**PHASE 1: Run docker images of MySQL and microservices together 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-user,

sql\_password=pass379,

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 projectdownload 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]To 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

**PHASE 2: Using docker-compose to run docker images of MySQL and microservices in one go**

It would be diffucult for every time to remember these two long commands for running mysql docker and application docker.

1. To run the mysql docker:

docker run --detach --env MYSQL\_ROOT\_PASSWORD=bjjd --env MYSQL\_USER=project-user --env MYSQL\_PASSWORD=pass379 --env MYSQL\_DATABASE=project-db --name mysql --publish 3306:3306 mysql:8.0.22

2. To the application

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

Docker-compose installation is required to 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

Following are the steps to run both docker container using one simple command

1. Create docker-compose.yaml file in the project-mgmt-service project with the following contents:

**version:** '3.7'

**services:**

**project-mgmt-service:**

**image:** rajivbansal2981/project-mgmt-service:0.0.1-RELEASE

#build:

#context: .

#dockerfile: Dockerfile

**ports:**

- "5379:5379"

**restart:** always

**depends\_on:** # Start the depends\_on first

- mysql

**environment:**

**DB\_HOSTNAME:** mysql

**DB\_PORT:** 3306

**DB\_NAME:** project\_db

**DB\_USERNAME:** project

**DB\_PASSWORD:** project

**mysql:**

**image:** mysql:5.7

**ports:**

- "3306:3306"

**restart:** always

**environment:**

**MYSQL\_ROOT\_PASSWORD:** bjjd

**MYSQL\_DATABASE:** project\_db

**MYSQL\_USER:** project

**MYSQL\_PASSWORD:** project

**volumes:**

- mysql-database-data-volume:/var/lib/mysql

# Volumes

**volumes:**

**mysql-database-data-volume:**

1. Run the following command from project-mgmt-service location.

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

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

1. It will start the mysql and then run the application and after that we can use the application
2. 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

Important Utility Command:

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

**PHASE 3: Using Gcloud kubernetes or minikube, a local kubernetes focusing on making it easy to** **push and run docker images of MySQL and microservices in Kubernetes cluster**

**Step 1.1 : Setup the environment to connect with Gcloud kubernetes**

Google Cloud SDK Installation

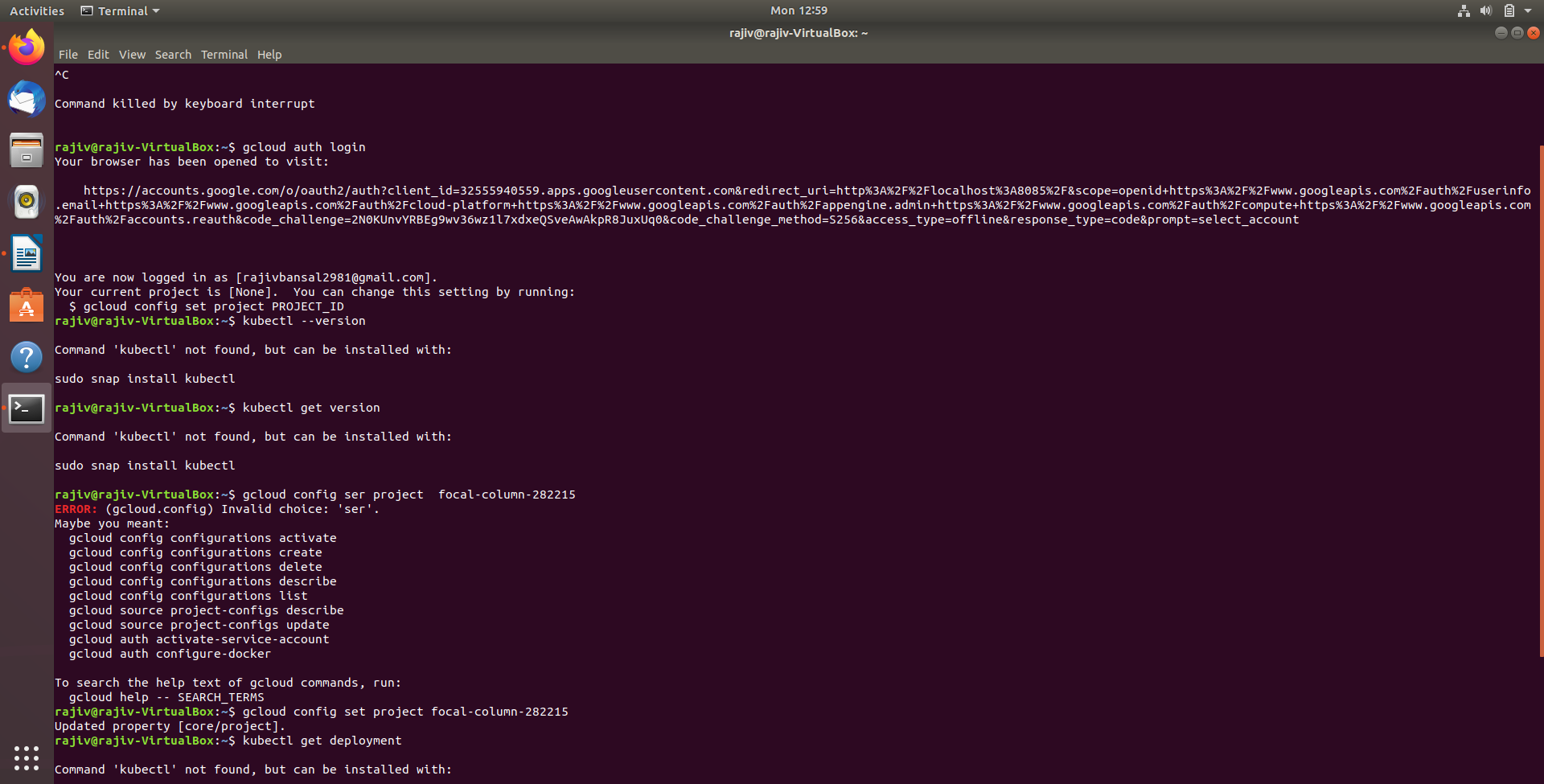
1. Download and install Google Cloud SDK by running the following command in your shell or Terminal:

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

2. Restart your shell or Terminal.

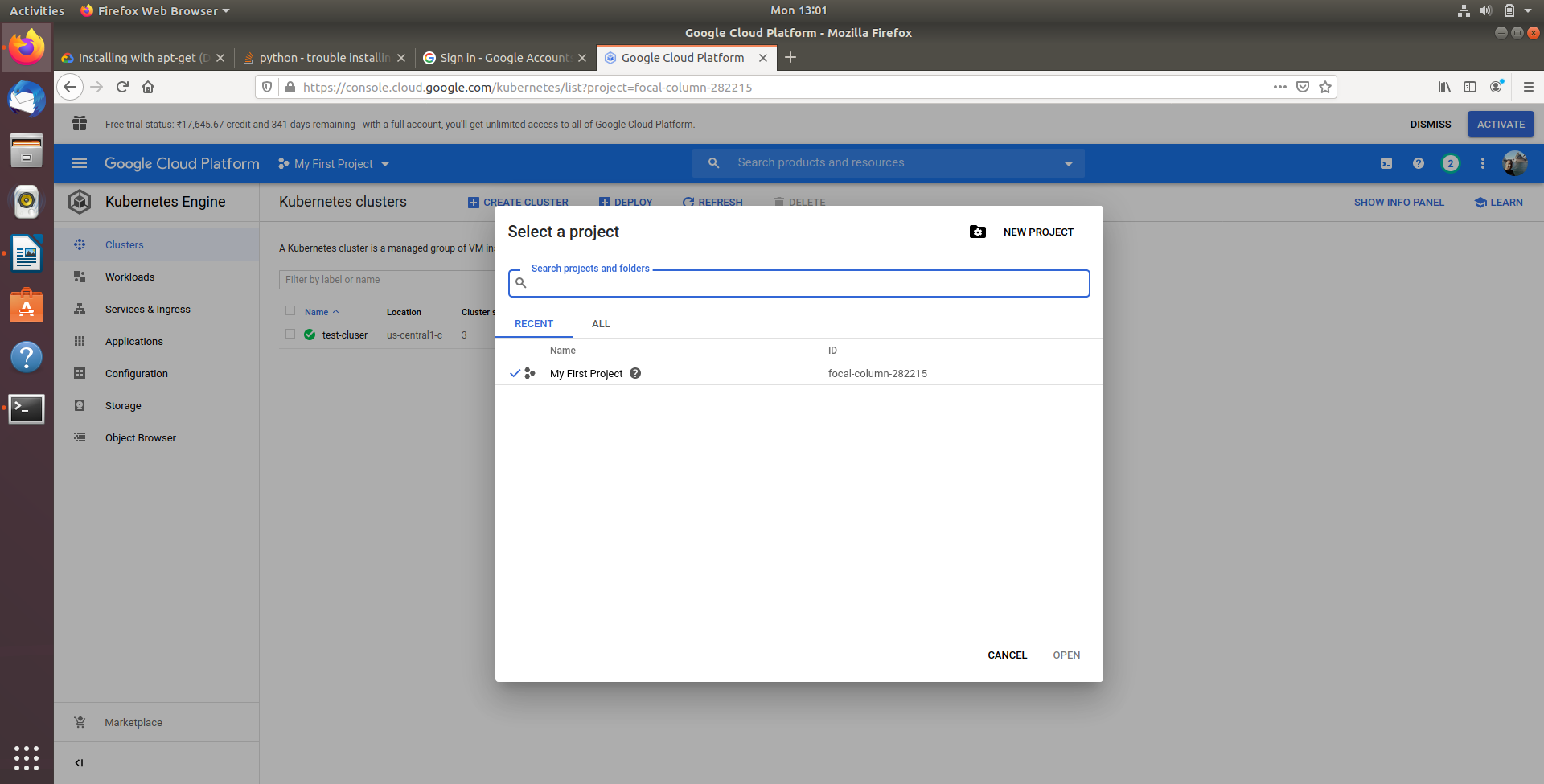
3. Authenticate to Google Cloud Platform by running gcloud auth login

4. It will move to brower where we will provide the Gloud credentials and we will be logged in Gloud.

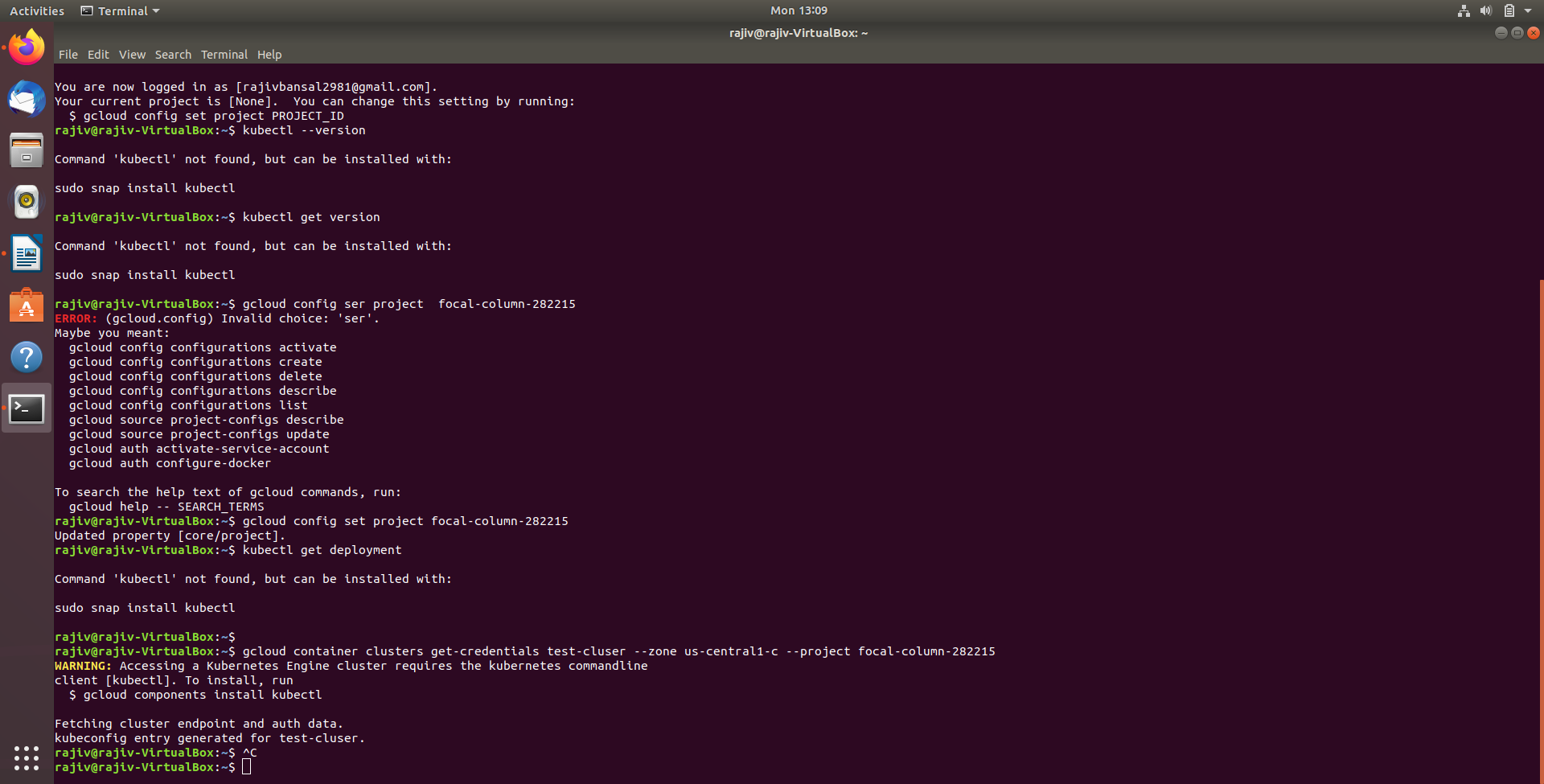


5. Now we need to provide the project-id ( can check into Google Console.

Command: gcloud config set project focal-column-282215



**Output:**

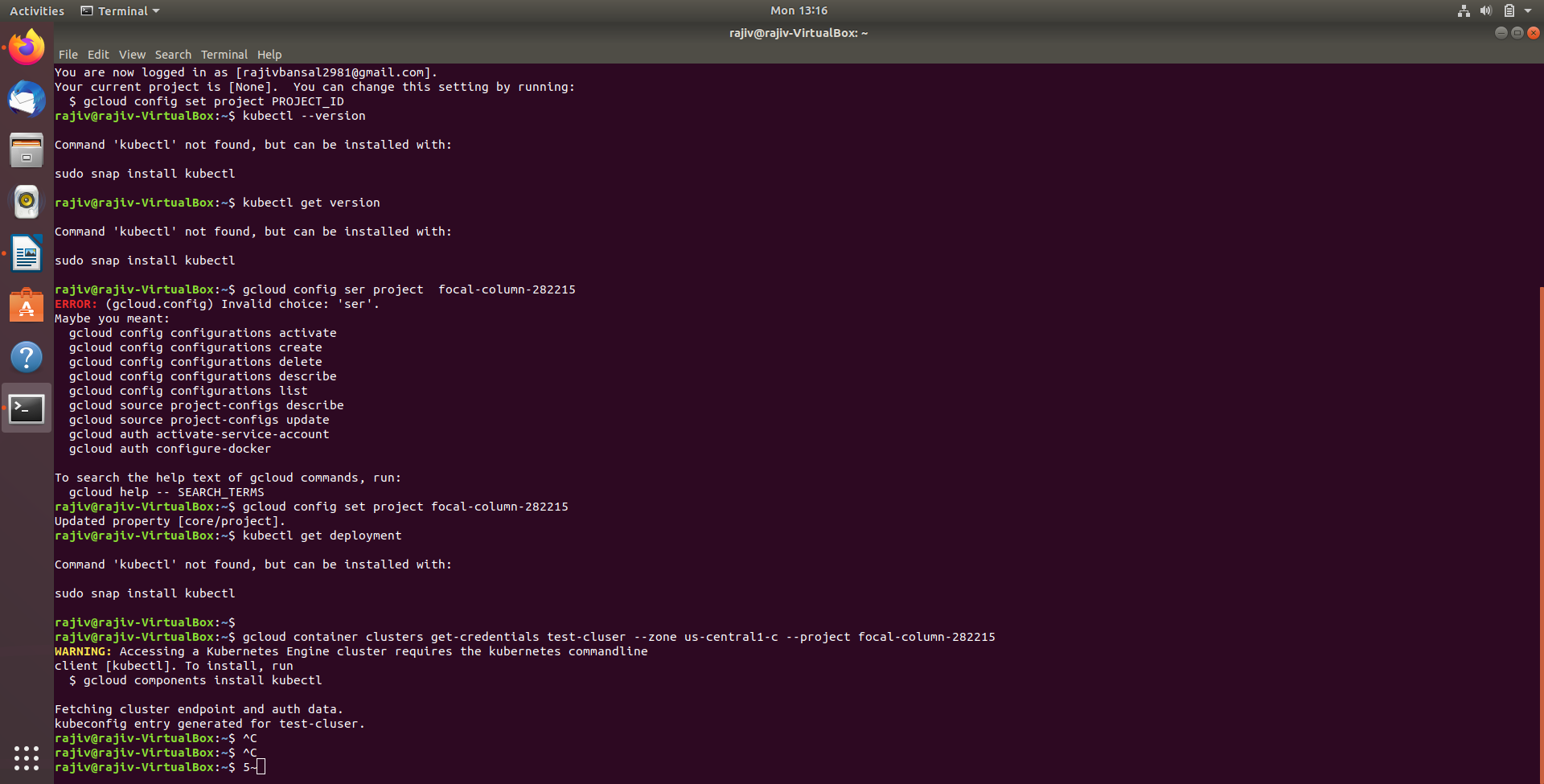
****

**6. Try to connect Kubernetes engine from Gcloud Interface but getting error says**

**“Accessing a Kubernetes Engine cluster requires the kubernetes commandline”**

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

**Output:**



**Kubectl installation**

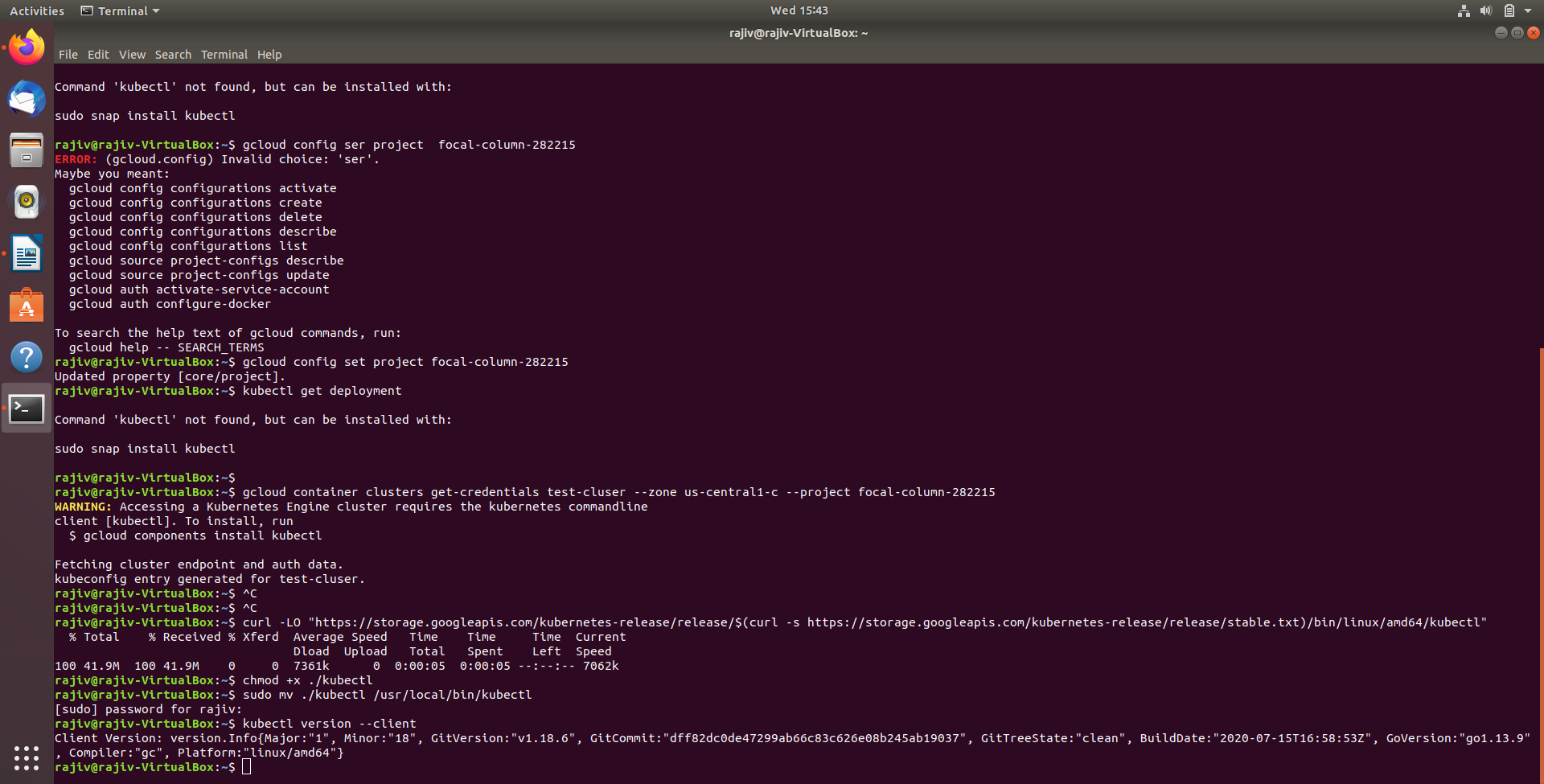
**7. Next step is to install Kubectl**

### **Install kubectl binary with curl on Linux**

* Download the latest release with the 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:

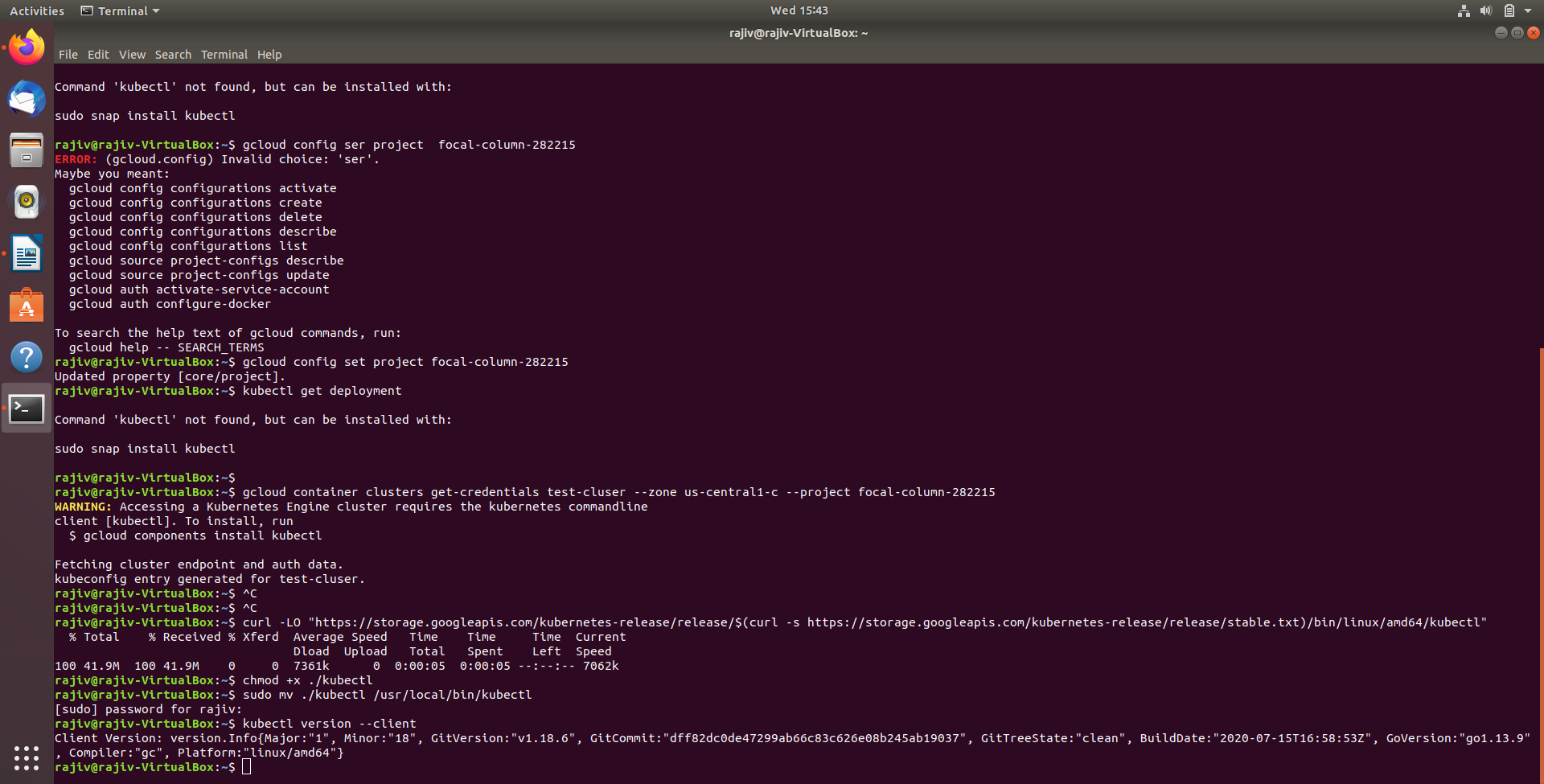


* Make the kubectl binary executable.

chmod +x ./kubectl

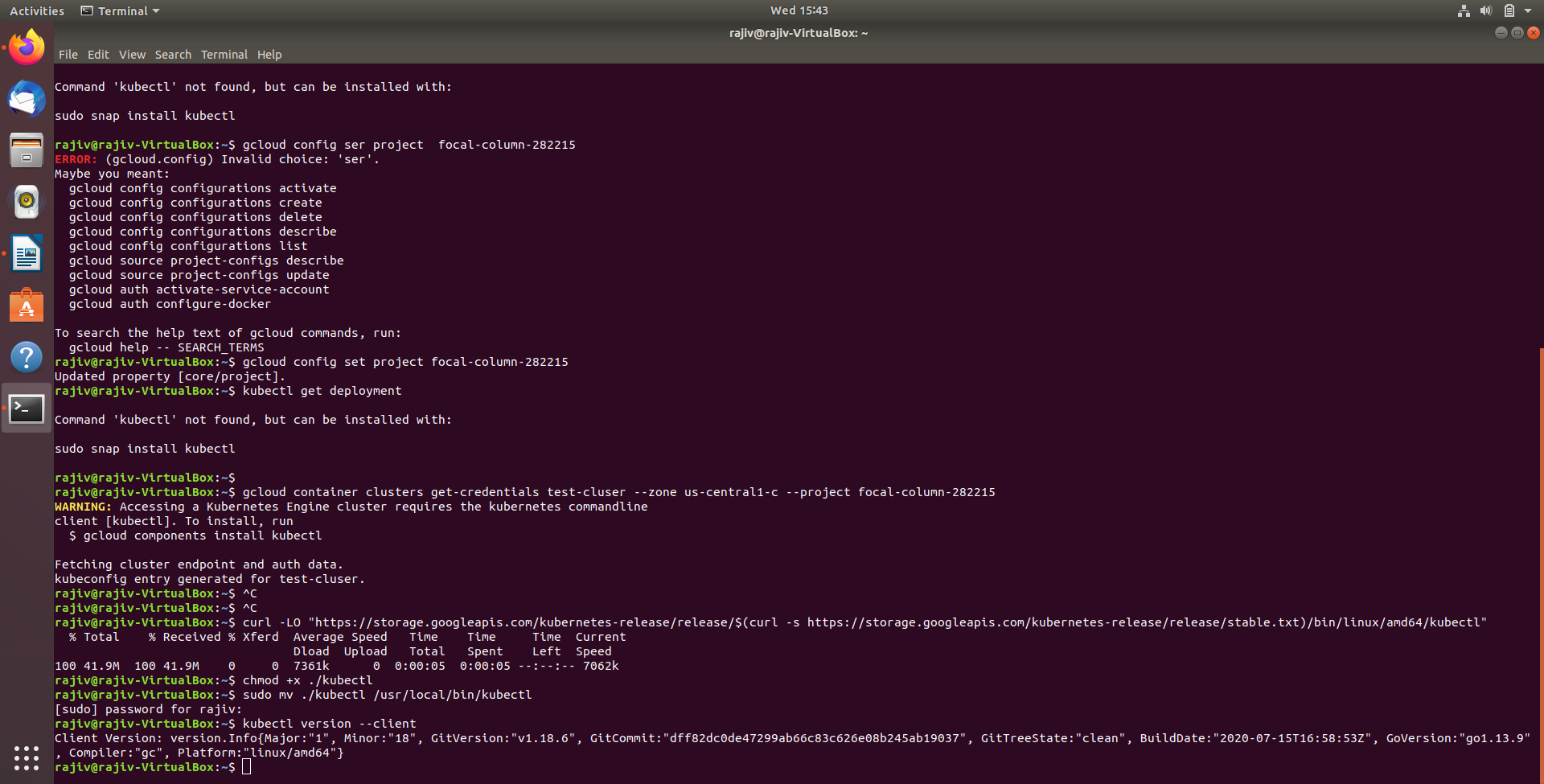
* Move the binary in to your PATH.

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



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

kubectl version --client



**Step 1.2: Setup the minikube environment**

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

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

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

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

Following are the steps to pull and deploy the images of mysql and aplication docker into kubernetes.

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

**Step 3: Setup the configuration files to pull the docker image of mysql and microservices from docker hub into the kubernetes**

## 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.

### 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.

### 2. Deploying By Yourself On VM, Or On-premises Machines

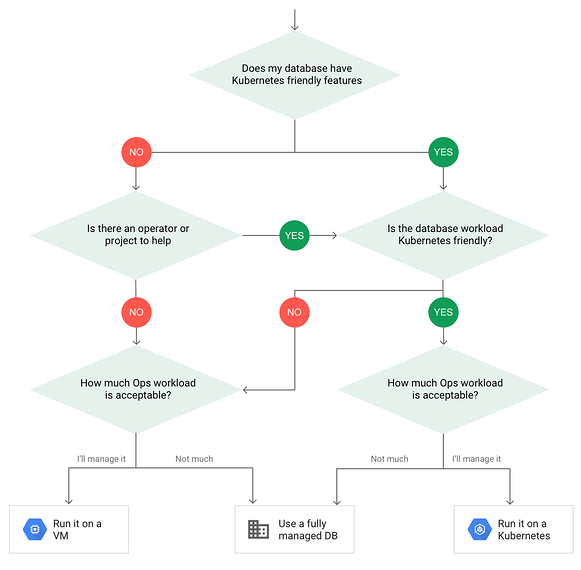
With this option you can deploy the database to any virtual machine (EC2 or Compute Engine), and you’ll have full control. You’ll be able to deploy any version of the database, and you can set your own security and backup plans. On the other hand, this means that you'll manage, patch, scale, or provision 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. 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?

Step 4: To create a Persistent Volume

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

mysql-database-data-volume-persistentvolumeclaim.yaml

**apiVersion:** v1

**kind:** PersistentVolume

**metadata:**

**name:** mysql-pv-volume

**labels:**

**type:** local

**spec:**

**storageClassName:** manual

**capacity:**

**storage:** 1000Mi

**accessModes:**

- ReadWriteOnce

**hostPath:**

**path:** "/mnt/data"

**---**

**apiVersion:** v1

**kind:** PersistentVolumeClaim

**metadata:**

**name:** mysql-pv-claim

**labels:**

**app:** project-mgmt-service

**spec:**

**accessModes:**

- ReadWriteOnce

**resources:**

**requests:**

**storage:** 100Mi

Step 5: To create the yaml file for the deployment and service of mysql

**apiVersion:** apps/v1

**kind:** Deployment

**metadata:**

**name:** project-mgmt-service-mysql

**labels:**

**app:** project-mgmt-service

**spec:**

**selector:**

**matchLabels:**

**app:** project-mgmt-service

**tier:** mysql

**strategy:**

**type:** Recreate

**template:**

**metadata:**

**labels:**

**app:** project-mgmt-service

**tier:** mysql

**spec:**

**containers:**

- **image:** mysql:5.7

**name:** mysql

**env:**

- **name:** MYSQL\_ROOT\_PASSWORD

**valueFrom:**

**secretKeyRef:**

**name:** project-mgmt-service-secrets

**key:** database.root.password

- **name:** MYSQL\_DATABASE

**valueFrom:**

**configMapKeyRef:**

**name:** project-mgmt-service-configmap

**key:** database.hostname

- **name:** MYSQL\_USER

**valueFrom:**

**configMapKeyRef:**

**name:** project-mgmt-service-configmap

**key:** database.username

- **name:** MYSQL\_PASSWORD

**valueFrom:**

**secretKeyRef:**

**name:** project-mgmt-service-secrets

**key:** database.password

**args:**

- "--ignore-db-dir=lost+found" #CHANGE

**ports:**

- **containerPort:** 3306

**name:** mysql

**volumeMounts:**

- **name:** mysql-persistent-storage

**mountPath:** /var/lib/mysql

**volumes:**

- **name:** mysql-persistent-storage

**persistentVolumeClaim:**

**claimName:** mysql-pv-claim

**---**

**apiVersion:** v1

**kind:** Service

**metadata:**

**name:** project-mgmt-service-mysql # DNS name

**labels:**

**app:** project-mgmt-service

**spec:**

**ports:**

- **name:** "mysql-port"

**port:** 3306

**targetPort:** 3306

**selector:**

**app:** project-mgmt-service

**tier:** mysql

**clusterIP:** None

1. kubectl apply -f mysql-database-data-volume-persistentvolumeclaim.yaml
2. kubectl apply -f mysql-deployment.yaml
3. kubectl apply -f project-mgmt-service-deployment.yaml
4. Check the MySQL Database created

#### Logging into the MySQL pod

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';

mysql> mysql -u project -pproject

mysql> use project\_db;

Database changed

mysql>

You can get the MySQL pod and use kubectl exec command to login to the Pod.

$ kubectl get pods

NAME READY STATUS RESTARTS AGE

polling-app-mysql-6b94bc9d9f-td6l4 1/1 Running 0 4m23s

$ kubectl exec -it polling-app-mysql-6b94bc9d9f-td6l4 -- /bin/bash

root@polling-app-mysql-6b94bc9d9f-td6l4:/#

kubectl get pvc

kubectl delete pvc mysql-pv-claim

1. kubectl delete all -l app=project-mgmt-service
2. kubectl get events --sort-by=.metadata.creationTimestamp

--If the application is not deployed then we can check the issue

kubectl get events --sort-by=.metadata.creationTimestamp

--If you want to see the log of the application deployment failure, then look into the containerized application logs

kubectl logs <pods-id>

kubectl logs project-mgmt-service-866dbc5dc4-tj5tm

--To get the deployment and service yaml into the project-mgmt-service project

kubectl get deployment project-mgmt-service -o yaml > deployment.yaml

kubectl get service project-mgmt-service -o yaml > service.yaml