***International Institute of Information Technology, Bangalore***

# MOSIP

# SERVERLESS MOSIP PLATFORM


**Team Members :**

| **MT2021133** | **MT2021128** | **MT2021133** | **MT2021111** |
|---|---|---|---|
| Shwetank Singh | Shree Garodia | Vishwajeet Deulkar | Rushikesh Kale |

| **MT2021147** | **MT2021018** |
|---|---|
| Tushar Sharma | Anmol Bisht |

**Under** **Guided by**

**Prof. B. Thangaraju** **Sanath Varammbally**

# Table of Contents

**6. References**

# 1. Serverless

## 1.1 What is serverless

Serverless does not mean that there are no servers, it just means that the developers don't have to worry about them.
In Serverless Computing, the machine resources are allocated on a demand basis. Serverless offloads all management responsibility for backend cloud infrastructure and operations tasks - provisioning, scheduling, scaling, etc. - to the cloud provider. Pricing is based on the actual amount of resources consumed. You are not paying for idle capacity.

## 1.2 Why serverless

By using serverless architecture, developers can focus on their core product, instead of worrying about managing and operating servers. This reduced overhead lets developers focus more on the front-end and business logic of the product. Thus this leads developers to make better quality products that scale well and that are reliable.

## 1.3 Technology Stack used

I. **Docker:** It is a set of platforms as service products that use OS-level virtualization to deliver software in packages called containers
II. **Docker Compose**: Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services.
III. **Kubernetes**: Kubernetes is an open-source container orchestration system

for automating software deployment, scaling, and management.

IV. **Kind:** A kind is a tool for running local Kubernetes clusters using Docker container "nodes". kind was primarily designed for testing Kubernetes itself.

V. **Knative:** Knative is an Open-Source Enterprise-level solution to build Serverless and Event-Driven Applications. Serverless Containers in Kubernetes environments.

## 1.4 Installation procedure

Step 1: Installation of curl: Used in CLI to transfer data.

```
$ sudo apt install curl
```

Step 2: Installation of Kind

```
$ curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.11.1/kind-linux-amd64
$ chmod +x ./kind
$ mv ./kind /usr/local/bin/kind
$ mv ./kind ./kinddir
```

Step 3: Installation of Kubernetes CLI

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"
```

Download the kubectl checksum file:

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

Validate the kubectl binary against the checksum file:

```
$ echo "$(<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
```

### Step 4 : Installation knative binary

Download the binary for your system from the kn release page about 63.5mb
From https://github.com/knative/client/releases install version asset v0.26.0 kn-linux-amd64

Rename the binary to kn and make it executable by running the commands:

```
$ mv <path-to-binary-file> kn
$ chmod +x kn
$ mv kn /usr/local/bin
$ kn version
```

### Step 5 : Installation of Knative “Quickstart” environment

```
$ kn quickstart kind
```

## 1.5 Output :

```
sharan@sharan-dell:~$ sudo kn quickstart kind
[sudo] password for sharan:
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 13s 💚
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: 9m38s
🎉 Now have some fun with Serverless and Event Driven Apps!
sharan@sharan-dell:~$
```

# 2. Running Hello World

## 2.1 Deploy Hello World Service

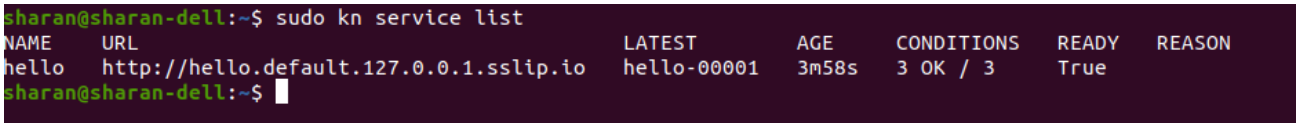
```
$ sudo kn service create hello --image gcr.io/knative-samples/helloworld-go
--port 8080 --env TARGET=World
```

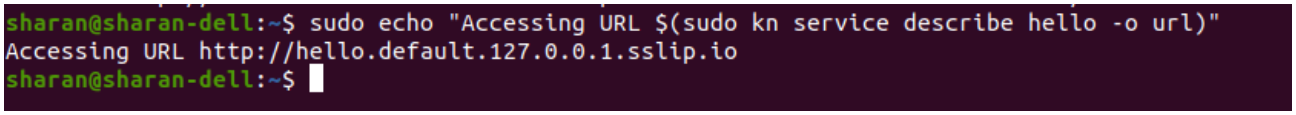
3

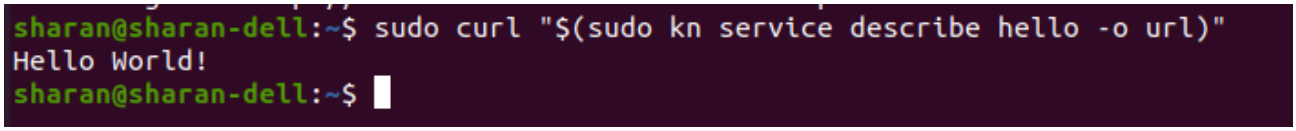
## 2.2 Access Hello World Service

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

## 2.3 Output

# 3. Using Kubernetes Dashboard

## 3.1 Deploying Kubernetes Dashboard

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

3

## 3.2 Accessing Kubernetes Dashboard

```
$ sudo kubectl proxy

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

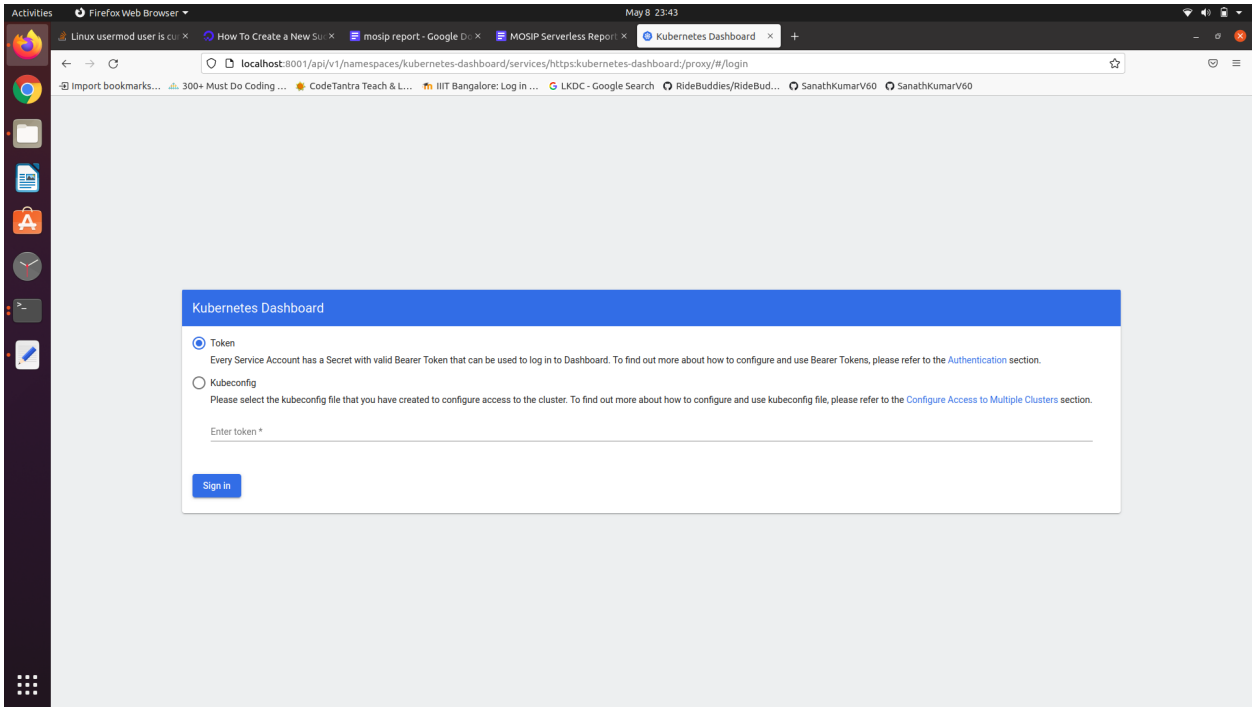
## 3.3 Kubernetes Dashboard Authentication

```
$ sudo kubectl create serviceaccount dashboard-admin-sa
$ sudo kubectl create clusterrolebinding dashboard-admin-sa
--clusterrole=cluster-admin --serviceaccount=default:dashboard-admin-sa
$ kubectl get secrets
$ sudo kubectl describe secret dashboard-admin-sa-token-wjbwz
```

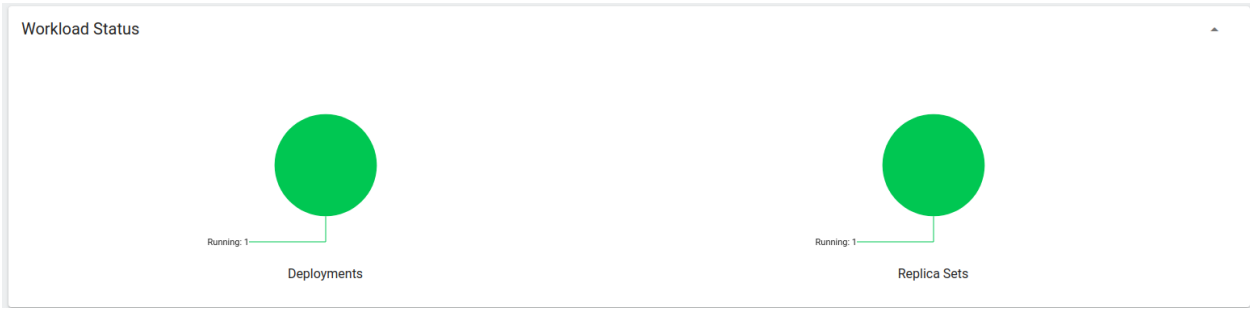
## 3.4 Output

```
shwetank@shwetank-hp-15:~$ sudo kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.3.1/aio/deploy/recommended.yaml
[sudo] password for shwetank:
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
deployment.apps/dashboard-metrics-scraper created
shwetank@shwetank-hp-15:~$
```

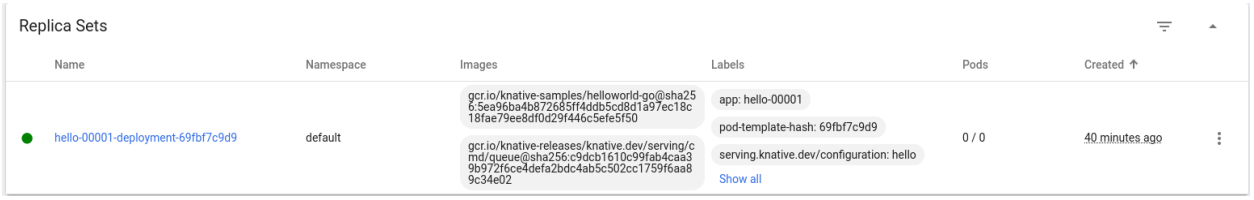
```
shwetank@shwetank-hp-15:~$ sudo kubectl proxy
Starting to serve on 127.0.0.1:8001
```



```
shwetank@shwetank-hp-15:~$ sudo kubectl describe secret dashboard-admin-sa-token-wjbwz
Name:         dashboard-admin-sa-token-wjbwz
Namespace:    default
Labels:       <none>
Annotations:  kubernetes.io/service-account.name: dashboard-admin-sa
              kubernetes.io/service-account.uid: d1583d61-af27-474f-aad8-10ed13119d41

Type:  kubernetes.io/service-account-token

Data
====
namespace:  7 bytes
token:      eyJhbGciOiJSUzI1NiIsImtpZCI6IkVFdVJOaHptT1hBVmI2WFhxZlFRT2RaQjJhRFEtVDBhaEVRWkVVMFJXVkkifQ.eyJpc3M
OiJkZWZhdWx0Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZWNyZXQubmFtZSI6ImRhc2hib2FyZC1hZG1pbi1zYS10b2tlbi13am
taW4tc2EiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC51aWQiOiJkMTU4M2Q2MS1hZjI3LTQ3NGYtYWFkO
EifQ.xOkRZrtraKc3khCrA2G8kLjwvjvBpb5hBf0PgU26jkr78ZDNTNQXWxOAOJZm7hBc2mybIiLlpaFRDNV9xg6ArktUcHUUYPotMjILc2OFo
yoX7EgsbC4foBKo_cKGMfXq2r4F_3ytIOYgtAiq16pOCs-Bgf3u8M22tVhyqModpXiCDjS2k-K_2fsW_S1rLqs7FFwZTrDleuGpuJk_VJHh7Vh
ca.crt:     1099 bytes
shwetank@shwetank-hp-15:~$
```







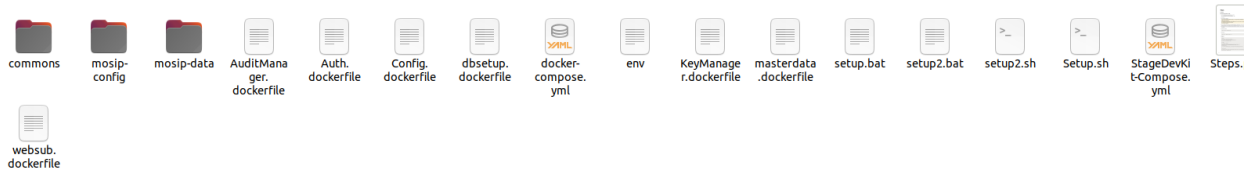

# 4. Kernel Service Setup

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

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

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

docker volume create --name=mosip_config

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

Outputs :




```
.
├── 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
```

**StageDevKit-Compose :**

```yaml
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
```

**Setup.sh**

```
#!/bin/sh
repository="https://github.com/mosip/commons.git"
local="/home/nupur/Desktop/IIITB/Semester/3rdSem/mosip/commons"
git clone -b 1.1.5.4 "$repository" "$local"
cd commons
mvn clean install -DskipTests
loader_path="kernel\kernel-auth-adapter\target\kernel-auth-adapter-1.2.0
-rc1.jar"
path="kernel\kernel-auth-service\target\kernel-auth-service-1.2.0-rc1.ja
r"
spring_profile="local"
java -jar -Dloader.path="$loader_path" -jar
-Dspring.profiles.active="$spring_profile" "$path"
cd ..
# get the config jar
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)

docker-compose -f StageDevKit-Compose.yml -d up
```

# 5. Running Kernel Auth Service

### 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:
 - name: pvo.0
   persistentVolumeClaim:
     claimName: claimname.0

---
kind: Pod
metadata:
 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
 terminationGracePeriodSeconds: 0
 volumes:
 - name: pvo.0
   persistentVolumeClaim:
     claimName: claimname.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:
           - shared-workspace:/opt/workspace

   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

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

Output :

```
Pulling kernel-auth-service       ... done
Pulling postgres                  ... done
Pulling kernel-masterdata-service ... done
```

```
Starting kernel-auth-service       ... done
Starting kernel-masterdata-service ... done
Starting postgres                  ... done
Attaching to kernel-masterdata-service, kernel-auth-service, postgres
postgres                    |
postgres                    | PostgreSQL Database directory appears to contain
a database; Skipping initialization
postgres                    |
```

```
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>(Zi
```

```
kernel-masterdata-service   | Exception in thread "main" java.util.zip.ZipException: zip file is empty
kernel-masterdata-service   |  at java.base/java.util.zip.ZipFile$Source.zerror(ZipFile.java:1607)
kernel-masterdata-service   |  at java.base/java.util.zip.ZipFile$Source.findEND(ZipFile.java:1410)
kernel-masterdata-service   |  at java.base/java.util.zip.ZipFile$Source.initCEN(ZipFile.java:1504)
kernel-masterdata-service   |  at java.base/java.util.zip.ZipFile$Source.<init>(ZipFile.java:1308)
kernel-masterdata-service   |  at java.base/java.util.zip.ZipFile$Source.get(ZipFile.java:1271)
kernel-masterdata-service   |  at java.base/java.util.zip.ZipFile$CleanableResource.<init>(ZipFile.java:831)
kernel-masterdata-service   |  at java.base/java.util.zip.ZipFile$CleanableResource$FinalizableResource.<init>(ZipFile.java:857)
kernel-masterdata-service   |  at java.base/java.util.zip.ZipFile$CleanableResource.get(ZipFile.java:846)
kernel-masterdata-service   |  at java.base/java.util.zip.ZipFile.<init>(ZipFile.java:248)
kernel-masterdata-service   |  at java.base/java.util.zip.ZipFile.<init>(ZipFile.java:177)
```

# 5. References


1) https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/
2) https://knative.dev/docs/getting-started/
3) https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/