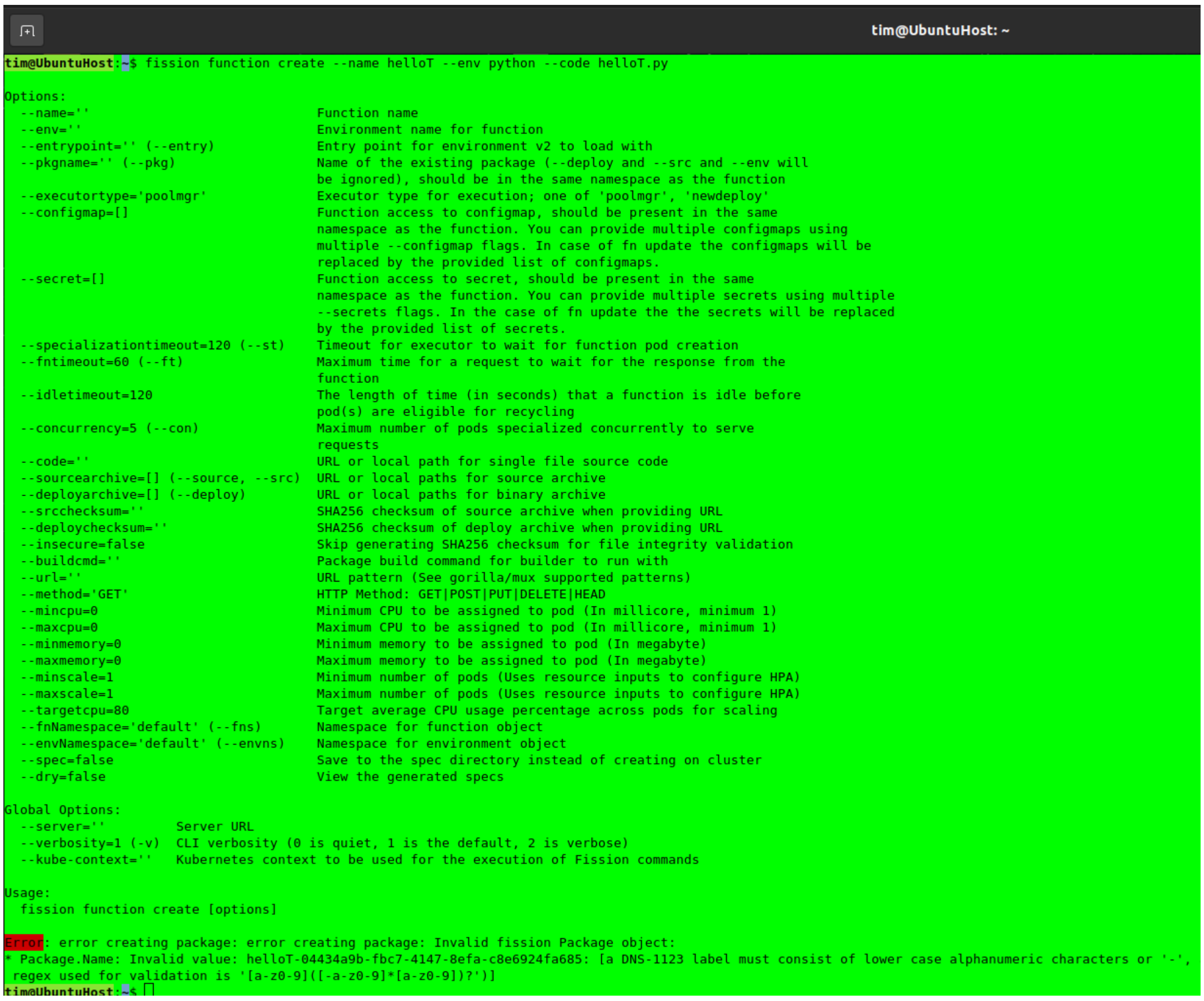
```
def main():
        return "Hello, Thesis Defense"
```

After successfully running the sample code we move forward to test our Fission framework with the same recursive function which was finding the nth Fibonacci number.

Here in our code we measure the time with time library to find the execution time of the function and measured those value for our comparison. We test one function 5 times to make an accurate result shown in figure: A.53.

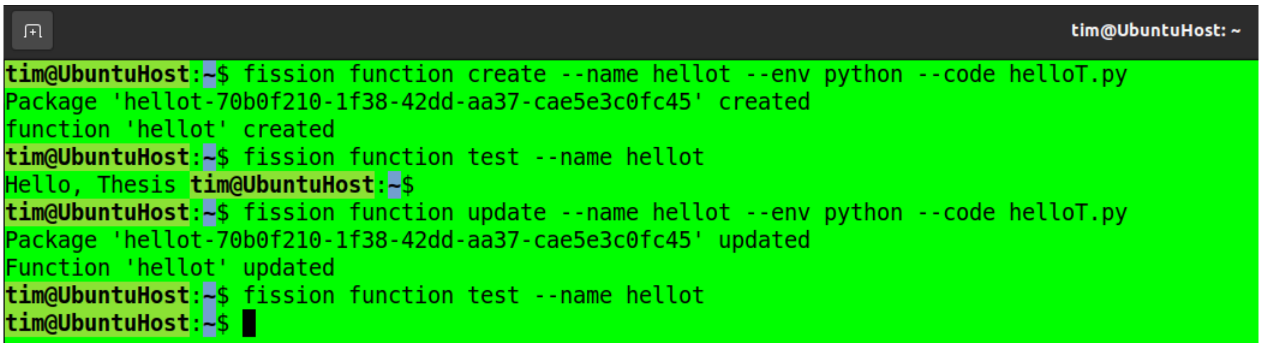
We get the value for finding the 30th, 35th and 38th Fibonacci number but we get an error to find the 40th Fibonacci number shown in figure: A.54.

Maybe Fission cannot utilize the memory properly.



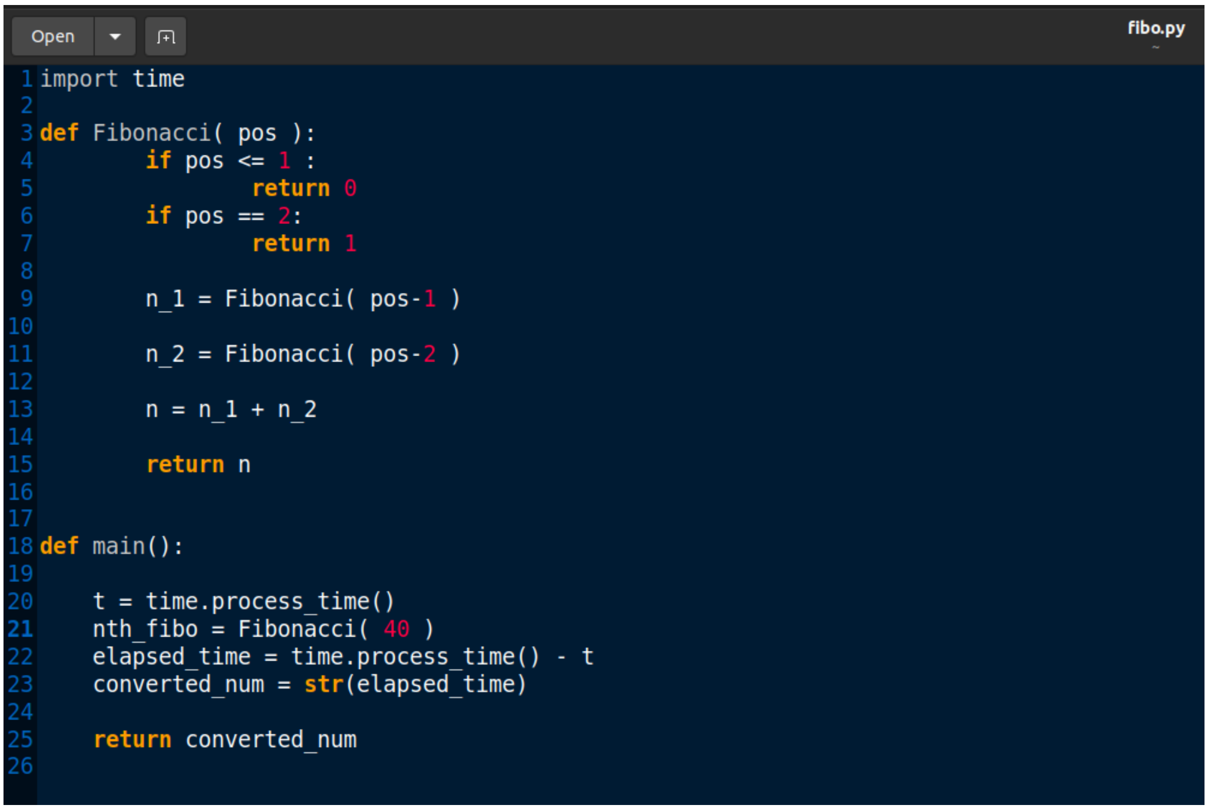
```
tim@UbuntuHost:~$ fission function create --name helloT --env python --code helloT.py

Options:
  --name=''                              Function name
  --env=''                               Environment name for function
  --entrypoint='' (--entry)              Entry point for environment v2 to load with
  --pkgname='' (--pkg)                   Name of the existing package (--deploy and --src and --env will
                                         be ignored), should be in the same namespace as the function
  --executortype='poolmgr'               Executor type for execution; one of 'poolmgr', 'newdeploy'
  --configmap=[]                         Function access to configmap, should be present in the same
                                         namespace as the function. You can provide multiple configmaps using
                                         multiple --configmap flags. In case of fn update the configmaps will be
                                         replaced by the provided list of configmaps.
  --secret=[]                            Function access to secret, should be present in the same
                                         namespace as the function. You can provide multiple secrets using multiple
                                         --secrets flags. In the case of fn update the the secrets will be replaced
                                         by the provided list of secrets.
  --specializationtimeout=120 (--st)     Timeout for executor to wait for function pod creation
  --fntimeout=60 (--ft)                  Maximum time for a request to wait for the response from the
                                         function
  --idletimeout=120                      The length of time (in seconds) that a function is idle before
                                         pod(s) are eligible for recycling
  --concurrency=5 (--con)                Maximum number of pods specialized concurrently to serve
                                         requests
  --code=''                              URL or local path for single file source code
  --sourcearchive=[] (--source, --src)   URL or local paths for source archive
  --deployarchive=[] (--deploy)          URL or local paths for binary archive
  --srcchecksum=''                       SHA256 checksum of source archive when providing URL
  --deploychecksum=''                    SHA256 checksum of deploy archive when providing URL
  --insecure=false                       Skip generating SHA256 checksum for file integrity validation
  --buildcmd=''                          Package build command for builder to run with
  --url=''                               URL pattern (See gorilla/mux supported patterns)
  --method='GET'                         HTTP Method: GET|POST|PUT|DELETE|HEAD
  --mincpu=0                             Minimum CPU to be assigned to pod (In millicore, minimum 1)
  --maxcpu=0                             Maximum CPU to be assigned to pod (In millicore, minimum 1)
  --minmemory=0                          Minimum memory to be assigned to pod (In megabyte)
  --maxmemory=0                          Maximum memory to be assigned to pod (In megabyte)
  --minscale=1                           Minimum number of pods (Uses resource inputs to configure HPA)
  --maxscale=1                           Maximum number of pods (Uses resource inputs to configure HPA)
  --targetcpu=80                         Target average CPU usage percentage across pods for scaling
  --fnNamespace='default' (--fns)        Namespace for function object
  --envNamespace='default' (--envns)     Namespace for environment object
  --spec=false                           Save to the spec directory instead of creating on cluster
  --dry=false                            View the generated specs

Global Options:
  --server=''          Server URL
  --verbosity=1 (-v)   CLI verbosity (0 is quiet, 1 is the default, 2 is verbose)
  --kube-context=''    Kubernetes context to be used for the execution of Fission commands

Usage:
  fission function create [options]

Error: error creating package: error creating package: Invalid fission Package object:
* Package.Name: Invalid value: helloT-04434a9b-fbc7-4147-8efa-c8e6924fa685: [a DNS-1123 label must consist of lower case alphanumeric characters or '-',
 regex used for validation is '[a-z0-9]([-a-z0-9]*[a-z0-9])?')]
tim@UbuntuHost:~$
```

Figure A.50: Error for function name structure miss match



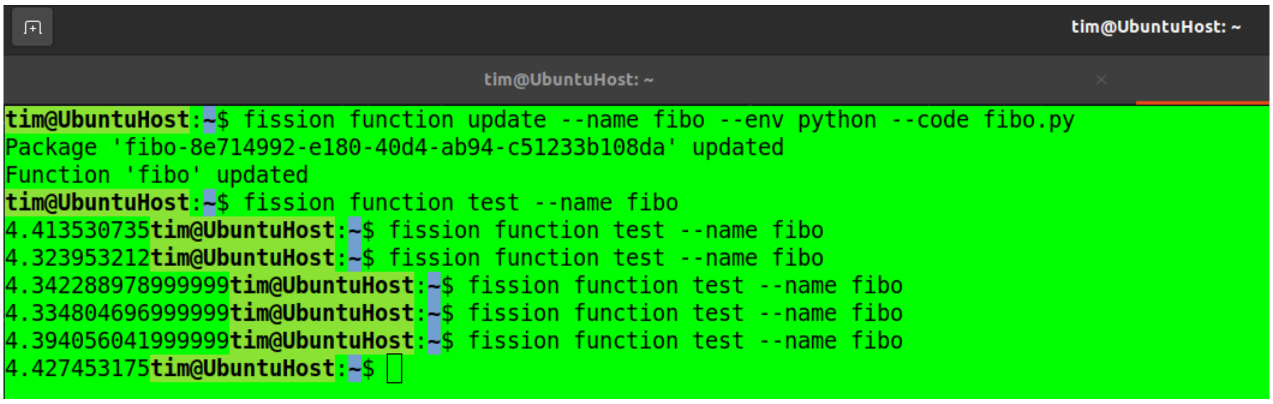
```
tim@UbuntuHost:~$ fission function create --name hellot --env python --code helloT.py
Package 'hellot-70b0f210-1f38-42dd-aa37-cae5e3c0fc45' created
function 'hellot' created
tim@UbuntuHost:~$ fission function test --name hellot
Hello, Thesis tim@UbuntuHost:~$
tim@UbuntuHost:~$ fission function update --name hellot --env python --code helloT.py
Package 'hellot-70b0f210-1f38-42dd-aa37-cae5e3c0fc45' updated
Function 'hellot' updated
tim@UbuntuHost:~$ fission function test --name hellot
tim@UbuntuHost:~$
```

Figure A.51: Successfully running a serverless Python function in Fission



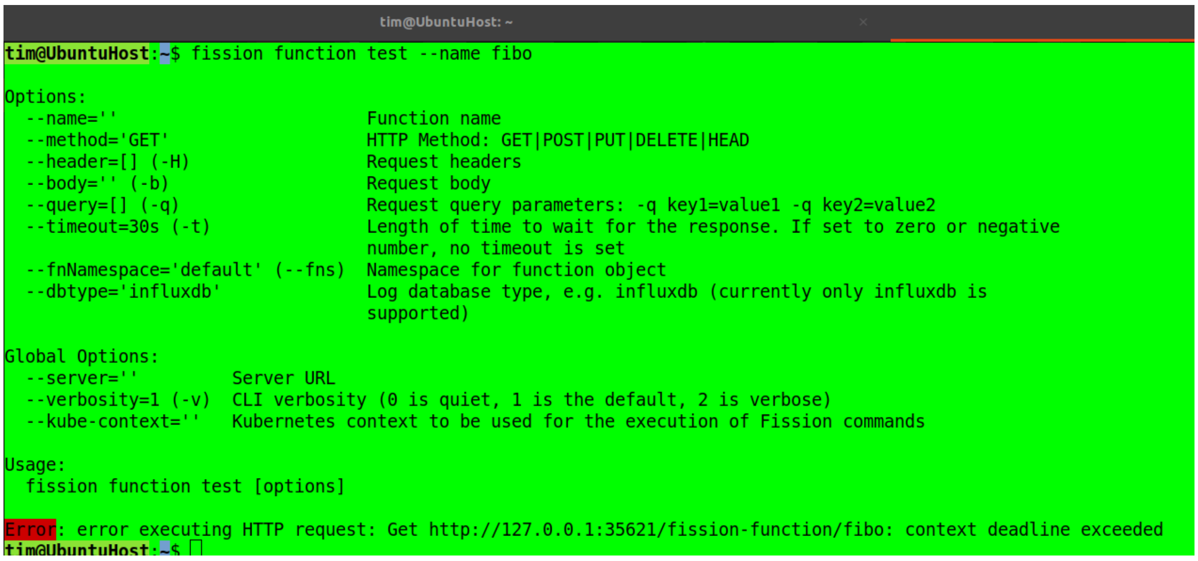
```python
import time

def Fibonacci( pos ):
        if pos <= 1 :
                return 0
        if pos == 2:
                return 1

        n_1 = Fibonacci( pos-1 )

        n_2 = Fibonacci( pos-2 )

        n = n_1 + n_2

        return n


def main():

    t = time.process_time()
    nth_fibo = Fibonacci( 40 )
    elapsed_time = time.process_time() - t
    converted_num = str(elapsed_time)

    return converted_num

```

Figure A.52: Recursive function and Execution time measurement function code for Fission



```
tim@UbuntuHost:~$ fission function update --name fibo --env python --code fibo.py
Package 'fibo-8e714992-e180-40d4-ab94-c51233b108da' updated
Function 'fibo' updated
tim@UbuntuHost:~$ fission function test --name fibo
4.413530735tim@UbuntuHost:~$ fission function test --name fibo
4.323953212tim@UbuntuHost:~$ fission function test --name fibo
4.342288978999999tim@UbuntuHost:~$ fission function test --name fibo
4.334804696999999tim@UbuntuHost:~$ fission function test --name fibo
4.394056041999999tim@UbuntuHost:~$ fission function test --name fibo
4.427453175tim@UbuntuHost:~$ 
```

Figure A.53: Execution time value for multiple time execution of the recursive function



```
tim@UbuntuHost:~$ fission function test --name fibo

Options:
  --name=''                          Function name
  --method='GET'                     HTTP Method: GET|POST|PUT|DELETE|HEAD
  --header=[] (-H)                   Request headers
  --body='' (-b)                     Request body
  --query=[] (-q)                    Request query parameters: -q key1=value1 -q key2=value2
  --timeout=30s (-t)                 Length of time to wait for the response. If set to zero or negative
                                     number, no timeout is set
  --fnNamespace='default' (--fns)    Namespace for function object
  --dbtype='influxdb'                Log database type, e.g. influxdb (currently only influxdb is
                                     supported)

Global Options:
  --server=''           Server URL
  --verbosity=1 (-v)    CLI verbosity (0 is quiet, 1 is the default, 2 is verbose)
  --kube-context=''     Kubernetes context to be used for the execution of Fission commands

Usage:
  fission function test [options]

Error: error executing HTTP request: Get http://127.0.0.1:35621/fission-function/fibo: context deadline exceeded
tim@UbuntuHost:~$
```

Figure A.54: Function error calculating 40th Fibonacci number

A.4 GCP:

We got 300USD credit for 3month for the Google Cloud Platform. For that, we had to give them a VISA card number for further billing and they mentioned they will not auto-charge after the trial period is finished. For authentication, they verify the card with a deduction of 1USD. After going throw the GPC console we realized that there is less feature than the AWS and Azure. But there are specific engines for machine Learning ad Artificial Intelligence work. Shown in figure: A.55.

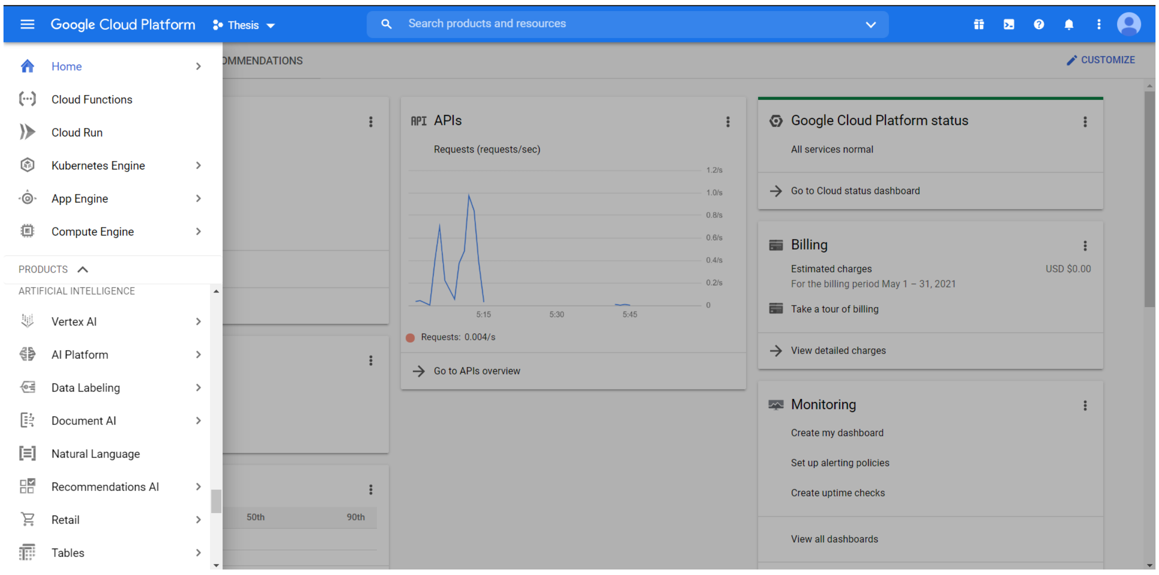
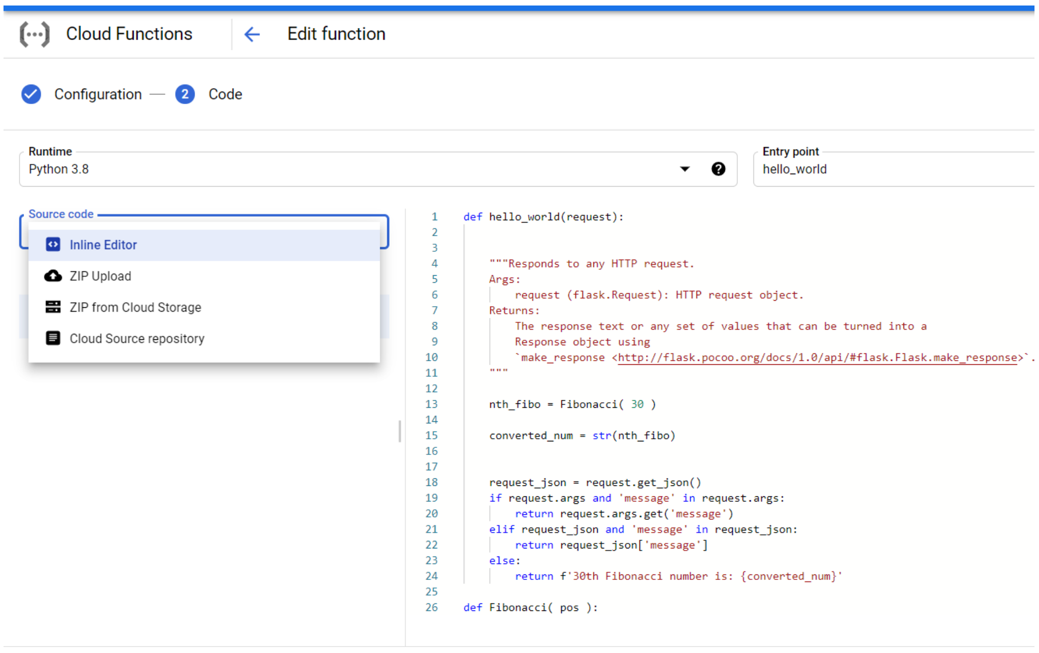


Figure A.55: Google Cloud Platform Console (Dashboard)

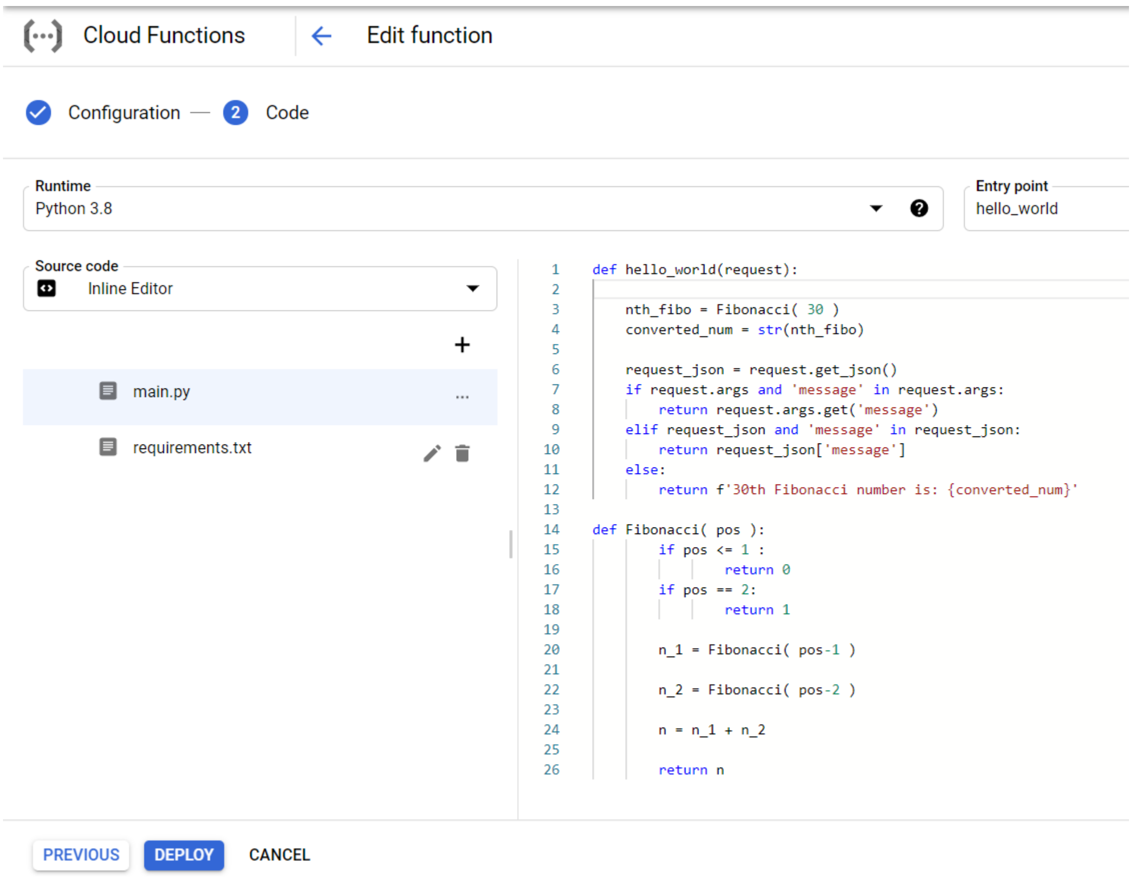
But for our work only needed the Serverless Functions feature which is called **Cloud Functions** in GCP. Here in the GCP function, only a few languages are supported compared to the other two provers but the good thing is we can use an inline code editor for all these programming languages and also from the zipped file. Shown in figure: A.56.



```python
def hello_world(request):


    """Responds to any HTTP request.
    Args:
        request (flask.Request): HTTP request object.
    Returns:
        The response text or any set of values that can be turned into a
        Response object using
        `make_response <http://flask.pocoo.org/docs/1.0/api/#flask.Flask.make_response>`.
    """

    nth_fibo = Fibonacci( 30 )

    converted_num = str(nth_fibo)


    request_json = request.get_json()
    if request.args and 'message' in request.args:
        return request.args.get('message')
    elif request_json and 'message' in request_json:
        return request_json['message']
    else:
        return f'30th Fibonacci number is: {converted_num}'

def Fibonacci( pos ):
```

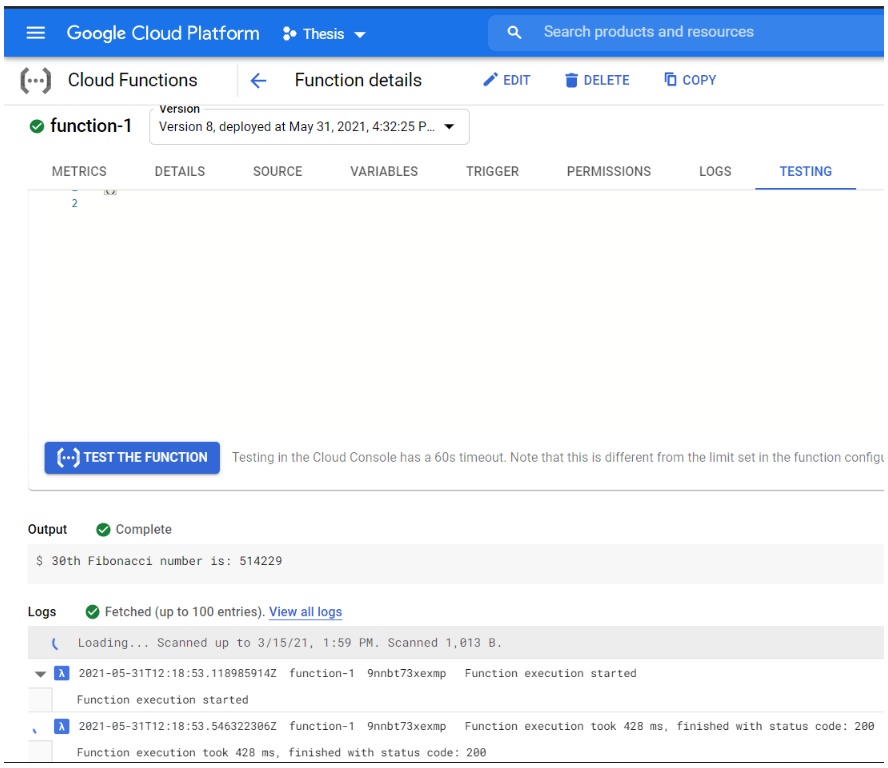
Figure A.56: Google Cloud Functions' in line code editor

For the comparison of a recursive function, we write a code for finding the Fibonacci number and measured the time of finding nth Fibonacci number same as we did in AWS Lambda. Shown in figure: A.57.
Here in Cloud Functions, it looks for the entry point method for running the code which is the main method here and that method return a string. After writing the code here we test the function and collect the result for finding the 30th, 35th, 38th and 40th Fibonacci number. Shown in figure: A.58.
In the metric tab, we can find the graph for various thing like memory utilization, execution time etc. Shown in figure: A.59.
Now it's time to run the image processing part. For doing that we needed a bucket from where the function can read the colourful image. Shown in figure: A.60.
First of all, we made a bucket in GCP storage and upload the colourful image. Shown in figure: A.61.
Then we need to set the roles or permissions to access the Cloud Functions. But after going in the permissions tab we can see the roles are already set maybe it good for a developer to save their time but it can be a security risk because from other functions the data for the bucket can be accessed. Shown in figure: A.62.
To continue, we created a function for image processing. To do that we needed NumPy, OpenCV, wand(for image reading), google-cloud-storage(for accessing the file from the bucket) and google-cloud-version libraries. On our first try, we were getting error for version mismatching of OpenCV and NumPy. Shown in figure: A.63.
In GCP there is no hassle of Layer we can write the library name with the specific version in the requirement.txt file. While deploying the function GCP automatically downloads the libraries so this is a very helpful feature in GPC comparing with AWS. Shown in figure: A.64.
Without any dependency or library mismatch error, we could deploy our function successfully after several tries. Then we wrote our code in the inline editor. Forsake



```python
def hello_world(request):

    nth_fibo = Fibonacci( 30 )
    converted_num = str(nth_fibo)

    request_json = request.get_json()
    if request.args and 'message' in request.args:
        return request.args.get('message')
    elif request_json and 'message' in request_json:
        return request_json['message']
    else:
        return f'30th Fibonacci number is: {converted_num}'

def Fibonacci( pos ):
        if pos <= 1 :
                return 0
        if pos == 2:
                return 1

        n_1 = Fibonacci( pos-1 )

        n_2 = Fibonacci( pos-2 )

        n = n_1 + n_2

        return n
```

Figure A.57: Deploying the recursive function in Google Cloud Functions


Google Cloud Platform
Thesis
Search products and resources
Cloud Functions
Function details
EDIT
DELETE
COPY
function-1
Version
Version 8, deployed at May 31, 2021, 4:32:25 P...
METRICS
DETAILS
SOURCE
VARIABLES
TRIGGER
PERMISSIONS
LOGS
TESTING
TEST THE FUNCTION
Testing in the Cloud Console has a 60s timeout. Note that this is different from the limit set in the function configu
Output
Complete
Logs
Fetched (up to 100 entries). View all logs

```
$ 30th Fibonacci number is: 514229
```

```
Loading... Scanned up to 3/15/21, 1:59 PM. Scanned 1,013 B.
2021-05-31T12:18:53.118985914Z  function-1  9nnbt73xexmp   Function execution started
Function execution started
2021-05-31T12:18:53.546322306Z  function-1  9nnbt73xexmp   Function execution took 428 ms, finished with status code: 200
Function execution took 428 ms, finished with status code: 200
```

Figure A.58: Log for running the recursive function in Google Cloud Functions

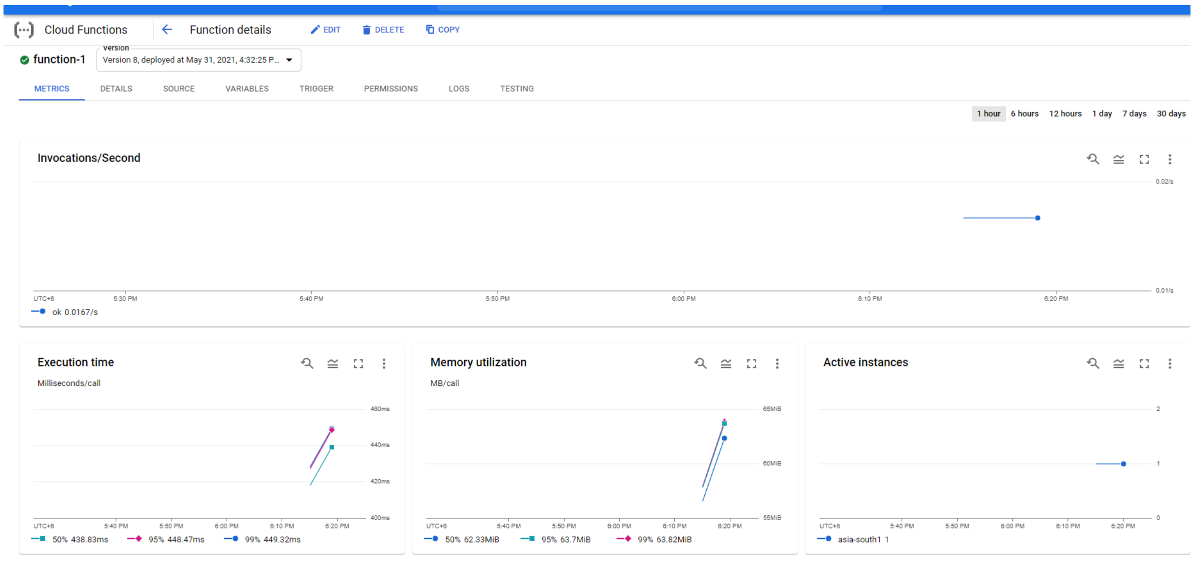


Figure A.59: Function details with graphical overview

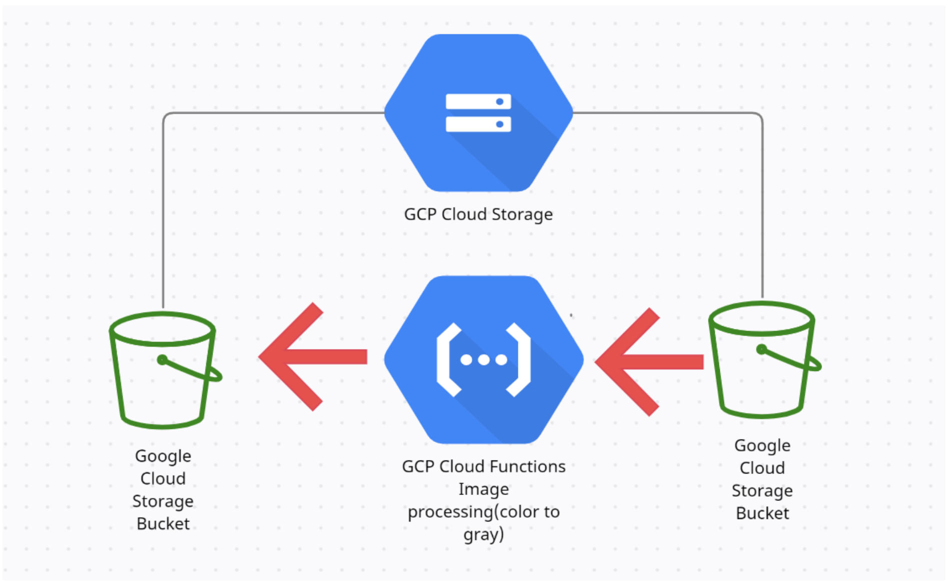
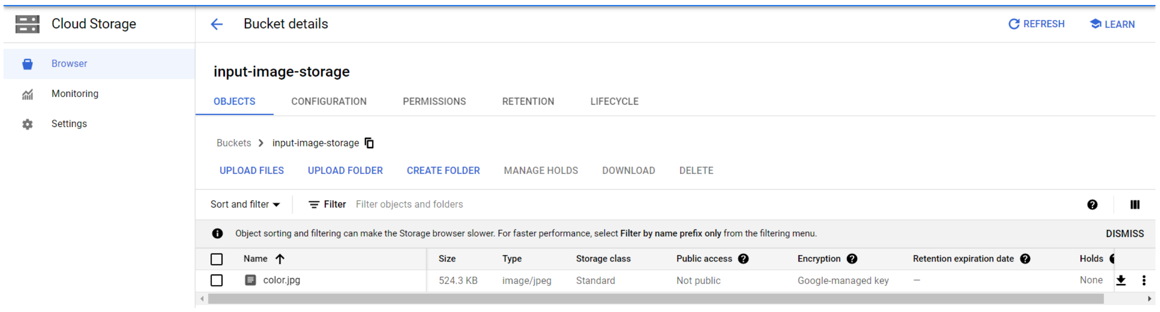

GCP Cloud Storage
Google Cloud Storage Bucket
GCP Cloud Functions Image processing(color to gray)
Google Cloud Storage Bucket

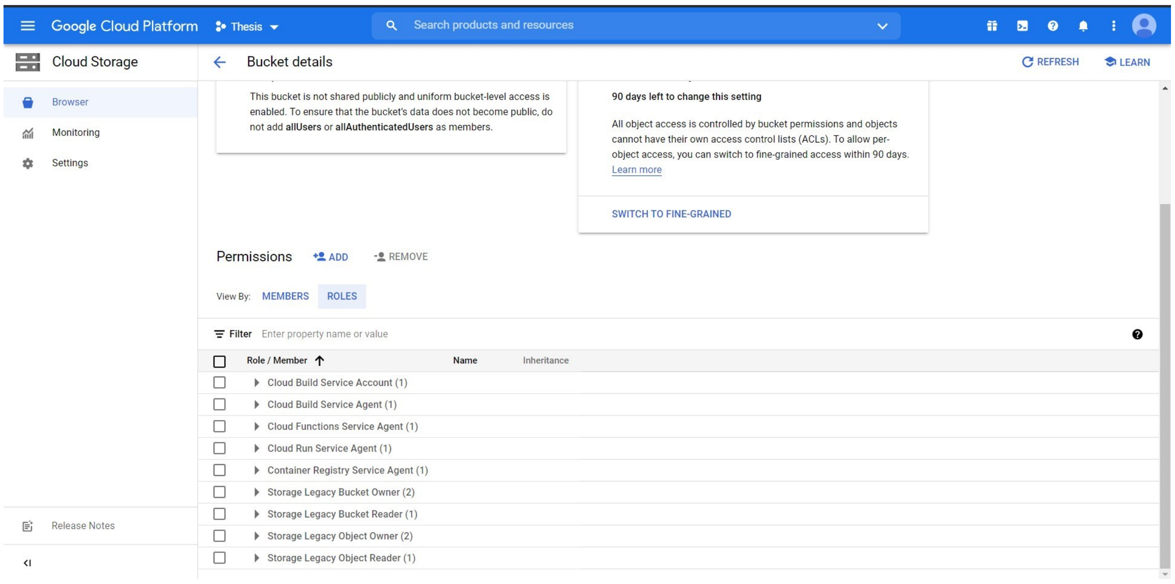
Figure A.60: Workflow of Image Processing in Google Cloud Functions



Object sorting and filtering can make the Storage browser slower. For faster performance, select **Filter by name prefix only** from the filtering menu.

Name	Size	Type	Storage class	Public access	Encryption	Retention expiration date	Holds
color.jpg	524.3 KB	image/jpeg	Standard	Not public	Google-managed key	–	None

Figure A.61: Bucket details with our input color image



This bucket is not shared publicly and uniform bucket-level access is enabled. To ensure that the bucket's data does not become public, do not add **allUsers** or **allAuthenticatedUsers** as members.

90 days left to change this setting

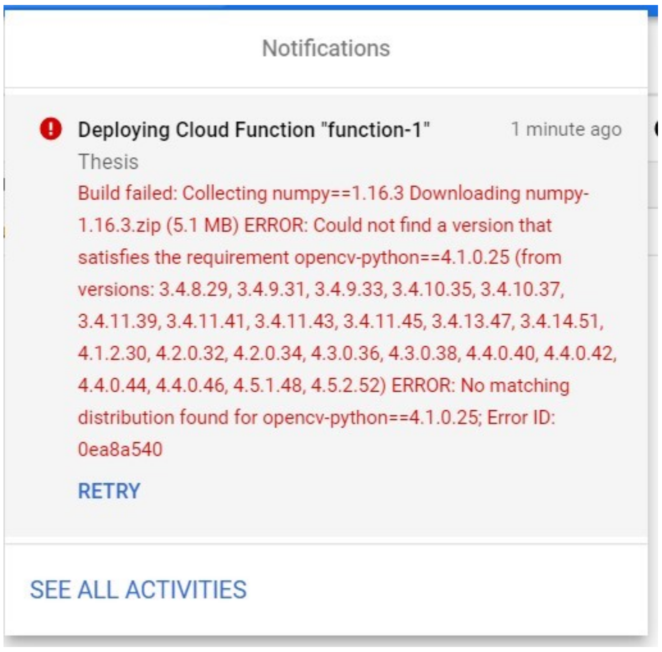
All object access is controlled by bucket permissions and objects cannot have their own access control lists (ACLs). To allow per-object access, you can switch to fine-grained access within 90 days. Learn more

Permissions

View By: MEMBERS ROLES

Role / Member	Name	Inheritance
Cloud Build Service Account (1)		
Cloud Build Service Agent (1)		
Cloud Functions Service Agent (1)		
Cloud Run Service Agent (1)		
Container Registry Service Agent (1)		
Storage Legacy Bucket Owner (2)		
Storage Legacy Bucket Reader (1)		
Storage Legacy Object Owner (2)		
Storage Legacy Object Reader (1)		

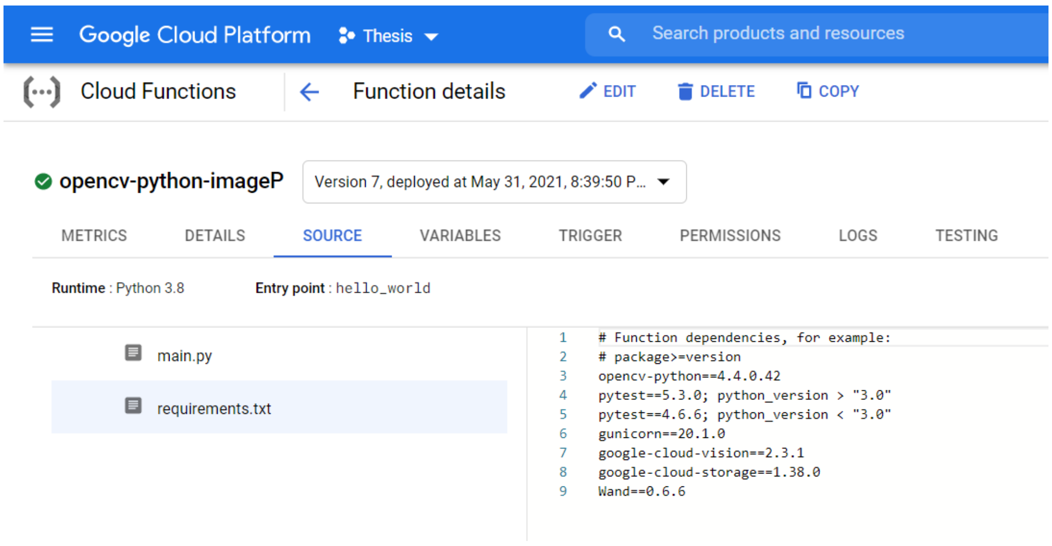
Figure A.62: Default bucket roles



Notifications

Deploying Cloud Function "function-1" 1 minute ago
Thesis
Build failed: Collecting numpy==1.16.3 Downloading numpy-1.16.3.zip (5.1 MB) ERROR: Could not find a version that satisfies the requirement opencv-python==4.1.0.25 (from versions: 3.4.8.29, 3.4.9.31, 3.4.9.33, 3.4.10.35, 3.4.10.37, 3.4.11.39, 3.4.11.41, 3.4.11.43, 3.4.11.45, 3.4.13.47, 3.4.14.51, 4.1.2.30, 4.2.0.32, 4.2.0.34, 4.3.0.36, 4.3.0.38, 4.4.0.40, 4.4.0.42, 4.4.0.44, 4.4.0.46, 4.5.1.48, 4.5.2.52) ERROR: No matching distribution found for opencv-python==4.1.0.25; Error ID: 0ea8a540

Figure A.63: Deployment error for miss matched library function for Python

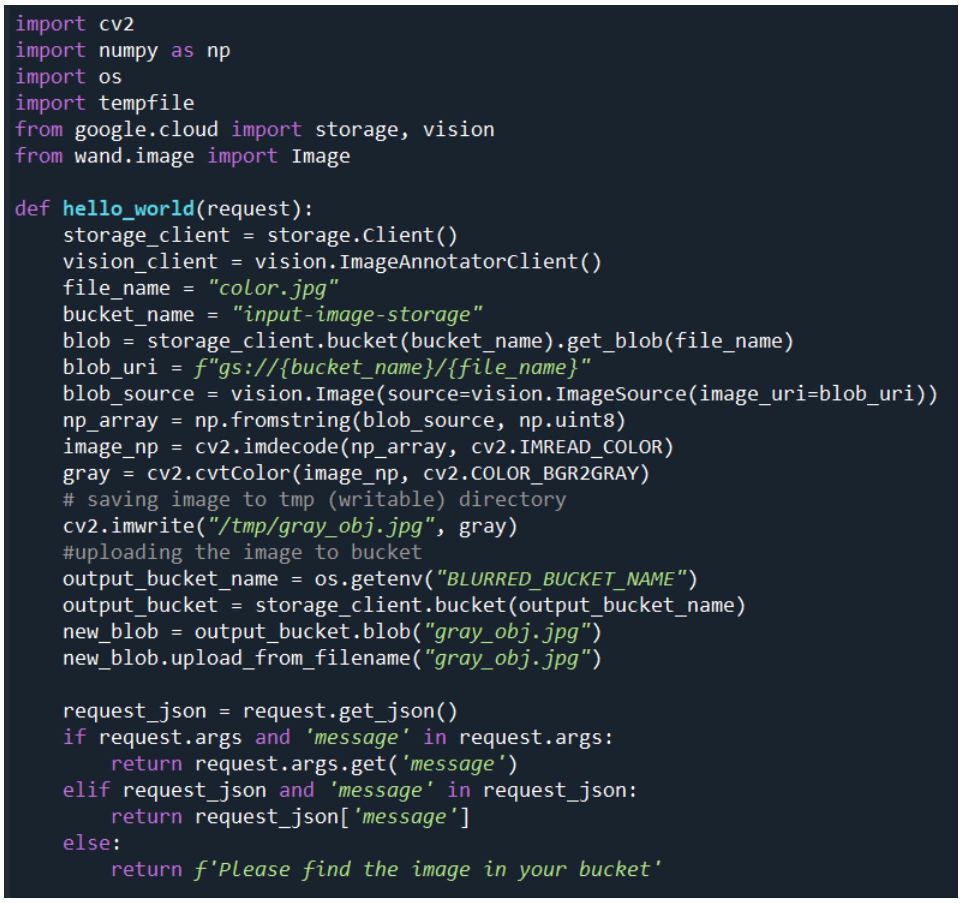


Runtime : Python 3.8 Entry point : hello_world

```
# Function dependencies, for example:
# package>=version
opencv-python==4.4.0.42
pytest==5.3.0; python_version > "3.0"
pytest==4.6.6; python_version < "3.0"
gunicorn==20.1.0
google-cloud-vision==2.3.1
google-cloud-storage==1.38.0
Wand==0.6.6
```

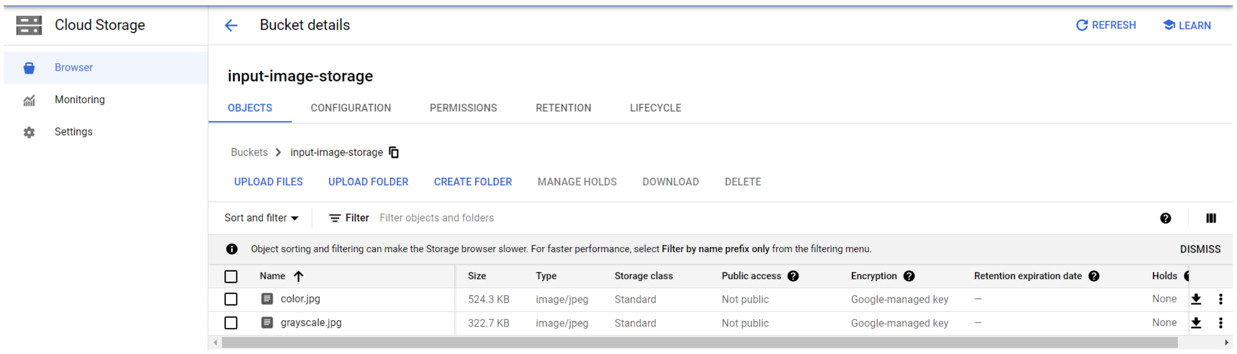
Figure A.64: Requirement library files for image processing with specific version

or good view we are showing it in the spider code editor. Shown in figure: A.65.
First of all, we imported everything that we need. Then declare the storage and get the storage client and image annotator. Then put the image in a NumPy array and processed the image in gray scale and upload it to the bucket. Shown in figure: A.66.
After going into the bucket we can find our grascale.jpg image file. GPC takes more than 2minutes to deploy a function with dependencies so we deleted the image ran it over again sometimes and collect the execution time. Shown in figure: A.67.



```python
import cv2
import numpy as np
import os
import tempfile
from google.cloud import storage, vision
from wand.image import Image

def hello_world(request):
    storage_client = storage.Client()
    vision_client = vision.ImageAnnotatorClient()
    file_name = "color.jpg"
    bucket_name = "input-image-storage"
    blob = storage_client.bucket(bucket_name).get_blob(file_name)
    blob_uri = f"gs://{bucket_name}/{file_name}"
    blob_source = vision.Image(source=vision.ImageSource(image_uri=blob_uri))
    np_array = np.fromstring(blob_source, np.uint8)
    image_np = cv2.imdecode(np_array, cv2.IMREAD_COLOR)
    gray = cv2.cvtColor(image_np, cv2.COLOR_BGR2GRAY)
    # saving image to tmp (writable) directory
    cv2.imwrite("/tmp/gray_obj.jpg", gray)
    #uploading the image to bucket
    output_bucket_name = os.getenv("BLURRED_BUCKET_NAME")
    output_bucket = storage_client.bucket(output_bucket_name)
    new_blob = output_bucket.blob("gray_obj.jpg")
    new_blob.upload_from_filename("gray_obj.jpg")

    request_json = request.get_json()
    if request.args and 'message' in request.args:
        return request.args.get('message')
    elif request_json and 'message' in request_json:
        return request_json['message']
    else:
        return f'Please find the image in your bucket'
```

Figure A.65: Image processing function code for Google Cloud Functions



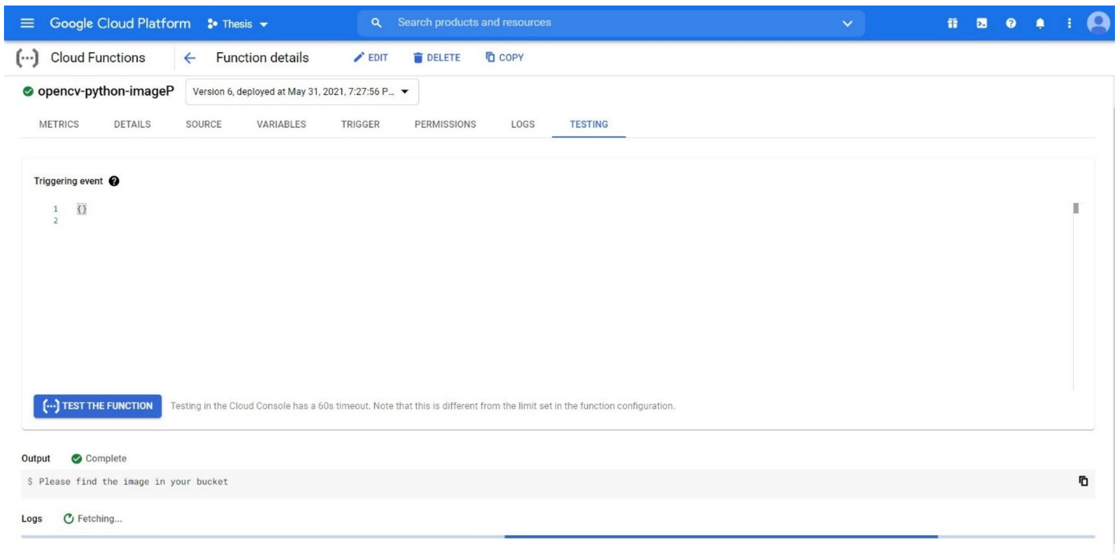
input-image-storage

Buckets > input-image-storage

Object sorting and filtering can make the Storage browser slower. For faster performance, select **Filter by name prefix only** from the filtering menu.

Name ↑	Size	Type	Storage class	Public access	Encryption	Retention expiration date	Holds
color.jpg	524.3 KB	image/jpeg	Standard	Not public	Google-managed key	–	None
grayscale.jpg	322.7 KB	image/jpeg	Standard	Not public	Google-managed key	–	None

Figure A.66: Finding processed gray-scale image in the bucket



```
{}
```

Testing in the Cloud Console has a 60s timeout. Note that this is different from the limit set in the function configuration.

```
$ Please find the image in your bucket
```

Figure A.67: Successfully running the image processing function in Google Cloud Functions