Frankfurt University of Applied Sciences

MSc. in High Integrity Systems

A Project Report on

Deploying Web based project (PPMT) to Google cloud

Submitted for the Requirements of MSc. Program Course
"Cloud Computing", Winter Semester, 2021/2022

Authors Name	Matriculation Number
Abhishek Saha	1339877
Md Moshfiqul Alam	1340060
Tonmoy Deb Chowdhury	1340439
Sm Habibur Rahaman Shabu	1338315

Under the Supervision of
Prof. Dr.-Ing. Dipl.-Inform. Eicke Godehardt
The Faculty of Computer Science and Engineering
Frankfurt University of Applied Sciences

Table of Contents

1. Introduction

1.1 Background

In the 4th Industrial revolution, Containers have introduced new dimensions to the way software is built, shipped, and maintained. A containerized process is highly portable and repeatable, allowing us to relocate and deploy container applications more easily than previously. Docker is one of the most popular container platforms for developing, shipping, and running applications. It provides loosely isolated environment called container to package and run the application. [2] Docker is the most widely used container virtualization system [1] and provides several useful tools for development, maintenance, and deployment of cloud software applications. There are two types of resources in docker: containers and images.

Managing the life cycle of containers, particularly in a large and dynamic environment, it is quite important to use a container orchestration platform. Kubernetes is the most widely used orchestration technology. By using kubernetes, we can allocate, and deploy containers, scale up, and down number of containers based on demand, load balancing of the service discovery among containers, and many other things [3].

1.2 Objective and Motivation

For providing a web-based service it is very important to reach to the user or customer who requires global access to the software. To provide such a dynamic access, it is necessary to use cloud-based platform like google cloud. In our project our target is to build a software service and deploy it to the google cloud platform for global access so that users can easily access and have no issues.

At first, we build a system that can make an easy way for users to track their project in the shortest possible time. There are always some issues with downloading speed and uploading speed alongside the bandwidth limit. It also causes delays to stream something. To find out the solution for this situation pointing out to nearby servers is much more needed.

So, it motivated us to deploy our web-based application in a public cloud platform. To energize the project, we have studied several cases online and wanted to learn about project management system from there. We have seen it has various uses and different ways to build it. Which also gave us enough motivation to build something like that. We worked hard and built enough strength to make it successful. However, on the journey, we had faced several errors while deploying and creating clusters online and team issues which made us a little disturbed to complete the project but at the end, we tried to build something useful. In the later part of this report, it has been described how all things have been built, implemented, finalized, and deployed as well as tested on the cloud.

The scope of this document would function as an installation guide that offers step-by-step instructions for the web application deployment procedure.

2. Systems Overview

Our application is a Personal project management tools (PPMT) that fetch data from an in-memory database of the application and shows the project update to the user.

For the deployment of our web application to google cloud initially we use Docker and create docker container. The docker container mainly help us to packed up our project code, Dependencies, and configuration from the local machine. We use docker so that the project runs smoothly and same performance in the cloud server as it is in the local machine and no uncertainty occurs.

After creating docker container we use Kubernetes. This is our main tool for the deployment. Kubernetes mainly provides us an API to control how and where those docker containers will run. It also allows us to run our Docker containers and workloads to tackle some of the operating complexities when moving to scale the containers and deployed it on the google cloud servers. The deployment covers both local, and cloud Kubernetes cluster.

Finally with the help of Kubernetes we successfully deployed our project to google cloud platform.

2.1 System Architecture

The backbone of our system architecture is kubernetes cluster which deploy the docker container to google cloud. We have used GitHub as source control repository, and docker hub as a container repository. First, we pull our git project into local machine then create local docker image using docker from our spring boot application. Then we push the local docker image (Backend) and docker image (Frontend) to docker hub and then from google cloud platform using google Kubernetes engine. we are able to run the docker image as a container in our

Kubernetes cluster. We tested the commands to make our application work smoothly. Our high-level system architecture is illustrated in figure 1.

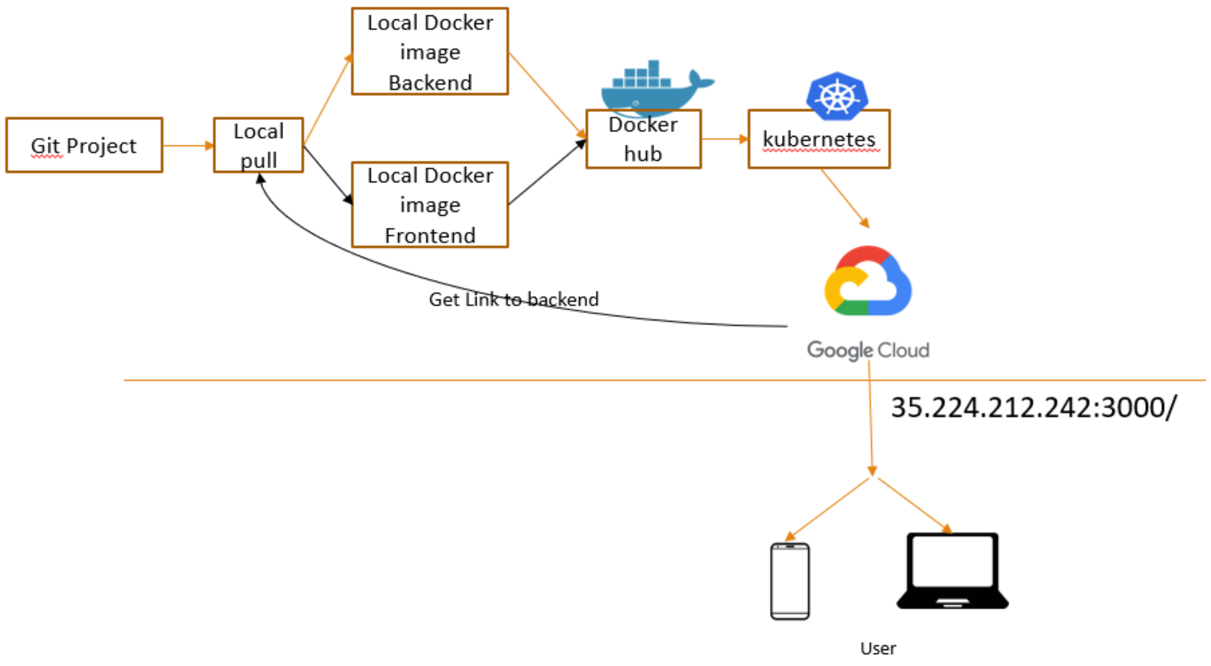


Figure 1: Systems Architecture

2.2 Tools and Methodologies

In this project, we have used number of tools including some DevOps tools, and the techniques which are listed below:

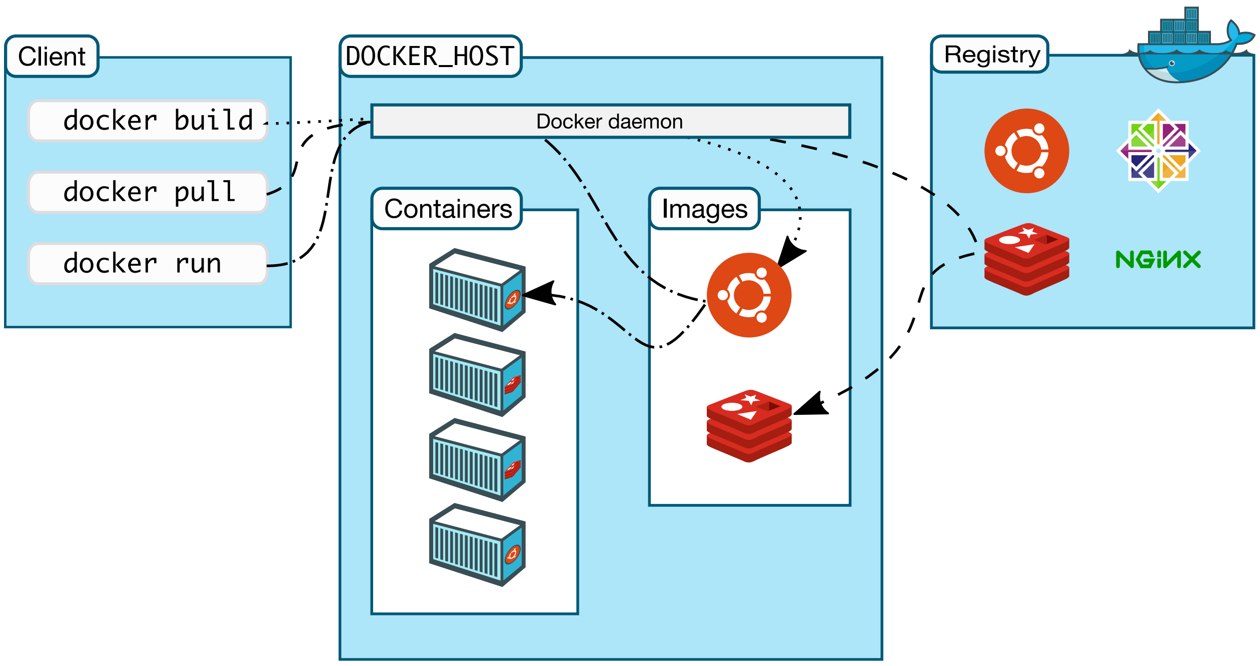
```
Programming
Language:Java

Database: MySQL
Framework: Spring Boot
Application Build Tool: Apache
Maven DevOps Tools: Docker
KubernetesCloud Platform: Google
Cloud Platform Source Control
Repository: GitHub Container
Repository: DockerHub
```

3. Deployment

3.1 Docker

Docker uses a client-server architecture. The Docker client connects to the Docker host, that can build, run, and distribute Docker containers. The Docker client and docker host can run on the same system. Then it can easily push to the docker hub. Docker is a free and open-source platform for creating, delivering, and operating applications. Docker allows us to separate our application from the infrastructure which allows us to manage our infrastructure in the same manner that we can control our applications. It helps us to minimize the time between writing code and executing it by utilizing Docker's methodology for fast implementation, testing, and deploying code of our project.

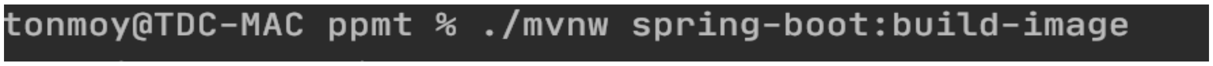
References: https://docs.docker.com/get-started/overview/



References: https://docs.docker.com/get-started/overview/

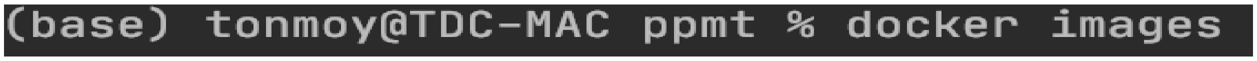
3.1.1 Docker Image Create (Backend)

1. First we install docker to our local machine. Opened ppmt project folder and run the following commands on the terminal for creating the docker image.



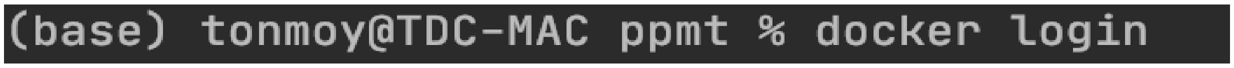
```
tonmoy@TDC-MAC ppmt % ./mvnw spring-boot:build-image
```

2. After creating the docker image we check the images of our project by running the following command.



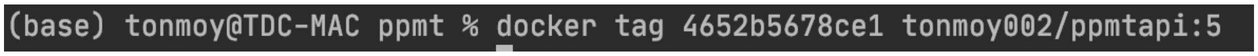
```
(base) tonmoy@TDC-MAC ppmt % docker images
```

3. Now we create a Docker hub account to push our local docker image into Docker hub. Login to docker hub by providing credentials using following commands.



```
(base) tonmoy@TDC-MAC ppmt % docker login
```

4. Now we need to tag a docker image before we push it into docker hub. For this we need the ID of our docker image give a name and tag it by 5 by using the following command.

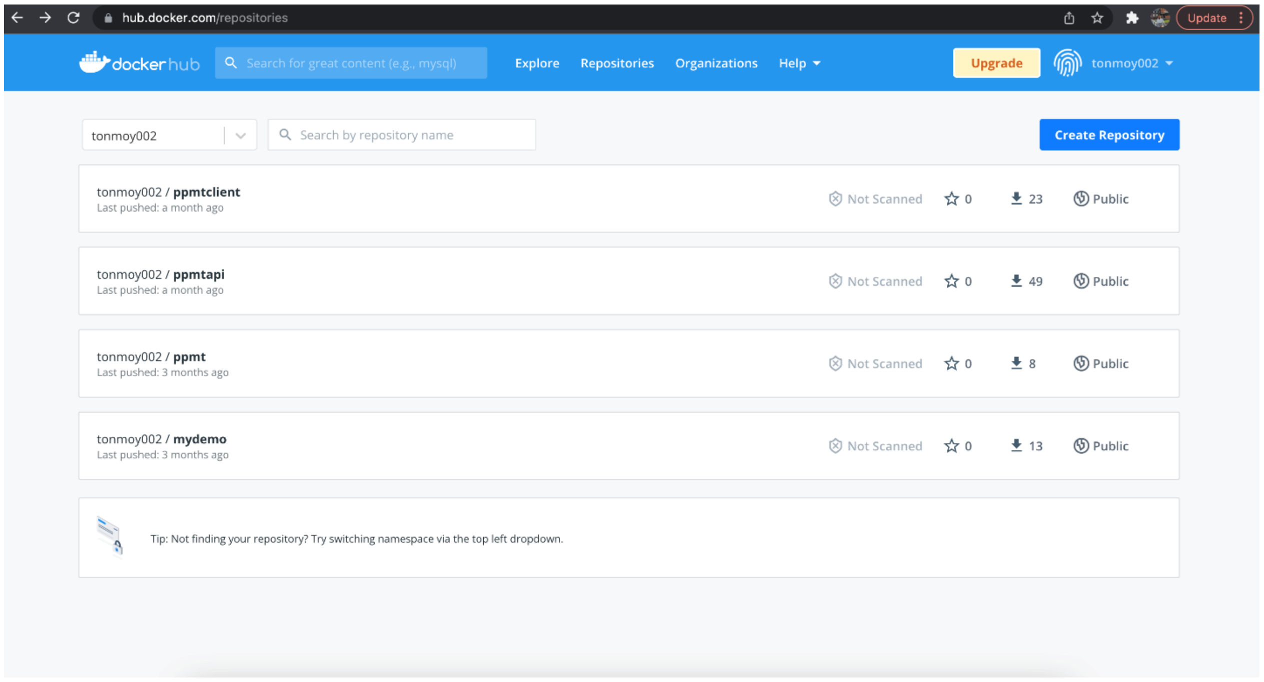


```
(base) tonmoy@TDC-MAC ppmt % docker tag 4652b5678ce1 tonmoy002/ppmtapi:5
```

5. Now we push our docker image into docker hub using the tag by the following command.

```
(base) tonmoy@TDC-MAC ppmt % docker push tonmoy002/ppmtapi:5
```

6. We can see that our docker image successfully uploaded to our docker hub account and no issue occurs.


hub.docker.com/repositories
docker hub
Search for great content (e.g., mysql)
Explore
Repositories
Organizations
Help
Upgrade
tonmoy002
Search by repository name
Create Repository
Update

tonmoy002 / **ppmtclient** Last pushed: a month ago — Not Scanned ☆ 0 ⬇ 23 Public

tonmoy002 / **ppmtapi** Last pushed: a month ago — Not Scanned ☆ 0 ⬇ 49 Public

tonmoy002 / **ppmt** Last pushed: 3 months ago — Not Scanned ☆ 0 ⬇ 8 Public

tonmoy002 / **mydemo** Last pushed: 3 months ago — Not Scanned ☆ 0 ⬇ 13 Public

Tip: Not finding your repository? Try switching namespace via the top left dropdown.

Project completed After that:

1. Create google cloud account.
2. Created two new projects on Google cloud named PPMT and PPMT-CLIENT respectively.
3. Created Database instance in PPMT project named **ppmtdb** and created database name: **ppmt**

3.2 Kubernetes

Kubernetes is a portable and scalable open-source platform which manage the container-based applications and services. It is a container orchestration system that organizes computer, network, and storage infrastructure. The master and node components are the most important parts of Kubernetes. The Kubernetes cluster is controlled and managed by the master node, which also comprises a set of worker nodes. As the smallest deployable unit on the nodes, the system orchestrates so-called pods. One or more containers are contained in each pod. The master node's storage and network resources are shared by these containers. Each container has its own set of instructions about how to execute it. They have a unique IP address that they use to access the pod. However, each time a pod fails, a new one must be produced, along with a new IP address.

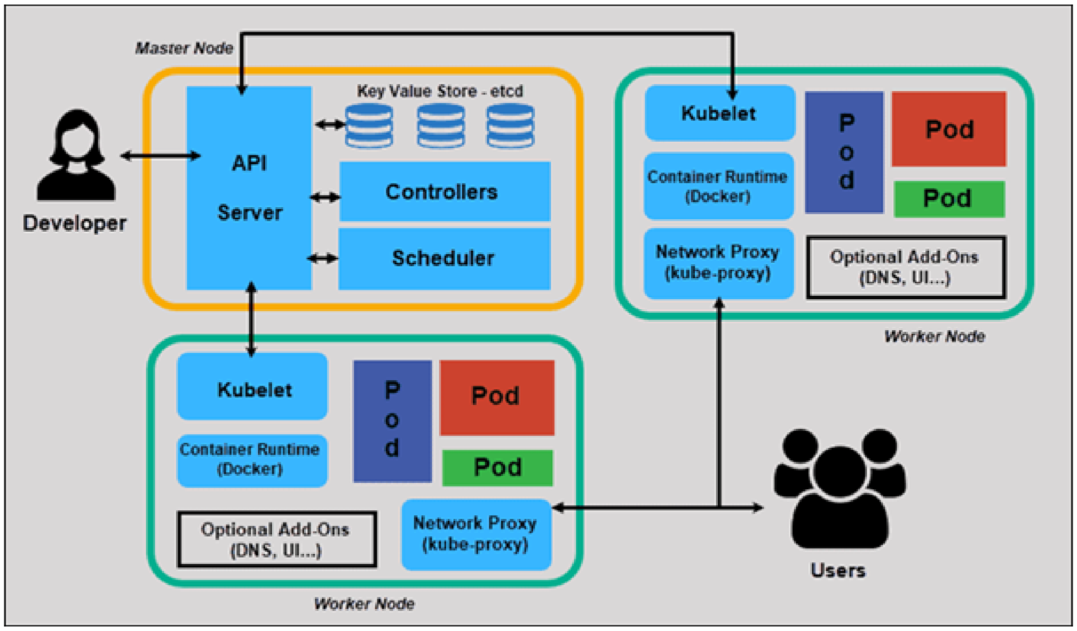
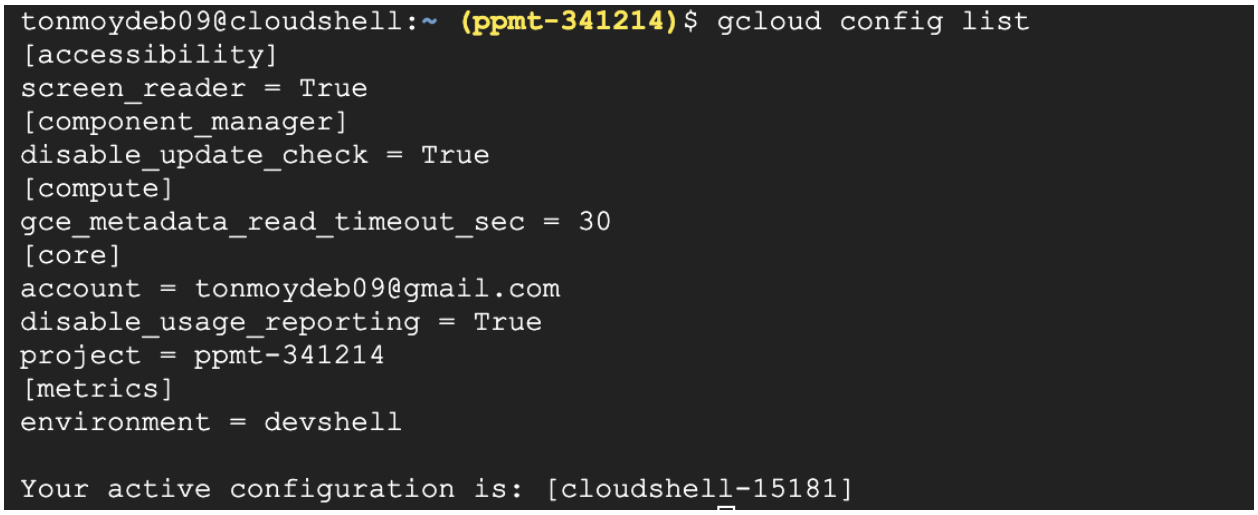

Master Node
Key Value Store - etcd
Developer
API Server
Controllers
Scheduler
Kubelet
Container Runtime (Docker)
Network Proxy (kube-proxy)
Pod
Pod
Pod
Optional Add-Ons (DNS, UI...)
Worker Node
Kubelet
Container Runtime (Docker)
Pod
Pod
Pod
Optional Add-Ons (DNS, UI...)
Network Proxy (kube-proxy)
Worker Node
Users

Fig: Kubernetes Architecture [References: https://www.ais.com/an-overview-of-kubernetes-architecture-and-container-deployment/]

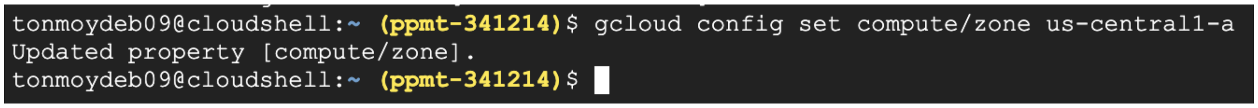
3.2.1 Kubernetes Implementation

1. We already have the docker image in the docker hub. Now we want to run our application as a pod in Kubernetes. For this, we first create a dummy project named PPMT on google cloud and open cloud shell. This cloud shell allows us to create resources and interact using this shell. First we check the configuration list by using the command **'gcloud config list'** and see we are assigned in and we have a project.



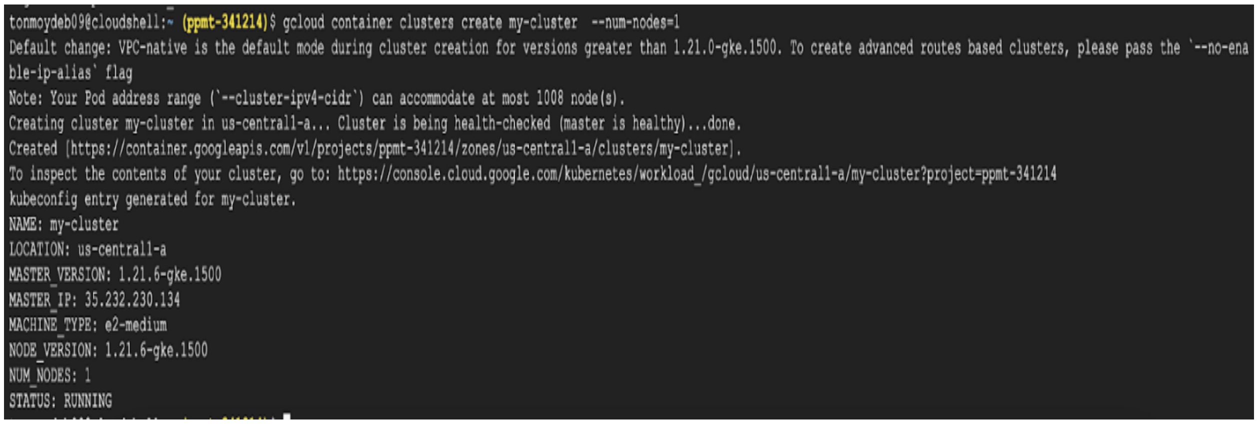
```
tonmoydeb09@cloudshell:~ (ppmt-341214)$ gcloud config list
[accessibility]
screen_reader = True
[component_manager]
disable_update_check = True
[compute]
gce_metadata_read_timeout_sec = 30
[core]
account = tonmoydeb09@gmail.com
disable_usage_reporting = True
project = ppmt-341214
[metrics]
environment = devshell

Your active configuration is: [cloudshell-15181]
```

2. Now we need to set our zone before we create our cluster for Kubernetes. To do that we have to use the command '**gcloud config set compute/zone us-centrall-a'**



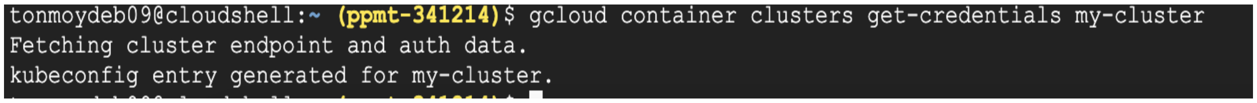
```
tonmoydeb09@cloudshell:~ (ppmt-341214)$ gcloud config set compute/zone us-central1-a
Updated property [compute/zone].
tonmoydeb09@cloudshell:~ (ppmt-341214)$
```

3. Now we are ready to create our container cluster. For creating the container cluster, we use **'gcloud container clusters my-cluster –num-nodes=**1' command and set the number of nodes 1. Because our project is small.



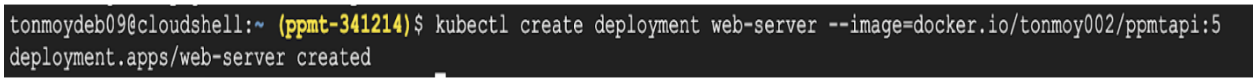
```
tonmoydeb09@cloudshell:~ (ppmt-341214)$ gcloud container clusters create my-cluster  --num-nodes=1
Default change: VPC-native is the default mode during cluster creation for versions greater than 1.21.0-gke.1500. To create advanced routes based clusters, please pass the `--no-ena
ble-ip-alias` flag
Note: Your Pod address range (`--cluster-ipv4-cidr`) can accommodate at most 1008 node(s).
Creating cluster my-cluster in us-central1-a... Cluster is being health-checked (master is healthy)...done.
Created [https://container.googleapis.com/v1/projects/ppmt-341214/zones/us-central1-a/clusters/my-cluster].
To inspect the contents of your cluster, go to: https://console.cloud.google.com/kubernetes/workload_/gcloud/us-central1-a/my-cluster?project=ppmt-341214
kubeconfig entry generated for my-cluster.
NAME: my-cluster
LOCATION: us-central1-a
MASTER_VERSION: 1.21.6-gke.1500
MASTER_IP: 35.232.230.134
MACHINE_TYPE: e2-medium
NODE_VERSION: 1.21.6-gke.1500
NUM_NODES: 1
STATUS: RUNNING
```

4. Now our cluster is ready, and we want to deploy our project to this cluster. We are now able to get the credentials for interacting with this cluster using kubectl. To do this we use the command below.



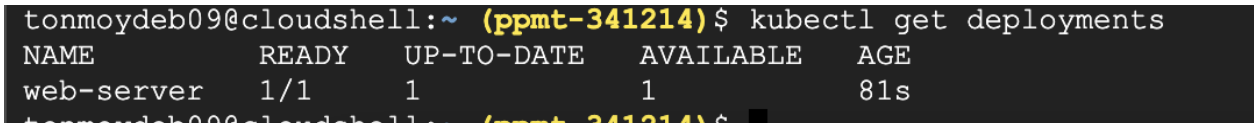
```
tonmoydeb09@cloudshell:~ (ppmt-341214)$ gcloud container clusters get-credentials my-cluster
Fetching cluster endpoint and auth data.
kubeconfig entry generated for my-cluster.
```

6. Now we can use kubeclt and want to create a deployment using the image that we created. For creating deployment of the web server, we use the following command.



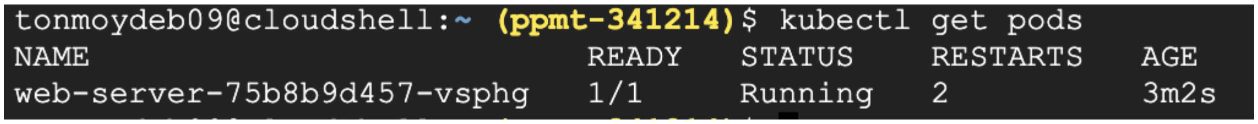
```
tonmoydeb09@cloudshell:~ (ppmt-341214)$ kubectl create deployment web-server --image=docker.io/tonmoy002/ppmtapi:5
deployment.apps/web-server created
```

7.Now we can get the deployment information by **‘kubectl get deployments’** command.



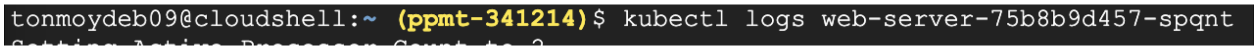
```
tonmoydeb09@cloudshell:~ (ppmt-341214)$ kubectl get deployments
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
web-server   1/1     1            1           81s
```

8. For checking the pods status we use **‘kubectl get pods’**.



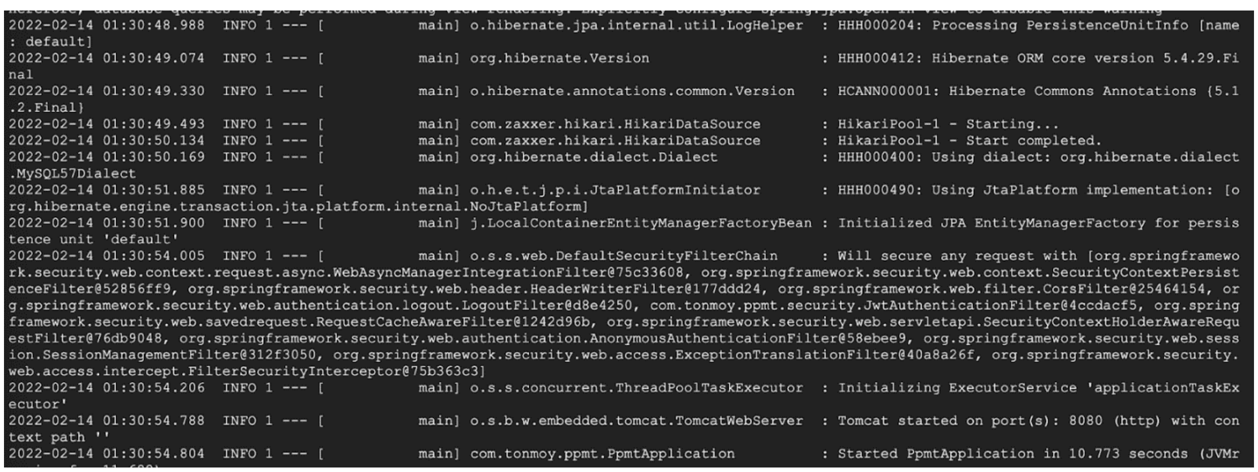
```
tonmoydeb09@cloudshell:~ (ppmt-341214)$ kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
web-server-75b8b9d457-vsphg   1/1     Running   2          3m2s
```

9.Now we check the spring boot logs status by using the below command.



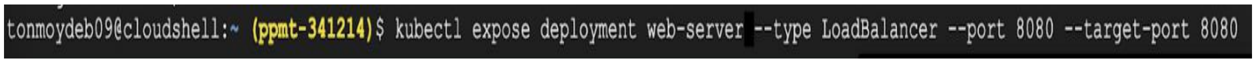
```
tonmoydeb09@cloudshell:~ (ppmt-341214)$ kubectl logs web-server-75b8b9d457-spqnt
```

10.After successfully run of the application it will show a status.



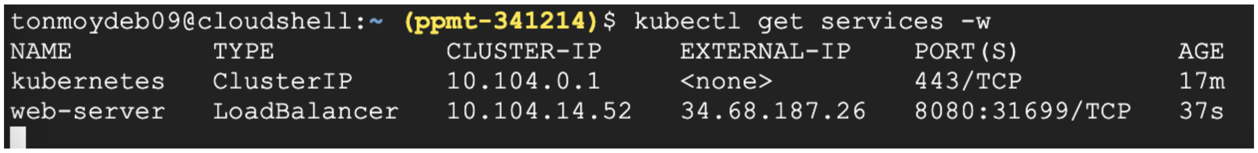
```
2022-02-14 01:30:48.988  INFO 1 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name
: default]
2022-02-14 01:30:49.074  INFO 1 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 5.4.29.Fi
nal
2022-02-14 01:30:49.330  INFO 1 --- [           main] o.hibernate.annotations.common.Version   : HCANN000001: Hibernate Commons Annotations {5.1
.2.Final}
2022-02-14 01:30:49.493  INFO 1 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2022-02-14 01:30:50.134  INFO 1 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2022-02-14 01:30:50.169  INFO 1 --- [           main] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibernate.dialect
.MySQL57Dialect
2022-02-14 01:30:51.885  INFO 1 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [o
rg.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2022-02-14 01:30:51.900  INFO 1 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persis
tence unit 'default'
2022-02-14 01:30:54.005  INFO 1 --- [           main] o.s.s.web.DefaultSecurityFilterChain     : Will secure any request with [org.springframewo
rk.security.web.context.request.async.WebAsyncManagerIntegrationFilter@75c33608, org.springframework.security.web.context.SecurityContextPersist
enceFilter@52856ff9, org.springframework.security.web.header.HeaderWriterFilter@177ddd24, org.springframework.web.filter.CorsFilter@25464154, or
g.springframework.security.web.authentication.logout.LogoutFilter@d8e4250, com.tonmoy.ppmt.security.JwtAuthenticationFilter@4ccdacf5, org.spring
framework.security.web.savedrequest.RequestCacheAwareFilter@1242d96b, org.springframework.security.web.servletapi.SecurityContextHolderAwareRequ
estFilter@76db9048, org.springframework.security.web.authentication.AnonymousAuthenticationFilter@58ebee9, org.springframework.security.web.sess
ion.SessionManagementFilter@312f3050, org.springframework.security.web.access.ExceptionTranslationFilter@40a8a26f, org.springframework.security.
web.access.intercept.FilterSecurityInterceptor@75b363c3]
2022-02-14 01:30:54.206  INFO 1 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskEx
ecutor'
2022-02-14 01:30:54.788  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with con
text path ''
2022-02-14 01:30:54.804  INFO 1 --- [           main] com.tonmoy.ppmt.PpmtApplication          : Started PpmtApplication in 10.773 seconds (JVMr
```

11.We already have the pods, but this pod cannot be accessed from the public internet. So, we need to expose it using services. So, we create a service using kubectl using the expose deployment command and the name of the deployment which is webserver and type that is Load Balancer after that we need to map our ports. Here we put our port 8080 and also our targeted port 8080.



```
tonmoydeb09@cloudshell:~ (ppmt-341214)$ kubectl expose deployment web-server --type LoadBalancer --port 8080 --target-port 8080
```

12. We use the command **'kubectl get services -w**' for the external IP. Below the screenshot we can see that our external IP is generated.

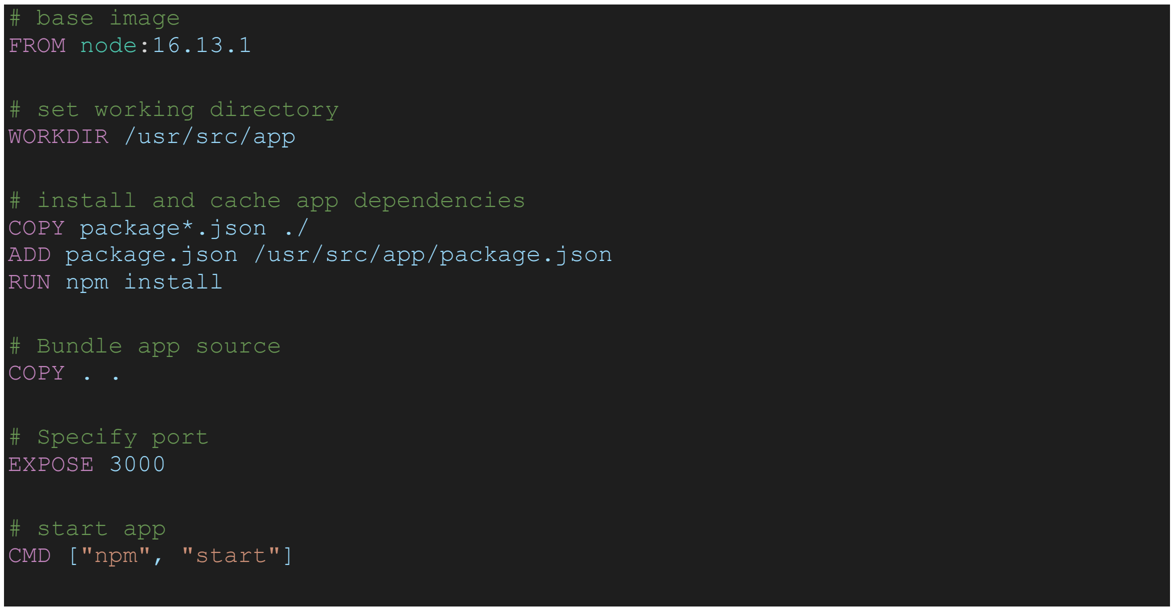


```
tonmoydeb09@cloudshell:~ (ppmt-341214)$ kubectl get services -w
NAME          TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)          AGE
kubernetes    ClusterIP      10.104.0.1     <none>         443/TCP          17m
web-server    LoadBalancer   10.104.14.52   34.68.187.26   8080:31699/TCP   37s
```

By using this external IP address, we can see that our pod is running our application which has a controller that return the services of the application.

Now we will deploy the react application for user interaction using docker **(Frontend):**

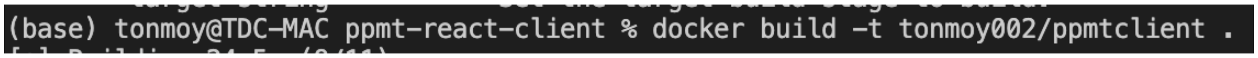
1. First create a Docker file without any extension and adding these commands in Docker file:



```
# base image
FROM node:16.13.1

# set working directory
WORKDIR /usr/src/app

# install and cache app dependencies
COPY package*.json ./
ADD package.json /usr/src/app/package.json
RUN npm install

# Bundle app source
COPY . .

# Specify port
EXPOSE 3000

# start app
CMD ["npm", "start"]
```

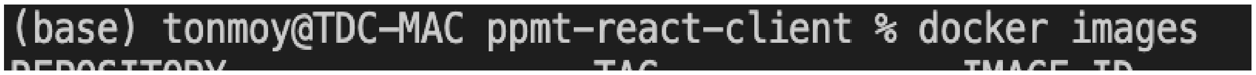
This command will initiate node package and install necessary packages from package.json file.

2. After the necessary package installation, we build an image file using the command in docker:



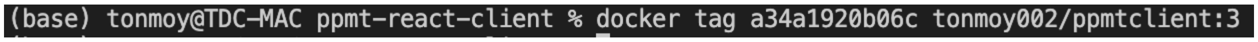
```
(base) tonmoy@TDC-MAC ppmt-react-client % docker build -t tonmoy002/ppmtclient .
```

3. We Check the docker images is created successfully by the following command:



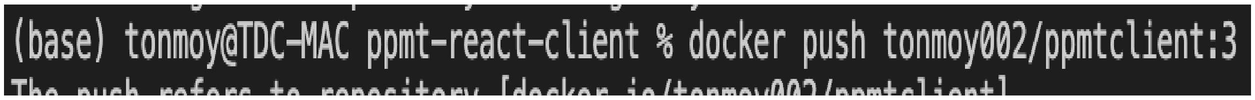
```
(base) tonmoy@TDC-MAC ppmt-react-client % docker images
```

4. Now we need the image tag id to push the docker image into docker hub. By using the commend we get the tag id.

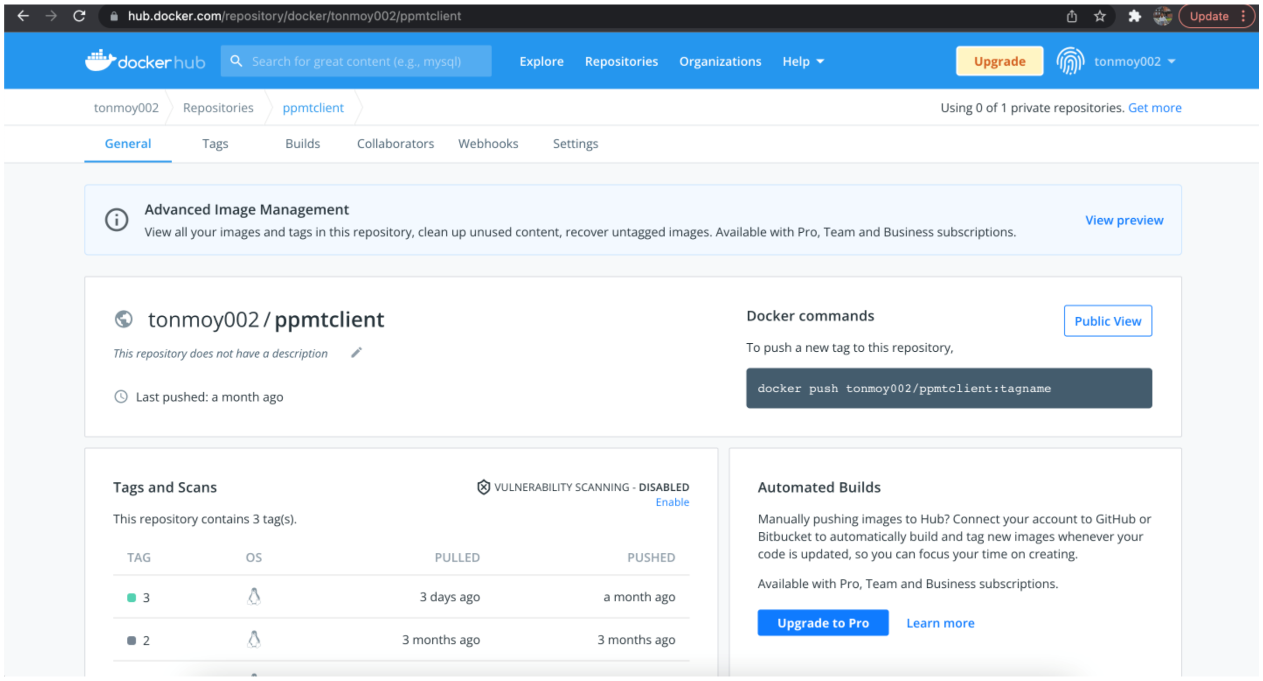


```
(base) tonmoy@TDC-MAC ppmt-react-client % docker tag a34a1920b06c tonmoy002/ppmtclient:3
```

5. Now we have the tag id and need to push it to docker hub. To upload our docker image we use the **'docker push'** command.



```
(base) tonmoy@TDC-MAC ppmt-react-client % docker push tonmoy002/ppmtclient:3
The push refers to repository [docker.io/tonmoy002/ppmtclient]
```



hub.docker.com/repository/docker/tonmoy002/ppmtclient

tonmoy002 › Repositories › ppmtclient

Using 0 of 1 private repositories. Get more

General | Tags | Builds | Collaborators | Webhooks | Settings

Advanced Image Management
View all your images and tags in this repository, clean up unused content, recover untagged images. Available with Pro, Team and Business subscriptions.

tonmoy002 / ppmtclient

This repository does not have a description

Last pushed: a month ago

Docker commands

To push a new tag to this repository,

```
docker push tonmoy002/ppmtclient:tagname
```

Tags and Scans

VULNERABILITY SCANNING - DISABLED

This repository contains 3 tag(s).

TAG	OS	PULLED	PUSHED
3		3 days ago	a month ago
2		3 months ago	3 months ago

Automated Builds

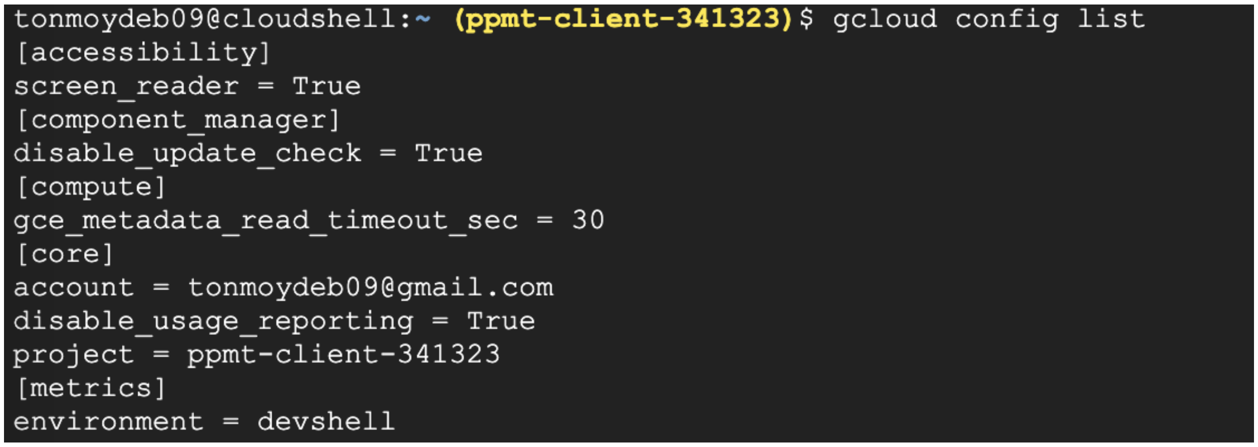
Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

Available with Pro, Team and Business subscriptions.

Now our docker image is ready to deploy to google cloud.

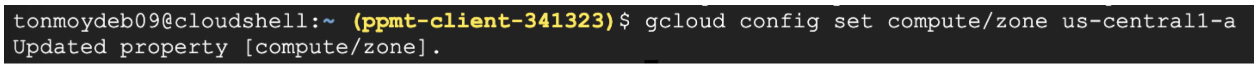
Initially we create a dummy google cloud project and named it PPMT-CLIENT. Open the cloud shell and set up the basic needs:

1. First, we opened the cloud shell and by the following command **'gcloud config list'** we Check the configuration list:



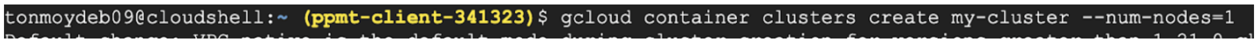
```
tonmoydeb09@cloudshell:~ (ppmt-client-341323)$ gcloud config list
[accessibility]
screen_reader = True
[component_manager]
disable_update_check = True
[compute]
gce_metadata_read_timeout_sec = 30
[core]
account = tonmoydeb09@gmail.com
disable_usage_reporting = True
project = ppmt-client-341323
[metrics]
environment = devshell
```

2. Now we have to update the server zone before creating the Kubernetes cluster. For this we use the following command.



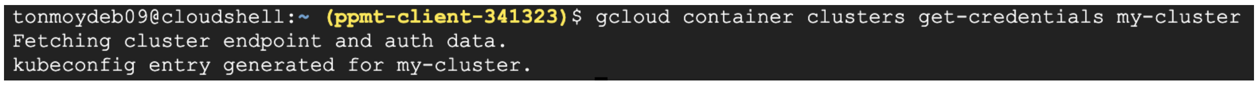
```
tonmoydeb09@cloudshell:~ (ppmt-client-341323)$ gcloud config set compute/zone us-central1-a
Updated property [compute/zone].
```

3. After updating the server zone, we enable Kubernetes API engine from that particular project. Then we run the following command to create cluster.



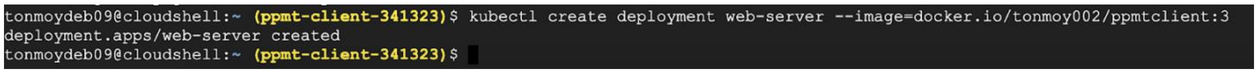
```
tonmoydeb09@cloudshell:~ (ppmt-client-341323)$ gcloud container clusters create my-cluster --num-nodes=1
```

5. Now we need the credential of the cluster to deploy it on google cloud. By using the following command, we get the cluster credentials.



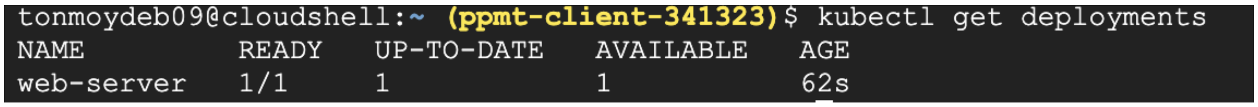
```
tonmoydeb09@cloudshell:~ (ppmt-client-341323)$ gcloud container clusters get-credentials my-cluster
Fetching cluster endpoint and auth data.
kubeconfig entry generated for my-cluster.
```

6. Now our cluster is ready for deployment using Kubectl. So, we use the command **"kubectl create deployment web-server –image=docker.io/tonmoy002/ppmtclient:3 "** and see that our deployment is created successfully.



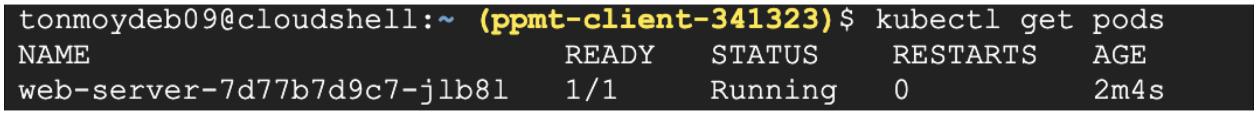
```
tonmoydeb09@cloudshell:~ (ppmt-client-341323)$ kubectl create deployment web-server --image=docker.io/tonmoy002/ppmtclient:3
deployment.apps/web-server created
tonmoydeb09@cloudshell:~ (ppmt-client-341323)$
```

7. For checking the deployment status, we use **"kubectl get deployments**" command and see the credentials.



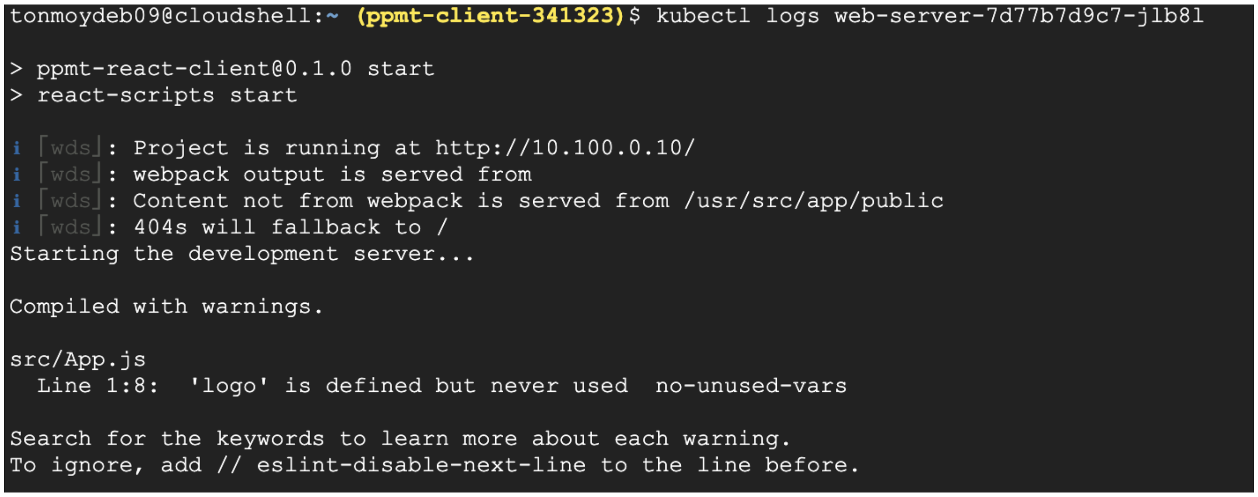
```
tonmoydeb09@cloudshell:~ (ppmt-client-341323)$ kubectl get deployments
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
web-server   1/1     1            1           62s
```

8. Now we use "**kubectl get pods"** to get the pod credentials which is needed to check the logs.



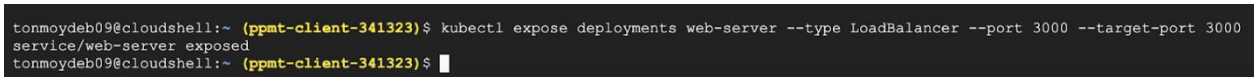
```
tonmoydeb09@cloudshell:~ (ppmt-client-341323)$ kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
web-server-7d77b7d9c7-jlb8l   1/1     Running   0          2m4s
```

9. We already have the pod name so we can easily check the react log status using "**kubectl logs**":



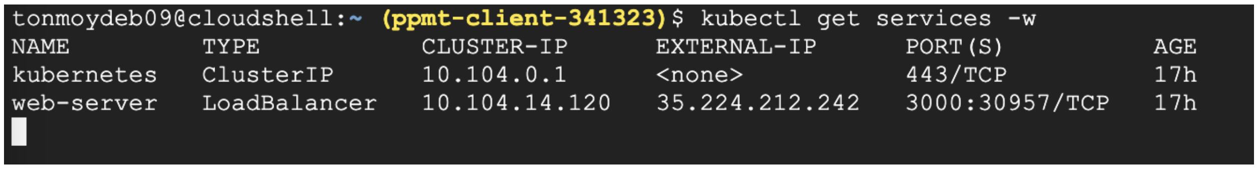
```
tonmoydeb09@cloudshell:~ (ppmt-client-341323)$ kubectl logs web-server-7d77b7d9c7-jlb8l

> ppmt-react-client@0.1.0 start
> react-scripts start

i ｢wds｣: Project is running at http://10.100.0.10/
i ｢wds｣: webpack output is served from
i ｢wds｣: Content not from webpack is served from /usr/src/app/public
i ｢wds｣: 404s will fallback to /
Starting the development server...

Compiled with warnings.

src/App.js
  Line 1:8:  'logo' is defined but never used  no-unused-vars

Search for the keywords to learn more about each warning.
To ignore, add // eslint-disable-next-line to the line before.
```

10. We have the pods, but this pod cannot be accessed from the public internet. So, we need to expose it using services. So, we create a service using kubectl using the expose deployment command and the name of the deployment which is webserver and type that is Load Balancer after that we need to map our ports. Here we put our port 3000 and also our targeted port is 3000.



```
tonmoydeb09@cloudshell:~ (ppmt-client-341323)$ kubectl expose deployments web-server --type LoadBalancer --port 3000 --target-port 3000
service/web-server exposed
tonmoydeb09@cloudshell:~ (ppmt-client-341323)$
```

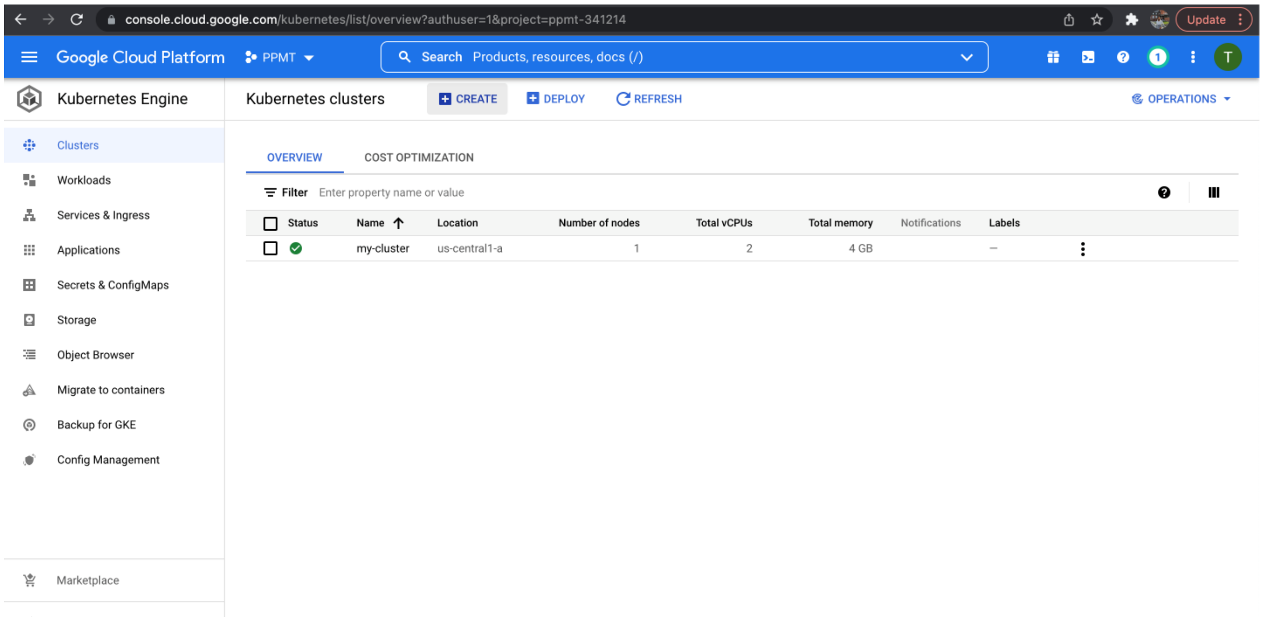
11. Now we successfully deployed our frontend to google cloud. For further use we need the external IP. For this we use **"kubectl get services -w"** command and our external IP is created.



```
tonmoydeb09@cloudshell:~ (ppmt-client-341323)$ kubectl get services -w
NAME         TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)          AGE
kubernetes   ClusterIP      10.104.0.1     <none>           443/TCP          17h
web-server   LoadBalancer   10.104.14.120  35.224.212.242   3000:30957/TCP   17h
```

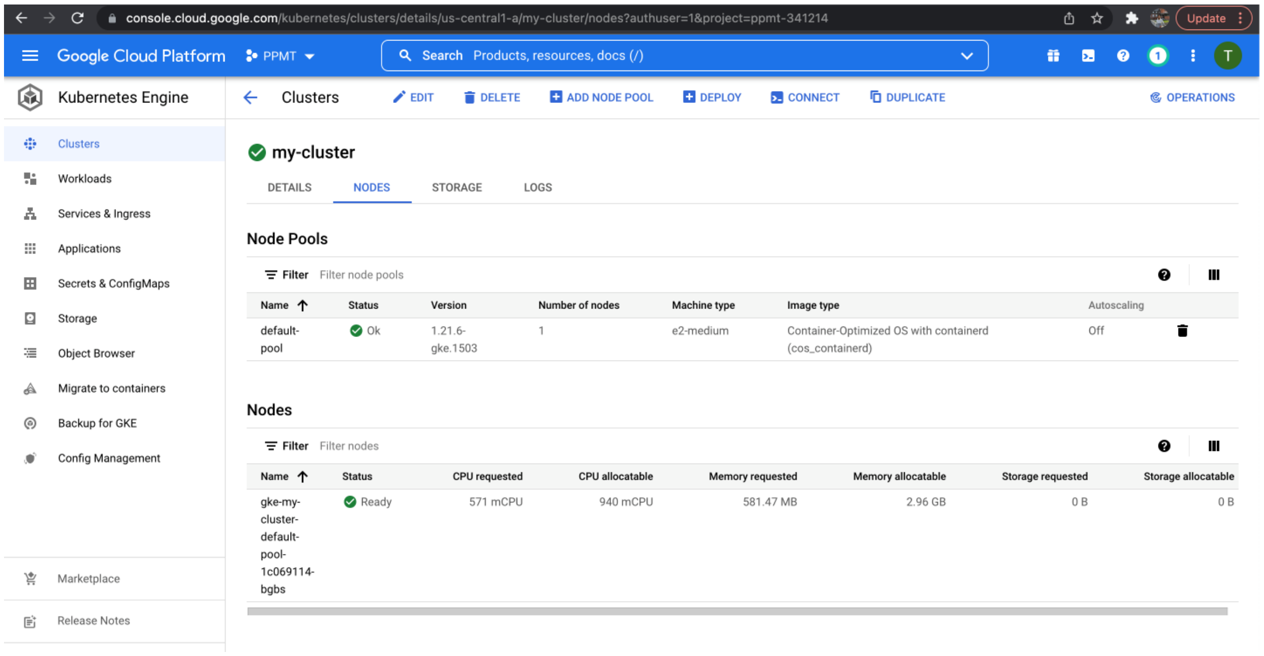
From the screenshot we get our external IP 35.224.212.242, and our port is 3000. So now anyone can easily get our services by using the IP address and port. Here is our final IP address.
http://35.224.212.242:3000/

12. Form the google cloud account We can see the overview status of our Kubernetes cluster.



Status	Name	Location	Number of nodes	Total vCPUs	Total memory	Notifications	Labels
✓	my-cluster	us-central1-a	1	2	4 GB		—

13. From the cluster we can see the Node status. We have only one node because our project is small.



Node Pools

Filter Filter node pools

Name	Status	Version	Number of nodes	Machine type	Image type	Autoscaling
default-pool	✓ Ok	1.21.6-gke.1503	1	e2-medium	Container-Optimized OS with containerd (cos_containerd)	Off

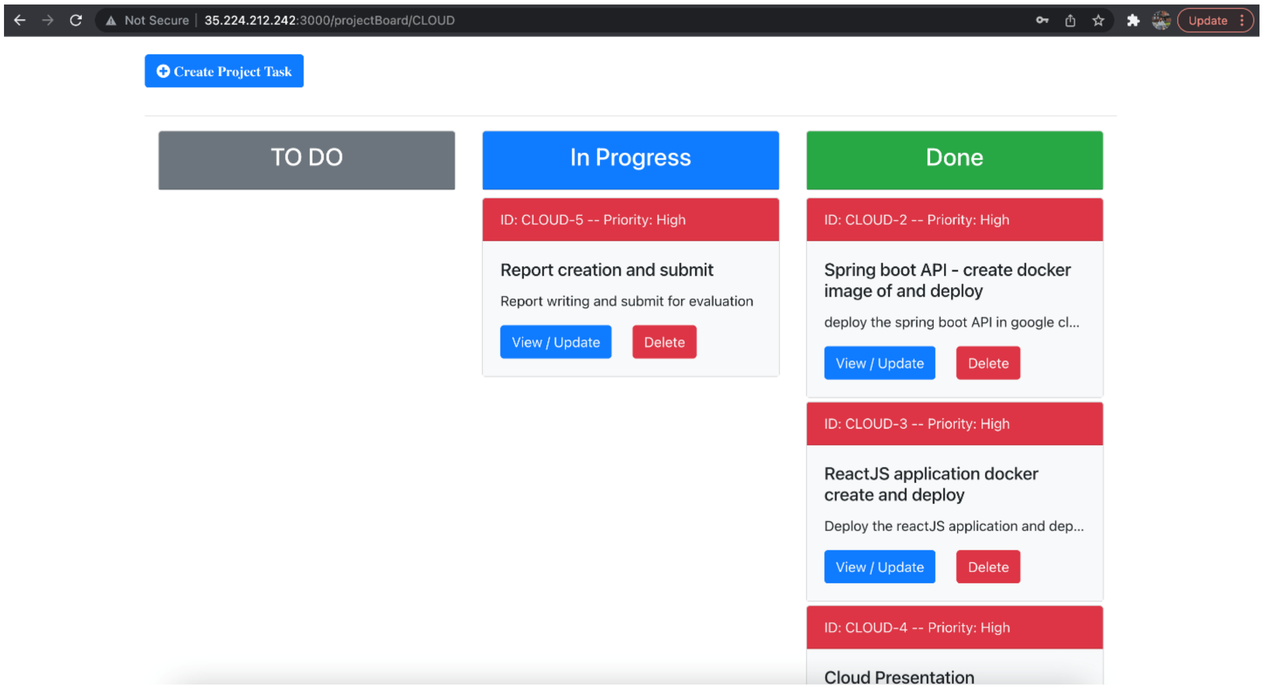
Nodes

Filter Filter nodes

Name	Status	CPU requested	CPU allocatable	Memory requested	Memory allocatable	Storage requested	Storage allocatable
gke-my-cluster-default-pool-1c069114-bgbs	✓ Ready	571 mCPU	940 mCPU	581.47 MB	2.96 GB	0 B	0 B

4. Result and Discussion

Finally, we are able to deploy our application on google cloud platform (GCP) using dockers and Kubernetes. We further went ahead to integrate with Docker and Kubernetes. We did validations and performance testing to checking our deployment. The test results were tremendous as services could be redirected by our gateway.



ID: CLOUD-5 -- Priority: High
Report creation and submit
Report writing and submit for evaluation

ID: CLOUD-2 -- Priority: High
Spring boot API - create docker image of and deploy
deploy the spring boot API in google cl...

ID: CLOUD-3 -- Priority: High
ReactJS application docker create and deploy
Deploy the reactJS application and dep...

ID: CLOUD-4 -- Priority: High
Cloud Presentation

5. Future work

We are planning to update our project and add more features. So, we will use more nodes in future, and we also have plan to automate our service and update by using Jenkins's pipeline. We will investigate other cloud platforms such as private and hybrid cloud and their services in the future. We'd also like to experiment with containerized microservice deployment and service discovery.

6. Conclusion

This paper explored docker container deployment using Kubernetes cluster to Google Cloud Platform (GCP) and our methodologies for doing so. It contains detailed instructions for configuring and installing the required tools and services. Finally, we went over the deployment, testing methods and reviewed the results of the deployments.

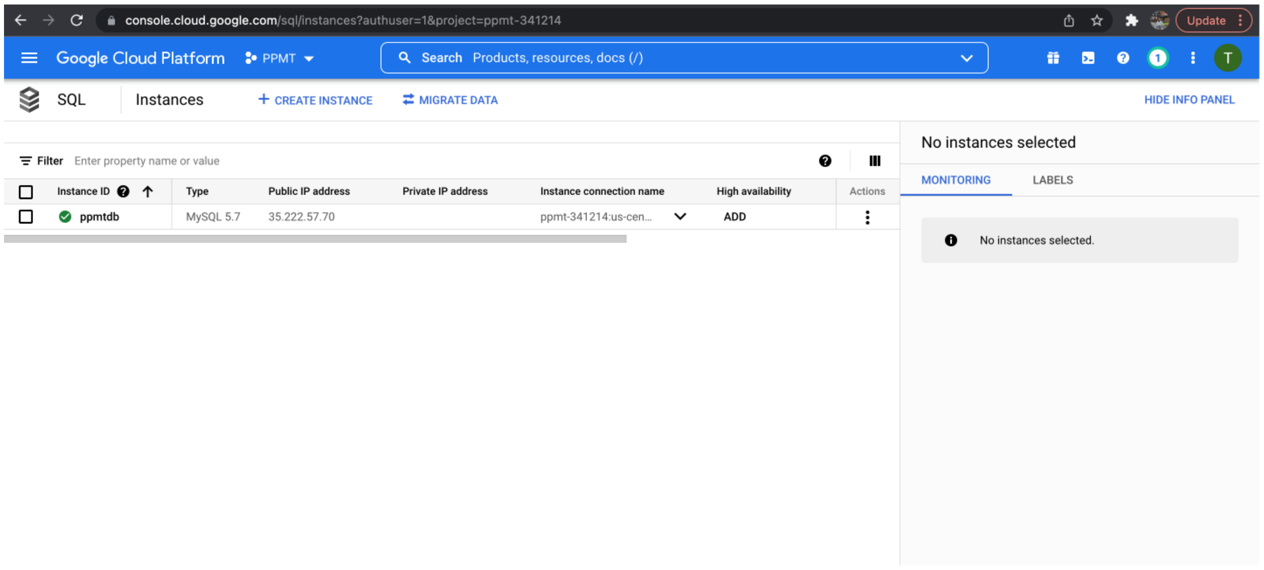
Acknowledgement

We would like to thank our mentor, Prof. Dr.-Ing. Dipl.-Inform. Eicke Godehardt, for his support and guidance throughout our project and Frankfurt University of Applied Sciences for providing us different opportunity for exhibiting our talents. The completion of this project could not have been possible without the expertise of Prof. Dr.-Ing. Dipl.-Inform. Eicke Godehardt, our beloved project adviser.

Database Deployment:

Google cloud SQL is a fully managed solution for the issues of scaling, replication, shorting database. So, we prefer and use this service.

First, we create a SQL instance named ppmtdb to our google cloud using the public IP address 35.222.57.70 and also generates the instance connection name.



console.cloud.google.com/sql/instances?authuser=1&project=ppmt-341214

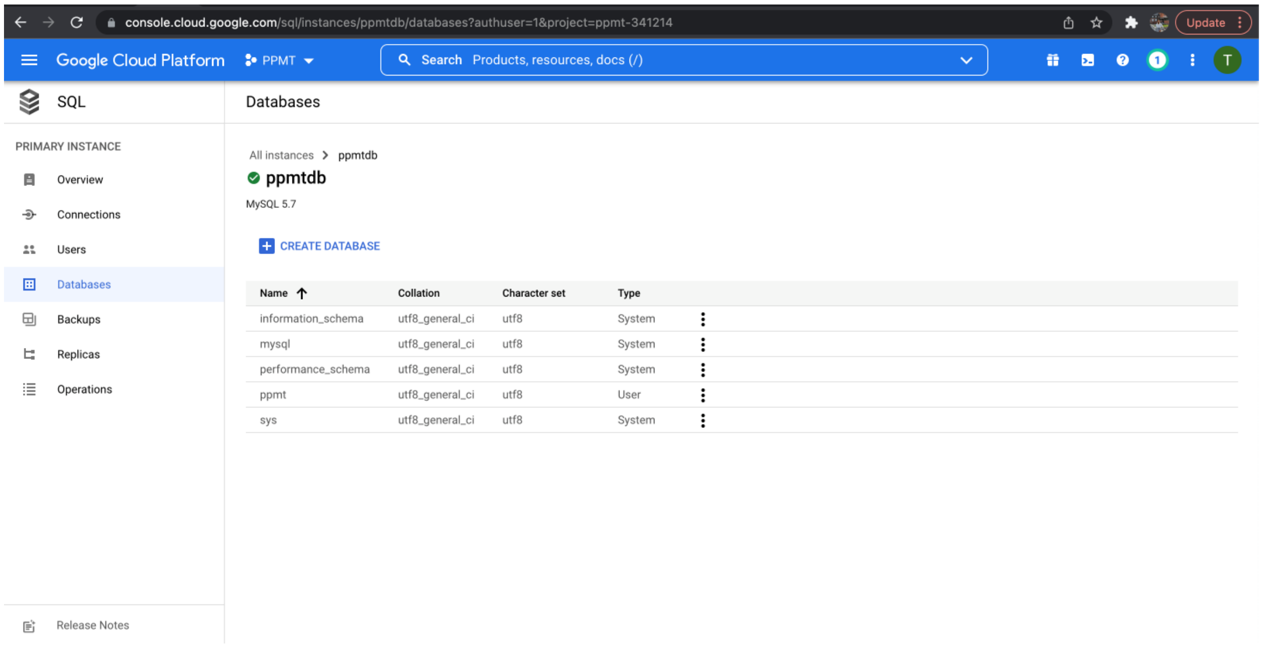
Instance ID	Type	Public IP address	Private IP address	Instance connection name	High availability	Actions
ppmtdb	MySQL 5.7	35.222.57.70		ppmt-341214:us-cen...	ADD	

No instances selected

No instances selected.

So, our MySQL server is successfully configured, and we can see the status.

2. Now we will access this database from locally installed MySQl workbench. By using the public IP we can easily connect the google cloud database to our local machine workbench. Here we can see the updated ppmtdb on google cloud.



Name ↑	Collation	Character set	Type
information_schema	utf8_general_ci	utf8	System
mysql	utf8_general_ci	utf8	System
performance_schema	utf8_general_ci	utf8	System
ppmt	utf8_general_ci	utf8	User
sys	utf8_general_ci	utf8	System

Appendix I - List of Figures

References

[1]. “What is containerization?”, by IBM Cloud Kubernetes Service (https://www.ibm.com/cloud/learn/containerization)

[2]. “Docker overview”, by Docker official documents page (https://docs.docker.com/get-started/overview/)

[3]. “What Is Container Orchestration?”, by Isaac Eldridge on Jul. 17th, 2018 (https://blog.newrelic.com/engineering/container-orchestration- explained/)