**Problem Statement: Deploying a Python Server Application using Kubernetes with Docker Desktop**

**Objective:**

The goal of this task is to **deploy a Python server application** using **Kubernetes** on **Docker Desktop**. You will containerize the Python application using **Docker**, create a **Kubernetes Deployment**, and expose it via a **NodePort service** so that it is accessible from other machines in the network.

**Requirements:**

You need to perform the following tasks:

1. **Set up Docker Desktop with Kubernetes**
   * Ensure **Docker Desktop** is installed and Kubernetes is enabled.
   * Verify Kubernetes is running with:
   * kubectl cluster-info
2. **Create a Python Server Application**
   * Implement a simple **Flask-based Python server** (app.py) that returns a welcome message when accessed at /.
3. **Containerize the Application using Docker**
   * Write a **Dockerfile** to package the application.
   * Build and test the **Docker image** locally.
4. **Deploy the Application on Kubernetes**
   * Create a **Kubernetes Deployment** (deployment.yaml) to manage the Python server pods.
   * Ensure the Deployment runs exactly **3 replicas**.
5. **Expose the Service using NodePort**
   * Create a **NodePort service** (service.yaml) to make the application accessible.
   * Ensure the application is accessible via <Docker-Desktop-IP>:<NodePort>.
6. **Test the Deployment**
   * Retrieve the **assigned NodePort** and test access using curl or a web browser.
   * Verify the application returns the expected response.
7. **Cleanup (Optional)**
   * Delete the Kubernetes resources after successful deployment.

**Expected Deliverables:**

1. **Python server application (app.py)**
2. **Dockerfile to containerize the app**
3. **Kubernetes Deployment YAML (deployment.yaml)**
4. **Kubernetes Service YAML (service.yaml)**
5. **Instructions on how to test the deployed application**

**Success Criteria:**

* The Python server should be **running in Kubernetes** on Docker Desktop.
* The application should be **accessible from a browser or via curl** using <Docker-Desktop-IP>:<NodePort>.
* Running kubectl get pods should show Exactly **3 running replicas**.

**Additional Notes:**

* Use kubectl describe service <service-name> to find the **NodePort**.
* The service should be accessible using:
* curl http://<Docker-Desktop-IP>:<NodePort>
* If using **Windows**, replace localhost with host.docker.internal if needed.

**Step-by-Step Solution**

This guide will help you **deploy a Python server application** using **Docker Desktop’s Kubernetes** and expose it using a **NodePort service**.

**🔹 Step 1: Verify Docker Desktop and Kubernetes are Running**

Ensure that Docker Desktop and Kubernetes are properly set up.

**1.1 Check if Kubernetes is Running**

kubectl version

**1.2 Check Docker Version**

docker --version

**🔹 Step 2: Create a Simple Python Server**

We will create a simple **Flask-based Python server**.

**2.1 Create a Project Directory**

mkdir python-k8s-app && cd python-k8s-app

**2.2 Create app.py**

Create a Python file (app.py) that runs a simple web server

**🔹 Step 3: Containerize the Application using Docker**

We need to create a Docker container for our Flask application.

**3.1 Create Dockerfile**

Create a Dockerfile in the same directory:

# Use an official Python runtime as a parent image

FROM python:3.9

# Set the working directory in the container

WORKDIR /app

# Copy the application files into the container

COPY app.py /app

# Install Flask

RUN pip install flask

# Expose the application port

EXPOSE 80

# Run the Python server

CMD ["python", "app.py"]

**3.2 Build the Docker Image**

Run the following command to build the image:

docker build -t my-app .

**3.3 Verify the Docker Image**

Check if the image is built:

docker images

**3.4 Test the Image Locally**

Run the container and test:

docker run -p 4000:80 my-app

Open a browser and visit:

http://localhost:5000

You should see:

Hello, Kubernetes with NodePort!

Press CTRL + C to stop the container.

**To load a Docker image from Docker Desktop to a Kubernetes cluster set up with kubeadm**

**Use a Private Registry (Optional)**

If multiple nodes need the image, push it to a private registry instead of manually transferring:

1. Start a local registry:

**docker run -d -p 5000:5000 --name registry registry:2**

1. Tag and push the image:

**docker tag my-python-app:latest localhost:5000/my-python-app:latest**

**docker push localhost:5000/my-python-app:latest**

1. Update the Kubernetes deployment to use **localhost:5000/my-python-app:latest**

**🔹 Step 5: Deploy the Application to Kubernetes (Updated for 3 Replicas)**

**5.1 deployment.yaml (3 Replicas)**

apiVersion: apps/v1

kind: Deployment

metadata:

  name: my-app

spec:

  replicas: 3

  selector:

    matchLabels:

      app: my-app

  template:

    metadata:

      labels:

        app: my-app

    spec:

      containers:

      - name: my-app

        image: localhost:5000/my-app:latest  # Local image name and tag

        ports:

        - containerPort: 80

**5.2 service.yaml**

apiVersion: v1

kind: Service

metadata:

  name: my-app-service

spec:

  type: NodePort

  selector:

    app: my-app

  ports:

  - protocol: TCP

    port: 80

    targetPort: 80

    nodePort: 30000

**🔹 Step 6: Deploy to Kubernetes**

Now we **apply the changes**.

**6.1 Apply Deployment and Service**

kubectl apply -f deployment.yaml

kubectl apply -f service.yaml

**6.2 Verify the Deployment**

Check if **all 3 replicas are running**:

kubectl get pods

Example output:

NAME READY STATUS RESTARTS AGE

python-server-xyz123 1/1 Running 0 30s

python-server-abc456 1/1 Running 0 30s

python-server-mno789 1/1 Running 0 30s

✅ **3 Pods are running**

**6.3 Verify the Number of Replicas**

kubectl get deployment python-server

You should see:

NAME READY UP-TO-DATE AVAILABLE AGE

python-server 3/3 3 3 30s

**🔹 Step 7: Access the Application**

Now, access the application using **Docker Desktop’s IP**.

**7.1 Find Docker Desktop’s IP**

kubectl get nodes -o wide

Look for INTERNAL-IP, e.g., 192.168.1.100.

**7.2 Open the App in a Browser**

Go to:

http://192.168.1.100:30007

or use curl:

curl http://192.168.1.100:30007

Expected output:

Hello, Kubernetes with NodePort!

**🔹 Step 8: Cleanup (Optional)**

To remove all resources:

kubectl delete -f deployment.yaml

kubectl delete -f service.yaml