**Docker and Kubernetes**

In this tutorial, we will learn how to create, run the docker image of application and MySQL. And after that we will see how to run the docker images of application and MySQL in Kubernetes.

# Build and run docker images of MySQL and microservices independently

## To work with MySQL docker image

* To download the MySQL image from docker hub repository

docker pull mysql:8.0.22

* One time activity to run the MySQL with the following details:
  + root\_password=bjjd,
  + sql\_user=project,
  + sql\_password=project,
  + sql\_database=project\_db

docker run --detach --env MYSQL\_ROOT\_PASSWORD=bjjd --env MYSQL\_USER=project --env MYSQL\_PASSWORD=project --env MYSQL\_DATABASE=project\_db --name mysql --publish 3306:3306 mysql:8.0.22

* Every time to run MySQL

docker start mysql

* Download MySQL Workbench IDE for mysql8.0.22

https://dev.mysql.com/downloads/workbench/

## To work with microservice application: project-mgmt-service

* To build the project after download the MySQL image from docker hub repository

mvn clean install -DskipTestCase=true

* To build the docker image

docker image build -t project-mgmt-service:0.0.1-RELEASE .

* [Optional if requires to push the image in Docker hub then tag and push the docker image into docker hub repository
  + To tag the docker image

docker tag project-mgmt-service:0.0.1-RELEASE rajivbansal2981/project-mgmt-service:0.0.1-RELEASE

* + To login into the docker hub and push the docker image

docker login docker.io

docker push rajivbansal2981/project-mgmt-service:0.0.1-RELEASE

* To run the docker image of project-mgmt-service where RDS\_HOSTNAME is mysql (name of the mysql docker container)

docker run -p 5379:5379 --link=mysql --env RDS\_HOSTNAME=mysql project-mgmt-service:0.0.1-RELEASE

# Use docker-compose to run docker images of MySQL and microservices in one go

Using docker-compose, we can run both docker container using one simple command as it would be difficult for every time to remember the following two long commands for running mysql docker and application docker.

* To run the mysql docker:

docker run --detach --env MYSQL\_ROOT\_PASSWORD=bjjd --env MYSQL\_USER=project --env MYSQL\_PASSWORD=project --env MYSQL\_DATABASE=project\_db --name mysql --publish 3306:3306 mysql:8.0.22

* To the application

docker run -p 5379:5379 --link=mysql --env RDS\_HOSTNAME=mysql project-mgmt-service:0.0.1-RELEASE

## Docker-compose installation

Docker-compose installation is required to use docker-compose tool so that it can run both docker container using one simple command

1. Download the docker-compose tool
2. Run this command to download the current stable release of Docker Compose:

sudo curl -L "https://github.com/docker/compose/releases/download/1.27.4/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

1. Apply executable permissions to the binary:

sudo chmod +x /usr/local/bin/docker-compose

1. If the command docker-compose fails after installation, check your path. You can also create a symbolic link to /usr/bin or any other directory in your path.

sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose

1. Test the installation.

$ docker-compose –version

## Create the docker-compose.yaml file

**Perquisite**: Docker image of project-mgmt-service:0.0.1-RELEASE and microservice should be build. Refer Section 1: Build and run docker images of MySQL and microservices independently

Now, create docker-compose.yaml file in the project-mgmt-service project with the following contents:



## Run the docker-compose.yaml file

Following are the steps to run the docker-compose.yaml file.

* Go to the project-mgmt-service location.

cd /home/rajiv/git/BJJD-Kubernetes/project-mgmt-service

1. Execute the docker-compose command. It will start the mysql and then run the application and after that we can use the application

rajiv@rajiv-VirtualBox:~/git/BJJD-Kubernetes/project-mgmt-service$ docker-compose up

1. To stop and remove all the docker resources related to the application.

NOTE: If you want your instance to be initialized, you have to start from scratch. It is quite easy to do with docker compose when using a named volume like in your case. Warning: this will permanently delete the contents in your db\_data volume, wiping out any previous database you had there. Create a backup first if you need to keep the contents.

docker-compose down -v

# Run docker images (microservice and mysql) in the Kubernetes Environment.

To run the docker images in the Kubernetes, need to setup the GCloud Kubernetes or minikube a local Kubernetes.

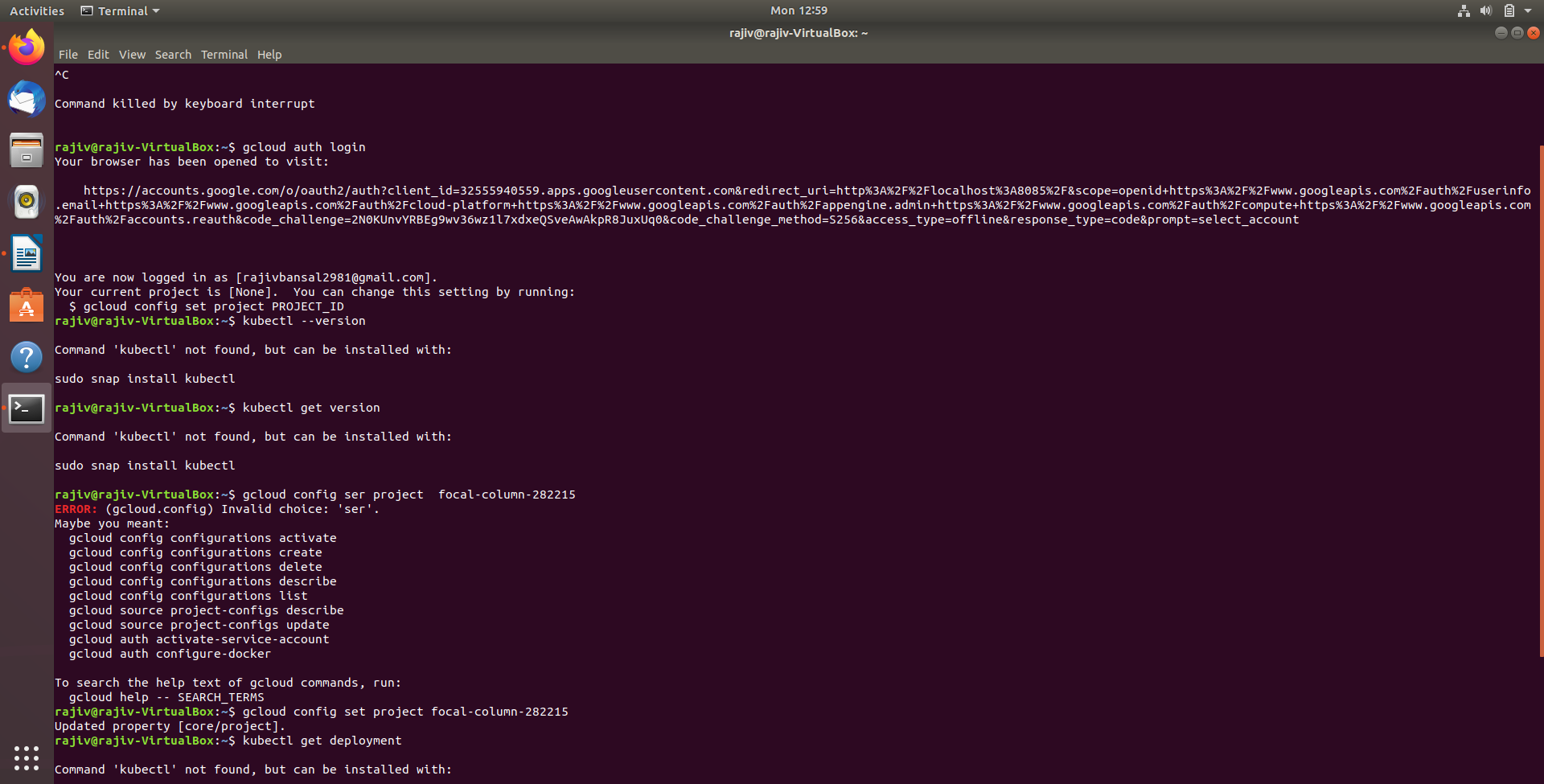
## Setup the environment to connect GCloud Kubernetes

* Download and install the Google Cloud SDK by running the following command in your shell or Terminal:

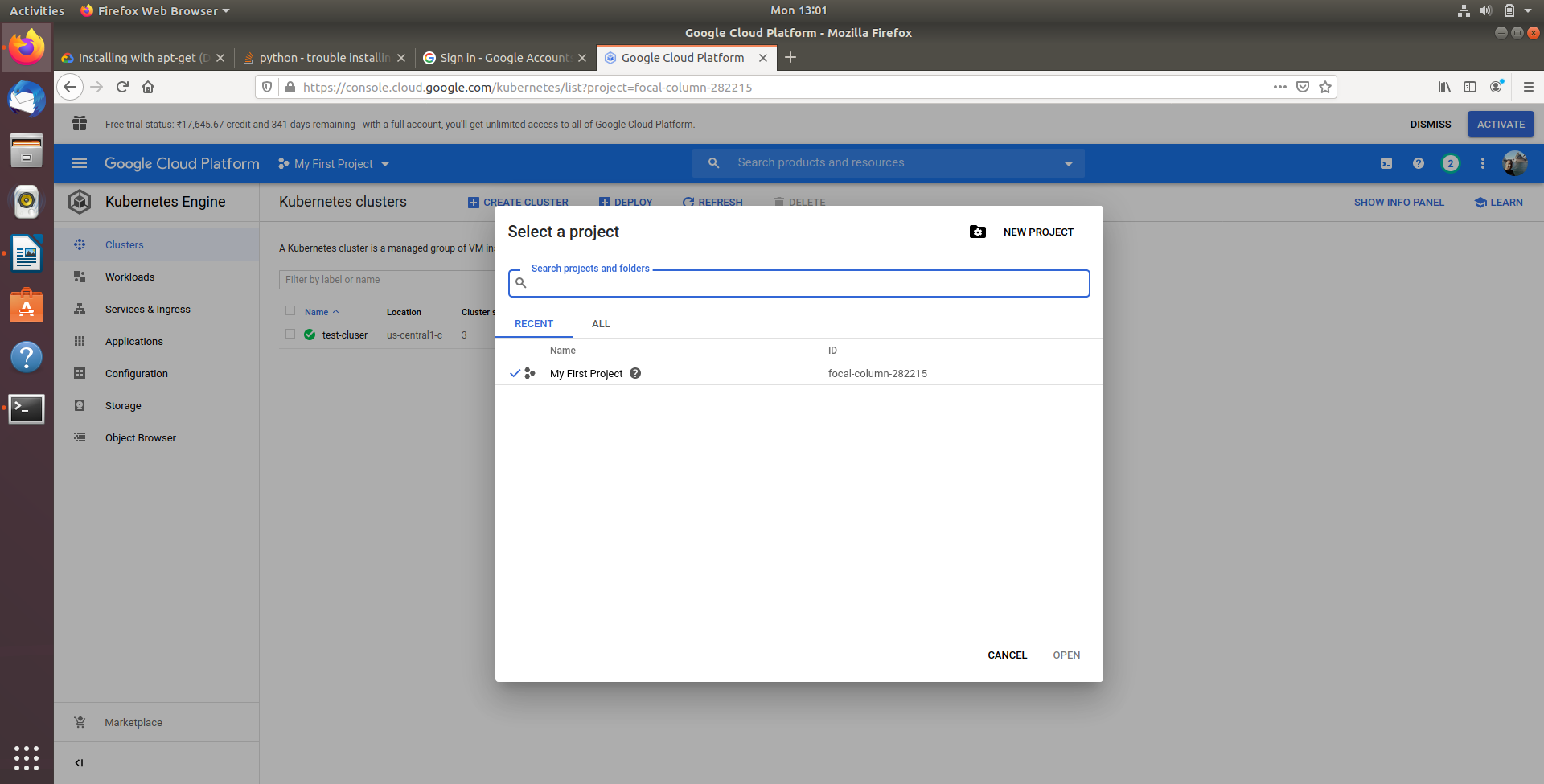
curl <https://dl.google.com/dl/cloudsdk/release/install_google_cloud_sdk.bash> | bash

* Restart your shell or Terminal.
* Authenticate to Google Cloud Platform by running the following command:

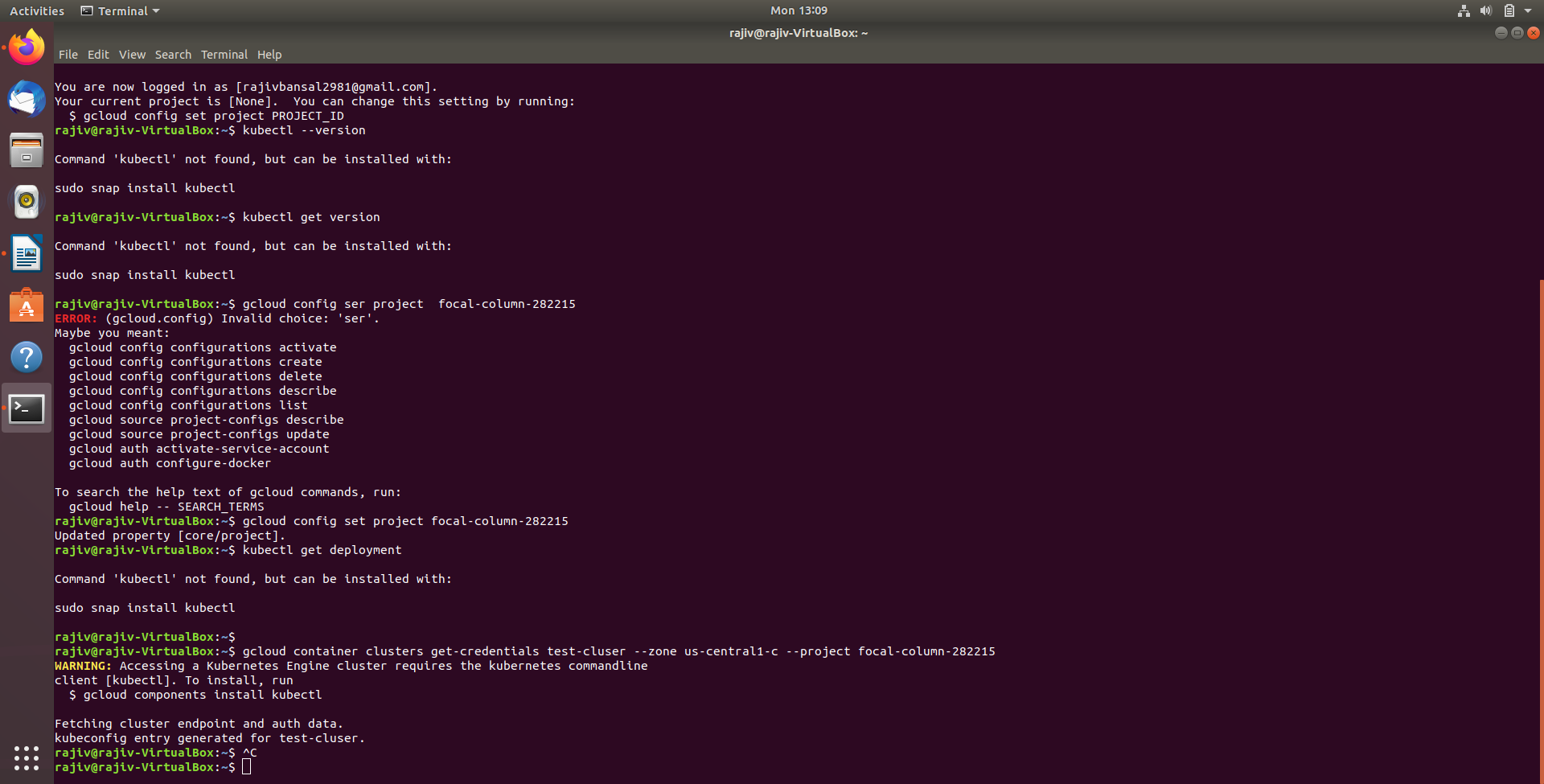
gcloud auth login

* It will move to browser where we will provide the Gloud credentials and we will be logged in Gloud.
* Now we need to provide the gcloud command along with project-id (can be checked into Google Console.)

gcloud config set project focal-column-282215

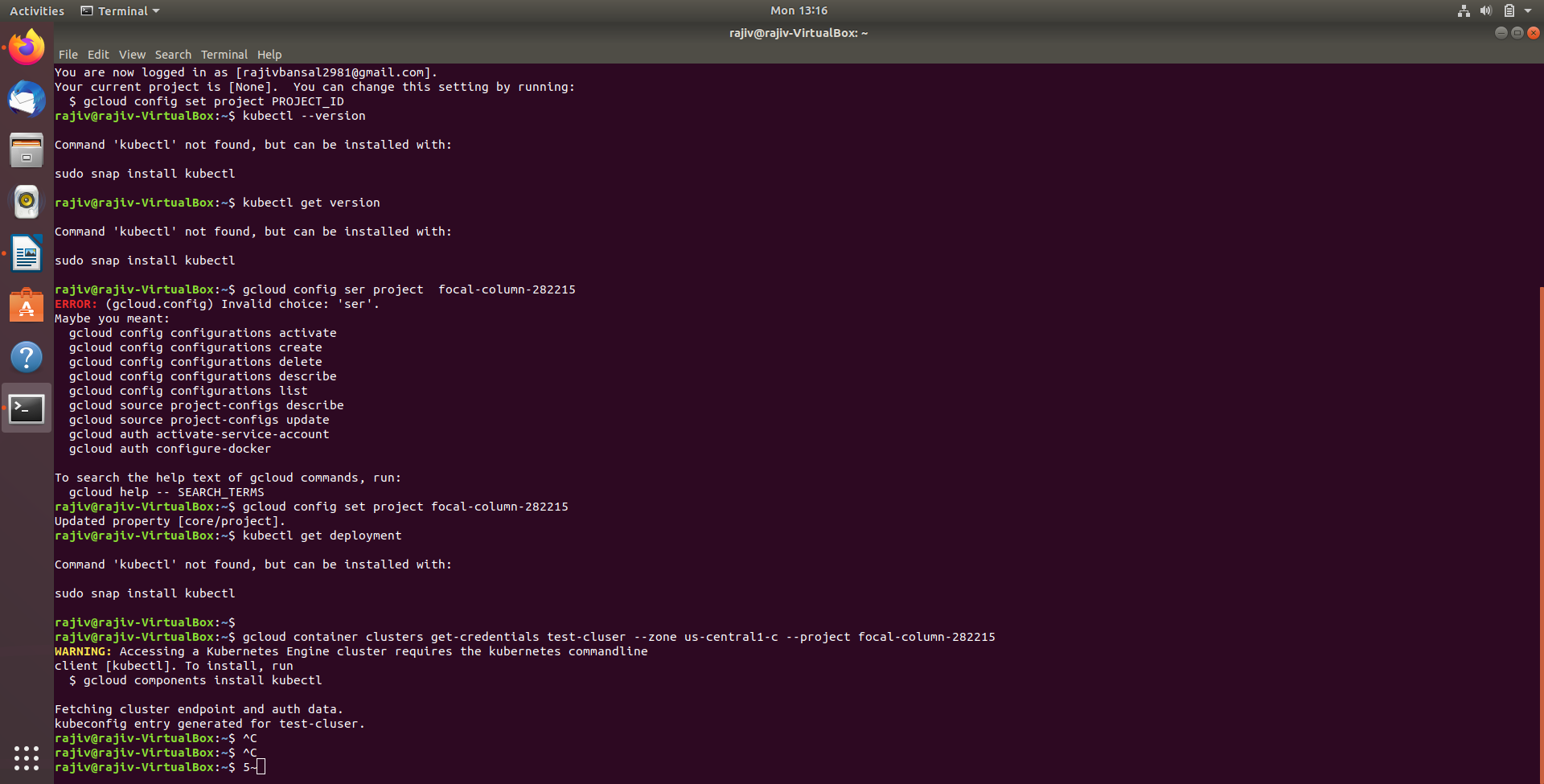


**Output:**

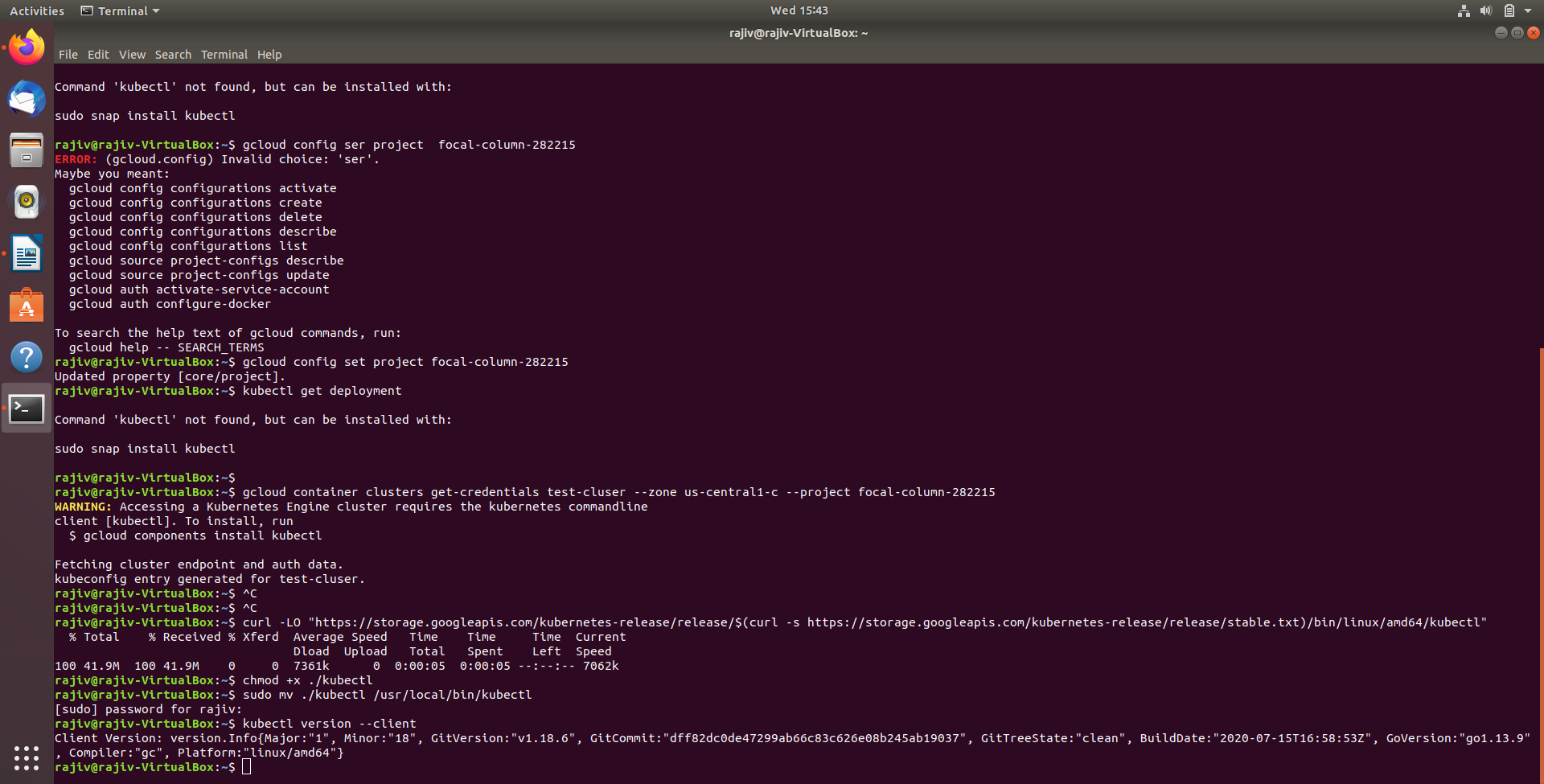
****

* Try to connect Kubernetes engine from Gcloud Interface using the following command it shows the error: “Accessing a Kubernetes Engine cluster requires the Kubernetes command line”

gcloud container clusters get-credentials test-cluser --zone us-central1-c --project focal-column-282215

**Output:**

* Download and install the Kubectl with latest release binary with curl on Linux using the following command:

curl -LO "https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl"

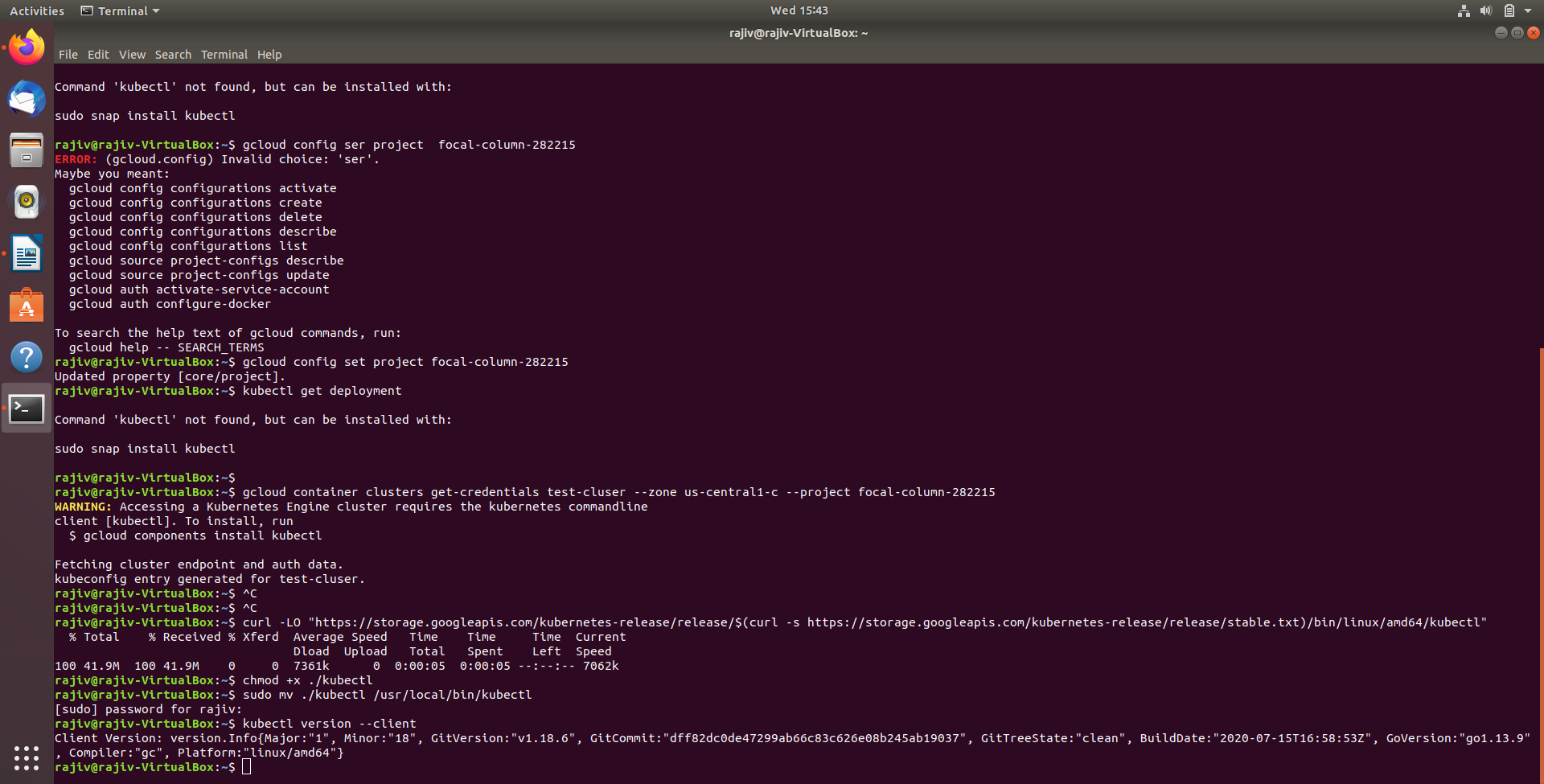
**Output:**

* Make the kubectl binary executable.

chmod +x ./kubectl

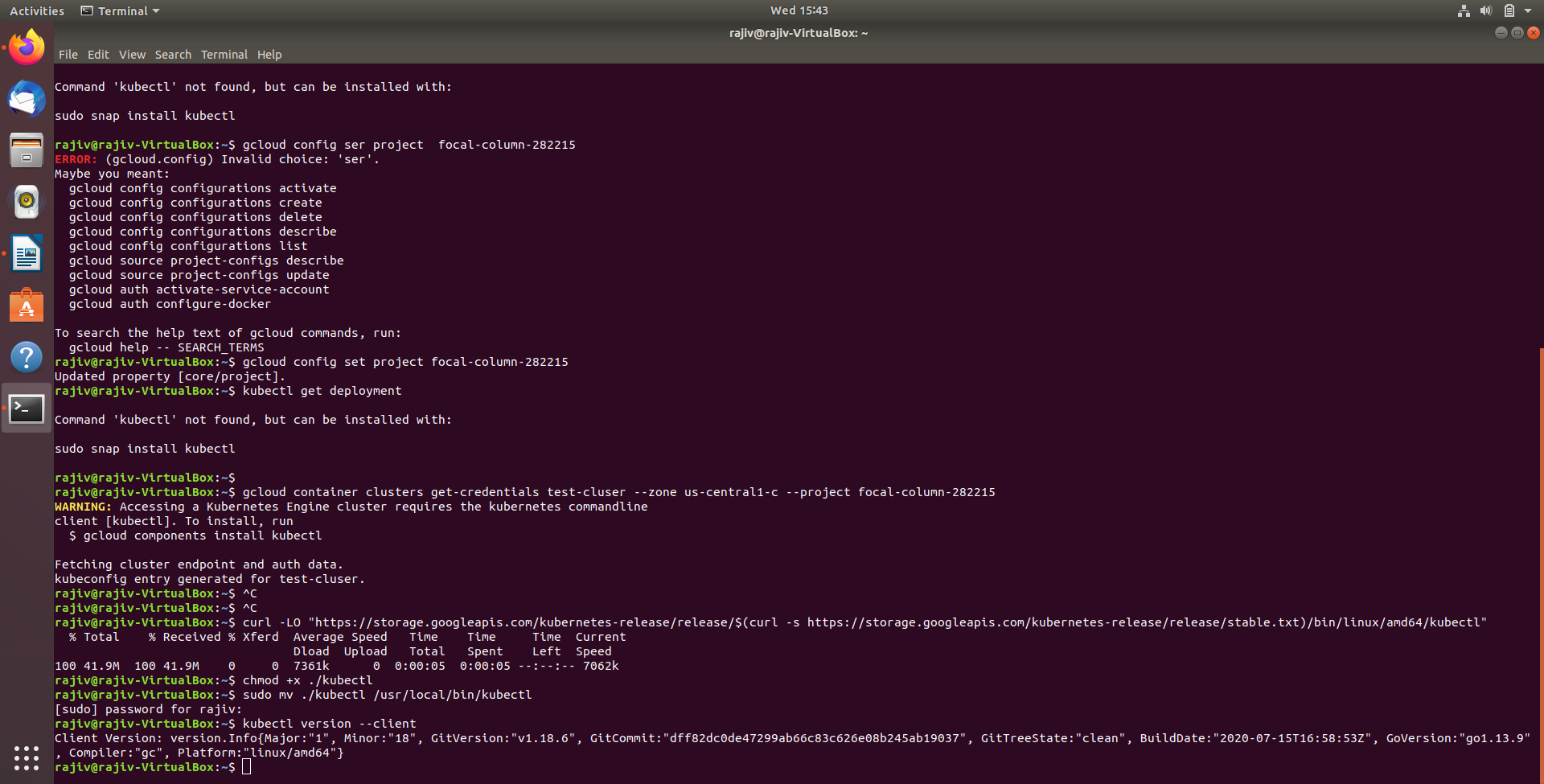
* Move the binary in to your PATH.

sudo mv ./kubectl /usr/local/bin/kubectl



* Test to ensure the version you installed is up-to-date:

kubectl version --client



## Setup the minikube, a local Kubernetes

minikube is local Kubernetes, focusing on making it easy to learn and develop for Kubernetes.

All you need is Docker (or similarly compatible) container or a Virtual Machine environment, and Kubernetes is a single command away: minikube start

* Download the minikube

curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

* Install the minikube

sudo install minikube-linux-amd64 /usr/local/bin/minikube

* Run the following command to install kubeadm and kubectl
  + sudo apt-get update && sudo apt-get install -y apt-transport-https curl
  + curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
  + cat <<EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list

deb https://apt.kubernetes.io/ kubernetes-xenial main

EOF

* + sudo apt-get update
  + sudo apt-get install -y kubelet kubeadm kubectl
  + sudo apt-mark hold kubelet kubeadm kubectl
* To start the minikube

minikube start –extra-config=kubeadm.ignore-preflight-errors=NumCPU –force –cpus 1

* To stop the minikube

minikube stop

## Is it feasible to deploy the database on Kubernetes?

In today’s world, there are more and more companies working on containerized technologies. Before doing a deep dive, let's review our options for running databases.

### 3.3.1. Fully Managed Databases

Fully managed databases are those that don’t have to provision or manage the database - this management can be done by cloud providers like AWS Google, Azure, or Digital Cloud. Managed databases include Amazon Web Services, Aurora DynamoDB, or Google Spanner and SQL. These databases are used because of a low-ops choice, cloud providers handle many of the maintenance tasks, such as backup, scaling patches, etc. You’ll just have to create a database to build the app, and let cloud providers handle the rest for you.

### 3.3.2. Deploying by Yourself On VM, Or On-premises Machines

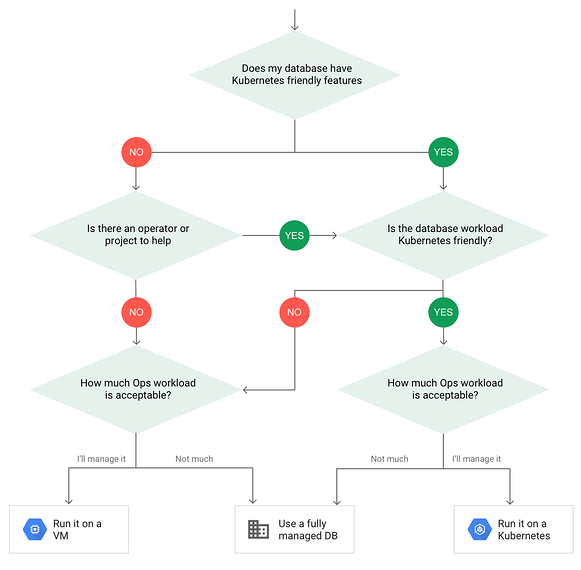
With this option you can deploy the database to any virtual machine (EC2 or Compute Engine), and you’ll have full control. You’ll be able to deploy any version of the database, and you can set your own security and backup plans. On the other hand, this means that you'll manage, patch, scale, or provision the database on your own. You’ll also have to have an administrator in place, who will manage and administer your database. This will add cost to your infrastructure, but has the advantage of flexibility.

### 3.3.3. Run It on Kubernetes

Here’s the main point, deploying the database in Kubernetes is closer to the full-ops option, but you’ll get some benefits in terms of the automation that Kubernetes provides to keep the database application up and running. It’s important to remember that pods are ephemeral, so the possibility that the database application restarts or fails is greater. Also, you’ll be responsible for the more specific database administrative tasks such as backup, scaling, etc.

Some important points to consider when choosing to deploy the database on Kubernetes are:

* There are some custom resources and operators available to manage the database on Kubernetes.
* Databases that have caching layers and more transient storage are better fits for Kubernetes.
* You have to understand the replication mode available in the database. Asynchronous modes of replication leave room for data loss, because the transactions might be committed to the primary database, but not to the secondary databases.



Above, we have a simple chart to show what the decision tree looks like when deploying databases on Kubernetes. First, we try to understand if the database has Kubernetes-friendly features, such as MySQL or PostgreSQL, then we’ll have to find/plan for kubernetes operators to package the database with additional features. The second question is - how much workload is acceptable given what we’ve seen is needed to deploy a database in Kubernetes? Do we have a team of operation site engineers, or would we find it feasible to deploy the database on a Managed DB?

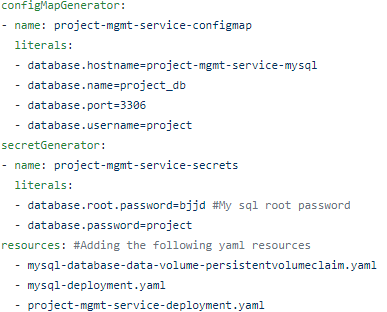
## Setup the configuration files to pull the docker image of mysql and microservices from docker hub into the Kubernetes

### 3.4.1. To create a kustomization.yaml

Kubectl supports the management of Kubernetes objects using a kustomization file. We can add the following objects in this file:

* we can create a Secret by generators
* We can generate the configmaps.
* We can add the resources files. For example, deployment and service yaml files.

[BJJD-Kubernetes](https://github.com/rajivbansal2981/BJJD-Kubernetes/tree/master)/[project-mgmt-service](https://github.com/rajivbansal2981/BJJD-Kubernetes/tree/master/project-mgmt-service)/[deployment](https://github.com/rajivbansal2981/BJJD-Kubernetes/tree/master/project-mgmt-service/deployment)/kustomization.yaml



### 3.4.2. To create a Persistent Volume and Persistent Volume Claim

We can deploy a database to Kubernetes as a stateful application. Usually, when we deploy pods, they have their own storage, but that storage is ephemeral - if the container kills its storage, it’s gone with it.

So, we’ll have a Kubernetes object to tackle that scenario: when we want our data to persist, we attach a pod with a respective persistent volume claim. By doing it this way, if our container kills our data, it will be in the cluster, and the new pod will access the data accordingly.

Pod -> PVC -> PV

* PV = Persistent Volume
* PVC = Persistent Volume Claim

A [PersistentVolume](https://kubernetes.io/docs/concepts/storage/persistent-volumes/) (PV) is a piece of storage in the cluster that has been manually provisioned by an administrator, or dynamically provisioned by Kubernetes using a [StorageClass](https://kubernetes.io/docs/concepts/storage/storage-classes).

Persistent volume will be attached with the pod, to ensure data safety on restart. Persistent volume claims 20GB from storage with ReadWriteOne access mode. Host path is /mnt/data where all our data will reside.

A [PersistentVolumeClaim](https://kubernetes.io/docs/concepts/storage/persistent-volumes/#persistentvolumeclaims) (PVC) is a request for storage by a user that can be fulfilled by a PV. PersistentVolumes and PersistentVolumeClaims are independent from Pod lifecycles and preserve data through restarting, rescheduling, and even deleting Pods.

When a PersistentVolumeClaim is created, a PersistentVolume is dynamically provisioned based on the StorageClass configuration.

[BJJD-Kubernetes](https://github.com/rajivbansal2981/BJJD-Kubernetes/tree/master)/[project-mgmt-service](https://github.com/rajivbansal2981/BJJD-Kubernetes/tree/master/project-mgmt-service)/[deployment](https://github.com/rajivbansal2981/BJJD-Kubernetes/tree/master/project-mgmt-service/deployment)/mysql-database-data-volume-persistentvolumeclaim.yaml

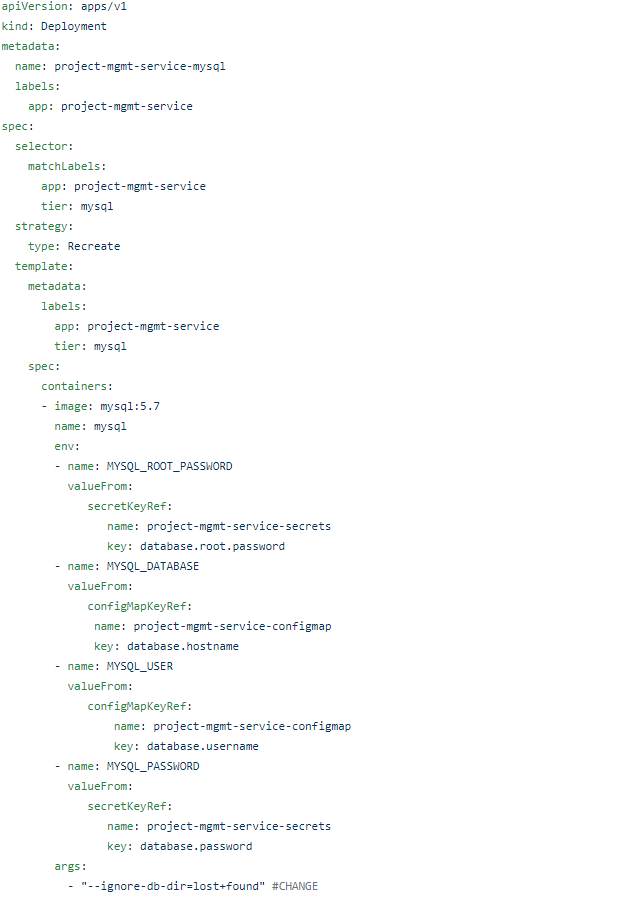


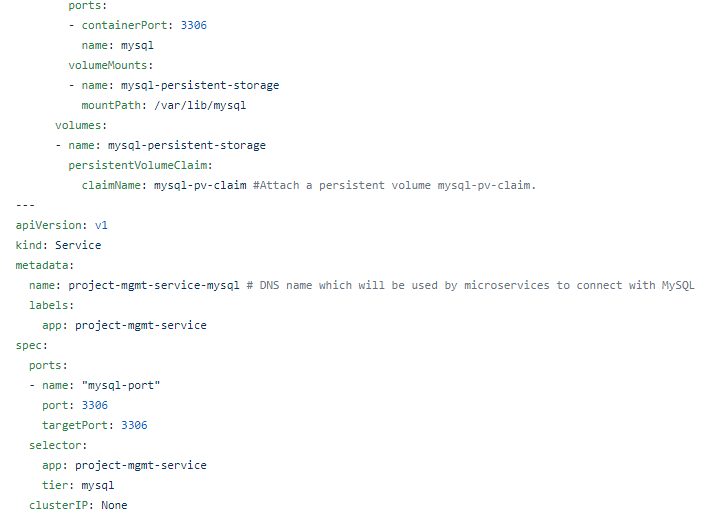
### 3.4.3. To create a deployment and service yaml for MySQL

The deployment creates pods with image MySQL, with 5.7 tags pulled from docker hub, with an environment variable password on port 3306. We’ll also attach a persistent volume **mysql-pv-claim.**

After that we deploy the service for MySQL database on port 3306, with all pods having label key app and value MySQL.

[BJJD-Kubernetes](https://github.com/rajivbansal2981/BJJD-Kubernetes/tree/master)/[project-mgmt-service](https://github.com/rajivbansal2981/BJJD-Kubernetes/tree/master/project-mgmt-service)/[deployment](https://github.com/rajivbansal2981/BJJD-Kubernetes/tree/master/project-mgmt-service/deployment)/mysql-deployment.yaml



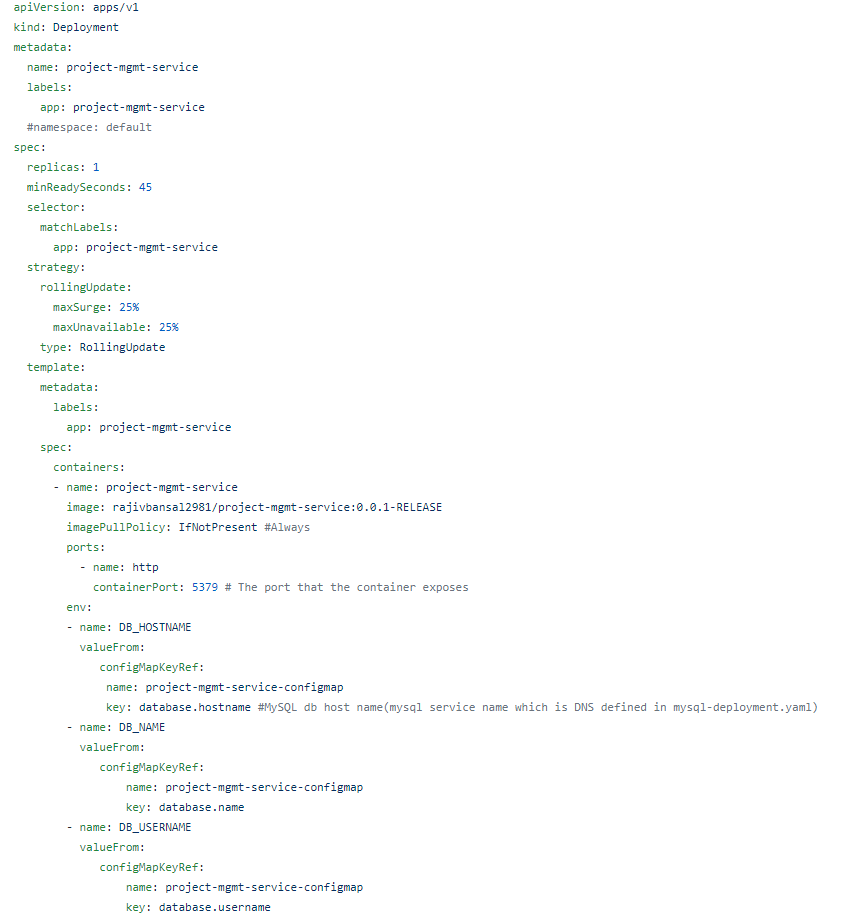


### 3.4.4. To create a deployment and service yaml for project-mgmt-service

Now we will create the deployment and service yaml for project-mgmt-service and will connect with MySQL docker image deployed in kubernetes.

The DB\_HOSTNAME environment variable sets the name of the MySQL Service defined above, and project-mgmt-service will access the database by Service.

[BJJD-Kubernetes](https://github.com/rajivbansal2981/BJJD-Kubernetes/tree/master)/[project-mgmt-service](https://github.com/rajivbansal2981/BJJD-Kubernetes/tree/master/project-mgmt-service)/[deployment](https://github.com/rajivbansal2981/BJJD-Kubernetes/tree/master/project-mgmt-service/deployment)/**project-mgmt-service-deployment.yaml**





### 3.4.5. To build and run the docker image in kubernetes

Following are the commands to build and run the docker images in the kubernetes:

1. Build the project

mvn clean install -DskipTestCase=true

1. To list out all the docker images

docker images

1. To remove the docker image

docker image rm rajivbansal2981/project-mgmt-service:0.0.1-RELEASE -f

1. To remove all the non-tagged images

docker rmi $(docker images --filter "dangling=true" -q --no-trunc) -f

1. To build the docker image

docker image build -t project-mgmt-service:0.0.1-RELEASE .

1. To tag and push the docker image into docker hub repository[Optional as it is required when tag and push the image in docker repository so that anybody can pull and use this docker image in kubernetes]

* To tag the docker image

docker tag project-mgmt-service:0.0.1-RELEASE rajivbansal2981/project-mgmt-service:0.0.1-RELEASE

* To login into the docker hub and push the docker image

docker login docker.io

docker push rajivbansal2981/project-mgmt-service:0.0.1-RELEASE

1. Start the minikube

minikube start --extra-config=kubeadm.ignore-preflight-errors=NumCPU --force --cpus 1

1. Delete the pods and deployment, service etc by label name of the application

kubectl delete all -l app=project-mgmt-service

* + Delete the pvc claim for mysql

kubectl delete pvc mysql-pv-claim

* + Delete the secrets

kubectl get secrets

kubectl get secrets <secret\_name>

kubectl delete secrets project-mgmt-service-secrets-bdtk6dh96g

* + Delete the configmaps

kubectl get configmaps

kubectl delete configmaps project-mgmt-service-configmap-8g7f57f86d

1. Deploy and run the docker image in kubernetes by applying following commands in the order or follow the direct step 10 to apply the following commands in one go:

kubectl apply -f mysql-database-data-volume-persistentvolumeclaim.yaml

kubectl apply -f mysql-deployment.yaml

kubectl apply -f project-mgmt-service-deployment.yaml

1. Deploy and run the docker image in kubernetes in one go using kustomization.yaml file

kubectl apply -k ./

1. To stop the minikube

minikube stop

1. Logging into the MySQL pod to check the MySQL Database created

kubectl run -it --rm --image=mysql:5.7 --restart=Never mysql-client -- mysql -h project-mgmt-service-mysql -pbjjd

* Creating username and password in mysql

mysql> create user 'project'@'project-mgmt-service-mysql' identified by 'project';

* Creating database schema in mysql

mysql> create database project\_db;

* Grant user named ‘project’ on project\_db;

mysql> grant all on project\_db.\* to 'project'@'project-mgmt-service-mysql';

* Switch to the project user

mysql> mysql -u project -pproject

mysql> use project\_db;

Database changed

mysql>

**Other Important Kubernetes commands**

* kubectl get events --sort-by=.metadata.creationTimestamp
* kubectl logs <pods-id>
* kubectl delete all
* kubectl get deployment
* kubectl get services
* kubectl get pods
* kubectl apply -f mysql-database-data-volume-persistentvolumeclaim.yaml
* kubectl apply -f mysql-deployment.yaml
* kubectl apply -f project-mgmt-service-deployment.yaml
* kubectl describe pod project-mgmt-service-5c88bd4b87-cpgzc
* kubectl get secrets
* kubectl get secrets <secret\_name>
* kubectl delete secrets project-mgmt-service-secrets-bdtk6dh96g
* kubectl get configmaps
* kubectl delete configmaps project-mgmt-service-configmap-8g7f57f86d

