

Kubernetesize - Deploy the application to kubernetes cluster

In this lab, we shall deploy the application to a kubernetes cluster.

Learning Outcomes

After completing the lab, you will be able to:

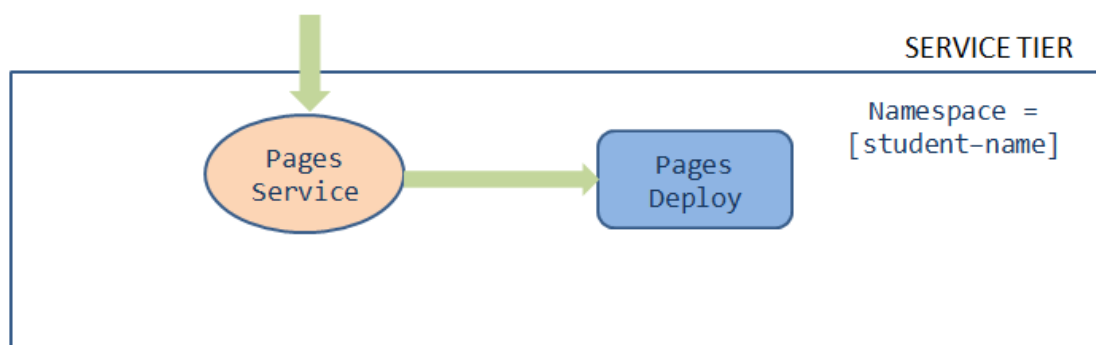
1. Describe how to create Kubernetes Objects
2. Describe how to write yaml files for pods, deployments and services
3. Run your Spring boot application within a kubernetes cluster
4. Basics of K8s service discovery concepts

Starting to Kubernetesize

Before starting the lab, verify the pages image created in the previous lab exists in [docker hub](#)

Below the initial version of our deployment architecture which will evolve as our application evolves.

Initial Deployment Architecture



Complete the following tasks to implement the above architecture

1. Create a unique namespace `[student-name]` in the cluster. For example, if your name is **Bryan Evans** you can replace `[student-name]` as **bryan** or **evans**

which ever is unique across the cluster.

2. Create a deployment object with the following specifications.

```
name -> pages
namespace -> [student-name]
labels -> app: pages
replicas -> 1
image -> [docker-username]/pages:[tag](name of the image c
created in the previous lab)
containerPort -> 8080
```

3. Create a service object to expose the deployment with the following specifications.

```
name -> pages
namespace -> [student-name]
labels -> app: pages
selector -> pages
Type -> NodePort
targetPort -> 8080
port -> 8080
protocol -> TCP
```

Solution guide:

Create manifest files for deployment

1. Create deployment/pages-namespace.yaml from the root project folder

```
apiVersion: v1
kind: Namespace
metadata:
  name: [student-name]
```

2. Create deployment/pages-deployment.yaml from the root project folder

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: pages
    name: pages
    namespace: [student-name]
spec:
  replicas: 1
  selector:
    matchLabels:
```

```
    app: pages
strategy: {}
template:
  metadata:
    labels:
      app: pages
  spec:
    containers:
      - image: [docker-username]/pages:[tag]
        name: pages
        ports:
          - containerPort: 8080
```

3. Create deployment/pages-service.yaml from the root project folder

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: pages
  name: pages
  namespace: [student-name]
spec:
  ports:
    - name: pages-service-port
      port: 8080
      protocol: TCP
      targetPort: 8080
  selector:
    app: pages
  type: NodePort
```

Start minikube locally

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. Deploy and test the application in minikube. Refer to [Deployment Guide](#)

Deploy the application to production cluster

1. Follow [Production Cluster Guide](#) to login/connect to the production cluster.
2. Deploy and test the application in production cluster. Refer to [Deployment Guide](#)
3. Commit code changes to the github repository

```
git add .  
git commit -m "Added K8 deployment objects"  
git push -u origin master
```

Deployment Guide

1. Create kubernetes objects

```
kubectl apply -f deployment/pages-namespace.yaml  
kubectl apply -f deployment/pages-service.yaml  
kubectl apply -f deployment/pages-deployment.yaml
```

2. Verify the created objects

```
kubectl get deployment pages --namespace [student-name]  
kubectl get service pages --namespace [student-name]
```

3. Set up [student-name] namespace to point to the current context

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

4. Access the pages application by port-forwarding using kubectl, enabling the application can be served via localhost on port 8080

```
kubectl port-forward svc/pages 8080:8080  
  
curl localhost:8080
```