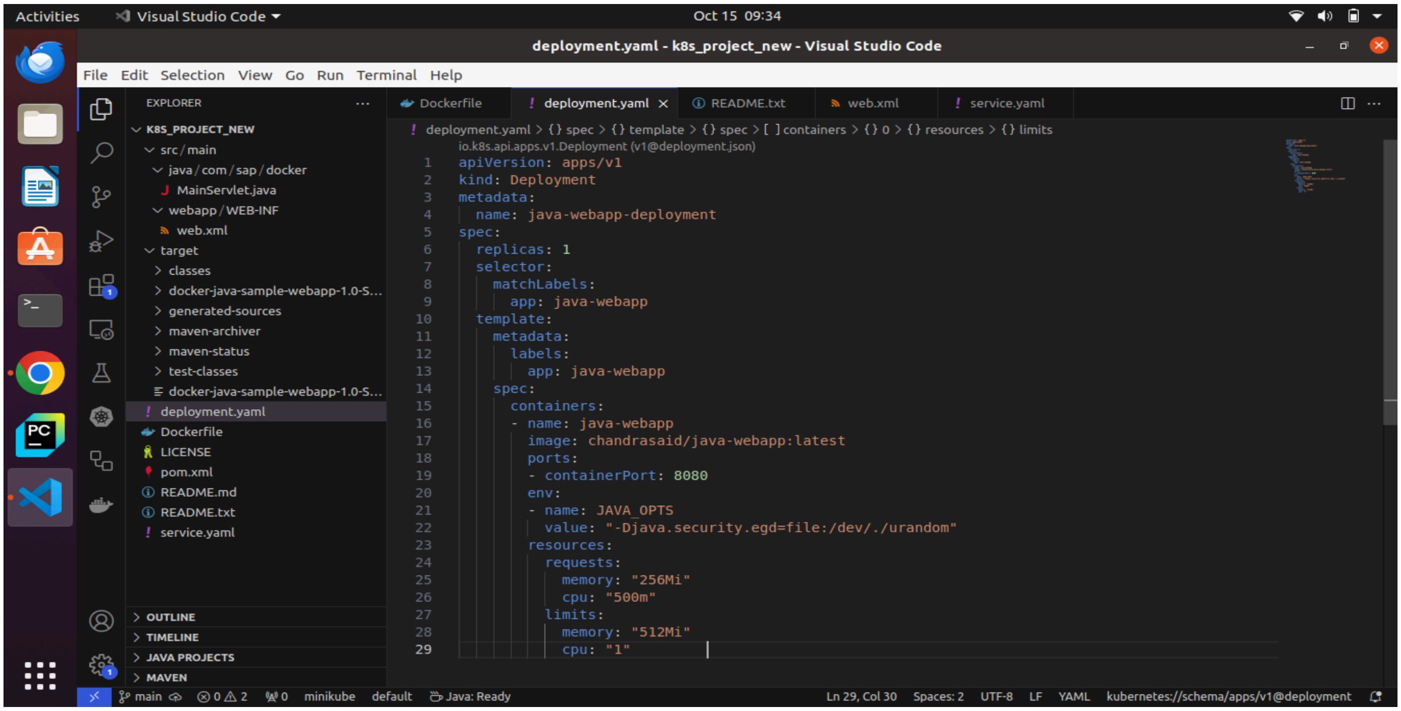
TAS 263 - Chandra Sai D

Deploying a Java Web Application on Kubernetes

Steps to Deploy the Service

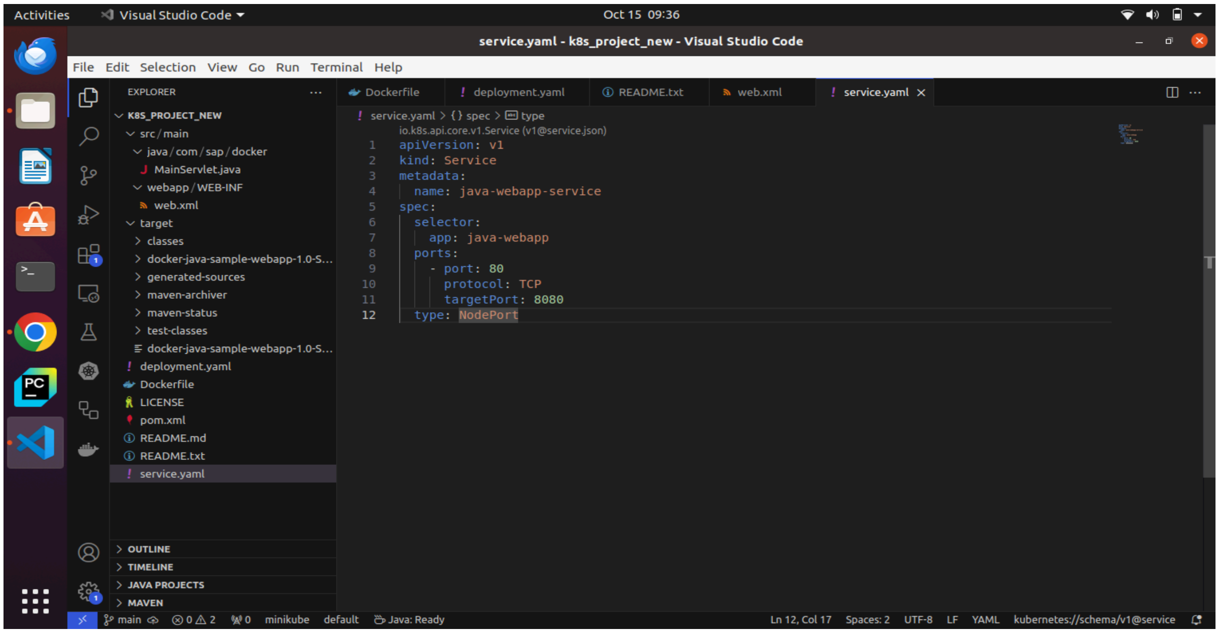
Step 1: Create YAML Files for the Service

1.1 Deployment YAML (deployment.yml)


deployment.yaml - k8s_project_new - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Dockerfile
deployment.yaml
README.txt
web.xml
service.yaml
EXPLORER
K8S_PROJECT_NEW
deployment.yaml > {} spec > {} template > {} spec > [ ] containers > {} 0 > {} resources > {} limits
io.k8s.api.apps.v1.Deployment (v1@deployment.json)
Ln 29, Col 30
YAML

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: java-webapp-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: java-webapp
  template:
    metadata:
      labels:
        app: java-webapp
    spec:
      containers:
      - name: java-webapp
        image: chandrasaid/java-webapp:latest
        ports:
        - containerPort: 8080
        env:
        - name: JAVA_OPTS
          value: "-Djava.security.egd=file:/dev/./urandom"
        resources:
          requests:
            memory: "256Mi"
            cpu: "500m"
          limits:
            memory: "512Mi"
            cpu: "1"
```

1.2 Service YAML (service.yml)


service.yaml - k8s_project_new - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Dockerfile
deployment.yaml
README.txt
web.xml
service.yaml
EXPLORER
K8S_PROJECT_NEW
service.yaml > {} spec > type
io.k8s.api.core.v1.Service (v1@service.json)
Ln 12, Col 17
YAML

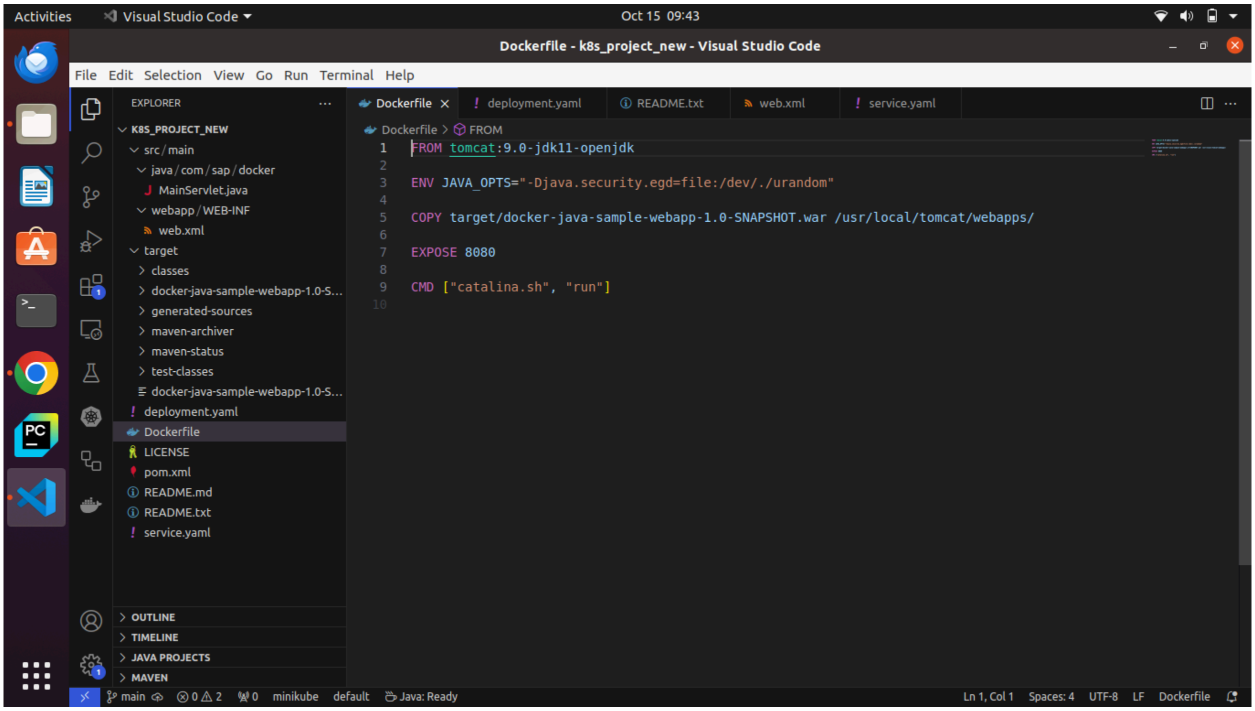
```
apiVersion: v1
kind: Service
metadata:
  name: java-webapp-service
spec:
  selector:
    app: java-webapp
  ports:
    - port: 80
      protocol: TCP
      targetPort: 8080
  type: NodePort
```

Step 2: Create JAR/WAR for the Service

Build the WAR File:

$ mvn clean package

Step 3: Create a Dockerfile



```
FROM tomcat:9.0-jdk11-openjdk

ENV JAVA_OPTS="-Djava.security.egd=file:/dev/./urandom"

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

EXPOSE 8080

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

Step 4: Build the Docker Image

$ docker build -t chandrasaid/my-java-webapp:latest .

Step 5: Deploy the Application in Kubernetes

5.1 Start Minikube

$ minikube start

5.2 Deploy the Application

$ kubectl apply -f deployment.yml

$ kubectl apply -f service.yml

Check the status of your pods:

$ kubectl get pods

Step 6: Access the Application

To access the application, you can use the minikube service command:

```
$ minikube service my-webapp --url
```

Step 7: It is designed to demonstrate how to deploy a Java web application on a Tomcat server using Kubernetes.

The application greets users by their name, which can be passed as a parameter in the URL.

To use this application, access it via the following URL format:

$ http://192.168.39.125:30160/docker-java-sample-webapp-1.0-SNAPSHOT/?name=Your_Name

Replace `Your_Name` with your actual name to see a personalized greeting.

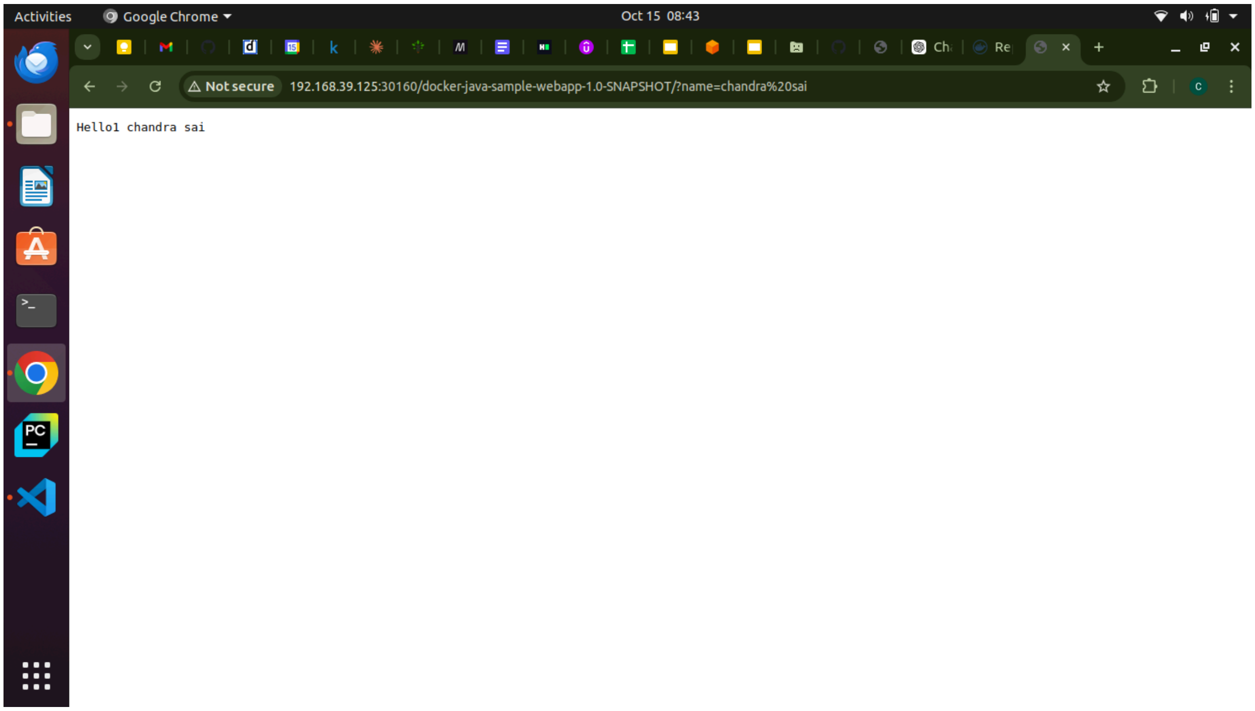
Example :

For instance, if you want to greet 'Chandra', use the following URL:

http://192.168.39.125:30160/docker-java-sample-webapp-1.0-SNAPSHOT/?name=Chandra

This will display the message:

`Hello, Chandra Welcome to our application!`



192.168.39.125:30160/docker-java-sample-webapp-1.0-SNAPSHOT/?name=chandra%20sai

```
Hello1 chandra sai
```