KUBERNETES ASSIGNMENT

Vikas K R(TAS241)

QUESTION STATEMENT:

1). You need to write required yaml files for this service.
2). You need to create a jar for this service first and then create a dockerfile for it.
3). After that to deploy this service in kubernetes you need to install minikube in your local or if you have any cloud platform account then it is also fine.
4). You can access this application on any port you want to (it's your choice).
5). This application is deploying a tomcat server in kubernetes. write a readfile for this.

Solution:

1. Set Up the Environment

Before starting, make sure the following are installed:

- **JDK 11 or later**: To compile your Java code.
- **Apache Maven**: To build the JAR/WAR file for your project.
- **Docker**: To build the Docker image for the Java service.
- **Minikube**: A local Kubernetes cluster (if using Minikube) or access to any cloud-based Kubernetes platform.
- **kubectl**: A command-line tool for Kubernetes to interact with the cluster.

1.1 Install Necessary Tools

- Install JDK.
- Install Maven.
- Install Docker.
- Install Minikube.

2. Create a Dockerfile

- **FROM**: This line specifies the base image for the Docker container. In this case, it's tomcat:9.0-jdk11-openjdk, which is an official Tomcat image with JDK 11, making it suitable for running Java web applications.
- **COPY**: This command copies the WAR file from the local `target` directory into the Tomcat container's `webapps` folder, renaming it as `ROOT.war`. This ensures that Tomcat runs the application as the default web application when it starts.
- **EXPOSE**: This exposes port `8080` on the container, which is Tomcat's default HTTP port. It tells Docker that the container will listen for incoming connections on this port.
- **CMD**: This specifies the command that the container will run when it starts. In this case, `catalina.sh run` starts the Tomcat server in the foreground.

Adding `WORKDIR /app` is not necessary in this case because Tomcat doesn't require the application to be in a specific directory other than its default webapps directory (`/usr/local/tomcat/webapps`). The application will be served directly by Tomcat from the `ROOT.war` file that is already placed in the correct location. Therefore, the `WORKDIR` command is irrelevant here for running the program on Tomcat.

Dockerfile M ● | Deployment.yaml | Service.yaml | *README.txt*

k8s_project_new > Dockerfile

```
FROM tomcat:9.0-jdk11-openjdk

COPY /target/docker-java-sample-webapp-1.0-SNAPSHOT.war /usr/local/tomcat/webapps/ROOT.war

EXPOSE 8080

CMD ["catalina.sh", "run"]
```

3. Create Deployment file

- The `apiVersion` specifies that this deployment uses version `apps/v1`, which is the stable API for deploying applications. The `kind` field indicates that the resource being defined is a **Deployment**.
- In the `metadata` section, `name: java-deployment-v` gives the deployment a unique name for identification purposes within the Kubernetes cluster. The `spec` section contains the deployment's specifications.
- The deployment will run 2 replicas, meaning two instances (pods) of the Java application will be created for better availability. The `selector` field tells Kubernetes which pods this deployment will manage by matching the `app: tomcat-app` label.
- The `template` section defines the pod specification. In the `metadata` subsection, the `labels` field assigns the `app: tomcat-app` label to the pod, which allows the deployment to identify the pods it creates.
- Under `spec`, it defines the container named `java-app-vikas`, which uses the image `vikaskarbail/java-app-vikas:latest` from Docker Hub. The `ports` section specifies that port `8080` will be exposed within the container, which is the port Tomcat uses to serve the web application.

Dockerfile M ● | ! Deployment.yaml × | ! Service.yaml | ⓘ *README.txt*

k8s_project_new > ! Deployment.yaml > {} spec > {} template > {} spec > [ ] containers > {} 0 > [ ] ports

all.json

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: java-deployment-v
spec:
  replicas: 2
  selector:
    matchLabels:
      app: tomcat-app
  template:
    metadata:
      labels:
        app: tomcat-app
    spec:
      containers:
      - name: java-app-vikas
        image: vikaskarbail/java-app-vikas:latest
        ports:
        - containerPort: 8080
```

4. Create Service file

- The `apiVersion: v1` specifies that this configuration uses version 1 of the Kubernetes API, and the `kind: Service` indicates that this resource is a Service, which is used to expose applications within or outside the cluster.
- In the `metadata` section, the `name: java-service-v` gives the service a unique identifier within the Kubernetes cluster.
- In the `spec` section, `type: NodePort` means the service will be exposed on a specific port on each node of the cluster, making it accessible from outside the cluster. The `selector` field uses the label `app: tomcat-app` to match the pods that the service will route traffic to.
- The `ports` section defines how traffic is handled. The service listens on `port 8080` and routes traffic to the containers' `targetPort 8080`, which is the port where the Tomcat server is running. The

`nodePort: 30008` specifies that the service will be available externally on port 30008, meaning users can access the Java application via any node's IP address on this port.

Dockerfile M ● | ! Deployment.yaml | ! Service.yaml ×

k8s_project_new > ! Service.yaml > ...

```yaml
apiVersion: v1
kind: Service
metadata:
  name: java-service-v
spec:
  type: NodePort
  selector:
    app: tomcat-app
  ports:
    - port: 8080
      targetPort: 8080
      nodePort: 30008
```

5. Deploy on Minikube

Run the following commands

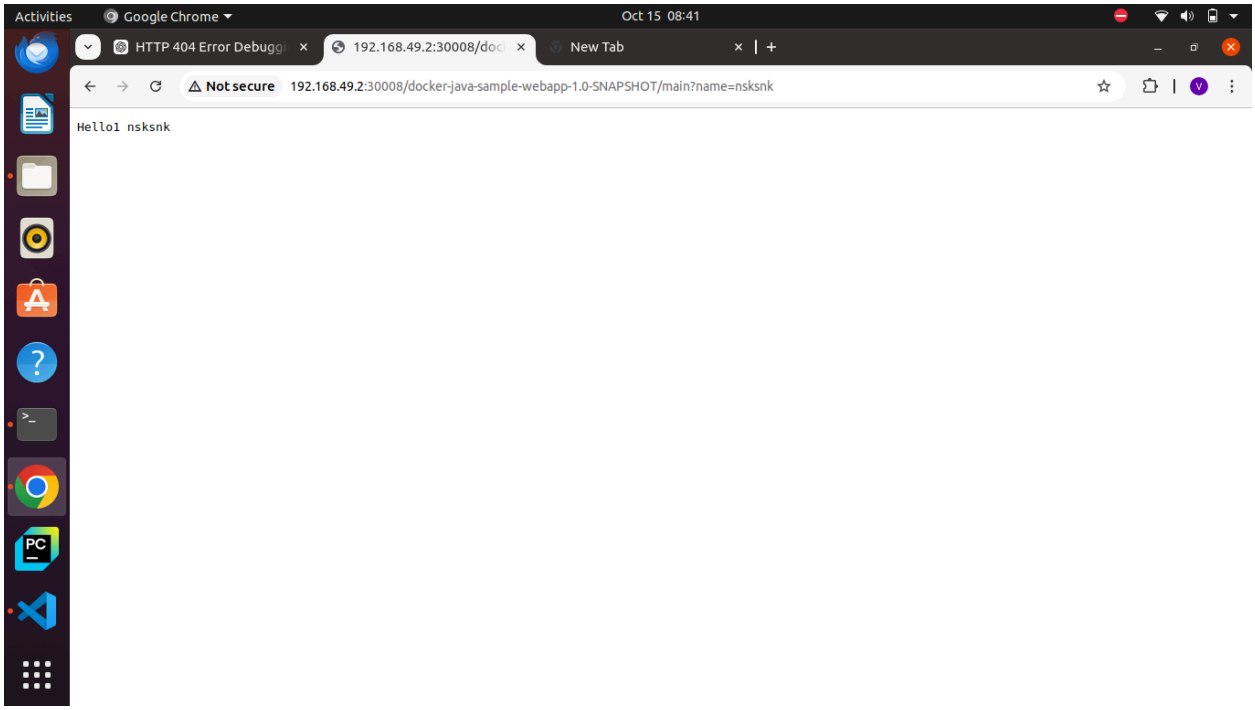
```
minikube start

kubectl apply -f deployment.yaml

kubectl apply -f service.yaml

minikube service java-tomcat-v --url
```

OUTPUT:



Github link: https://github.com/VikasKR123/Kuberntes_Assignment_Vikas