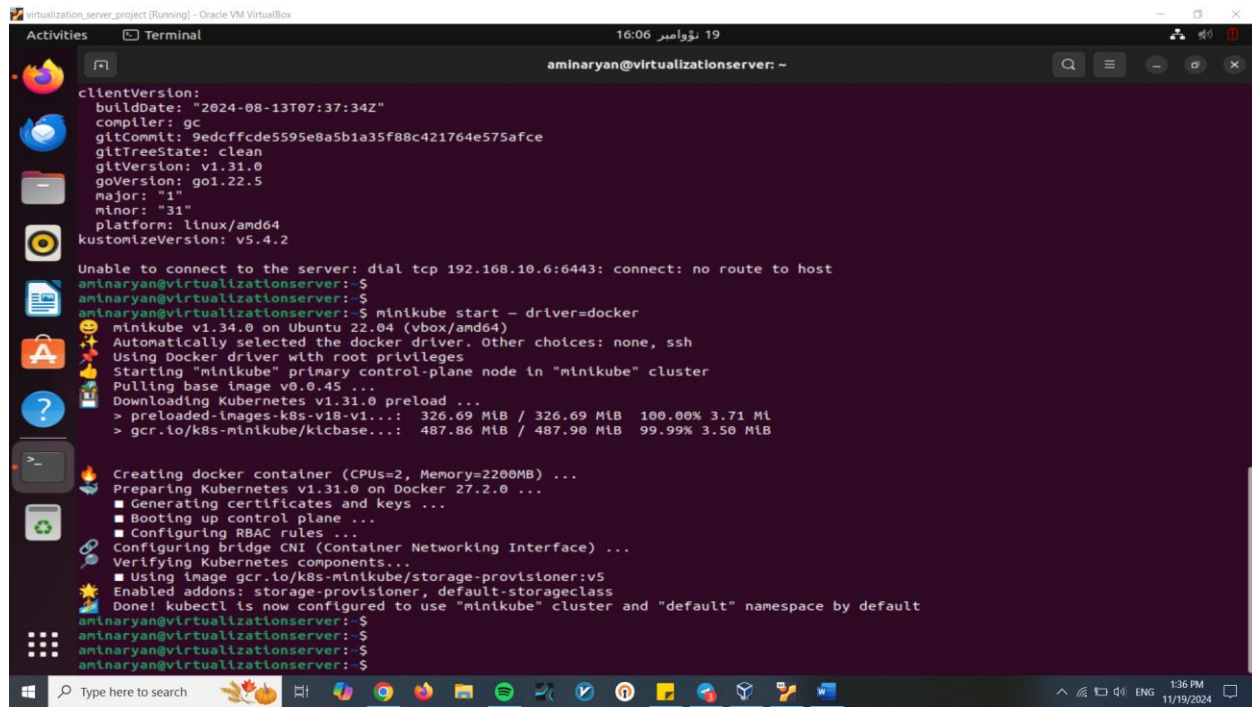


Project 1

First we install minikube:



A terminal window titled 'virtualization_server_project [Running] - Oracle VM VirtualBox' showing the installation of minikube. The user 'aminaryan@virtualizationserver: ~' runs several commands. First, they check the minikube version and build date. Then, they attempt to connect to a server at 192.168.10.6:6443, which fails due to no route to host. Next, they run 'minikube start --driver=docker'. The terminal shows that minikube v1.34.0 is installed on Ubuntu 22.04. It automatically selects the docker driver and starts the 'minikube' primary control-plane node. It then downloads the Kubernetes v1.31.0 preload and the gcr.io/k8s-minikube/kicbase image. Finally, it creates a docker container (CPUs=2, Memory=2200MB), prepares Kubernetes v1.31.0 on Docker 27.2.0, and configures the bridge CNI. The cluster is now configured to use the 'minikube' cluster and 'default' namespace.

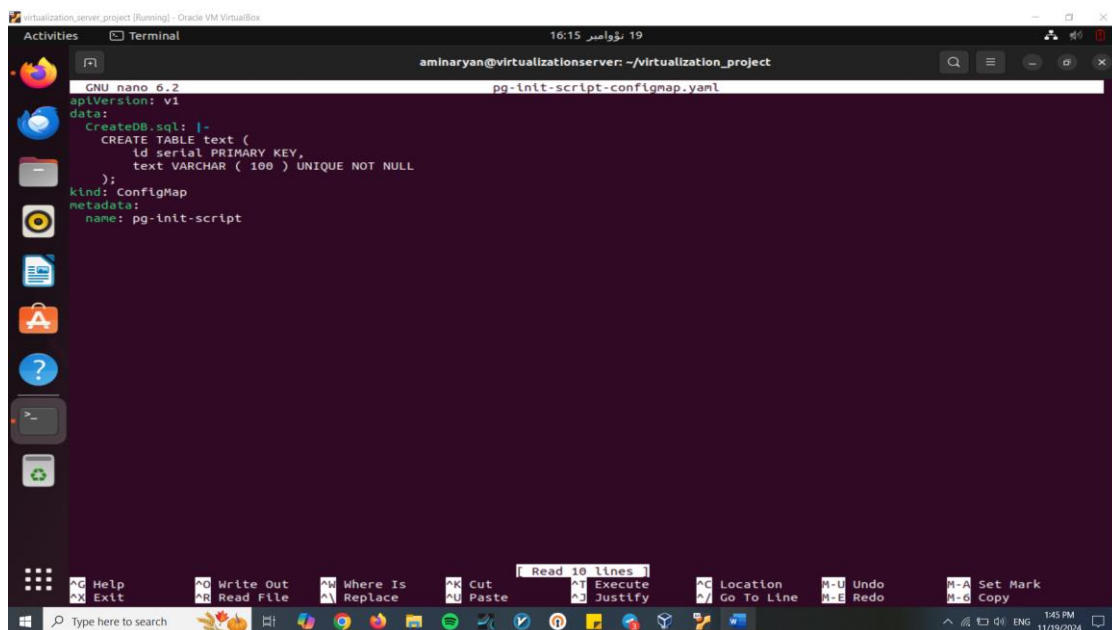
```
clientVersion:
  buildDate: "2024-08-13T07:37:34Z"
  compiler: gc
  gitCommit: 9edcffe5595e8a5b1a35f88c421764e575afce
  gitTreeState: clean
  gitVersion: v1.31.0
  goVersion: go1.22.5
  major: "1"
  minor: "31"
  platform: linux/amd64
  kustomizeVersion: v5.4.2

Unable to connect to the server: dial tcp 192.168.10.6:6443: connect: no route to host
aminaryan@virtualizationserver:~$
aminaryan@virtualizationserver:~$
aminaryan@virtualizationserver:~$ minikube start --driver=docker
minikube v1.34.0 on Ubuntu 22.04 (vbox/amd64)
Automatically selected the docker driver. Other choices: none, ssh
Using Docker driver with root privileges
Starting "minikube" primary control-plane node in "minikube" cluster
Pulling base image v0.0.45 ...
Downloading Kubernetes v1.31.0 preload ...
> preloaded-images-k8s-v18-v1...: 326.69 MiB / 326.69 MiB 100.00% 3.71 Mi
> gcr.io/k8s-minikube/kicbase...: 487.86 MiB / 487.90 MiB 99.99% 3.50 MiB

Creating docker container (CPUs=2, Memory=2200MB) ...
Preparing Kubernetes v1.31.0 on Docker 27.2.0 ...
  ■ Generating certificates and keys ...
  ■ Booting up control plane ...
  ■ Configuring RBAC rules ...
  ■ Configuring bridge CNI (Container Networking Interface) ...
  ■ Verifying Kubernetes components...
  ■ Using image gcr.io/k8s-minikube/storage-provisioner:v5
Enabled addons: storage-provisioner, default-storageclass
Done! kubectrl is now configured to use "minikube" cluster and "default" namespace by default
aminaryan@virtualizationserver:~$
aminaryan@virtualizationserver:~$
aminaryan@virtualizationserver:~$
aminaryan@virtualizationserver:~$
```

Database

Now we try to deploy postgres to minikube. First we write a configmap for the table:

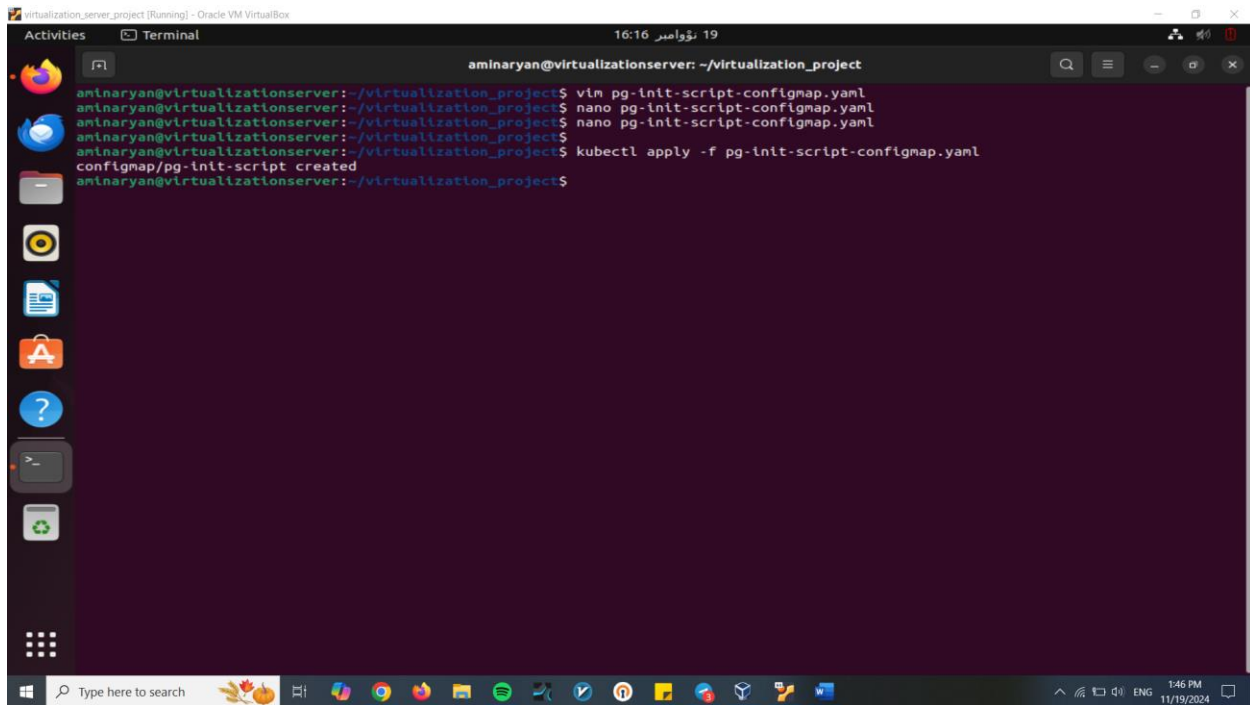


A terminal window titled 'virtualization_server_project [Running] - Oracle VM VirtualBox' showing the creation of a ConfigMap. The user 'aminaryan@virtualizationserver: ~' runs 'nano pg-init-script-configmap.yaml'. The file content is as follows:

```
apiVersion: v1
data:
  CreateDB.sql: |-
    CREATE TABLE text (
      id serial PRIMARY KEY,
      text VARCHAR ( 100 ) UNIQUE NOT NULL
    );
kind: ConfigMap
metadata:
  name: pg-init-script
```

The terminal shows the file being edited in nano. The bottom of the terminal displays the nano editor's status bar with various shortcuts like 'Ctrl+G Help', 'Ctrl+W Write Out', etc.

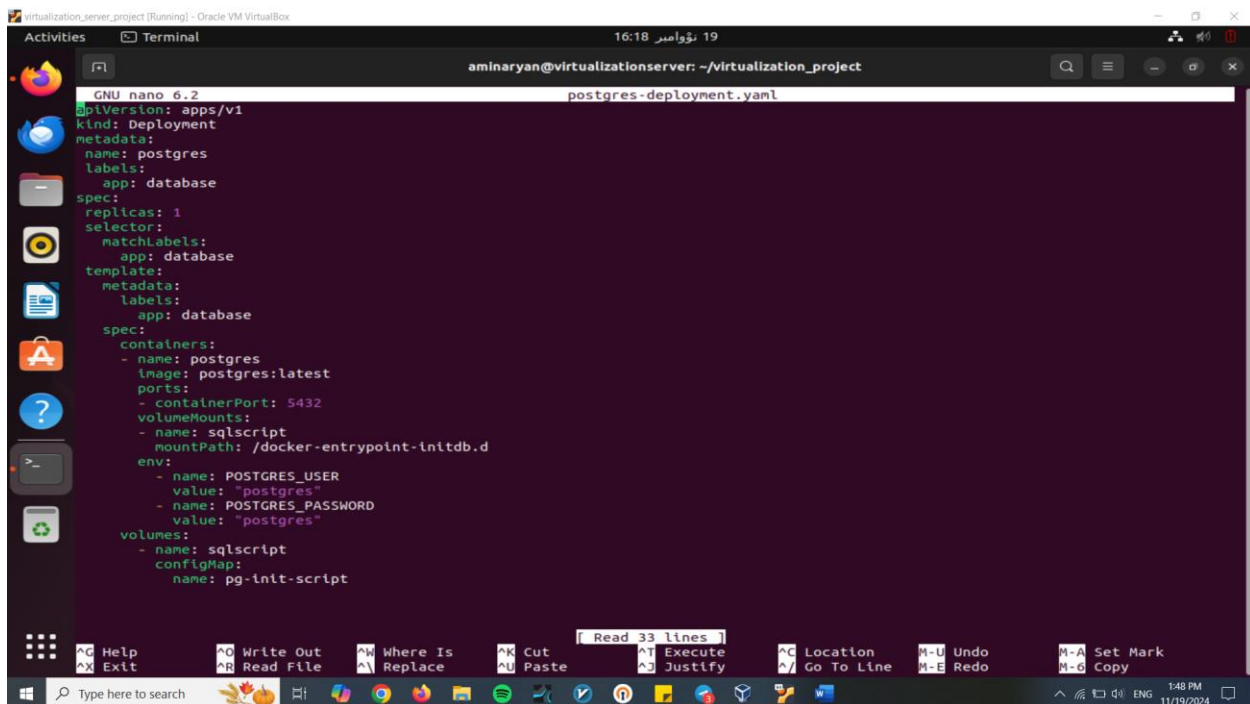
Now we apply it:



A terminal window titled 'virtualization_server_project [Running] - Oracle VM VirtualBox' with a dark background. The prompt is 'aminaryan@virtualizationserver: ~/virtualization_project'. The user enters the following commands: 'vim pg-init-script-configmap.yaml', 'nano pg-init-script-configmap.yaml', 'nano pg-init-script-configmap.yaml', and 'kubectl apply -f pg-init-script-configmap.yaml'. The output of the last command is 'configmap/pg-init-script created'. The terminal window is part of a desktop environment with a sidebar on the left and a taskbar at the bottom.

```
aminaryan@virtualizationserver: ~/virtualization_project
aminaryan@virtualizationserver:~/virtualization_project$ vim pg-init-script-configmap.yaml
aminaryan@virtualizationserver:~/virtualization_project$ nano pg-init-script-configmap.yaml
aminaryan@virtualizationserver:~/virtualization_project$ nano pg-init-script-configmap.yaml
aminaryan@virtualizationserver:~/virtualization_project$ kubectl apply -f pg-init-script-configmap.yaml
configmap/pg-init-script created
aminaryan@virtualizationserver:~/virtualization_project$
```

Now we write a postgres deployment:

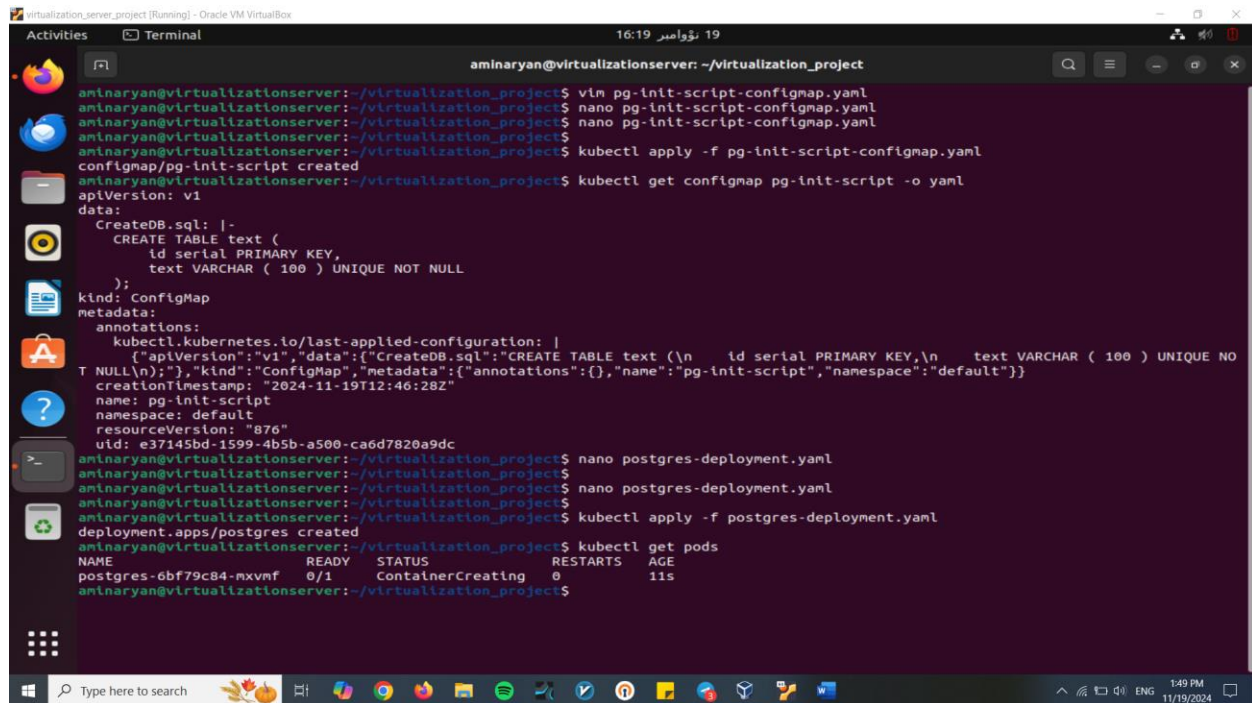


A terminal window titled 'virtualization_server_project [Running] - Oracle VM VirtualBox' with a dark background. The prompt is 'aminaryan@virtualizationserver: ~/virtualization_project'. The user has opened a file named 'postgres-deployment.yaml' in the nano editor. The content of the file is a Kubernetes Deployment for PostgreSQL. The terminal window is part of a desktop environment with a sidebar on the left and a taskbar at the bottom.

```
aminaryan@virtualizationserver: ~/virtualization_project
GNU nano 6.2 postgres-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres
  labels:
    app: database
spec:
  replicas: 1
  selector:
    matchLabels:
      app: database
  template:
    metadata:
      labels:
        app: database
    spec:
      containers:
        - name: postgres
          image: postgres:latest
          ports:
            - containerPort: 5432
          volumeMounts:
            - name: sqlscript
              mountPath: /docker-entrypoint-initdb.d
          env:
            - name: POSTGRES_USER
              value: "postgres"
            - name: POSTGRES_PASSWORD
              value: "postgres"
      volumes:
        - name: sqlscript
          configMap:
            name: pg-init-script
```

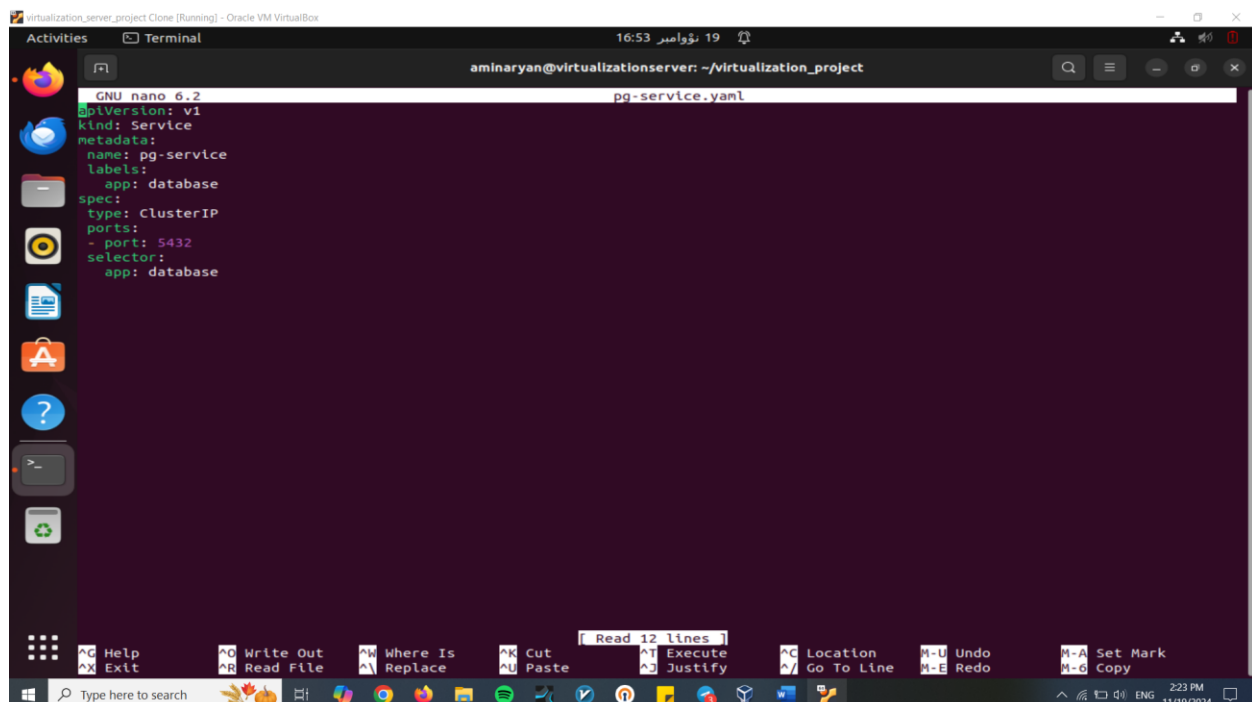
and now we apply it:

Mohammadamin Aryan – Virtualization Project



```
aminaryan@virtualizationserver: ~/virtualization_project
aminaryan@virtualizationserver:~/virtualization_project$ vim pg-init-script-configmap.yaml
aminaryan@virtualizationserver:~/virtualization_project$ nano pg-init-script-configmap.yaml
aminaryan@virtualizationserver:~/virtualization_project$ nano pg-init-script-configmap.yaml
aminaryan@virtualizationserver:~/virtualization_project$ kubectl apply -f pg-init-script-configmap.yaml
configmap/pg-init-script created
aminaryan@virtualizationserver:~/virtualization_project$ kubectl get configmap pg-init-script -o yaml
apiVersion: v1
data:
  CreateDB.sql: |-
    CREATE TABLE text (
      id serial PRIMARY KEY,
      text VARCHAR ( 100 ) UNIQUE NOT NULL
    );
kind: ConfigMap
metadata:
  annotations:
    kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","data":{"CreateDB.sql":"CREATE TABLE text (\n  id serial PRIMARY KEY,\n  text VARCHAR ( 100 ) UNIQUE NO
T NULL\n);"},"kind":"ConfigMap","metadata":{"annotations":{},"name":"pg-init-script","namespace":"default"}}
  creationTimestamp: "2024-11-19T12:46:28Z"
  name: pg-init-script
  namespace: default
  resourceVersion: "876"
  uid: e37145bd-1599-4b5b-a500-ca6d7820a9dc
aminaryan@virtualizationserver:~/virtualization_project$ nano postgres-deployment.yaml
aminaryan@virtualizationserver:~/virtualization_project$ nano postgres-deployment.yaml
aminaryan@virtualizationserver:~/virtualization_project$ kubectl apply -f postgres-deployment.yaml
deployment.apps/postgres created
aminaryan@virtualizationserver:~/virtualization_project$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
postgres-6bf79c84-mxvmf             0/1     ContainerCreating   0          11s
aminaryan@virtualizationserver:~/virtualization_project$
```

Now because we do not want users to communicate with the database directly, and want the database to communicate with the backend, we apply this service:



```
aminaryan@virtualizationserver: ~/virtualization_project
aminaryan@virtualizationserver:~/virtualization_project$ nano pg-service.yaml
aminaryan@virtualizationserver:~/virtualization_project$ kubectl apply -f pg-service.yaml
service/pg-service created
aminaryan@virtualizationserver:~/virtualization_project$ kubectl get service
NAME    TYPE        PORT(S)
pg-svc  ClusterIP   5432
aminaryan@virtualizationserver:~/virtualization_project$
```

Mohammadamin Aryan – Virtualization Project

```
virtualization_server_project Clone [Running] - Oracle VM VirtualBox
Activities Terminal 16:56 19 نوفمبر
aminaryan@virtualizationserver: ~/virtualization_project
aminaryan@virtualizationserver:~/virtualization_project$ kubectl get pods
NAME                READY   STATUS    RESTARTS   AGE
postgres-6bf79c84-mxvmf 1/1     Running   1 (5m50s ago)   36m
aminaryan@virtualizationserver:~/virtualization_project$ kubectl exec -it postgres-6bf79c84-mxvmf postgres -- /bin/bash
root@postgres-6bf79c84-mxvmf:/# psql -U postgres
psql (17.1 (Debian 17.1-1.pgdg120+1))
Type "help" for help.

postgres=# \dt
              List of relations
Schema | Name | Type | Owner
-----+-----+-----+-----
public | text | table | postgres
(1 row)

postgres=#
```

Backend

Now we try to build the docker image for the backend:

```
virtualization_server_project Clone [Running] - Oracle VM VirtualBox
Activities Terminal 17:16 19 نوفمبر
aminaryan@virtualizationserver: ~/virtualization_project/fullstackapp/backend
aminaryan@virtualizationserver:~/virtualization_project/fullstackapp/backend$ docker build -t backend .
[+] Building 242.2s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 227B
=> [internal] load metadata for docker.io/library/python:3.7
=> [auth] library/python:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load build context
=> => transferring context: 1.52kB
=> [1/5] FROM docker.io/library/python:3.7@sha256:eedf63967cdb57d8214db38ce21f105003ed4e4d0358f02bedc057341bcf92a0
=> resolve docker.io/library/python:3.7@sha256:eedf63967cdb57d8214db38ce21f105003ed4e4d0358f02bedc057341bcf92a0
=> sha256:2011a37d2a08fe83dd9ff923e0f83bdf7290152e2e6afe359bde1453170d9bdc 2.01kB / 2.01kB
=> sha256:167b8a53ca4504bc6aa3182e336fa96f4ef76875d158c1933d3e2fa19c57e0c3 49.56MB / 49.56MB
=> sha256:eedf63967cdb57d8214db38ce21f105003ed4e4d0358f02bedc057341bcf92a0 1.86kB / 1.86kB
=> sha256:16d93ae3411be3db255b6b52fdcf155a0dff0f697c2e4e3d862caf8d978830b2 8.13kB / 8.13kB
=> sha256:b47a222d28fa95600198398973d0a29b82a968f03e7ef361cc8ded562e4d84a3 24.03MB / 24.03MB
=> sha256:debce5f9f3a9709085f7f2ad3cf41f036a3b57b400b27ba3a883928315787042 64.11MB / 64.11MB
=> sha256:1d7ca7cd2e060ae77ac6284a9d027f72a31a02a18bfc2a249ef2e7b01074338b 211.04MB / 211.04MB
=> sha256:ff3119008f58beeef8336fa83370b0fe914db94ca0b7bb55abe3e1bf2b1ad56 6.39MB / 6.39MB
=> extracting sha256:167b8a53ca4504bc6aa3182e336fa96f4ef76875d158c1933d3e2fa19c57e0c3
=> sha256:c2423a76a32b7fffb2ee7bb6d1e0c74bb1811237eddcb3200594daf7a52d4f378 14.70MB / 14.70MB
=> sha256:elc98ca4926a91839805ce7ed76a70225e303007331ee0f45dfabbbf55fd8c8 2.44B / 2.44B
=> sha256:3b62c8e1d79b6554a8bffc1196ff5dd822858c179f1f8dc0f0c74a288859a6fb 2.85MB / 2.85MB
=> extracting sha256:b47a222d28fa95600198398973d0a29b82a968f03e7ef361cc8ded562e4d84a3
=> extracting sha256:debce5f9f3a9709085f7f2ad3cf41f036a3b57b400b27ba3a883928315787042
=> extracting sha256:1d7ca7cd2e060ae77ac6284a9d027f72a31a02a18bfc2a249ef2e7b01074338b
=> extracting sha256:ff3119008f58beeef8336fa83370b0fe914db94ca0b7bb55abe3e1bf2b1ad56
=> extracting sha256:c2423a76a32b7fffb2ee7bb6d1e0c74bb1811237eddcb3200594daf7a52d4f378
=> extracting sha256:elc98ca4926a91839805ce7ed76a70225e303007331ee0f45dfabbbf55fd8c8
=> extracting sha256:3b62c8e1d79b6554a8bffc1196ff5dd822858c179f1f8dc0f0c74a288859a6fb
=> [2/5] RUN mkdir /app
=> [3/5] WORKDIR /app
=> [4/5] ADD . /app/
=> [5/5] RUN pip install -r requirements.txt
=> exporting to image
=> exporting layers
=> writing image sha256:179e00345548eeef64ec354cd10bdc810b9c5fc48c7dd1933d274cd85b58b053
=> naming to docker.io/library/backend
aminaryan@virtualizationserver:~/virtualization_project/fullstackapp/backend$
```


And now we apply the backend deployment and service:

```
virtualization_server_project Clone [Running] - Oracle VM VirtualBox
Activities Terminal 17:18 19
aminaryan@virtualizationserver: ~/virtualization_project/fullstackapp/kubernetes

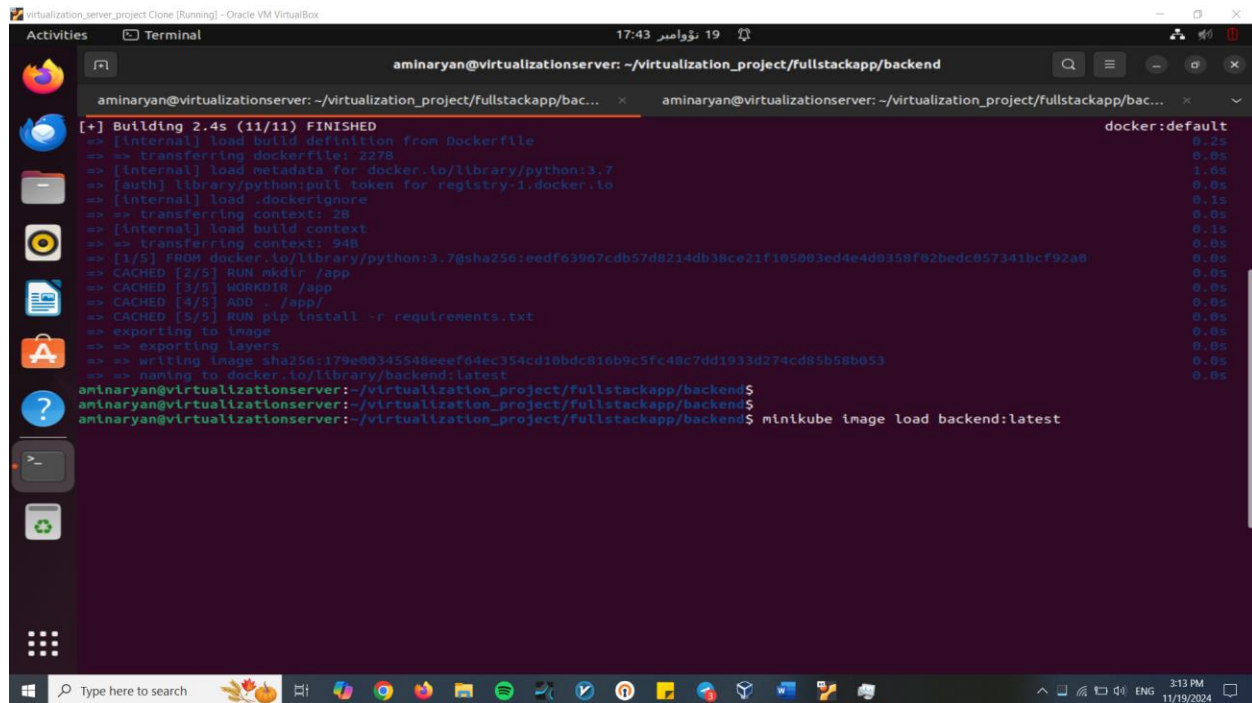
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
  labels:
    app: backend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
        - name: flask-backend
          image: backend:latest
          imagePullPolicy: Never
          ports:
            - containerPort: 5000
          env:
            - name: DATABASE_URI
              value: pg-service
---
apiVersion: v1
kind: Service
metadata:
  name: flask-service
labels:
  app: backend
spec:
  type: ClusterIP
  selector:
    app: backend
  ports:
    - port: 5000
      targetPort: 5000
"backend.yaml" 39L, 641B
1,1 All
```

```
virtualization_server_project Clone [Running] - Oracle VM VirtualBox
Activities Terminal 17:21 19
aminaryan@virtualizationserver: ~/virtualization_project/fullstackapp/kubernetes

aminaryan@virtualizationserver:~/virtualization_project/fullstackapp/kubernetes$ kubectl get pods
NAME                                READY   STATUS             RESTARTS   AGE
backend-555cf5958d-fjsrf            0/1     ErrImageNeverPull   0           2m11s
postgres-6bf79c84-mxvmf            1/1     Running             1 (31m ago)  61m
aminaryan@virtualizationserver:~/virtualization_project/fullstackapp/kubernetes$ kubectl logs backend-555cf5958d-fjsrf
Error from server (BadRequest): container "flask-backend" in pod "backend-555cf5958d-fjsrf" is waiting to start: ErrImageNeverPull
aminaryan@virtualizationserver:~/virtualization_project/fullstackapp/kubernetes$
```

We see that we have an error. It is because we have to load the built image into minikube:

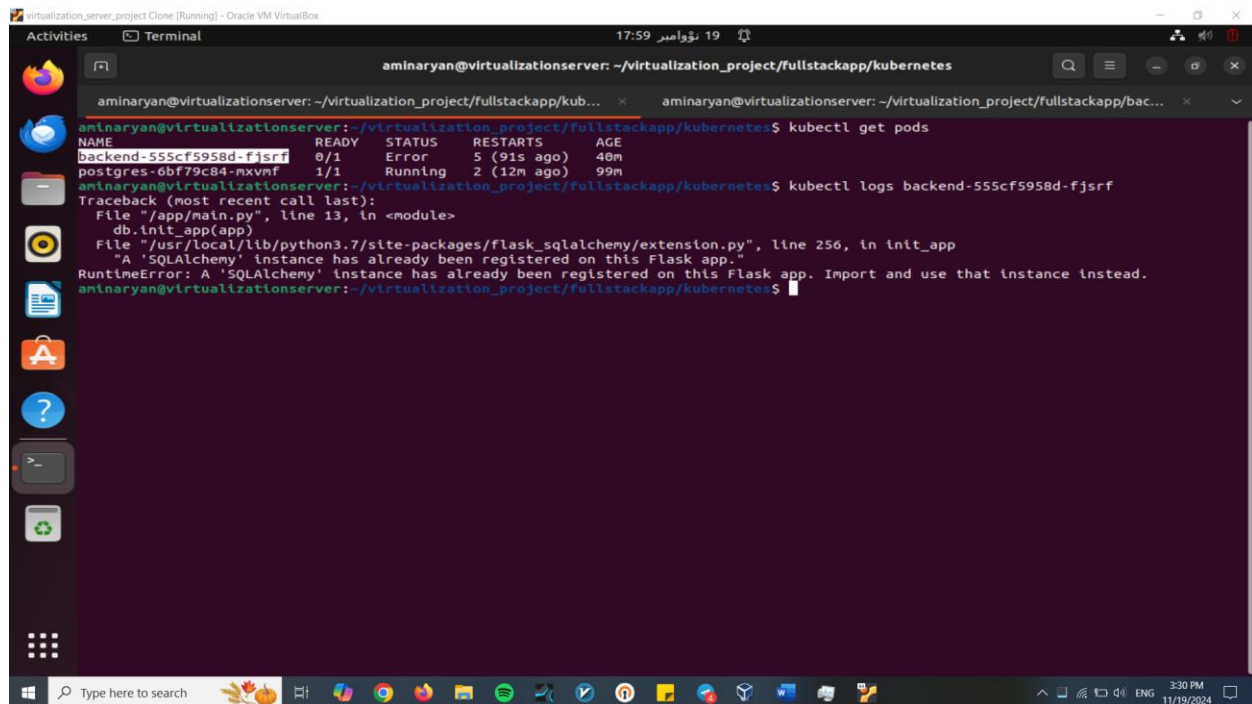
Mohammadamin Aryan – Virtualization Project



The screenshot shows a terminal window titled "virtualization_server_project Clone [Running] - Oracle VM VirtualBox". The terminal is running a Docker build command. The output shows the build progress, including the transfer of the Dockerfile, loading of metadata, and the final image export. The build is completed successfully, and the image is named "docker:default".

```
aminaryan@virtualizationserver: ~/virtualization_project/fullstackapp/backend
aminaryan@virtualizationserver: ~/virtualization_project/fullstackapp/backend$ docker build -t backend:latest .
[+] Building 2.4s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 227B
=> [internal] load metadata for docker.io/library/python:3.7
=> [auth] library/python:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load build context
=> => transferring context: 94B
=> [1/5] FROM docker.io/library/python:3.7@sha256:eedf63967c8b57d8214db38ce21f105003ed4e4d0358f02bedc057341bcf92a0
=> CACHED [2/5] RUN mkdir /app
=> CACHED [3/5] WORKDIR /app
=> CACHED [4/5] ADD . /app/
=> CACHED [5/5] RUN pip install -r requirements.txt
=> exporting image
=> => exporting layers
=> => writing image sha256:179e00345548eeef64ec354cd10bdc816b9c5fc48c7dd1933d274cd85b58b053
=> => naming to docker.io/library/backend:latest
aminaryan@virtualizationserver: ~/virtualization_project/fullstackapp/backend$
aminaryan@virtualizationserver: ~/virtualization_project/fullstackapp/backend$ minikube image load backend:latest
```

Now we try again:



The screenshot shows a terminal window titled "virtualization_server_project Clone [Running] - Oracle VM VirtualBox". The terminal is running a Kubernetes command to get the status of the pods. The output shows a table of pods with their names, statuses, restart counts, and ages. The pod "backend-555cf5958d-fjsrf" is in a "Ready" state. The terminal also shows the logs for this pod, which indicate a "RuntimeError: A 'SQLAlchemy' instance has already been registered on this Flask app. Import and use that instance instead."

```
aminaryan@virtualizationserver: ~/virtualization_project/fullstackapp/kubernetes$ kubectl get pods
NAME                                READY  STATUS   RESTARTS   AGE
backend-555cf5958d-fjsrf            0/1    Error    5 (91s ago)  40m
postgres-6bf79c84-mxvmf            1/1    Running  2 (12m ago)  99m
aminaryan@virtualizationserver: ~/virtualization_project/fullstackapp/kubernetes$ kubectl logs backend-555cf5958d-fjsrf
Traceback (most recent call last):
  File "/app/main.py", line 13, in <module>
    db.init_app(app)
  File "/usr/local/lib/python3.7/site-packages/flask_sqlalchemy/extension.py", line 256, in init_app
    "A 'SQLAlchemy' instance has already been registered on this Flask app."
RuntimeError: A 'SQLAlchemy' instance has already been registered on this Flask app. Import and use that instance instead.
aminaryan@virtualizationserver: ~/virtualization_project/fullstackapp/kubernetes$
```

As you can see the image error has been fixed. Now we have a Runtime error. We will fix the code (remove line 13 because it is not needed) and try again:

Frontend

The screenshot shows a terminal window titled "Virtualization Server [Running] - Oracle VM VirtualBox". The user is logged in as "amlnaryan" on a "virtualizationserver" machine. The current directory is "~/virtualization_project/fullstackapp/frontend".

The user runs the following commands:

```
amlnaryan@virtualizationserver:~/virtualization_project/fullstackapp/frontend$ cat Dockerfile
FROM node:16-alpine

ADD . /frontend
WORKDIR /frontend
RUN npm install --silent

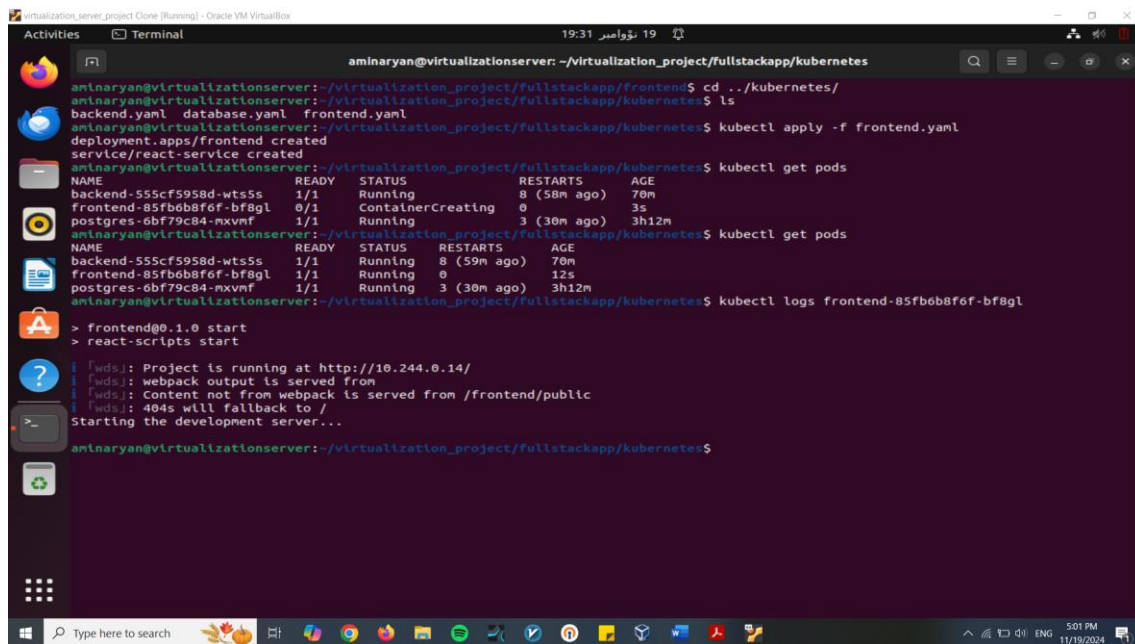
CMD ["npm", "start"]amlnaryan@virtualizationserver:~/virtualization_project/fullstackapp/frontend$
amlnaryan@virtualizationserver:~/virtualization_project/fullstackapp/frontend$ eval $(minikube docker-env)
amlnaryan@virtualizationserver:~/virtualization_project/fullstackapp/frontend$ docker build -t frontend:latest .
[+] Building 4.6s (5/9)
```

The output of the Docker build command is shown on the right side of the terminal:

```
docker:default
==> [internal] load build definition from Dockerfile
==> transferring dockerfile: 140B
==> [internal] load metadata for docker.io/library/node:16-alpine
==> [auth] library/node:pull token for registry-1.docker.io
==> [internal] load .dockerignore
==> transferring context: 116B
==> [internal] load build context
==> transferring context: 6.25kB
==> [1/4] FROM docker.io/library/node:16-alpine@sha256:a1f9d027912b58a7c75be7716c97cfbc6d3099f3a97ed84aa490be9dee20e787
==> resolve docker.io/library/node:16-alpine@sha256:a1f9d027912b58a7c75be7716c97cfbc6d3099f3a97ed84aa490be9dee20e787
==> sha256:72e89a80be58c922ed7b1475e50cf15153767e470695dd106521738b060e139d 1.43kB / 1.43kB
==> sha256:72e89a80be58c922ed7b1475e50cf15153767e470695dd106521738b060e139d 1.16kB / 1.16kB
==> sha256:2573171e8124bb95d14d128728a52a97bb917ef45d7c4f80cfe70bc44aa78b73 6.73kB / 6.73kB
==> sha256:7264adbb415046d30d16ba98b79770e18acce0ffa71850405994cffa9be7de 3.40MB / 3.40MB
==> sha256:eee371b9ce3ffdbb8aa703b9a14d318801ddc3468f096bb6cfeabeb715147f9 4.19MB / 36.63MB
==> sha256:93b3025fe1039271d06ec0d012a9ffa2039d706a322aac099c0831dd93382c2 2.34MB / 2.34MB
==> extracting sha256:7264adbb415046d30d16ba98b79770e18acce0ffa71850405994cffa9be7de
==> sha256:d9059661ce700924f66d2773666584fc8addcb78a2be03f720022f4875577ea9 452B / 452B
```

Mohammadamin Aryan – Virtualization Project

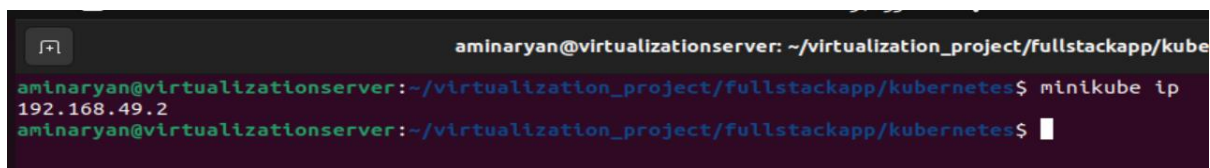
Now that we have our image in minikube, we deploy the frontend into Kuber:



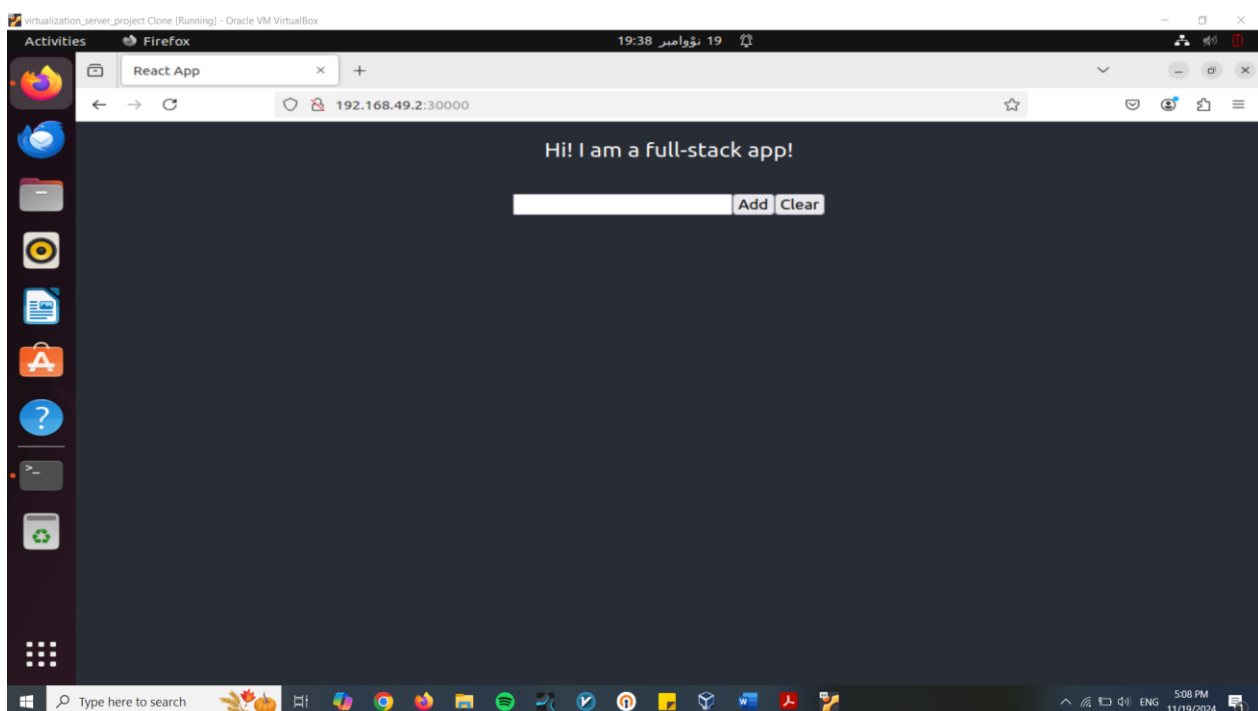
```
aminaryan@virtualizationserver: ~/virtualization_project/fullstackapp/kubernetes
aminaryan@virtualizationserver:~/virtualization_project/fullstackapp/frontend$ cd ../kubernetes/
aminaryan@virtualizationserver:~/virtualization_project/fullstackapp/kubernetes$ ls
backend.yaml database.yaml frontend.yaml
aminaryan@virtualizationserver:~/virtualization_project/fullstackapp/kubernetes$ kubectl apply -f frontend.yaml
deployment.apps/frontend created
aminaryan@virtualizationserver:~/virtualization_project/fullstackapp/kubernetes$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
backend-555cf5958d-wts5s            1/1     Running   8 (58m ago)  70m
frontend-85fb6b8f6f-bf8gl           0/1     ContainerCreating   0           3s
postgres-6bf79c84-mxvmf            1/1     Running   3 (30m ago)  3h12m
aminaryan@virtualizationserver:~/virtualization_project/fullstackapp/kubernetes$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
backend-555cf5958d-wts5s            1/1     Running   8 (59m ago)  70m
frontend-85fb6b8f6f-bf8gl           1/1     Running   0           12s
postgres-6bf79c84-mxvmf            1/1     Running   3 (30m ago)  3h12m
aminaryan@virtualizationserver:~/virtualization_project/fullstackapp/kubernetes$ kubectl logs frontend-85fb6b8f6f-bf8gl
> frontend@0.1.0 start
> react-scripts start

i [vds]: Project is running at http://10.244.0.14/
i [vds]: webpack output is served from
i [vds]: Content not from webpack is served from /frontend/public
i [vds]: 404s will fallback to /
Starting the development server...
```

We successfully deployed the project. Now we are able to open the project on browser using minikube ip:



```
aminaryan@virtualizationserver: ~/virtualization_project/fullstackapp/kube
aminaryan@virtualizationserver:~/virtualization_project/fullstackapp/kubernetes$ minikube ip
192.168.49.2
aminaryan@virtualizationserver:~/virtualization_project/fullstackapp/kubernetes$
```

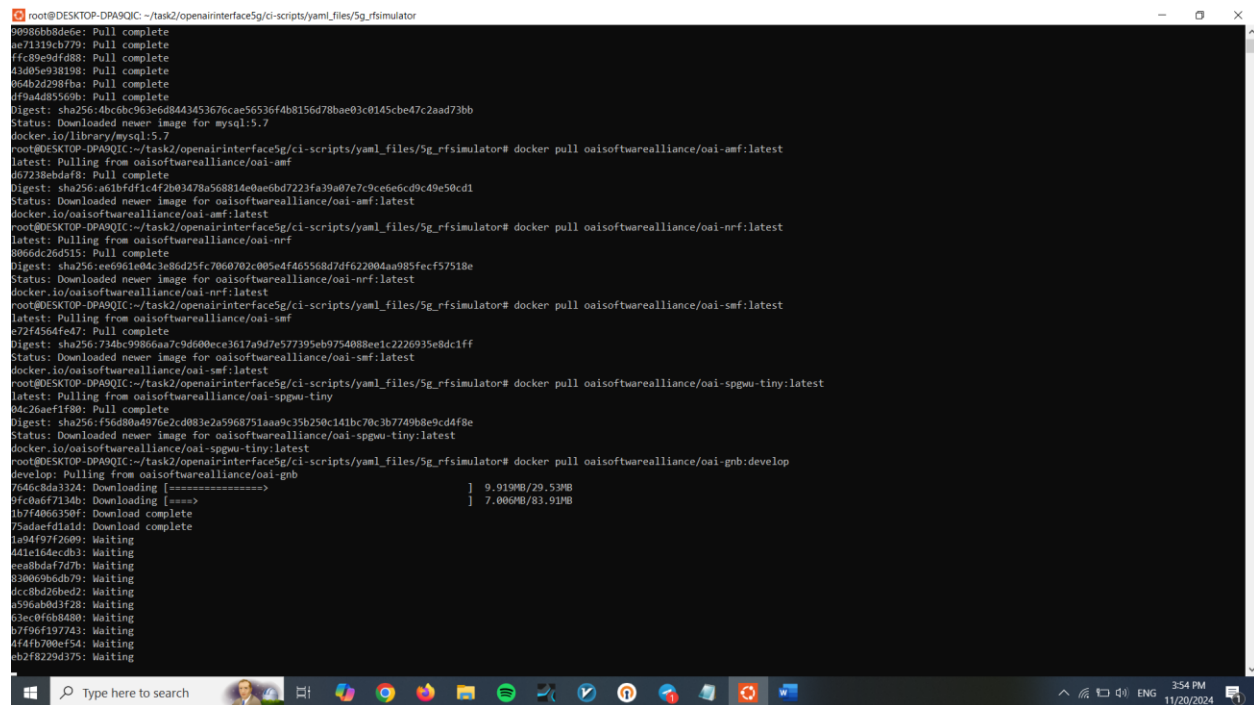


Project 2

We go step by step with the readme in github project. First we pull images from new docker repositories:

```
docker pull mysql:5.7
docker pull oaisoftwarealliance/oai-amf:latest
docker pull oaisoftwarealliance/oai-nrf:latest
docker pull oaisoftwarealliance/oai-smf:latest
docker pull oaisoftwarealliance/oai-spgwu-tiny:latest

docker pull oaisoftwarealliance/oai-gnb:develop
docker pull oaisoftwarealliance/oai-nr-ue:develop
```



```
root@DESKTOP-DPA9QIC: ~/task2/openairinterface5g/ci-scripts/yaml_files/5g_rfsimulator
0098bb8bde: Pull complete
ae71319cb779: Pull complete
ffc89e9df888: Pull complete
43a0e9931198: Pull complete
064b24298fba: Pull complete
d9a4d85569b: Pull complete
Digest: sha256:4bc6bc963e6d84435367cae56536f4b8156d78bae03c0145cbe47c2aad73bb
Status: Downloaded newer image for mysql:5.7
docker.io/library/mysql:5.7
root@DESKTOP-DPA9QIC:~/task2/openairinterface5g/ci-scripts/yaml_files/5g_rfsimulator# docker pull oaisoftwarealliance/oai-amf:latest
latest: Pulling from oaisoftwarealliance/oai-amf
d67238ebdaf8: Pull complete
Digest: sha256:a61bfdf1c4f2b03478a568814e0ae6bd7223fa39a07e7c9ce6e6cd9c49e50cd1
Status: Downloaded newer image for oaisoftwarealliance/oai-amf:latest
docker.io/oaisoftwarealliance/oai-amf:latest
root@DESKTOP-DPA9QIC:~/task2/openairinterface5g/ci-scripts/yaml_files/5g_rfsimulator# docker pull oaisoftwarealliance/oai-nrf:latest
latest: Pulling from oaisoftwarealliance/oai-nrf
8066dc26d515: Pull complete
Digest: sha256:ee6961e0dc3e86d25fc7060702c085e4f465568d7df622004aa985fecf57518e
Status: Downloaded newer image for oaisoftwarealliance/oai-nrf:latest
docker.io/oaisoftwarealliance/oai-nrf:latest
root@DESKTOP-DPA9QIC:~/task2/openairinterface5g/ci-scripts/yaml_files/5g_rfsimulator# docker pull oaisoftwarealliance/oai-smf:latest
latest: Pulling from oaisoftwarealliance/oai-smf
e22f4564fe47: Pull complete
Digest: sha256:734bc99860aa79d60bce3617a0d7e577395eb9754080ee1c2220935e8dc1ff
Status: Downloaded newer image for oaisoftwarealliance/oai-smf:latest
docker.io/oaisoftwarealliance/oai-smf:latest
root@DESKTOP-DPA9QIC:~/task2/openairinterface5g/ci-scripts/yaml_files/5g_rfsimulator# docker pull oaisoftwarealliance/oai-spgwu-tiny:latest
latest: Pulling from oaisoftwarealliance/oai-spgwu-tiny
04c26aeef1f80: Pull complete
Digest: sha256:f56d00a4976c2cd083e2a5968751aa9c35b250c141bc70c3b7740b8e9cd4f8e
Status: Downloaded newer image for oaisoftwarealliance/oai-spgwu-tiny:latest
docker.io/oaisoftwarealliance/oai-spgwu-tiny:latest
root@DESKTOP-DPA9QIC:~/task2/openairinterface5g/ci-scripts/yaml_files/5g_rfsimulator# docker pull oaisoftwarealliance/oai-gnb:develop
develop: Pulling from oaisoftwarealliance/oai-gnb
764dc0da3324: Downloading [=====] 9.919MB/29.53MB
9fc0a6f7134b: Downloading [=====] 7.006MB/83.91MB
1b7f4066350f: Download complete
75adaefda1d: Download complete
1a94f97f2609: Waiting
441e104ecd3: Waiting
eeabdaf7d7b: Waiting
830069b6db79: Waiting
dcb8bd26bed2: Waiting
a596ab0d3f28: Waiting
63ec0f6b8480: Waiting
b7f96f19743: Waiting
4f4fb700ef54: Waiting
eb2f8229d375: Waiting
```

And then we retag them to work with docker-compose:

```
root@DESKTOP-DPA9QIC: ~/task2/openairinterface5g/ci-scripts/yaml_files/5g_rfsimulator
root@DESKTOP-DPA9QIC:~/task2/openairinterface5g/ci-scripts/yaml_files/5g_rfsimulator# docker image tag oaisoftwarealliance/oai-amf:latest oai-amf:latest
root@DESKTOP-DPA9QIC:~/task2/openairinterface5g/ci-scripts/yaml_files/5g_rfsimulator# docker image tag oaisoftwarealliance/oai-nrf:latest oai-nrf:latest
root@DESKTOP-DPA9QIC:~/task2/openairinterface5g/ci-scripts/yaml_files/5g_rfsimulator# docker image tag oaisoftwarealliance/oai-smf:latest oai-smf:latest
root@DESKTOP-DPA9QIC:~/task2/openairinterface5g/ci-scripts/yaml_files/5g_rfsimulator# docker image tag oaisoftwarealliance/oai-spgwu-tiny:latest oai-spgwu-tiny:latest
root@DESKTOP-DPA9QIC:~/task2/openairinterface5g/ci-scripts/yaml_files/5g_rfsimulator# docker image tag oaisoftwarealliance/oai-gnb:develop oai-gnb:develop
root@DESKTOP-DPA9QIC:~/task2/openairinterface5g/ci-scripts/yaml_files/5g_rfsimulator# docker image tag oaisoftwarealliance/oai-nr-ue:develop oai-nr-ue:develop
root@DESKTOP-DPA9QIC:~/task2/openairinterface5g/ci-scripts/yaml_files/5g_rfsimulator#
```

Mohammadamin Aryan – Virtualization Project

After that, we use docker-compose to deploy the project:

```
root@DESKTOP-DPA9QIC: ~/task2/openairinterface5g/ci-scripts/yaml_files/5g_rfsimulator
root@DESKTOP-DPA9QIC:~/task2/openairinterface5g/ci-scripts/yaml_files/5g_rfsimulator# docker-compose up -d mysql oai-nrf oai-amf oai-smf oai-spgwu oai-ext-dn
[+] Running 2/2
  oai-ext-dn 1 layers [0] 00/00 Pulled
  7c457f21c76 Already exists
[+] Running 8/8
  Network rfsim5g-oai-traffic-net-net Created
  Network rfsim5g-oai-public-net Created
  Container rfsim5g-mysql Started
  Container rfsim5g-oai-nrf Started
  Container rfsim5g-oai-spgwu Started
  Container rfsim5g-oai-amf Started
  Container rfsim5g-oai-smf Started
  Container rfsim5g-oai-ext-dn Started
root@DESKTOP-DPA9QIC:~/task2/openairinterface5g/ci-scripts/yaml_files/5g_rfsimulator# docker-compose ps -a
NAME                                COMMAND                                SERVICE    CREATED     STATUS                                     PORTS
rfsim5g-mysql                       mysql:5.7                             mysql      25 seconds ago Up 24 seconds (health: starting) 3306/tcp, 33060/tcp
rfsim5g-oai-amf                     oai-amf:latest                        "/openair-amf/bin/oa..." oai-amf    25 seconds ago Up 22 seconds (health: starting) 80/tcp, 9090/tcp, 38412/sctp
rfsim5g-oai-ext-dn                  ubuntu:bionic                          "/bin/bash -c 'apt -..." oai-ext-dn 24 seconds ago Up 20 seconds (health: starting)
rfsim5g-oai-nrf                     oai-nrf:latest                        "/openair-nrf/bin/oa..." oai-nrf    25 seconds ago Up 24 seconds (health: starting) 80/tcp, 9090/tcp
rfsim5g-oai-smf                     oai-smf:latest                        "/bin/bash -c 'open..." oai-smf    25 seconds ago Up 22 seconds (health: starting) 80/tcp, 8080/tcp, 8805/udp
rfsim5g-oai-spgwu                   oai-spgwu-tiny:latest                 "python3 /openair-sp..." oai-spgwu  25 seconds ago Up 22 seconds (health: starting) 2152/udp, 8805/udp
root@DESKTOP-DPA9QIC:~/task2/openairinterface5g/ci-scripts/yaml_files/5g_rfsimulator#
```

```
rfsim5g-public: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 192.168.71.129 netmask 255.255.255.192 broadcast 192.168.71.191
inet6 fe80::42:65ff:fe32:2edc prefixlen 64 scopeid 0x20<link>
ether 02:42:65:32:2e:dc txqueuelen 0 (Ethernet)
RX packets 81 bytes 9190 (9.1 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 101 bytes 13730 (13.7 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

rfsim5g-traffic: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 192.168.72.129 netmask 255.255.255.192 broadcast 192.168.72.191
inet6 fe80::42:48ff:feea:2190 prefixlen 64 scopeid 0x20<link>
ether 02:42:48:ea:21:90 txqueuelen 0 (Ethernet)
RX packets 6487 bytes 379441 (379.4 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 12638 bytes 30444662 (30.4 MB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

As you can see, the project is being run with docker-compose.

In regarding to installing the docker-compose, I already had docker-compose on my system but this is how it can be installed:

```
1. sudo curl -L
   "https://github.com/docker/compose/releases/download/1.29.2/docker-
   compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

2. sudo chmod +x /usr/local/bin/docker-compose
```

Project 3

First we install microk8s on ubuntu:

```
root@new-server:~#  
root@new-server:~#  
root@new-server:~# sudo snap install microk8s --classic  
2024-11-25T12:28:57Z INFO Waiting for automatic snapd restart...  
microk8s (1.31/stable) v1.31.2 from Canonical installed  
root@new-server:~#  
root@new-server:~# sudo microk8s status  
microk8s is not running. Use microk8s inspect for a deeper inspection.  
root@new-server:~#  
root@new-server:~#  
root@new-server:~#  
root@new-server:~#  
root@new-server:~#  
root@new-server:~#  
root@new-server:~#  
root@new-server:~# sudo microk8s status  
microk8s is not running. Use microk8s inspect for a deeper inspection.  
root@new-server:~# Run ps -ef | grep kubelet  
Command 'Run' not found, did you mean:  
  command 'Run' from snap bun-js (1.1.36)  
  command 'Run' from deb python3-zunclient (4.4.0-0ubuntu1)  
See 'snap info <snapname>' for additional versions.  
root@new-server:~#  
root@new-server:~#  
root@new-server:~#  
root@new-server:~#  
root@new-server:~#  
root@new-server:~# sudo microk8s status  
microk8s is running  
high-availability: no  
datastore master nodes: 127.0.0.1:19001  
datastore standby nodes: none  
addons:  
enabled:  
  dns # (core) CoreDNS  
  ha-cluster # (core) Configure high availability on the current node  
  helm # (core) Helm - the package manager for Kubernetes  
  helm3 # (core) Helm 3 - the package manager for Kubernetes  
disabled:  
  cert-manager # (core) Cloud native certificate management  
  cis-hardening # (core) Apply CIS K8s hardening  
  community # (core) The community addons repository  
  dashboard # (core) The Kubernetes dashboard
```

Then I decided to add an alias to simplify the use of microk8s:

```
root@new-server:~# alias kubect1='sudo microk8s kubect1'  
root@new-server:~#  
root@new-server:~#  
root@new-server:~#  
root@new-server:~#  
root@new-server:~# kubect1 get nodes  
NAME          STATUS    ROLES    AGE    VERSION  
new-server    Ready    <none>    2m     v1.31.2  
root@new-server:~# kubect1 get nodes  
NAME          STATUS    ROLES    AGE    VERSION  
new-server    Ready    <none>    2m2s   v1.31.2  
root@new-server:~#
```


Now we have microk8s running and ready. We try to enable DNS and Storage for it:

```
root@new-server: ~/virtualization_project/task3
root@new-server:~/virtualization_project/task3# sudo microk8s enable dns storage
Infer repository core for addon dns
Infer repository core for addon storage
WARNING: Do not enable or disable multiple addons in one command.
        This form of chained operations on addons will be DEPRECATED in the future.
        Please, enable one addon at a time: 'microk8s enable <addon>'
Addon core/dns is already enabled
DEPRECATION WARNING: 'storage' is deprecated and will soon be removed. Please use 'hostpath-storage' instead.

Infer repository core for addon hostpath-storage
Enabling default storage class.
WARNING: Hostpath storage is not suitable for production environments.
        A hostpath volume can grow beyond the size limit set in the volume claim manifest.

deployment.apps/hostpath-provisioner created
storageclass.storage.k8s.io/microk8s-hostpath created
serviceaccount/microk8s-hostpath created
clusterrole.rbac.authorization.k8s.io/microk8s-hostpath created
clusterrolebinding.rbac.authorization.k8s.io/microk8s-hostpath created
Storage will be available soon.
root@new-server:~/virtualization_project/task3#
```

It's time to write the needed deployments and services for this project. Here, I tried to read the docker-compose file for each service and write a compatible deployment and service yaml file for microk8s. We start by database:

```
mysql:
  container_name: "rfsim5g-mysql"
  image: mysql:5.7
  volumes:
    - ./oai_db.sql:/docker-entrypoint-initdb.d/oai_db.sql
    - ./mysql-healthcheck.sh:/tmp/mysql-healthcheck.sh
  environment:
    - TZ=Europe/Paris
    - MYSQL_DATABASE=oai_db
    - MYSQL_USER=test
    - MYSQL_PASSWORD=test
    - MYSQL_ROOT_PASSWORD=linux
  healthcheck:
    test: /bin/bash -c "/tmp/mysql-healthcheck.sh"
    interval: 10s
    timeout: 5s
    retries: 5
  networks:
    public_net:
      ipv4_address: 192.168.71.131
```

First of all, we have a file for our database. Therefore, we should create a configmap to map the file we have to microk8s. we create a configmap called "db-scripts":

```
root@new-server: ~/virtualization_project/task3
root@new-server:~/virtualization_project/task3# kubectl create configmap db-scripts --from-file=oai_db.sql
configmap/db-scripts created
root@new-server:~/virtualization_project/task3# kubectl get configmaps
NAME          DATA   AGE
db-scripts    1       6s
kube-root-ca.crt 1      35h
root@new-server:~/virtualization_project/task3#
```

Now we write the db-deployment for our mysql database:

```
root@new-server: ~/virtualization_project/task3
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - name: mysql
        image: mysql:5.7
        env:
        - name: MYSQL_DATABASE
          value: "oai_db"
        - name: MYSQL_USER
          value: "test"
        - name: MYSQL_PASSWORD
          value: "test"
        - name: MYSQL_ROOT_PASSWORD
          value: "linux"
        volumeMounts:
        - name: db-scripts
          mountPath: /docker-entrypoint-initdb.d/oai_db.sql
          subPath: oai_db.sql
      volumes:
      - name: db-scripts
        configMap:
          name: db-scripts
```

We set env for every deployment based on environment in docker-compose. Also, we add volumes and configmap to our deployment for it to be able to access the database file.

We also write a db-service for our other deployments to access the database:

```
root@new-server: ~/virt
apiVersion: v1
kind: Service
metadata:
  name: mysql
spec:
  selector:
    app: mysql
  ports:
  - protocol: TCP
    port: 3306
    targetPort: 3306
```

Now we apply them to microk8s:

```
root@new-server:~/virtualization_project/task3# vim db-deployment.yaml
root@new-server:~/virtualization_project/task3# vim db-service.yaml
root@new-server:~/virtualization_project/task3#
root@new-server:~/virtualization_project/task3# kubectl apply -f db-deployment.yaml
deployment.apps/mysql created
root@new-server:~/virtualization_project/task3# kubectl apply -f db-service.yaml
service/mysql created
root@new-server:~/virtualization_project/task3# kubectl get services
NAME                TYPE          CLUSTER-IP      EXTERNAL-IP  PORT(S)    AGE
kubernetes           ClusterIP     10.152.183.1     <none>       443/TCP    35h
mysql                 ClusterIP     10.152.183.223   <none>       3306/TCP   7s
root@new-server:~/virtualization_project/task3# kubectl get pods
NAME                READY   STATUS    RESTARTS   AGE
mysql-c78f5d65b-vv6cb 1/1     Running   0           20s
root@new-server:~/virtualization_project/task3#
```

Now we move on to the next deployment which is “oai-nrf”:

```
oai-nrf:
  container_name: "rfsim5g-oai-nrf"
  image: oai-nrf:latest
  environment:
    - NRF_INTERFACE_NAME_FOR_SBI=eth0
    - NRF_INTERFACE_PORT_FOR_SBI=80
    - NRF_INTERFACE_HTTP2_PORT_FOR_SBI=9090
    - NRF_API_VERSION=v1
    - INSTANCE=0
    - PID_DIRECTORY=/var/run
  networks:
    public_net:
      ipv4_address: 192.168.71.130
  volumes:
    - ./nrf-healthcheck.sh:/openair-nrf/bin/nrf-healthcheck.sh
  healthcheck:
    test: /bin/bash -c "/openair-nrf/bin/nrf-healthcheck.sh"
    interval: 10s
    timeout: 5s
    retries: 5
```

We write a deployment for it:

```
root@new-server: ~/virtualization_project/task3
apiVersion: apps/v1
kind: Deployment
metadata:
  name: oai-nrf
  labels:
    app: oai-nrf
spec:
  replicas: 1
  selector:
    matchLabels:
      app: oai-nrf
  template:
    metadata:
      labels:
        app: oai-nrf
    spec:
      containers:
        - name: oai-nrf
          image: oaisoftwarealliance/oai-nrf:latest
          env:
            - name: NRF_INTERFACE_NAME_FOR_SBI
              value: eth0
            - name: NRF_INTERFACE_PORT_FOR_SBI
              value: "80"
            - name: NRF_INTERFACE_HTTP2_PORT_FOR_SBI
              value: "9090"
            - name: NRF_API_VERSION
              value: v1
            - name: INSTANCE
              value: "0"
            - name: PID_DIRECTORY
              value: /var/run
          ports:
            - containerPort: 8080
              name: http
```

Like before, the env is just a copy of environment from docker-compose. After that, we should use the new image mentioned in the project description and also we have to expose port 8080

for it because this images runs a listener on this port. Also because of that, we have to write a service for it:

```
apiVersion: v1
kind: Service
metadata:
  name: oai-nrf
  labels:
    app: oai-nrf
spec:
  selector:
    app: oai-nrf
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
      name: http
  type: ClusterIP
```

Now we do the same for oai-amf. The thing in oai-amf is that it needs to connect to mysql and oai-nrf. In docker-compose, because these two services has some static IP, oai-amf needed the IP of these services but in microk8s, they both have a service instead of static IP. Therefore, we have to pass the service name instead of an static IP to oai-amf in its environment variables:

```
- name: NRF_IPV4_ADDRESS
  value: "oai-nrf"
- name: NRF_PORT
  value: "80"
- name: NRF_API_VERSION
  value: "v1"
- name: NRF_FQDN
  value: "oai-nrf"
- name: MYSQL_SERVER
  value: "mysql"
```

 root@new-server: ~/virtualization_project/task3

```
root@new-server:~/virtualization_project/task3# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
mysql-c78f5d65b-vv6cb               1/1     Running   0           16h
oai-amf-645cfb678b-gqkcs           1/1     Running   0           3m38s
oai-nrf-6db6c99cc4-zgskw           1/1     Running   0           3h53m
root@new-server:~/virtualization_project/task3#
```

oai-amf:

```

root@new-server: ~/virtualization_project/task3
[2024-11-27 17:54:50.396] [amf_sbi] [info] Could not get JSON content from the response
[2024-11-27 17:54:50.397] [amf_sbi] [debug] NF Update, response from NRF, JSON data:
null
[2024-11-27 17:54:50.397] [amf_app] [debug] Received SBI_UPDATE_NF_INSTANCE_RESPONSE
[2024-11-27 17:54:50.397] [amf_app] [debug] Handle NF Update response
[2024-11-27 17:54:50.397] [amf_app] [debug] Set a timer to the next Heart-beat (10)
[2024-11-27 17:55:00.398] [amf_app] [debug] Send ITTI msg to SBI task to trigger NRF Heartbeat
[2024-11-27 17:55:00.400] [amf_sbi] [info] Receive Update NF Instance Request, handling ...
[2024-11-27 17:55:00.400] [amf_sbi] [debug] Send NF Update to NRF
[2024-11-27 17:55:00.401] [amf_sbi] [debug] Send NF Update to NRF, Msg body [{"op":"replace","path":"/nfStatus","value":"REGISTERED"}]
[2024-11-27 17:55:00.402] [amf_sbi] [debug] Send HTTP message to http://oai-nrf:8080/nrf-nfm/v1/nf-instances/0bcee430-2277-4c8b-92ef-69423da0368d
[2024-11-27 17:55:00.402] [amf_sbi] [info] Send HTTP message to http://oai-nrf:8080/nrf-nfm/v1/nf-instances/0bcee430-2277-4c8b-92ef-69423da0368d
[2024-11-27 17:55:00.403] [amf_sbi] [info] HTTP message Body: [{"op":"replace","path":"/nfStatus","value":"REGISTERED"}]
[2024-11-27 17:55:00.403] [amf_sbi] [debug] Send a simple HTTP request
[2024-11-27 17:55:00.422] [amf_sbi] [info] Get response with HTTP code (204)
[2024-11-27 17:55:00.423] [amf_sbi] [info] Could not get JSON content from the response
[2024-11-27 17:55:00.423] [amf_sbi] [debug] NF Update, response from NRF, JSON data:
null
[2024-11-27 17:55:00.423] [amf_app] [debug] Received SBI_UPDATE_NF_INSTANCE_RESPONSE
[2024-11-27 17:55:00.423] [amf_app] [debug] Handle NF Update response
[2024-11-27 17:55:00.423] [amf_app] [debug] Set a timer to the next Heart-beat (10)
[2024-11-27 17:55:09.558] [amf_app] [info]

-----gNBs' Information-----
Index | Status | Global Id | gNB Name | PLMN
-----
-----UEs' Information-----
Index | 5GMM State | IMSI | GUTI | RAN UE NGAP ID | AMF UE NGAP ID | PLMN | Cell Id
-----

[2024-11-27 17:55:10.424] [amf_app] [debug] Send ITTI msg to SBI task to trigger NRF Heartbeat
[2024-11-27 17:55:10.424] [amf_sbi] [info] Receive Update NF Instance Request, handling ...
[2024-11-27 17:55:10.426] [amf_sbi] [debug] Send NF Update to NRF
[2024-11-27 17:55:10.426] [amf_sbi] [debug] Send NF Update to NRF, Msg body [{"op":"replace","path":"/nfStatus","value":"REGISTERED"}]
[2024-11-27 17:55:10.427] [amf_sbi] [debug] Send HTTP message to http://oai-nrf:8080/nrf-nfm/v1/nf-instances/0bcee430-2277-4c8b-92ef-69423da0368d
[2024-11-27 17:55:10.427] [amf_sbi] [info] Send HTTP message to http://oai-nrf:8080/nrf-nfm/v1/nf-instances/0bcee430-2277-4c8b-92ef-69423da0368d
[2024-11-27 17:55:10.428] [amf_sbi] [info] HTTP message Body: [{"op":"replace","path":"/nfStatus","value":"REGISTERED"}]
[2024-11-27 17:55:10.444] [amf_sbi] [debug] Send a simple HTTP request
[2024-11-27 17:55:10.444] [amf_sbi] [info] Get response with HTTP code (204)
[2024-11-27 17:55:10.445] [amf_sbi] [info] Could not get JSON content from the response
[2024-11-27 17:55:10.445] [amf_sbi] [debug] NF Update, response from NRF, JSON data:
null
[2024-11-27 17:55:10.446] [amf_app] [debug] Received SBI_UPDATE_NF_INSTANCE_RESPONSE
[2024-11-27 17:55:10.446] [amf_app] [debug] Handle NF Update response
[2024-11-27 17:55:10.446] [amf_app] [debug] Set a timer to the next Heart-beat (10)
root@new-server:~/virtualization_project/task3#

```

oai-nrf:

```

root@new-server: ~/virtualization_project/task3
[2024-11-27 16:54:40.381] [nrf_sbi] [info] Got a request to update an NF instance, Instance ID: 0bcee430-2277-4c8b-92ef-69423da0368d
[2024-11-27 16:54:40.382] [nrf_app] [info] Handle Update NF Instance request, NF type AMF (HTTP version 2)
[2024-11-27 16:54:40.383] [nrf_app] [info] NF Heart-Beat procedure!
[2024-11-27 16:54:40.383] [nrf_app] [debug] Replace member nfStatus with new value REGISTERED
[2024-11-27 16:54:40.383] [nrf_app] [info] Updated the NF profile (profile ID 0bcee430-2277-4c8b-92ef-69423da0368d)
[2024-11-27 16:54:40.384] [nrf_app] [debug] NF update for Heartbeat, current time 1732726480384
[2024-11-27 16:54:40.384] [nrf_app] [debug] Set NF status to REGISTERED
[2024-11-27 16:54:50.394] [nrf_sbi] [info] Got a request to update an NF instance, Instance ID: 0bcee430-2277-4c8b-92ef-69423da0368d
[2024-11-27 16:54:50.394] [nrf_app] [info] Handle Update NF Instance request, NF type AMF (HTTP version 2)
[2024-11-27 16:54:50.395] [nrf_app] [info] NF Heart-Beat procedure!
[2024-11-27 16:54:50.395] [nrf_app] [debug] Replace member nfStatus with new value REGISTERED
[2024-11-27 16:54:50.395] [nrf_app] [info] Updated the NF profile (profile ID 0bcee430-2277-4c8b-92ef-69423da0368d)
[2024-11-27 16:54:50.395] [nrf_app] [debug] NF update for Heartbeat, current time 1732726490395
[2024-11-27 16:54:50.395] [nrf_app] [debug] Set NF status to REGISTERED
[2024-11-27 16:55:00.419] [nrf_sbi] [info] Got a request to update an NF instance, Instance ID: 0bcee430-2277-4c8b-92ef-69423da0368d
[2024-11-27 16:55:00.419] [nrf_app] [info] Handle Update NF Instance request, NF type AMF (HTTP version 2)
[2024-11-27 16:55:00.420] [nrf_app] [info] NF Heart-Beat procedure!
[2024-11-27 16:55:00.420] [nrf_app] [debug] Replace member nfStatus with new value REGISTERED
[2024-11-27 16:55:00.420] [nrf_app] [info] Updated the NF profile (profile ID 0bcee430-2277-4c8b-92ef-69423da0368d)
[2024-11-27 16:55:00.420] [nrf_app] [debug] NF update for Heartbeat, current time 1732726500420
[2024-11-27 16:55:00.420] [nrf_app] [debug] Set NF status to REGISTERED
[2024-11-27 16:55:10.440] [nrf_sbi] [info] Got a request to update an NF instance, Instance ID: 0bcee430-2277-4c8b-92ef-69423da0368d
[2024-11-27 16:55:10.441] [nrf_app] [info] Handle Update NF Instance request, NF type AMF (HTTP version 2)
[2024-11-27 16:55:10.441] [nrf_app] [info] NF Heart-Beat procedure!
[2024-11-27 16:55:10.441] [nrf_app] [debug] Replace member nfStatus with new value REGISTERED
[2024-11-27 16:55:10.441] [nrf_app] [info] Updated the NF profile (profile ID 0bcee430-2277-4c8b-92ef-69423da0368d)
[2024-11-27 16:55:10.442] [nrf_app] [debug] NF update for Heartbeat, current time 1732726510442
[2024-11-27 16:55:10.442] [nrf_app] [debug] Set NF status to REGISTERED
[2024-11-27 16:55:20.466] [nrf_sbi] [info] Got a request to update an NF instance, Instance ID: 0bcee430-2277-4c8b-92ef-69423da0368d
[2024-11-27 16:55:20.466] [nrf_app] [info] Handle Update NF Instance request, NF type AMF (HTTP version 2)
[2024-11-27 16:55:20.467] [nrf_app] [info] NF Heart-Beat procedure!
[2024-11-27 16:55:20.467] [nrf_app] [debug] Replace member nfStatus with new value REGISTERED
[2024-11-27 16:55:20.467] [nrf_app] [info] Updated the NF profile (profile ID 0bcee430-2277-4c8b-92ef-69423da0368d)
[2024-11-27 16:55:20.467] [nrf_app] [debug] NF update for Heartbeat, current time 1732726520467
[2024-11-27 16:55:20.467] [nrf_app] [debug] Set NF status to REGISTERED
[2024-11-27 16:55:30.492] [nrf_sbi] [info] Got a request to update an NF instance, Instance ID: 0bcee430-2277-4c8b-92ef-69423da0368d
[2024-11-27 16:55:30.498] [nrf_app] [info] Handle Update NF Instance request, NF type AMF (HTTP version 2)
[2024-11-27 16:55:30.498] [nrf_app] [info] NF Heart-Beat procedure!
[2024-11-27 16:55:30.498] [nrf_app] [debug] Replace member nfStatus with new value REGISTERED
[2024-11-27 16:55:30.498] [nrf_app] [info] Updated the NF profile (profile ID 0bcee430-2277-4c8b-92ef-69423da0368d)
[2024-11-27 16:55:30.498] [nrf_app] [debug] NF update for Heartbeat, current time 1732726530498
[2024-11-27 16:55:30.498] [nrf_app] [debug] Set NF status to REGISTERED
[2024-11-27 16:55:40.524] [nrf_sbi] [info] Got a request to update an NF instance, Instance ID: 0bcee430-2277-4c8b-92ef-69423da0368d
[2024-11-27 16:55:40.525] [nrf_app] [info] Handle Update NF Instance request, NF type AMF (HTTP version 2)
[2024-11-27 16:55:40.525] [nrf_app] [info] NF Heart-Beat procedure!
[2024-11-27 16:55:40.525] [nrf_app] [debug] Replace member nfStatus with new value REGISTERED
[2024-11-27 16:55:40.526] [nrf_app] [info] Updated the NF profile (profile ID 0bcee430-2277-4c8b-92ef-69423da0368d)
[2024-11-27 16:55:40.526] [nrf_app] [debug] NF update for Heartbeat, current time 1732726540526
[2024-11-27 16:55:40.526] [nrf_app] [debug] Set NF status to REGISTERED
root@new-server:~/virtualization_project/task3#

```

Next we try to apply oai-smf to microk8s. First of all, because there is a conf file for this service, we have to add it to microk8s using configmap:

```
root@new-server: ~/virtualization_project/task3
root@new-server:~/virtualization_project/task3# kubectl create configmap oai-smf-config --from-file=oai-smf.conf
configmap/oai-smf-config created
root@new-server:~/virtualization_project/task3#
```

Now we write a deployment and a service for oai-smf. Like oai-nrf, we write all the envs for it and replace the static IPs with service names of oai-nrf and oai-spgwu (not deployed yet) and we expose the port 80 for this service. Also we write a volume part for it to read from the conf file:

```
volumeMounts:
  - name: smf-config-volume
    mountPath: /openair-smf/bin/oai-smf.conf
    subPath: oai-smf.conf
volumes:
  - name: smf-config-volume
    configMap:
      name: oai-smf-config
```

```
root@new-server: ~/virtualization_project/task3
root@new-server:~/virtualization_project/task3# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
mysql-c78f5d65b-vv6cb               1/1     Running   0           18h
oai-amf-645cfb678b-gqkcs            1/1     Running   0           106m
oai-nrf-6db6c99cc4-zgskw            1/1     Running   0           5h35m
oai-smf-d8946bf58-wwjxv             1/1     Running   0           86m
root@new-server:~/virtualization_project/task3# kubectl get services
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP  PORT(S)          AGE
kubernetes    ClusterIP   10.152.183.1    <none>       443/TCP          2d6h
mysql         ClusterIP   10.152.183.223  <none>       3306/TCP         18h
oai-nrf       ClusterIP   10.152.183.117  <none>       8080/TCP         18h
oai-smf       ClusterIP   10.152.183.176  <none>       80/TCP,9090/TCP  88m
root@new-server:~/virtualization_project/task3#
```

Now we try to deploy oai-spgwu. Like before, we write a deployment and a service to expose port 80 and we write all their environment variables and replace IPs with suited service names. You can see the deployment and service in the github project.