International Institute of Informational Technology, Bangalore

Serverless platform - MOSIP Resident Services

IDA Core functions required for Resident Service

MT2021124
Shivang Agarwal

MT2021129
Shresth Pandey

MT2021125
Shivani Sharma

MT2021123
Shivam Kumar Rai

UNDER :
Prof B Thangaraju

Guided by:
Sanath Varambally

Introduction & Background

Modular Open-Source Identity Platform (MOSIP)

Mosip is an ambitious project of the International Institute of Information Technology Bangalore (IIIT-B). It is funded by the Bill and Melinda Gates Foundation, Omidyar Network and Tata Trust. The project aims at providing key components, modules, and functionalities in open source to help governments and agencies across the globe build a robust, scalable and secure platform for foundational identity.

Tasks Assigned and Performed:

1. **Technologies and Tools Learned and Used :**

   I. ***Knative*** *:- Knative is an open source project based on the Kubernetes platform for building, deploying and managing serverless*

   II. ***Kubernetes*** *:- Kubernetes is a portable, extensible, open source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available*

   III. ***Docker*** *:- Docker is a set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers. The service has both free and premium tiers. The software that hosts the containers is called Docker Engine*

   IV. ***Kind:*** *- kind is a tool for running local Kubernetes clusters using Docker container "nodes". Kind was primarily designed for testing Kubernetes itself, but may be used for local development or CI.*

   V. ***Serverless Architecture:*** *- A serverless architecture is a way to build and run applications and services without having to manage infrastructure.*

2. Going through the Reference Links :

a. https://www.youtube.com/watch?v=69OfdJ5BIzs
b. https://www.youtube.com/channel/UCxvEtyjmc3_KR45BOKMLJLA/playlists
c. https://knative.dev/docs/
d. https://www.youtube.com/watch?v=zx0_DIG6698
e. https://jamesdefabia.github.io/docs/user-guide/kubectl-overview/
f. https://developers.redhat.com/blog/2020/06/30/kourier-a-lightweight-knative-serving-i ngress#what_is_knative_
g. https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/
h. https://github.com/kubernetes/dashboard/blob/master/docs/user/access-control/creati ng-sample-user.md
i. https://serverlessworkflow.io/
j. https://kogito.kie.org/
k. https://www.youtube.com/watch?v=zx0_DIG6698
l. https://vocon-it.com/2018/12/10/kubernetes-4-persistent-volumes-hello-world/
m. https://dzone.com/refcardz/getting-started-with-quarkus-serverless-functions
n. https://dzone.com/articles/bind-a-cloud-event-to-knative
o. MOSIP Homepage (https://www.mosip.io/)

3. Installing KNative :

Reference Link Provided : https://knative.dev/docs/install/

We are installing Knative with the help of quickstart :

a. kind (Kubernetes in Docker) or minikube is being used so that we can run a local Kubernetes cluster with Docker container nodes

```
curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.12.0/kind-linux-
amd64 chmod +x ./kind
mv ./kind /some-dir-in-your-PATH/kind
```

b. The Kubernetes CLI (kubectl) is being used to run the commands against Kubernetes clusters. Kubectl can be used in order to deploy applications, manage as well as inspect cluster resources, and view logs.

c. Installing the Knative CLI :

i. Firstly, we download the binary for our system from the quickstart release page.

```
Download the latest release with the command:

curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

Validate the binary (optional)

curl -LO "https://dl.k8s.io/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl.sha256"
echo "$(cat kubectl.sha256)  kubectl" | sha256sum --check

Install kubectl

sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

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

kubectl version --client
```

ii. Now, we rename the file in order to remove the OS as well as the architecture information. For example, rename kn-quickstart-amd64 to kn-quickstart.

iii. Next, we make the plugin executable. For example, chmod +x kn-quickstart.

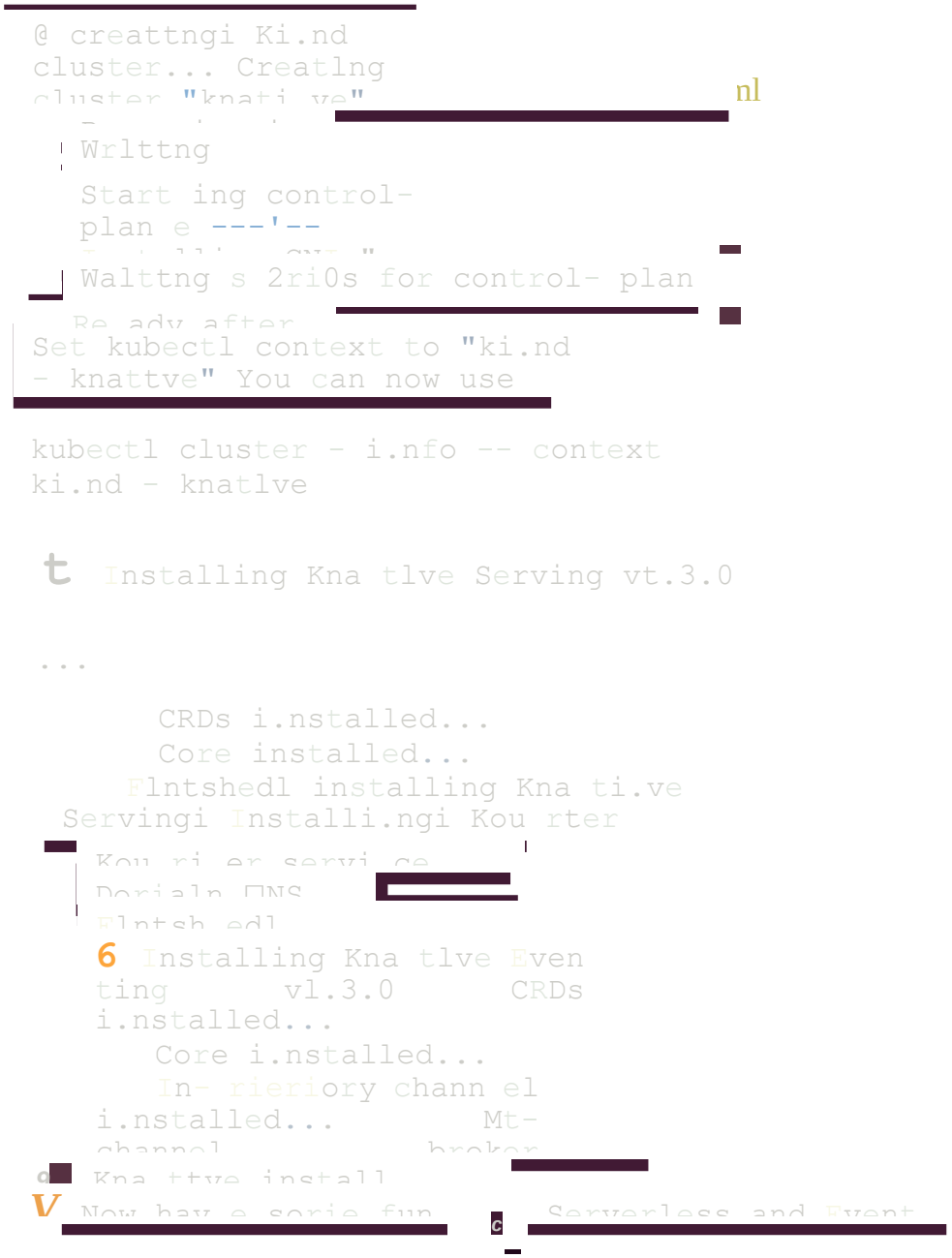
iv. After that, we move the executable binary file to a directory on your PATH, for example, in /usr/local/bin.

v. Verify that the plugin is working by running the command: kn quickstart --help

d. Now we run the Knative quickstart plugin :
The following functions are completed by the quickstart plugin:

1. **Checks if we have the selected Kubernetes instance installed**
2. **Creates a cluster called knative**
3. **Installs Knative Serving** with Kourier as the default networking layer, and sslip.io as the DNS
4. **Installs Knative Eventing** and creates an in-memory Broker and Channel implementation


shivang@shiv-Inspiron-5567:~$ sudo kn quickstart kind
[sudo] password for shivang:
Running Knative Quickstart using Kind
✅ Checking dependencies...
Kind version is: 0.12.0
☸ Creating Kind cluster...
Creating cluster "knative" ...
✓ Ensuring node image (kindest/node:v1.23.3) 🖼
✓ Preparing nodes 📦
✓ Writing configuration 📜
✓ Starting control-plane 🕹
✓ Installing CNI 🔌
✓ Installing StorageClass 💾
✓ Waiting ≤ 2m0s for control-plane = Ready ⏳
• Ready after 14s 💚
Set kubectl context to "kind-knative"
You can now use your cluster with:
kubectl cluster-info --context kind-knative
Have a nice day! 👋
🍿 Installing Knative Serving v1.3.0 ...
CRDs installed...
Core installed...
Finished installing Knative Serving
🕸 Installing Kourier networking layer v1.3.0 ...
Kourier installed...
Ingress patched...
Finished installing Kourier Networking layer
🕸 Configuring Kourier for Kind...
Kourier service installed...
Domain DNS set up...
Finished configuring Kourier
🔥 Installing Knative Eventing v1.3.0 ...
CRDs installed...
Core installed...
In-memory channel installed...
Mt-channel broker installed...
Example broker installed...
Finished installing Knative Eventing
🚀 Knative install took: 11m43s
🎉 Now have some fun with Serverless and Event Driven Apps!


@ creattngi Ki.nd
cluster... Creatlng
nl
Wrlttng
Start ing control-
plan e ---'--
Walttng s 2ri0s for control- plan
Set kubectl context to "ki.nd
- knattve" You can now use
kubectl cluster - i.nfo -- context
ki.nd - knatlve
t Installing Kna tlve Serving vt.3.0
...
CRDs i.nstalled...
Core installed...
Flntshedl installing Kna ti.ve
Servingi Installi.ngi Kou rter
6 Installing Kna tlve Even
ting vl.3.0 CRDs
i.nstalled...
Core i.nstalled...
In- rieriory chann el
i.nstalled... Mt-
V

4. Running a Sample Application (Hello World) :

a. Firstly, we are deploying the hello world service

```
Deploy the Service by running the command:

sudo kn service create hello --image
gcr.io/knative-samples/helloworld-go --port 8080 --env TARGET=World
```

```
shivang@shiv-Inspiron-5567:~$ sudo kn service create hello --image gcr.io/knative-samples/helloworld-go --port 8080 --env TARGET=World
[sudo] password for shivang:
Creating service 'hello' in namespace 'default':

  2.187s The Route is still working to reflect the latest desired specification.
  2.362s ...
  2.756s Configuration "hello" is waiting for a Revision to become ready.
129.827s ...
130.654s Ingress has not yet been reconciled.
132.475s Waiting for load balancer to be ready
133.759s Ready to serve.

Service 'hello' created to latest revision 'hello-00001' is available at URL:
http://hello.default.127.0.0.1.sslip.io
```

b. Now, viewing all the services

```
View a list of Knative services by running the command:

kn service list
```

```
shivang@shiv-Inspiron-5567:~$ sudo kn service list
[sudo] password for shivang:
NAME    URL                                   LATEST        AGE   CONDITIONS   READY   REASON
hello   http://hello.default.127.0.0.1.sslip.io   hello-00001   97m   3 OK / 3     True
```

c. We access our Knative Service by opening the previous URL in our browser or by running the command:

```
echo "Accessing URL $(kn service describe hello -o url)"
```

```
shivang@shiv-Inspiron-5567:~$ sudo echo "Accessing URL $(sudo kn service describe hello -o url)"
Accessing URL http://hello.default.127.0.0.1.sslip.io
```

```
curl "$(kn service describe hello -o url)"
```

```
shivang@shiv-Inspiron-5567:~$ sudo curl "$(sudo kn service describe hello -o url)"
Hello World!
```

5. Install Kubernetes Dashboard using Kubectl :

a. Deploying the Kubernetes dashboard using Kubectl:

```
sudo kubectl apply -f
https://raw.githubusercontent.com/kubernetes/dashboard/v2.3.1/
aio/deploy/recommended.yaml
```

```
shivang@shiv-Inspiron-5567:~$ sudo kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.3.1/aio/deploy/recommended.yaml
[sudo] password for shivang:
namespace/kubernetes-dashboard created
serviceaccount/kubernetes-dashboard created
service/kubernetes-dashboard created
secret/kubernetes-dashboard-certs created
secret/kubernetes-dashboard-csrf created
secret/kubernetes-dashboard-key-holder created
configmap/kubernetes-dashboard-settings created
role.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrole.rbac.authorization.k8s.io/kubernetes-dashboard created
rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
deployment.apps/kubernetes-dashboard created
service/dashboard-metrics-scraper created
Warning: spec.template.metadata.annotations[seccomp.security.alpha.kubernetes.io/pod]: deprecated since v1.19, non-functional in v1.25+; use the "seccompProfile" field instead
deployment.apps/dashboard-metrics-scraper created
```

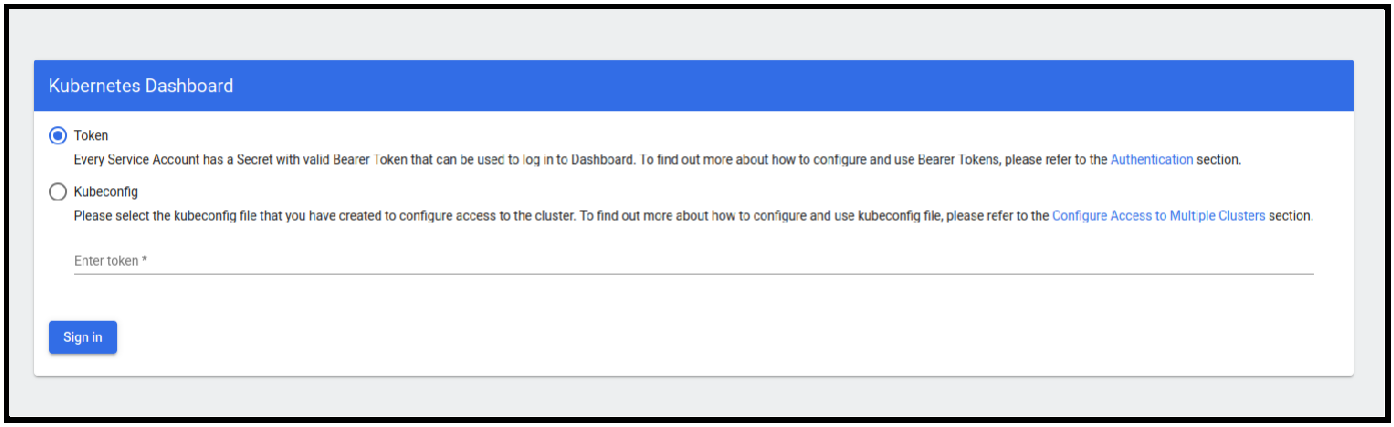
b. Now, accessing Kubernetes Dashboard using Kubectl :

```
sudo kubectl proxy
```

```
shivang@shiv-Inspiron-5567:~$ sudo kubectl proxy
Starting to serve on 127.0.0.1:8001
```

c. Accessing Kubernetes Dashboard using following url

http://localhost:8001/api/v1/namespaces/kubernetes-dashboard
/services/https:kubernetes-dashboard:/proxy/



d. Kubernetes Dashboard Authentication :

i. Firstly, we create a Service Account :

```
sudo kubectl create serviceaccount dashboard-admin-sa
```

```
shivang@shiv-Inspiron-5567:~$ sudo kubectl create serviceaccount dashboard-admin-sa
[sudo] password for shivang:
serviceaccount/dashboard-admin-sa created
```

ii. Binding the dashboard-admin-service-account service account to the cluster-admin role and get secrets

```
sudo kubectl create clusterrolebinding dashboard-admin-
sa
--clusterrole=cluster-admin
--serviceaccount=default:dashboard-admin-

sa kubectl get secrets
```

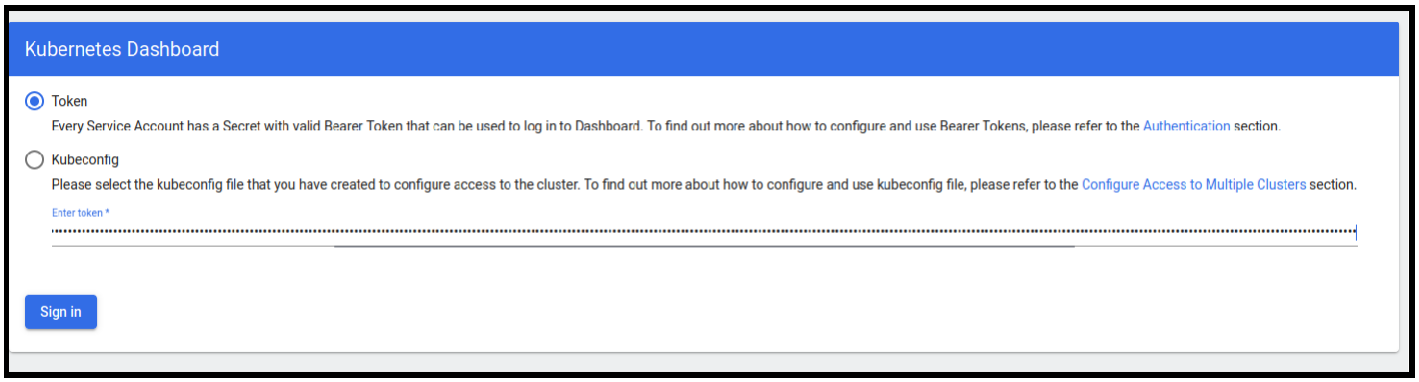
iii. Using kubectl describe to get the access token:

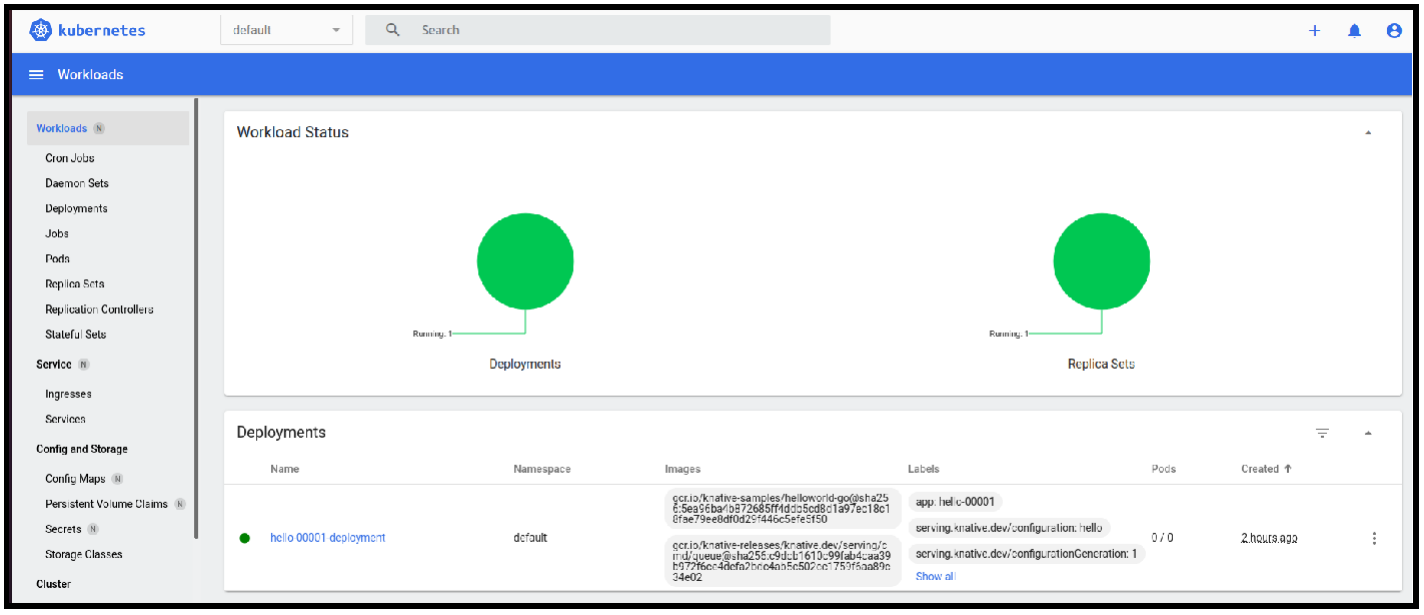
```
sudo kubectl describe secret
dashboard-admin-sa-token-lnjc8
```

```
shivang@shiv-Inspiron-5567:~$ sudo kubectl create clusterrolebinding dashboard-admin-sa --clusterrole=cluster-admin --serviceaccount=default:dashboard-admin-sa
clusterrolebinding.rbac.authorization.k8s.io/dashboard-admin-sa created
shivang@shiv-Inspiron-5567:~$ kubectl get secrets
error: error loading config file "/home/swaraj/.kube/config": read /home/swaraj/.kube/config: is a directory
shivang@shiv-Inspiron-5567:~$ sudo kubectl get secrets
NAME                           TYPE                                  DATA   AGE
dashboard-admin-sa-token-lnjc8   kubernetes.io/service-account-token   3      6m31s
default-token-9ljmz            kubernetes.io/service-account-token   3      170m
shivang@shiv-Inspiron-5567:~$
```

```
shivang@shiv-Inspiron-5567:~$ sudo kubectl describe secret dashboard-admin-sa-token-lnjc8
Name:         dashboard-admin-sa-token-lnjc8
Namespace:    default
Labels:       <none>
Annotations:  kubernetes.io/service-account.name: dashboard-admin-sa
              kubernetes.io/service-account.uid: 3bdfb554-c7de-4004-bdf9-079c6cdaa2ce

Type:  kubernetes.io/service-account-token

Data
====
ca.crt:     1099 bytes
namespace:  7 bytes
token:      eyJhbGciOiJSUzI1NiIsImtpZCI6Ikx0Qy1ibTVSSnVBX3RSdlY3ZU1JSVpPVk95ZnA3bmVPUUx6bFB1WEpwN1UifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW5
OiJkZWZhdWx0Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZWNyZXQubmFtZSI6ImRhc2hib2FyZC1hZG1pbi1zYS10b2tlbi1sbmpjOCIsImt1YmVybmV0ZXMuaW8vc2VydmljZW
taW4tc2EiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC51aWQiOiIzYmRmYjU1NC1jN2RlLTQwMDQtYmRmOS0wNzljNmNkYWEyY2UiLCJzdWIiOiJzeXN0Z
EifQ.z1bWv8B7xFT-FTLqFR6eChrBc6vyU9LBKN6ScqzdMJFtMTiCLdhXciebmy5QGMl_zNMNgJlve2J9h2SPrhZrWlheUM9vunLSu7FX2RRJ3QBXD9xqNq-8R9SVuQlmCm0268AeHDIiF_IFT
3yll4Nur38YBm3VKJAc23EwPFbjFOIWPnAbLaOOt8NZ2MDCngIWRBn2Hf7FUfGKZGdFopc9mnWEITXQQ8BoCE1U1Kfcs0DOoChl2wCgpLWqLYqy5p68F4W0W3nrIQ5dtzqMIN5u-isE84wdw
shivang@shiv-Inspiron-5567:~$
```

iv. Enter the token in the login field :





6. Kernel Service Setup :

```
Git repo clone (branch: 1.1.5.4)

 git clone https://github.com/mosip/commons.git : DDL
 git clone https://github.com/mosip/mosip-data.git :
 DML

 cd commons

 mvn clean install -DskipTests
Changed mosip.auth.adapter.impl.basepackage to
io.mosip.kernel.auth.defaultadapter in
AuditManagerBootApplication
```

Added spring.h2.console.settings.web-allow-others=true in commons/kernel/kernel-auditmanager- service/target/classes/application-local.properties

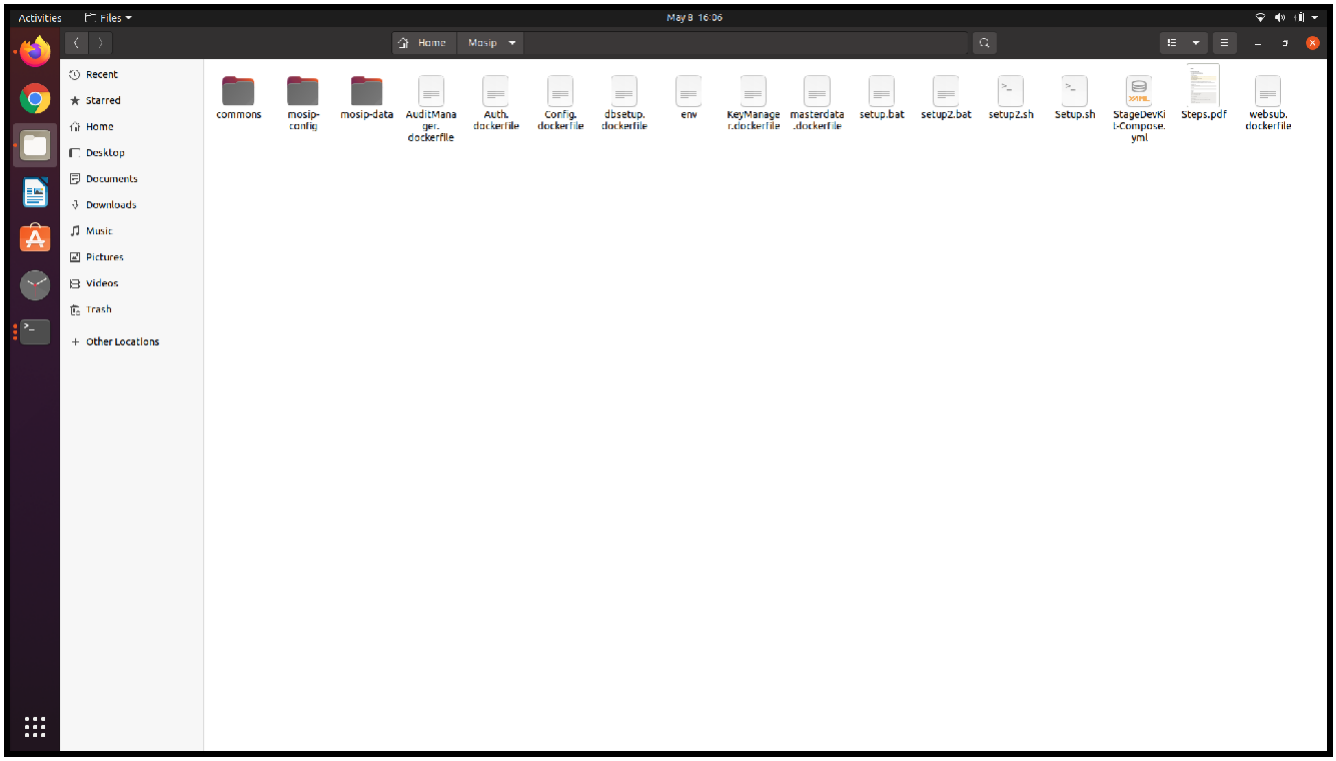
mvn clean install -DskipTests

4. java -jar -Dloader.path=kernel\kernel-auth-adapter\target\kernel-auth-adapter- 1.2.0-rc1.jar -jar -Dspring.profiles.active=local kernel\kernel-auth-service\target\kernel-auth-service-1.2.0-rc1.jar

5. wget https://repo1.maven.org/maven2/io/mosip/kernel/kernel-config-server/ 1.1.2/kernel-config-server-1.1.2.jar -P mosip-config

6. docker volume create --name=mosip_config

7. docker-compose -f StageDevKit-Compose.yml up



StageDevKit-Compose :

```
version: '3.8'
volumes:
    shared-workspace:
        name: "devkit-distributed-file-system" driver: local
    mosip_config:
        external: true name:
        mosip_config
services:
    mosip_auth_service: build:
            context: .
            dockerfile: Auth.dockerfile image:
        kernel/auth:v1 container_name:
        mosip_auth_service ports:
            - 8091:8091
        volumes:
            - shared-workspace:/opt/workspace
    mosip_config_service:
        build:
            context: .
            dockerfile: Config.dockerfile image:
        kernel/config:v1 container_name:
        mosip_config_service ports:
            - 51000:51000
        volumes:
            – shared-workspace:/opt/workspace
            – mosip_config:/config
        environment:
            – AUTH_SERVICE=http://mosip_auth_service:8091
        depends_on:
            – mosip_auth_service
    mosip_audit_service:
        build:
            context: .
            dockerfile: AuditManager.dockerfile
```

```
        image: kernel/auditmanager:v1
        container_name:
        mosip_audit_service ports:
            - 8081:8081
        volumes:
                        shared-
              workspace:/opt/workspace
              environment:
                        AUTH_SERVICE=http://mosip_auth_service:8091
                        CONFIG_SERVICE=mosip_config_
              service:51000 depends_on:
                        mosip
              _auth_service
              postgres:
        image:
        debezium/postgres
        container_name:
        postgres ports:
            - 5432:5432
        volumes:
                        shared-
              workspace:/opt/workspace
              environment:
                        POSTGRES_PASSWORD=root
                        PGDATA=/data/pgdata
                        POSTGRES_DB=kernel111
```

.sh setup

```
#!/bin/sh
repository="https://gi
thub.com/mosip/commons
.git"
local="/home/shivani/Desktop/IIITB/S
emester/2ndSem/mosip/commons" git
clone -b 1.1.5.4 "$repository"
"$local"
cd commons
mvn clean install -DskipTests
loader path="kernel\kernel-auth-
```

```
wget
https://repo1.maven.org/maven2/io/mosip/kernel/kernel-config-
server/1.1.2/ kernel-config-server-1.1.2.jar -P mosip-config
docker volume create --
name=mosip_config # build docker
images
docker-compose -f StageDevKit-Compose.yml up
```

Setup2.sh

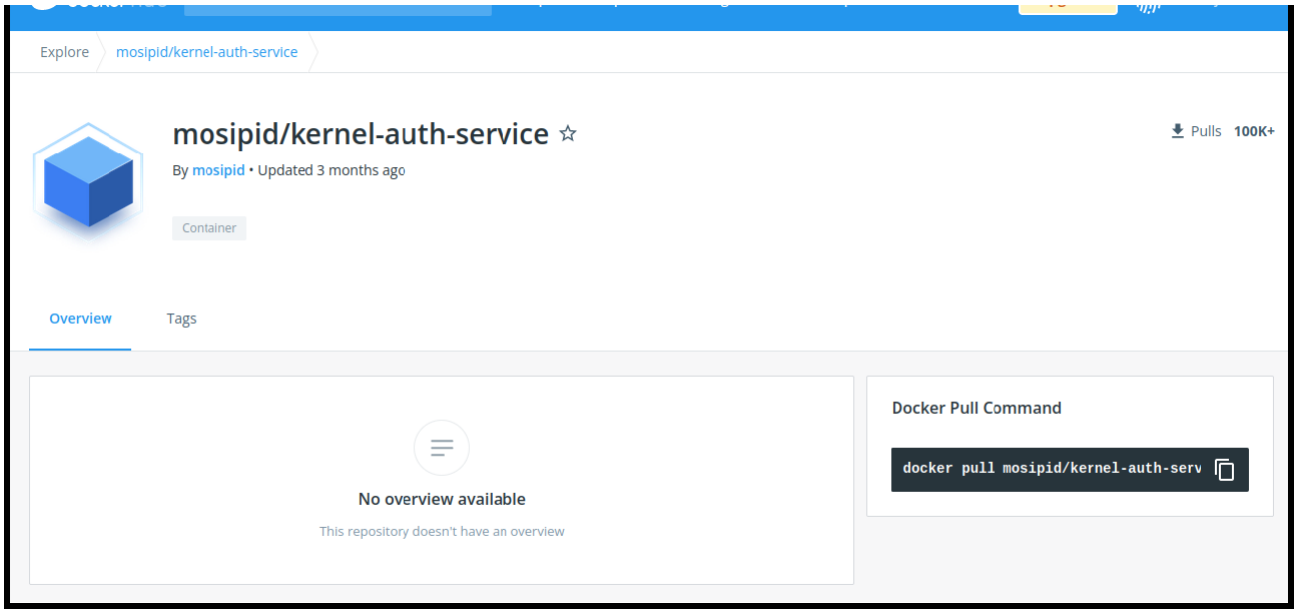
```
VERSION=1.2.0-rc1
#git clone https://github.com/mosip/commons.git
#cd commons


#git checkout $VERSION
#mvn clean install -DskipTests
#java -jar
-Dloader.path=kernel/kernel-auth-adapter/target/kernel-auth-adapter-1.2.0-
rc1.jar -jar -Dspring.profiles.active=local
kernel/kernel-auth-service/target/kernel-auth-service-1.2.0-rc1.jar

#cd ..
#wget
https://repo1.maven.org/maven2/io/mosip/kernel/kernel-config-server/1.1.2/
kernel-config-server-1.1.2.jar -P mosip-config

echo $VERSION
docker build -f Auth.dockerfile --build-arg version=$VERSION -t
kernel/auth:v1 .
docker build -f AuditManager.dockerfile --build-arg version=$VERSION -t
kernel/auditmanager:v1 .
docker build -f Config.dockerfile --build-arg version=$VERSION -t
kernel/config:v1 .
```

```
#docker build -f test.dockerfile
docker volume create --

name=mosip_config docker rm -f

$(docker ps -a -q)
```

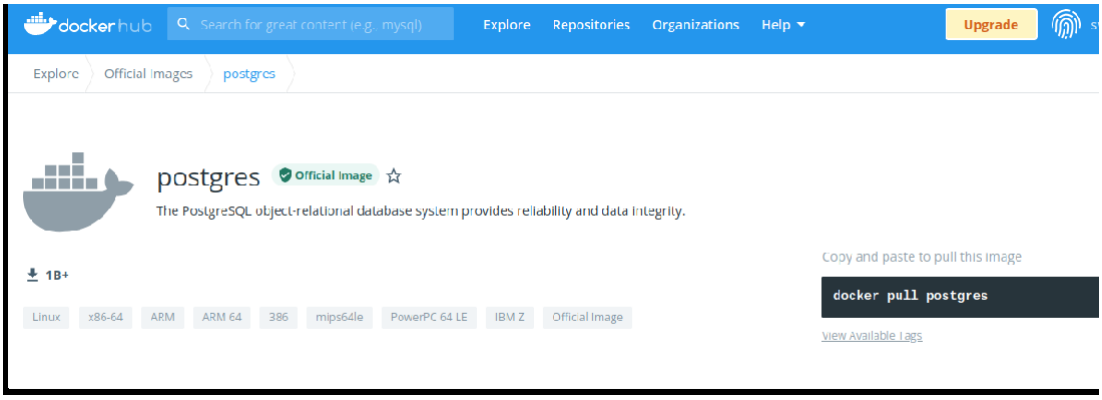
```
shivang@shiv-Inspiron-5567:~/Mosip$ tree -L 1
.
├── AuditManager.dockerfile
├── Auth.dockerfile
├── commons
├── Config.dockerfile
├── dbsetup.dockerfile
├── env
├── KeyManager.dockerfile
├── masterdata.dockerfile
├── mosip-config
├── mosip-data
├── setup2.bat
├── setup2.sh
├── setup.bat
├── Setup.sh
├── StageDevKit-Compose.yml
├── Steps.pdf
└── websub.dockerfile

3 directories, 14 files
```

7. Running Kernel Auth Service :

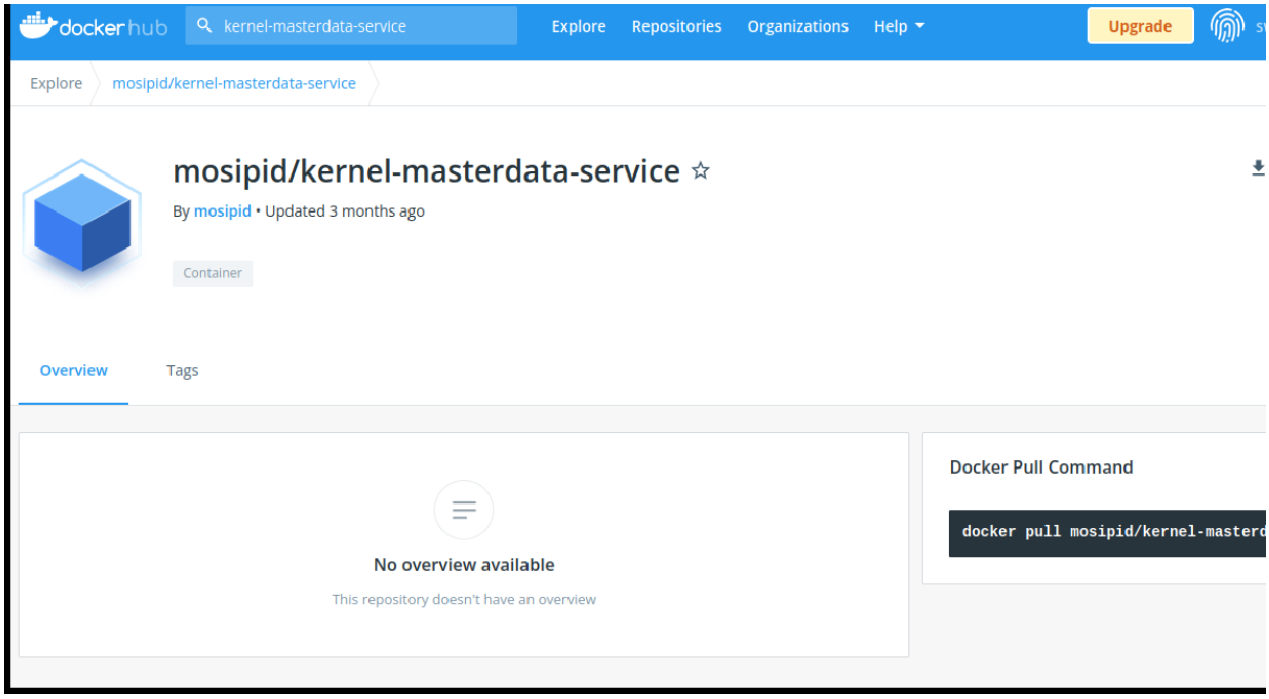
a. We Pull and run ‘kernel-auth-service’ using KNative Serving.



b. Pulling and running postgres:



c. Pulling and running ‘kernel-master data-service’ using KNative Serving.



Yaml File

```
---
kind: Pod
metadata:
 labels:
    app: demo.40
 namespace: default
spec:
 containers:
   image: mosipid/kernel-auth-
  service name: kernel-auth-service
   ports:
       containerPort:
    8091 name:
    portname.0
    protocol: tcp
  volumeMounts:
       mountPath:
    /opt/workspace name:
    pvo.0
 terminationGracePeriodSeconds: 0
 volumes:
```

```
     claimName: claimname.0

---
kind:
Pod
metadat
a:
 labels:
    app: demo.91
 namespace:
 default
spec:
 containers:
    image: mosipid/kernel-masterdata-
   service name: kernel-masterdata-
   service
   ports:
        containerPort:
     8092 name:
     portname.0
     protocol: tcp
   volumeMounts:
        mountPath:
     /opt/workspace name:
     pvo.0
```

Docker Compose File:

```
version: '3.8'
volumes:
   shared-workspace:
       name: "devkit-distributed-file-system"
       driver: local
   mosip_config:
       external: true
       name: mosip_config
services:
   kernel-auth-service:
       image: mosipid/kernel-auth-service
       container_name: kernel-auth-service
```

```
    ports:
            - 8091:8091
    volumes:
```

```
postgres:
        image:
    debezium/postgres
    container_name:
    postgres ports:
                - 5432:5432
        volumes:
                  shared-
          workspace:/opt/workspace
          environment:
```

```
kernel-masterdata-service:
        image: mosipid/kernel-masterdata-
    service container_name: kernel-
    masterdata-service ports:
                - 8092:8092
        volumes:
                - shared-
        workspace:/opt/workspace
```

```
Attaching to kernel-masterdata-service, postgres, kernel-auth-service
kernel-auth-service          | Exception in thread "main" java.util.zip.ZipException: zip file is empty
kernel-auth-service          |  at java.base/java.util.zip.ZipFile$Source.zerror(ZipFile.java:1607)
kernel-auth-service          |  at java.base/java.util.zip.ZipFile$Source.findEND(ZipFile.java:1410)
kernel-auth-service          |  at java.base/java.util.zip.ZipFile$Source.initCEN(ZipFile.java:1504)
kernel-auth-service          |  at java.base/java.util.zip.ZipFile$Source.<init>(ZipFile.java:1308)
kernel-auth-service          |  at java.base/java.util.zip.ZipFile$Source.get(ZipFile.java:1271)
kernel-auth-service          |  at java.base/java.util.zip.ZipFile$CleanableResource.<init>(ZipFile.java:831)
kernel-auth-service          |  at java.base/java.util.zip.ZipFile$CleanableResource$FinalizableResource.<init>(ZipFile.java:857)
kernel-auth-service          |  at java.base/java.util.zip.ZipFile$CleanableResource.get(ZipFile.java:846)
kernel-auth-service          |  at java.base/java.util.zip.ZipFile.<init>(ZipFile.java:248)
kernel-auth-service          |  at java.base/java.util.zip.ZipFile.<init>(ZipFile.java:177)
kernel-auth-service          |  at java.base/java.util.jar.JarFile.<init>(JarFile.java:350)
kernel-auth-service          |  at java.base/java.util.jar.JarFile.<init>(JarFile.java:321)
kernel-auth-service          |  at java.base/java.util.jar.JarFile.<init>(JarFile.java:287)
kernel-auth-service          |  at org.springframework.boot.loader.jar.JarFile.<init>(JarFile.java:115)
kernel-auth-service          |  at org.springframework.boot.loader.jar.JarFile.<init>(JarFile.java:109)
kernel-auth-service          |  at org.springframework.boot.loader.jar.JarFile.<init>(JarFile.java:95)
kernel-auth-service          |  at org.springframework.boot.loader.jar.JarFile.<init>(JarFile.java:86)
kernel-auth-service          |  at org.springframework.boot.loader.archive.JarFileArchive.<init>(JarFileArchive.java:60)
kernel-auth-service          |  at org.springframework.boot.loader.archive.JarFileArchive.<init>(JarFileArchive.java:56)
kernel-auth-service          |  at org.springframework.boot.loader.archive.ExplodedArchive.getNestedArchive(ExplodedArchive.java:119)
kernel-auth-service          |  at org.springframework.boot.loader.archive.ExplodedArchive.getNestedArchives(ExplodedArchive.java:106)
kernel-auth-service          |  at org.springframework.boot.loader.PropertiesLauncher.getClassPathArchives(PropertiesLauncher.java:449)
kernel-auth-service          |  at org.springframework.boot.loader.Launcher.launch(Launcher.java:49)
kernel-auth-service          |  at org.springframework.boot.loader.PropertiesLauncher.main(PropertiesLauncher.java:593)
```

```
postgres                     | syncing data to disk ... ok
postgres                     |
postgres                     | Success. You can now start the database server using:
postgres                     |
postgres                     |     pg_ctl -D /data/pgdata -l logfile start
postgres                     |
postgres                     |
postgres                     | WARNING: enabling "trust" authentication for local connections
postgres                     | You can change this by editing pg_hba.conf or using the option -A, or
postgres                     | --auth-local and --auth-host, the next time you run initdb.
postgres                     | waiting for server to start....LOG:  database system was shut down at 2022-05-08 11:23:21 GMT
postgres                     | LOG:  MultiXact member wraparound protections are now enabled
postgres                     | LOG:  autovacuum launcher started
postgres                     | LOG:  database system is ready to accept connections
postgres                     |  done
postgres                     | server started
postgres                     | CREATE DATABASE
postgres                     |
postgres                     |
postgres                     | /usr/local/bin/docker-entrypoint.sh: sourcing /docker-entrypoint-initdb.d/init-permissions.sh
postgres                     |
postgres                     | LOG:  received fast shutdown request
postgres                     | waiting for server to shut down...LOG:  aborting any active transactions
postgres                     | LOG:  autovacuum launcher shutting down
postgres                     | .LOG:  shutting down
postgres                     | LOG:  database system is shut down
postgres                     |  done
postgres                     | server stopped
postgres                     |
postgres                     | PostgreSQL init process complete; ready for start up.
postgres                     |
postgres                     | LOG:  database system was shut down at 2022-05-08 11:24:19 GMT
postgres                     | LOG:  MultiXact member wraparound protections are now enabled
postgres                     | LOG:  autovacuum launcher started
postgres                     | LOG:  database system is ready to accept connections
```