

FRANKFURT UNIVERSITY OF APPLIED  
SCIENCES

**Deployment of a Microservice Architecture  
for Implementing a Redirection Gateway  
in the Cloud by using Docker and  
Kubernetes**

by

Tomusange Brian 1291025

Sarath Chandra Mallineni 1291986

Abdullah Al Noman 1323731

Asad Ahmed 1321474

A SEMESTER PROJECT REPORT  
(CLOUD COMPUTING)  
Winter Semester 2020/21

*UNDER THE GUIDANCE OF*

Prof.Dr.Christian BAUN

February 15, 2021

### Abstract

Cloud computing and microservices technologies are offering a lot of facilities to handle interoperability, flexibility, orchestration and security in software engineering. This project uses container based technologies that are being used inside the microservice architecture which will ease the use of and the efficiency of resource usages. We wrapped our microservices in a container within the Docker and deployed it in kubernetes and google cloud. Finally the research explores how Dockers and Kubernetes technology can be used in microservices making it more flexible, effective and efficient

## 1 Introduction

Dockers and Kubernetes have recently become a great area of interest in the development of interoperability, flexible, and easy to use web-services. This section will give a brief background and the objectives of this project using these technologies.

### 1.1 Background

The need for quality and flexible microservice in different software has consequently led to quality and flexible Information Technology solutions. Consequently, today's competitive environment, customers with ever changing requirements, an appropriate software development technology is crucial. Microservice using Dockers and Kubernetes is a pervasive strategy for developing software applications in such an environment. [7][1][2]

Globally, workflows have been used for transaction-led organizations, and organizations have appreciated the benefits of business process automation. It is evident that if technology is used to automate these processes, then time-consuming and error-prone manual intervention can be eliminated. This helps in achieving more, in less time, with fewer mistakes. However, the real benefit is greater competitive advantage, as costs are reduced and productivity is improved. The key drivers for process automation are improved business efficiency and compliance. As businesses evolve, processes that have been fundamental to the operation of an organization can become error-prone, inefficient, or expensive.

### 1.2 Objectives

The major objectives of this project are the deployment of redirection services on a private cloud by using open source tools such as Docker and Kubernetes and the Google cloud to combine a cloud platform.

Specific Objective are:

1. To Analyze the microservices technology.
2. To design and deploy a Microservice Architecture using Dockers and Kubernetes.
3. Integrating CI/CD Pipeline with Docker and kubernetes.
4. Test and validate the deployment.

## 2 Methodology

This section gives a brief review of the technologies, approaches and methods used in microservices. Finally a detailed explanation of the methodology used to develop these microservices is presented.

### 2.1 Microservice Technology

Microservices are mainly used to solve the problems that come from monolithic structures. Microservices is a style which represents the structure of an application by combining all the autonomous services of that application.[6]. Figure 1, illustrates how microservices can be combined in an application. Our main focus is a redirection gateway meaning if someone wants to connect to a particular service, this redirection gateway will help to fetch the service and handover to the user. Redirection gateway should also be seen as a service. The following services which are available in our project are;

1. Service A
2. Simulator service.

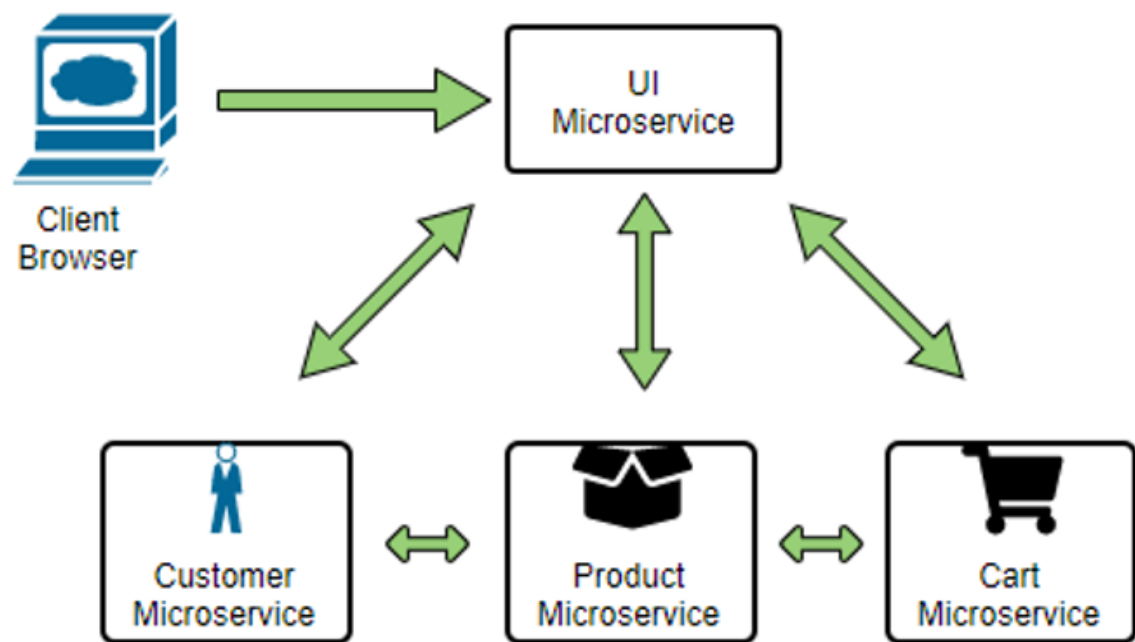


Figure 1: Example for Microservices[3]

### 2.2 Microservice Architecture

The API gateway of the microservices gives the opportunity to the clients to interact with the services. The services are integrated with the docker images which are developed with all the other services. It gives the possibility to access these docker images on any platform. It has the same functionality. The gateway is realized as an REST

API and is accessible via the provided UI and simple HTTP requests. The task of the gateway is to autoscale based on the user specifications. Finally using kubernetes we can merge these containers created above and orchestrate and deploy in one of the cloud environments, see figure 2. It provides us a distributed platform to access the service.

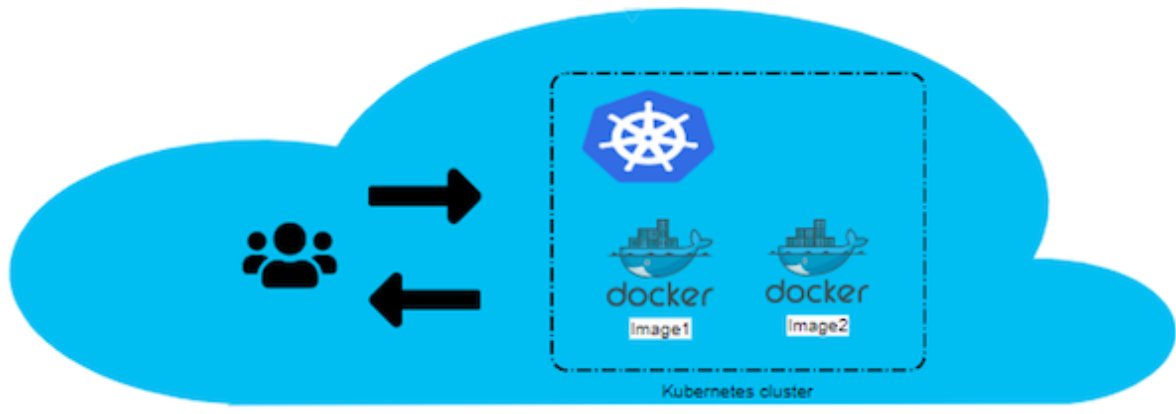


Figure 2: Top Level Architecture

We used google cloud platform to deploy our services on it. We created docker images for the services and created a cluster using kubernetes and deployed in google cloud environment as shown below in figure 3.

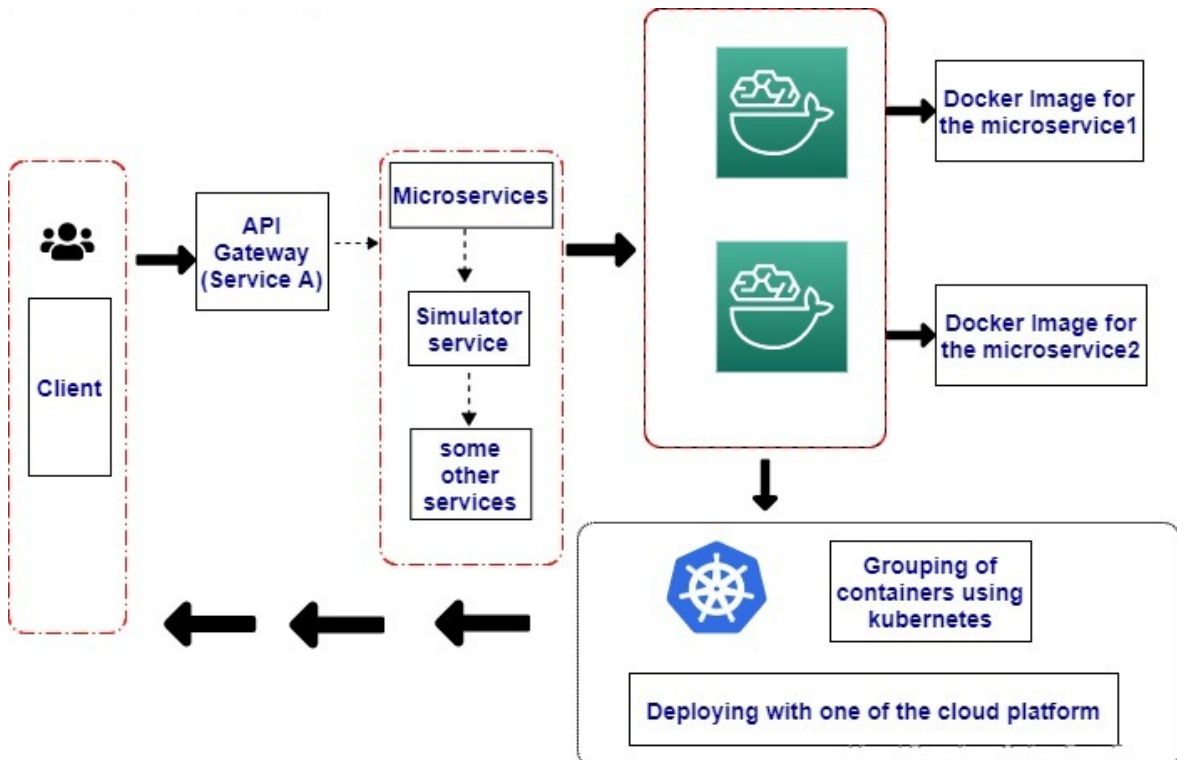


Figure 3: Detailed Architecture

## 2.3 Microservice Architecture using Docker and Kubernetes

We developed our services using spring boot. We also used my-sql database which was added to our services. So, we are creating docker containers for our two services and a mysql server and then connecting each of them with a network bridge. It serves the purpose to record and store information. The basic idea of deploying these services are explained below in figure 4.

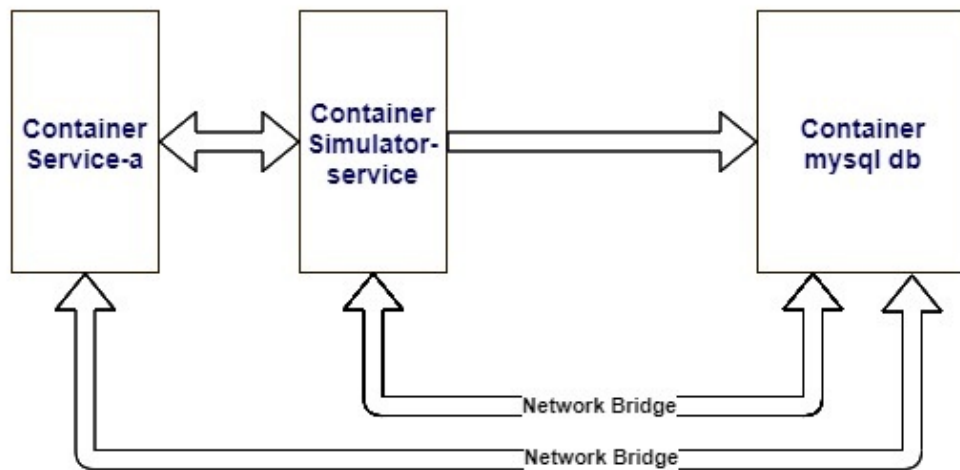


Figure 4: Network between Services[4]

## 3 Implementation

The section details the implementation steps towards achieving our desired objectives. We used a hybrid approach in our development, although we had some specific milestones which we have detailed in this section.

### 3.1 Creating a Docker Image

We created two services and one mysql server, so three images were created. An account on docker hub was created first so that we can push our microservices image and mysql image on docker hub which is a prerequisite for kubernetes deployment. We had set up the environment in our local machine for building and pushing the docker images in the docker hub.

Therefore we first created a network for mysql, and this would be running as a docker container later. We also configured the database credential for making it compatible

with our services. Table 1 below shows the commands for creating the images successfully.

```
$ docker network create service-tracker-db

$ docker container run --mysql db --
  network service-tracker-db-eMYSQL_ROOT_PASSWORD
  =root123-e MYSQL_DATABASE=service_tracker_db-dmysql:8.0.22

$ docker container logs -f mysql db

$ docker container exec -it mysql db bash
```

Table 1: Command for Creating Images.

We then created two docker images of our services and before uploading to the docker hub we added a tag and then pushed it to the docker hub repository by using some commands as shown in Table 2. Our services were loaded in the docker hub and thus they were exposed locally in our machine and available for pull.

```
$ docker image build -t service-a

$ docker image build -t simulator-service

$ docker container run --network service-
tracker-db --nameservice-a-container -p
8080:8080 -d service-a

$ docker container run --network service-tracker-db--name
simulator-service-container -p 8081:8081-dsimulator-service

$ docker tag service-a
abdullah1122/cloud_deployment:service-a-0.0.1-SNAPSHOT

$ docker push abdullah1122/cloud_deployment:
service-a-0.0.1-SNAPSHOT

$ docker tag simulator-serviceabdullah1122/cloud_deployment:
simulator-service-0.0.1-SNAPSHOT

$ docker push abdullah1122/cloud_deployment:
simulator-service-0.0.1-SNAPSHOT

$ docker tag mysql:8.0.22abdullah1122/cloud_deployment:
mysql-0.0.1-SNAPSHOT

$ docker push abdullah1122/cloud_deployment:
mysql-0.0.1-SNAPSHOT
```

Table 2: Command for loading services to docker hub.

After executing the above command in Table 2, our microservice images was on the docker hub like below in Figure 5.

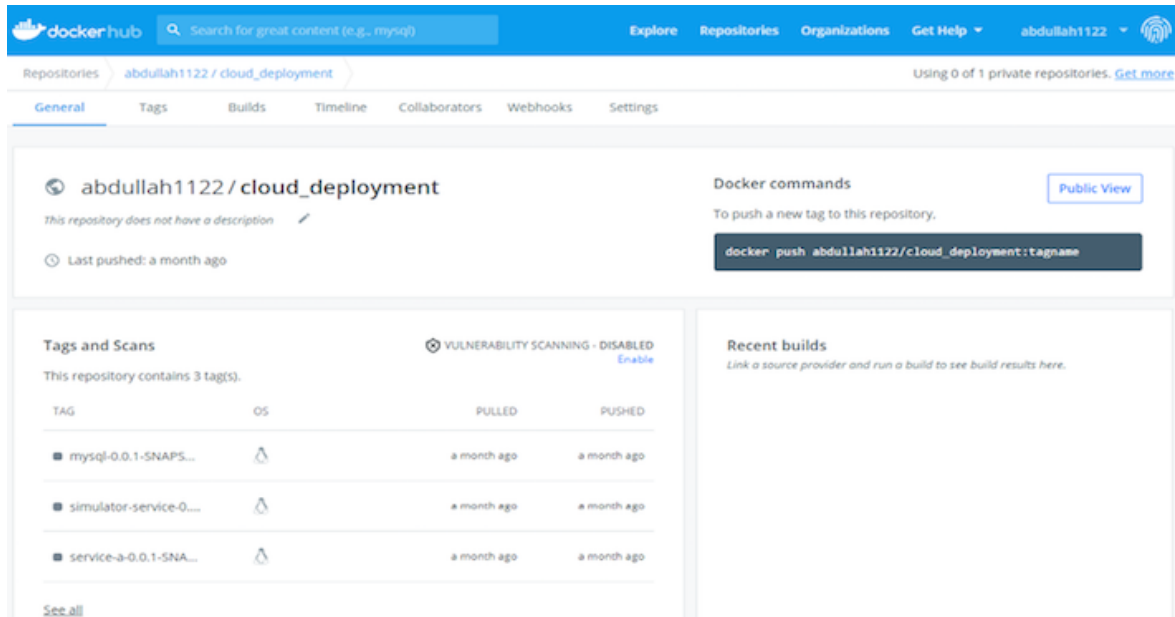


Figure 5: Image on Docker Hub

## 3.2 Kubernetes Deployment

We configured the environment for kubernetes deployment in our system and set up mysql for persisting volume and secret[5]. We also wrote the yaml file for deploying everything in the kubernetes cluster. We deployed our services and exposed them to the cloud. We did this using following commands in Table 3 and eventually our services are exposed to the external IP address from where we could access them.



```

$ kubectl create deployment service-a--abdullah1122
/cloud_deployment:service-a-0.0.1-SNAPSHOT

$ kubectl create deployment simulator-service--abdullah1122/
cloud_deployment:simulator-service-0.0.1-SNAPSHOT

$ kubectl create deployment mysql
--abdullah1122/cloud_deployment:mysql-0.0.1-SNAPSHOT

$ kubectl expose deployment simulator-service --type=LoadBalancer
--port 81 --target-port 8081

$ kubectl expose deployment service-a --type=LoadBalancer
--port 80 --target-port 8080
    
```

Table 3: Command for Creating Images.

The above deployment commands in Table 3, deployed and exposed our services which resulted in a running pod in kubernetes cluster. Finally we had successfully deployed our microservices to Kubernetes as shown in Figure 6 and we can access it through external IP.

```

PS C:\Users\USER> kubectl get deployments
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
service-a     1/1     1             1           40d
simulator-service 1/1     1             1           40d
PS C:\Users\USER> kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
service-a-5568fcf44f-jg4wp 1/1     Running   3           40d
simulator-service-fdf55d75c-f98lg 1/1     Running   3           40d
PS C:\Users\USER> kubectl get svc
NAME          TYPE          CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
kubernetes   ClusterIP     10.96.0.1       <none>        443/TCP          40d
service-a    LoadBalancer 10.96.253.117   127.0.0.1     8080:32162/TCP  40d
simulator-service NodePort      10.108.164.12   <none>        8081:32008/TCP  40d
    
```

Figure 6: Exposed Services

### 3.3 Uploading Docker Images to Google Cloud

As we have already set up the minikube, kubernetes and deployed our microservices in the local kubernetes cluster. Here we deployed the same services in the google cloud following these steps

1. We needed to register for google cloud platform first. Then after we logged in to the google cloud platform and we created a project named CloudDeployment. Then we created a cluster named as cluster-1. we have our docker images in docker hub and we would need to push the docker images to the google container registry.[8] The google container registry is the same as docker hub. Before pushing we had to set another environment name Google Cloud SDK (Command line interface). So, we have downloaded Google SDK and installed it on our local system by following the google SDK installation instruction.[7]
2. After installing the google SDK on our system, we then ran the initialization command in Table 4.

```
$ gcloud init
```

Table 4: Initialization Command

It asked to pick the configuration to use, so here we re-initialized the configuration that we had created in google cloud platform as shown in Figure. 7, so we selected 1.

```
Pick configuration to use:  
[1] Re-initialize this configuration [default] with new settings  
[2] Create a new configuration  
Please enter your numeric choice: 1
```

Figure 7: Image of initialization window

In the next step it asked us to choose the account to perform the operation so we selected our own account and then logged to the system. We set the compute region and zone. We then followed the steps to configuring this environment.

3. Now we are ready to push our docker images to Google cloud registry using the command in Table 5. In that step we need to tag the docker image with the project name so we have run the following command.

```
$ docker tag service-a gcr.io/clouddeployment-298120/  
/service-a  
  
$ docker tag simulation-service gcr.io/clouddeployment-  
298120/simulation-service
```

Table 5: Command for Tagging services

Then for pushing it to the google container registry we used the command in Table 6.

```
$ gcloud docker push gcr.io/clouddeployment-298120/  
/service-a  
  
$ gcloud docker push gcr.io/clouddeployment-298120/  
simulation-service
```

Table 6: Command for pushing services to google container

First time it failed because of authentication issues so we run the command in Table 7 to solve that issue.

```
$ gcloud auth configure-docker
```

Table 7: Authentication command

Finally our first service “service-a” was deployed on google container registry. For our other service and mysql docker images we followed the same procedure and uploaded it on the google container registry. We used the commands in Table 8 for pushing the image on to the google cloud.

```
$ dockertagsimulator-servicegcr.io/clouddeployment-  
298120/simulator-service:v1  
  
$ dockerpushgcr.io/clouddeployment-298120/simulator  
-service:v1
```

Table 8: Commands for pushing images to cloud.

Following the final output for pushing it in the google container registry, we deployed our images in the Kubernetes cluster, so that they were connected with cloudshell. We ran the deployment command in Table 9 for our services

```

$ kubectl create deployment service-a --image=gcr.io/
clouddeployment-298120/service-a

$ kubectl create deployment simulator-service--image=gcr.
io/clouddeployment-298120/simulator-service:v1
    
```

Table 9: Commands for deployment.

We exposed the deployed services, using the commands in Table 10;

```

$ kubectl expose deployment simulator-service --type=
LoadBalancer--port 81 --target-port 8081

$ kubectl expose deployment service-a --type=LoadBalancer
--port 80 --target-port 8080
    
```

Table 10: Commands for exposing the services to cloud.

Our services were exposed now in google cloud. We could check it by running the following command and we also found the external IP of the service so that we could solve the external IP part as seen in Figure 8.

```

Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to clouddeployment-298120.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
aalnoman638@cloudshell:~ (clouddeployment-298120) $ kubectl get service
NAME                TYPE                CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
kubernetes          ClusterIP           10.8.0.1      <none>         443/TCP          41d
service-a           LoadBalancer       10.8.0.158    35.225.228.133 80:32051/TCP     33d
simulator-service   LoadBalancer       10.8.10.4     35.232.204.143 81:31744/TCP     33d
aalnoman638@cloudshell:~ (clouddeployment-298120) $
    
```

Figure 8: Exposed Services on google cloud

Now we can access our rest services with the external IP and port as shown in Figure 9 and Figure 10. The following is the final deployed output and we could access our service globally.

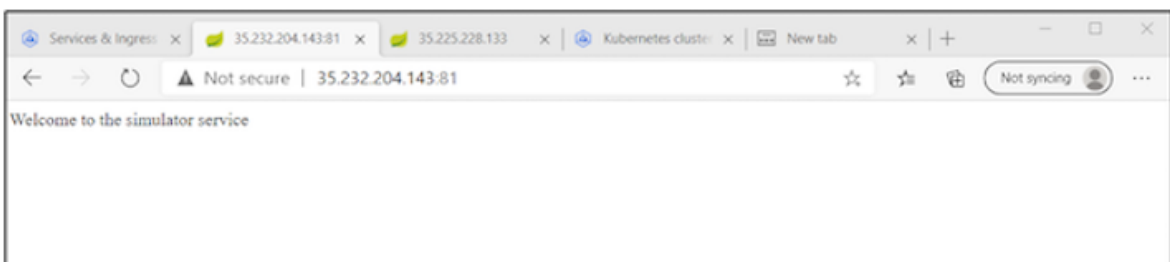


Figure 9: Response of our REST services (A)

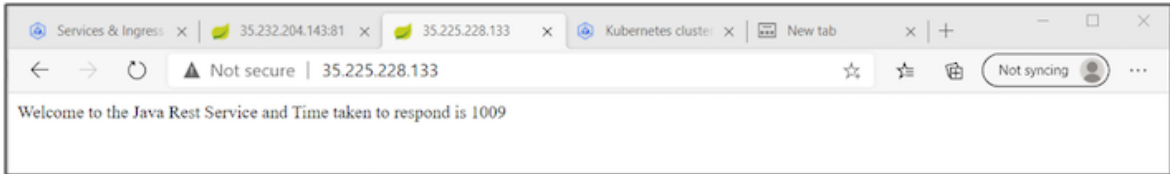


Figure 10: Response of our REST services (B)

## 4 Integrating CI/CD Pipeline for Kubernetes Deployment

This section explains how we integrated CI/CD pipeline for kubernetes in our project and deployed it on google cloud platform.

### 4.1 Installing Jenkins on Google Cloud

In the first phase of that we needed three virtual machine

1. Jenkins Server
2. Kubernetes Master (k8smaster)
3. Kubernetes Worker (k8sworker)

Our Jenkins server was connected to k8smaster which did the continuous deployment. We set up the Github and Docker hub that fits in CI/CD integration. The image in Figure 11, below shows the necessary steps

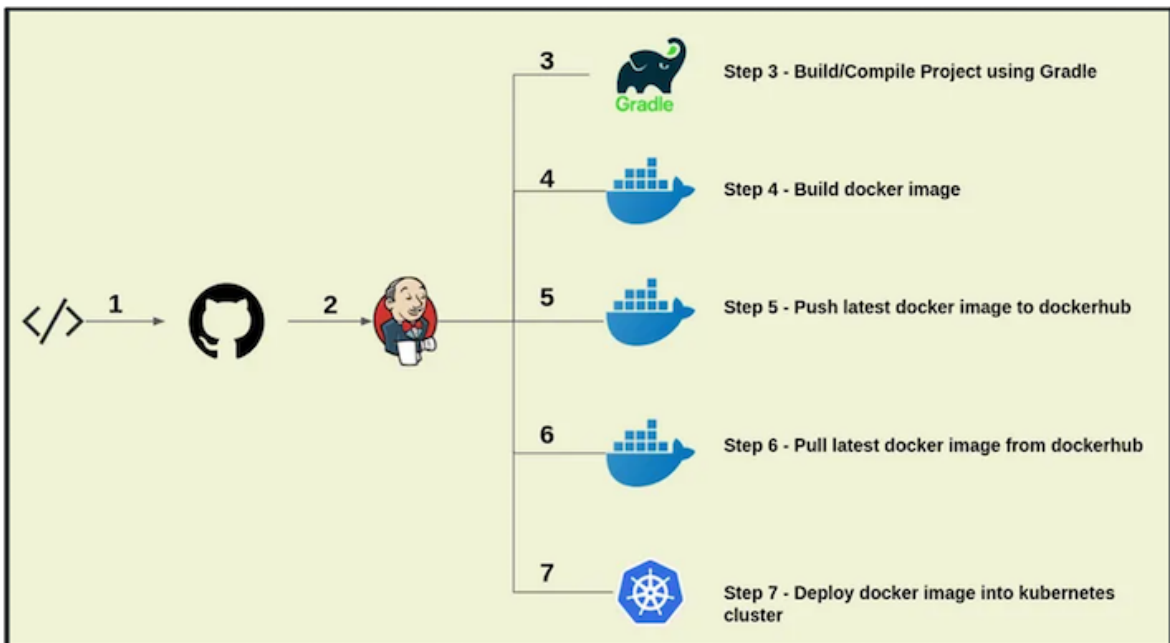


Figure 11: Installation steps

1. Checkin/Push our code to GitHub
2. Pull our code from Github into our Jenkins server.
3. Use Grade/Maven build tool for building the artifacts
4. Create docker images
5. Push latest docker images to DockerHub
6. Pull the latest images from DockerHub into Jenkins
7. Using our deployment file to deploy our applications inside our kubernetes cluster

The first thing we did was to create a GCE Virtual Machine. After creating a VM on GCE, we went to GCP Compute Engine from the navigation menu and click Create Instance with the name, region and the zone. We also configured the firewall rules for this virtual machine named jenkins see Figure 12.



Figure 12: Jenkins initialization

To install Jenkins, we first connected with the ssh terminal of our virtual machine then installed the java first using the commands in table 11.

```
$ sudo apt update // update the software packages  
  
$ sudo apt install default-jre // install jre jenkins environment
```

Table 11: Jenkins Environment Set Up Command.

Then we added a repository key to the system which we did by importing the GPG keys of the Jenkins. This command gave the output as ok and then we needed to append the Debian package repository address to the server's sources.list using the commands in table 12.

```
$ wget -q -o - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add-  
  
$ sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stablebinary/ > /etc/apt/sources.list.d/jenkins.list'
```

Table 12: Commands for appending the Debian package.

We installed Jenkins using the commands in Table 13 after setting up a correct environment.

```
$ sudo apt-get update // update the software packages  
  
$ sudo apt-get install jenkins // install jenkins
```

Table 13: Commands for installing Jenkins.

## 4.2 Creating a Jenkins pipeline script

After installing jenkins we verified our installation by accessing our initial login page and then change the username and password for the jenkins virtual machine. Our installation was successful completed and we setup the SSH pipeline for jenkins, setting up the maven, and docker for Jenkins.[9] We added a current jenkins user to the docker group so that Jenkins used Docker for building and pushing the docker images. We created a Jenkins pipeline script. So for running the pipeline we needed to store the github, docker and kubernetes, k8smaster server credentials in our jenkins. Jenkins pipeline script worked step by step and eventually upload it into the Google Cloud kubernetes and we were able to see that our services were running on the google cloud. The final script is on the github. The link is in Appendix A. In summary the CI/CD integration for our microservice deployment project was a success.

## 5 Discussions and Conclusion

Finally, we were able to deploy our redirection service gateway using dockers and kubernetes. We further went ahead to integrate the CI/CD Pipeline with Docker and kubernetes. We did validations and performance testing to checking our deployment. The results were tremendous as services could be redirected by our gateway. However some areas of further improvement include the following;

1. In the case of high traffic there is a likelihood of system failure. We did not interest ourselves in analyzing the throughput of the systems as it was beyond our area of research.
2. Security is a major issue in cloud technologies and more work needs to be done in this area to identify the security gaps and thus find solutions to these breach.
3. The dynamics of data being managed by an external third party need to be carefully studied particularly when deploying this technology.

## References

- [1] Yoshiteru Ishida Akhmad Alimudin. Service-based container deployment on kubernetes using stable marriage problem. *Proceedings of the 2020 The 6th International Conference on Frontiers of Educational Technologies*, 2020.
- [2] Ruth G. Lennon Brandon Thurgood. Cloud computing with kubernetes cluster elastic scaling. *Proceedings of the 3rd International Conference on Future Networks and Distributed Systems*, 2019.
- [3] Edureka. Microservices architecture training. Available at <https://www.edureka.co/blog/what-is-microservices/> (01.01.2021).
- [4] Javainuse. Deploy spring boot + mysql application to docker. Available at <https://www.javainuse.com/devOps/docker/docker-mysql> (21.01.2021).
- [5] Jhooq. Deployment of dockers and kubernetes. Available at <https://jhooq.com/deploy-spring-boot-microservices-on-kubernetes/#step-5> (17.01.2021).
- [6] Harold Castro-Mauricio Verano Lorena Salamanca Rubby Casallas Mario Vilamizar, Oscar Garcés. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. *International Conference on Emerging Trends in Information Technology and Engineering*, 2020.
- [7] Jaimeel M shah Nikhil Marathe, Ankita Gandhi. Docker swarm and kubernetes in cloud computing environment at proceedings of the third international conference on trends in electronics and informatics. 2019.
- [8] Iustin-Alexandru Ivanciu<sup>1</sup> Robert Botez<sup>1</sup>, Calin-Marian Iurian<sup>1</sup> and Virgil Dobrota<sup>1</sup>. Deploying a dockerized application with kubernetes on google cloud platform. *13th International Conference on Communications*, 2020.
- [9] Vaibhav Bejgam Sriniketan Mysari. Continuous integration and continuous deployment pipeline automation using jenkins ansible. 2020.

## A Appendix

The full code of our implementation and integration can be found on the github link:- [https://github.com/abdullahalnoman8/cloud\\_deployment](https://github.com/abdullahalnoman8/cloud_deployment)