Category microservice

In this lab we will look into an existing brown field category microservice and deploy it to the K8s cluster.

Learning Outcomes

After completing the lab, you will be able to:

1. Understand working with brown-field applications.
2. Lift and shift a brown-field application into K8s cluster
3. Deploy category microservice without modifying its functionality

Download the codebase category into workspace directory.

Create a repository called `category` in your GitHub account. Add this repository as a remote called origin of your local repository.

We will start by pushing the initial commit to GitHub, complete with the start and solutions tags.

```
git push origin master --tags
```

Before starting the lab, checkout the `distributed-start` tag into a new feature branch.

```
git checkout distributed-start -b category-wip
```

Category microservice design & implementation

1. Open the source code in intellij.
2. Take time to do a code walkthrough and understand the design and functionality of category micoservice.
3. The service uses `mongodb` for persisting the categories.
4. Remember, this is a brown field application which is already developed and your main goal is to lift and shift to K8s cluster.
5. Build the source code and test it using curl/postman.

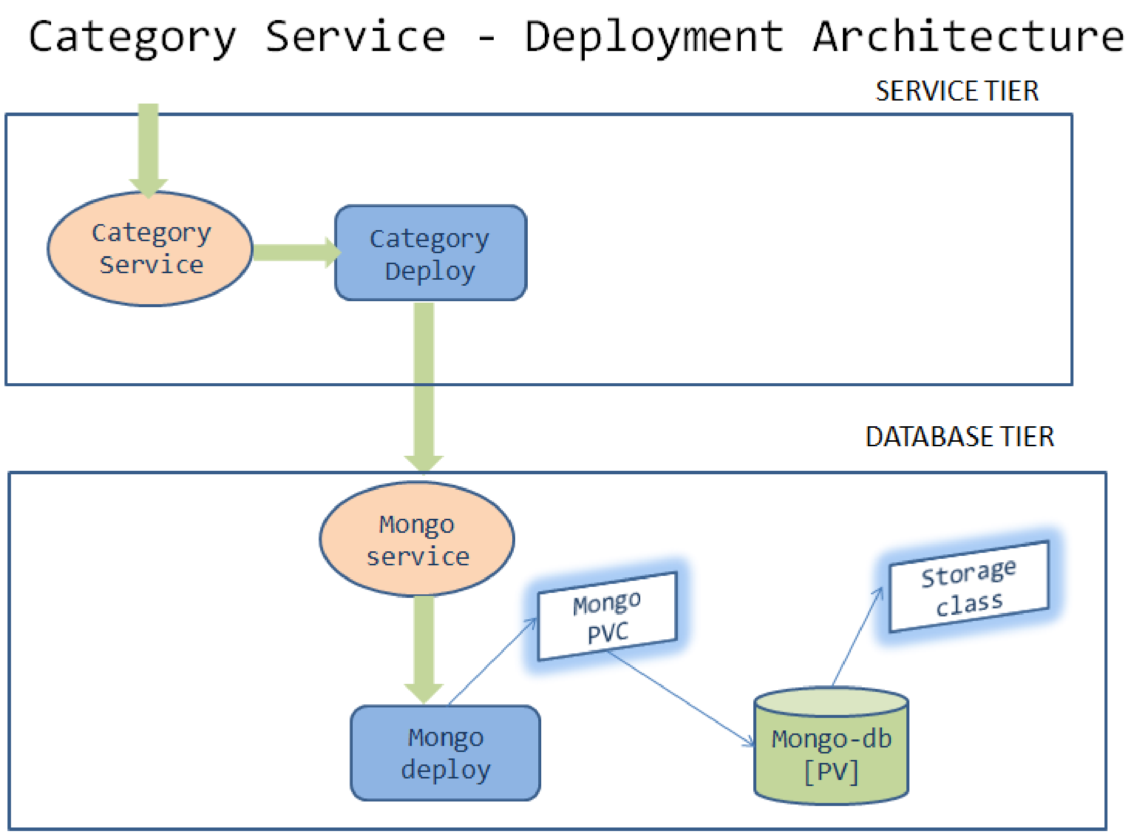
   `./gradlew bootRun`
6. Refer Curl Guide for testing and proceed with the next steps
7. Build the jar file and dockerize the category service

```
./gradlew clean build

docker build -t [docker-username]/category:distributed .

docker push [docker-username]/category:distributed
```

Deploying category microservice to K8s



1. Observe the deployments directory, which contains the manifest files for K8s Deployments
2. Walkthrough the yaml files & understand the solution to the deployment architecture.
3. Before we start deploying, replace `[student-name]` with your namespace in all the yaml files. Also, update the image name in the category deployment with `[docker-username]/category:distributed` replacing with your docker user name
4. We will first deploy our application on minikube and then deploy it to the production cluster

Deploy and test locally using minikube

1. Start minikube locally `minikube start --driver=virtualbox`

2. Verify the kubectl context `kubectl config get-contexts` is set to minikube. If not, set it to minikube `kubectl config use-context minikube`
3. Follow the Deployment Guide to deploy in the minikube and test the application locally.

Deploy and test in the production cluster

1. Verify the kubectl context `kubectl config get-contexts` is set to production cluster. If not, set it to the production cluster `kubectl config use-context [cluster-name]`
2. Follow the Deployment Guide to deploy and test the application in production

Deployment Guide

1. Set up `[student-name]` namespace to point to the current context. If the namespace is not created, the deployments will not work.

```
kubectl config set-context --current --namespace=[student-name]
```

2. Create the Database tier

```
kubectl apply -f deployment/mongo-storage-class.yaml
kubectl apply -f deployment/mongo-pv.yaml
kubectl apply -f deployment/mongo-pvc.yaml
kubectl apply -f deployment/mongo-service.yaml
kubectl apply -f deployment/mongo-deployment.yaml
```

3. Verify the deployment of database tier

```
kubectl get deployment mongo
kubectl get service mongo
kubectl get pvc
```

4. Proceed further if there are no errors, otherwise troubleshoot and fix them.
5. Create the service tier

```
kubectl apply -f deployment/category-service.yaml
kubectl apply -f deployment/category-deployment.yaml
```

6. Verify the deployment of service tier

```
kubectl get deployment category
kubectl get service category
```

7. Access the category application

```
kubectl port-forward svc/category 8080:8080
```

8. Refer Curl Guide for testing and proceed with the next steps

9. Commit code changes to the github repository

```
git add .
git commit -m "Category Start"
git push -u origin category-wip
```

Task Accomplished

We successfully deployed a 2 tier category microservice application to K8s cluster.