**Deploy Spring-Boot application-Mysql with Kubernetes**

**Pre-requisites:**

* Install Git
* Install Maven
* Install docker
* EKS Cluster

Get Source Code from github:

<https://github.com/Naresh240/employee-jdbc.git>

Build artifact by using below command:

mvn clean install

Create docker image:

docker build -t naresh240/employee-jdbc .

Docker login:

We need to login before push image to docker hub

docker login

Tag docker image:

docker tag employee-jdbc naresh240/employee-jdbc

push docker image to docker hub:

docker push naresh240/employee-jdbc

**Deploying Mysql on Kubernetes using PersistentVolume and Secrets:**

Following is the Kubernetes manifest for MySQL deployment. I’ve added comments alongside each configuration to make sure that its usage is clear to you.

**Creating the secrets:**

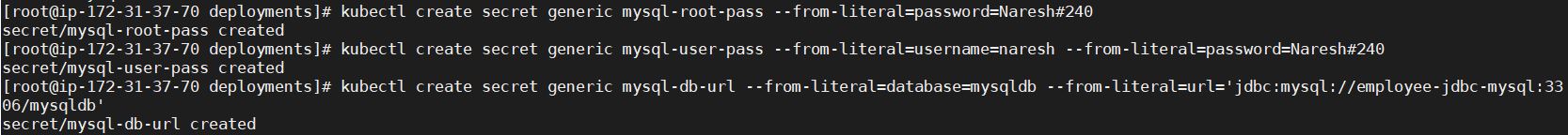
You can create secrets manually from a literal or file using the kubectl create secret command, or you can create them from a generator using Kustomize.

In this article, we’re gonna create the secrets manually:

kubectl create secret generic mysql-root-pass --from-literal=password=Naresh#240

kubectl create secret generic mysql-user-pass --from-literal=username=naresh --from-literal=password=Naresh#240

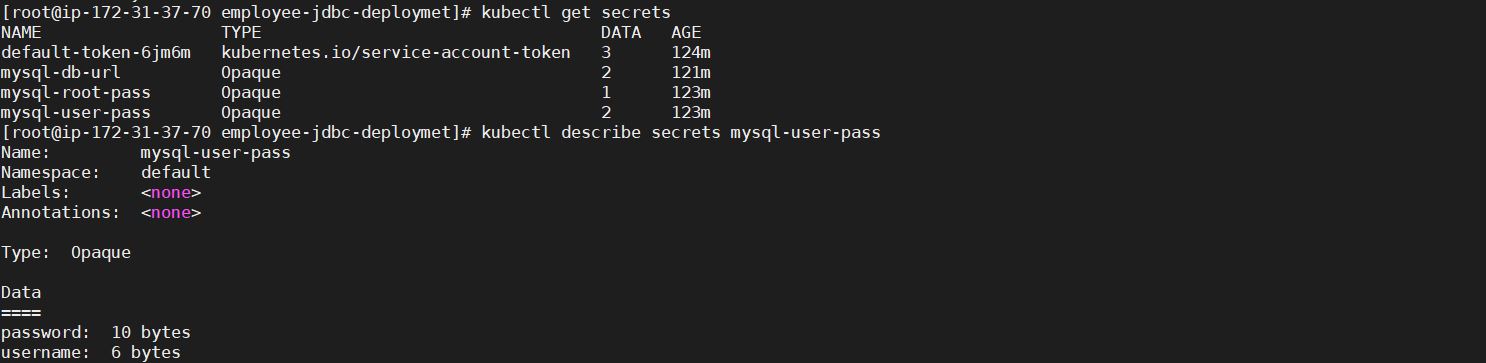
kubectl create secret generic mysql-db-url --from-literal=database=mysqldb --from-literal=url='jdbc:mysql://employee-jdbc-mysql:3306/mysqldb'



You can get the secrets like this –

kubectl get secrets

kubectl describe secrets mysql-user-pass



**Make directory:**

mkdir deployments

cd deployments

Create PersistentVolume with the name of **mysql-pv.yml**:

vi mysql-pv.yml

apiVersion: v1

kind: PersistentVolume # Create a PersistentVolume

metadata:

name: mysql-pv

labels:

type: local

spec:

storageClassName: standard # Storage class. A PV Claim requesting the same storageClass can be bound to this volume.

capacity:

storage: 250Mi

accessModes:

- ReadWriteOnce

hostPath: # hostPath PersistentVolume is used for development and testing. It uses a file/directory on the Node to emulate network-attached storage

path: "/mnt/data"

persistentVolumeReclaimPolicy: Retain # Retain the PersistentVolume even after PersistentVolumeClaim is deleted. The volume is considered “released”. But it is not yet available for another claim because the previous claimant’s data remains on the volume.

Create PersistentVolume Claim with the name of **mysql-pv-claim.yml**:

vi mysql-pv-claim.yml

---

apiVersion: v1

kind: PersistentVolumeClaim # Create a PersistentVolumeClaim to request a PersistentVolume storage

metadata: # Claim name and labels

name: mysql-pv-claim

labels:

app: employee-jdbc

spec: # Access mode and resource limits

storageClassName: standard # Request a certain storage class

accessModes:

- ReadWriteOnce # ReadWriteOnce means the volume can be mounted as read-write by a single Node

resources:

requests:

storage: 250Mi

Create Mysql Service file with the name of **employee-jdbc-service.yml**:

vi employee-jdbc-service.yml

---

apiVersion: v1 # API version

kind: Service # Type of kubernetes resource

metadata:

name: employee-jdbc-mysql # Name of the resource

labels: # Labels that will be applied to the resource

app: employee-jdbc

spec:

ports:

- port: 3306

selector: # Selects any Pod with labels `app=polling-app,tier=mysql`

app: employee-jdbc

tier: mysql

clusterIP: None

Create Mysql Deployment file with name of **employee-jdbc-deployment.yml:**

vi employee-jdbc-deployment.yml

---

apiVersion: apps/v1

kind: Deployment # Type of the kubernetes resource

metadata:

name: employee-jdbc-mysql # Name of the deployment

labels: # Labels applied to this deployment

app: employee-jdbc

spec:

selector:

matchLabels: # This deployment applies to the Pods matching the specified labels

app: employee-jdbc

tier: mysql

strategy:

type: Recreate

template: # Template for the Pods in this deployment

metadata:

labels: # Labels to be applied to the Pods in this deployment

app: employee-jdbc

tier: mysql

spec: # The spec for the containers that will be run inside the Pods in this deployment

containers:

- image: mysql:5.6 # The container image

name: mysql

env: # Environment variables passed to the container

- name: MYSQL\_ROOT\_PASSWORD

valueFrom: # Read environment variables from kubernetes secrets

secretKeyRef:

name: mysql-root-pass

key: password

- name: MYSQL\_DATABASE

valueFrom:

secretKeyRef:

name: mysql-db-url

key: database

- name: MYSQL\_USER

valueFrom:

secretKeyRef:

name: mysql-user-pass

key: username

- name: MYSQL\_PASSWORD

valueFrom:

secretKeyRef:

name: mysql-user-pass

key: password

ports:

- containerPort: 3306 # The port that the container exposes

name: mysql

volumeMounts:

- name: mysql-persistent-storage # This name should match the name specified in `volumes.name`

mountPath: /var/lib/mysql

volumes: # A PersistentVolume is mounted as a volume to the Pod

- name: mysql-persistent-storage

persistentVolumeClaim:

claimName: mysql-pv-claim

**Deploying MySQL:**

Kubectl apply -f .



Check deployments, pods and services:

To check deployments:

kubectl get deployments

To check pods:

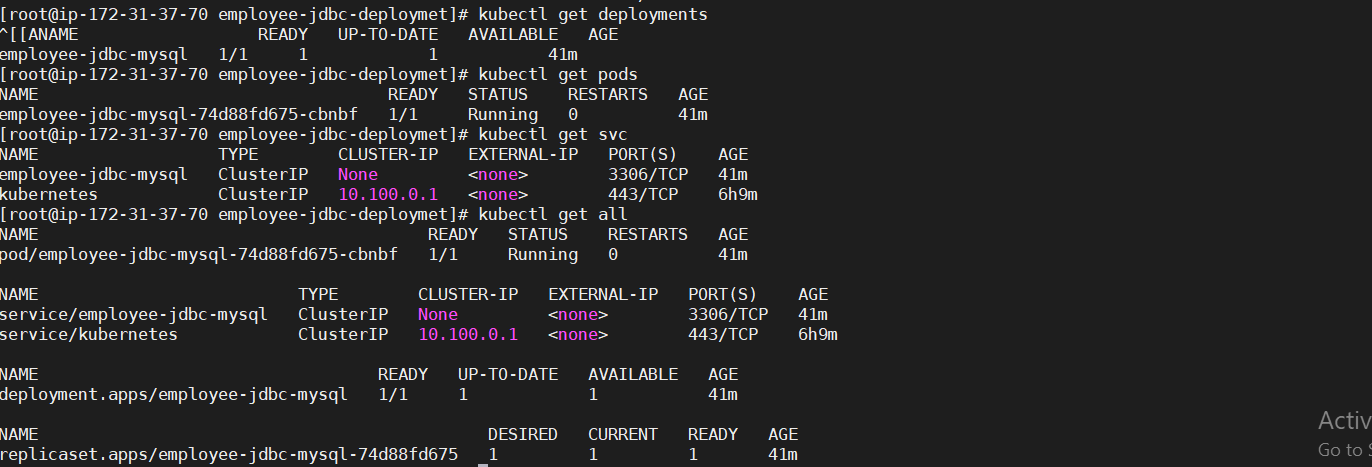
kubectl get pods

To check services:

kubectl get svc

To check all which are running:

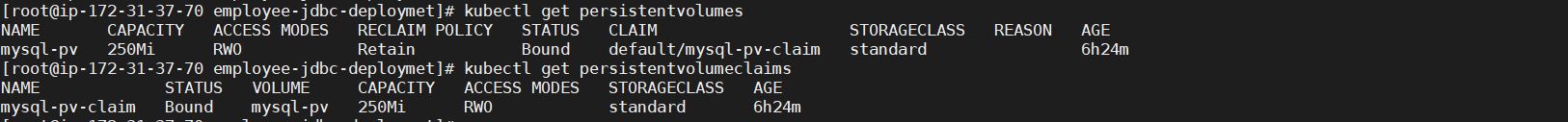
kubectl get all



To check Persistent volumes and Persistentvolumeclaim:

kubectl get persistentvolumes

kubectl get persistentvolumeclaims

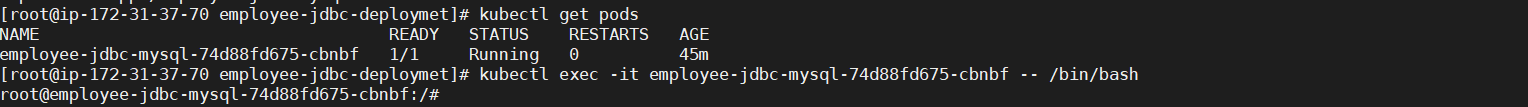


**Logging into the MySQL pod:**

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

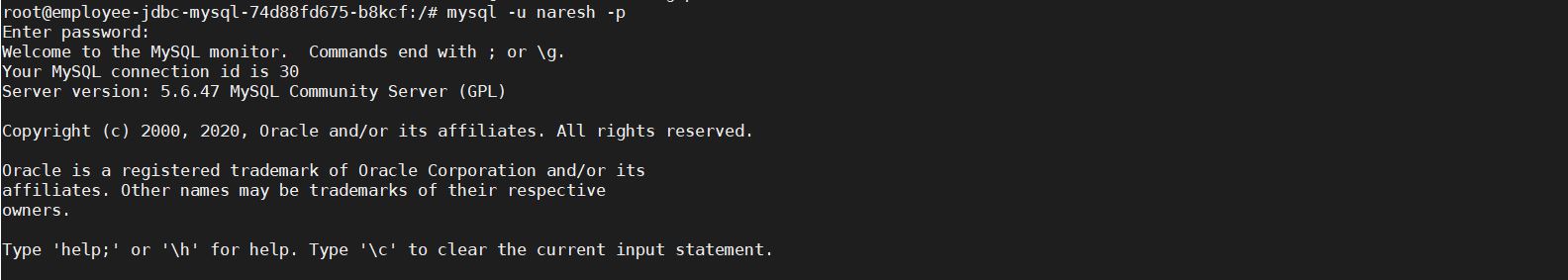
kubectl get pods

kubectl exec -it employee-jdbc-mysql-74d88fd675-cbnbf -- /bin/bash



Login in to mysql:

mysql -u naresh -p



Show databases:

show databases;



Change database to **mysqldb**:

use mysqldb;



Create table with the name of **employee**:

create table employee(empId varchar(40), empName varchar(40));

D:\Naresh\snipping\Capture.JPG

Now exit from pod:

exit

Create directory with the name of **employee-jdbc-deploymet**:

mkdir employee-jdbc-deploymet

cd employee-jdbc-deploymet

Create a file with the name of **employee-jdbc-server.yml**:

vi employee-jdbc-server.yml

---

apiVersion: v1 # API version

kind: Service # Type of the kubernetes resource

metadata:

name: employee-jdbc-server # Name of the kubernetes resource

labels: # Labels that will be applied to this resource

app: employee-jdbc-server

spec:

type: LoadBalancer # The service will be exposed by opening a Port on each node and proxying it.

selector:

app: employee-jdbc-server # The service exposes Pods with label `app=employee-jdbc-server`

ports: # Forward incoming connections on port 8080 to the target port 8080

- name: http

port: 8080

targetPort: 8080

Create a file with the name of **employee-jdbc-deployment.yml**:

vi employee-jdbc-deployment.yml

---

apiVersion: apps/v1 # API version

kind: Deployment # Type of kubernetes resource

metadata:

name: employee-jdbc-server # Name of the kubernetes resource

labels: # Labels that will be applied to this resource

app: employee-jdbc-server

spec:

replicas: 1 # No. of replicas/pods to run in this deployment

selector:

matchLabels: # The deployment applies to any pods mayching the specified labels

app: employee-jdbc-server

template: # Template for creating the pods in this deployment

metadata:

labels: # Labels that will be applied to each Pod in this deployment

app: employee-jdbc-server

spec: # Spec for the containers that will be run in the Pods

containers:

- name: employee-jdbc-server

image: naresh240/employee-jdbc

imagePullPolicy: IfNotPresent

ports:

- name: http

containerPort: 8080 # The port that the container exposes

resources:

limits:

cpu: 0.2

memory: "200Mi"

env: # Environment variables supplied to the Pod

- name: SPRING\_DATASOURCE\_USERNAME # Name of the environment variable

valueFrom: # Get the value of environment variable from kubernetes secrets

secretKeyRef:

name: mysql-user-pass

key: username

- name: SPRING\_DATASOURCE\_PASSWORD

valueFrom:

secretKeyRef:

name: mysql-user-pass

key: password

- name: SPRING\_DATASOURCE\_URL

valueFrom:

secretKeyRef:

name: mysql-db-url

key: url

Deploying the Spring Boot app on Kubernetes:

kubectl apply -f .



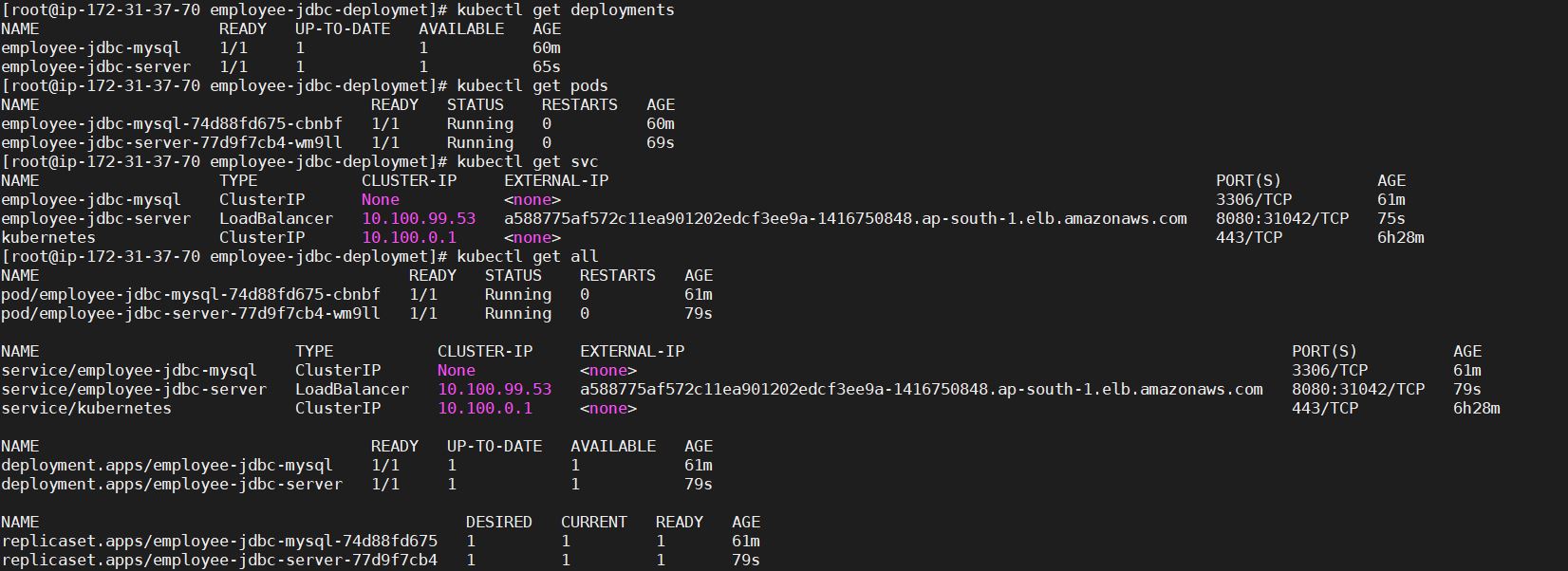
To check deployments, pods and svc:

kubectl get deployments

kubectl get pods

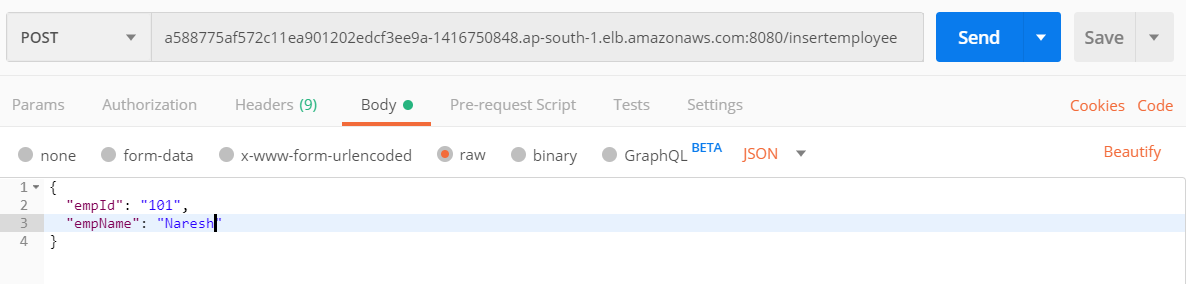
kubectl get svc

kubectl get all



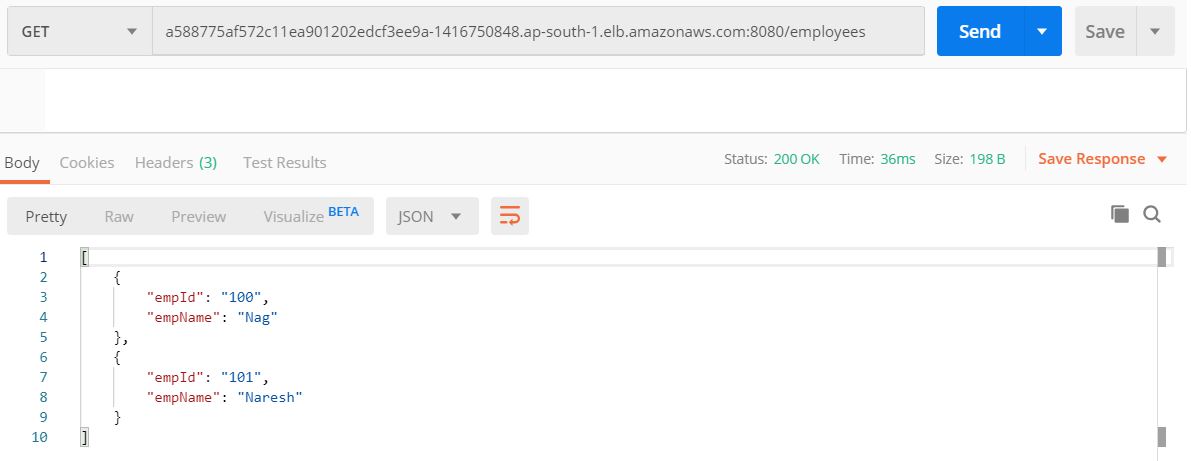
Copy EXTERNAL-IP from services and check with postman app:

a588775af572c11ea901202edcf3ee9a-1416750848.ap-south-1.elb.amazonaws.com:8080/insertemployee



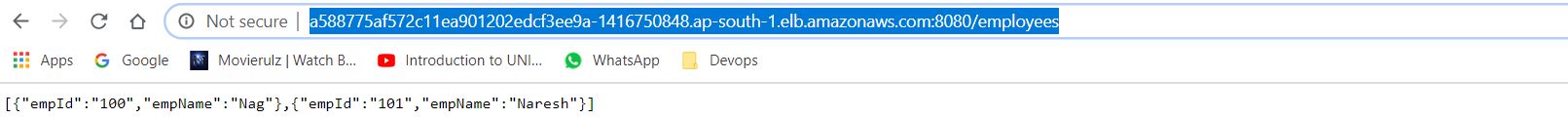
To check another API which used with in the source code:

a588775af572c11ea901202edcf3ee9a-1416750848.ap-south-1.elb.amazonaws.com:8080/employees



To check from UI:

<http://a588775af572c11ea901202edcf3ee9a-1416750848.ap-south-1.elb.amazonaws.com:8080/employees>



**To delete deployments and svc:**

kubectl delete deployments employee-jdbc-server employee-jdbc-mysql

kubectl delete svc employee-jdbc-mysql employee-jdbc-server

