Day 4-Kubernetes

1.Setup Directory Structure

First, create a project directory to keep all files organized.

```
mkdir E-commerce && cd E-commerce
```

Backend Setup

Create a backend directory and navigate into it.

```
mkdir backend && cd backend
```

Create `products.csv`

This file will store product details in CSV format.

```
nano products.csv
```

Paste the following sample data:

```
id,name,price,quantity
1,Smartphone,15000,25
2,Laptop,45000,15
3,Headphones,1500,50
4,Smartwatch,8000,30
5,Tablet,20000,20
6,Wireless Mouse,700,100
7,Bluetooth Speaker,1200,60
8,External Hard Drive,4000,40 9,USB
Flash Drive,500,150
10,Monitor,10000,10
```

Create `app.py`

This script sets up a Flask server to read the CSV file and return product data as JSON.

```
nano app.py
```

Paste the following Python script:

```
from flask import Flask
import pandas as pd

app = Flask(__name__)

@app.route("/products", methods=['GET']) def
read_data():
    df = pd.read_csv("products.csv")  # Ensure products.csv exists
```

```
    json_data = df.to_json() return
    json_data

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5005)
```

Create `requirements.txt`

This file lists the dependencies required for the backend.

```
nano requirements.txt
```

Add dependencies:

```
flask
pandas
```

Create `Dockerfile`

This Dockerfile defines how to package the backend application into a container.

```
nano Dockerfile
```

Paste the following:

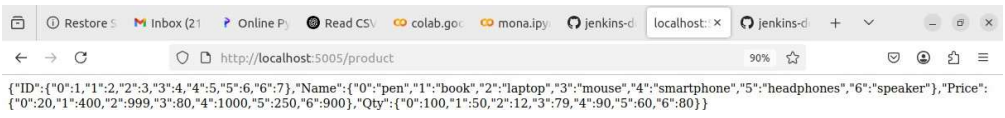
```
FROM python:3.11
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt COPY
. .
EXPOSE 5005
CMD ["python", "app.py"]
```

Build & Run Backend Container

Build and run the backend container.

```
docker build -t backend:latest . docker
run -itd -p 5005:5005 backend
docker logs $(docker ps -q --filter "ancestor=backend")
```

Run the application in the 5005/product



Frontend Setup

Create a frontend directory and navigate into it.

```
cd ..
mkdir frontend && cd frontend
```

Create `index.html`

This HTML file loads the product list from the backend.

```
nano index.html
```

Paste the following:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>E-Commerce Store</title>
    <script>
        async function fetchProducts() {
            const response = await fetch("http://localhost:5050/products");
            const products = await response.json();
            let output = "<h2>Product List</h2><ul>";
            for (const id in products.name) {
                output += `<li>${products.name[id]} -
$${products.price[id]}</li>`;
            }
            output += "</ul>";
            document.getElementById("product-list").innerHTML = output;
        }
     </script>
</head>
<body onload="fetchProducts()">
    <h1>Welcome to Our Store</h1>
    <div id="product-list">Loading...</div>
</body>
</html>
```

Create `Dockerfile`

This Dockerfile packages the frontend as an Nginx container.

```
nano Dockerfile
```

Paste:

```
FROM nginx:alpine
COPY index.html /usr/share/nginx/html/index.html
```

Build & Run Frontend Container

```
docker build -t frontend:latest .
```

1. Kubernetes Deployment

Create a `k8s` directory for Kubernetes configuration files.

```
cd ..
mkdir k8s && cd k8s
```

Backend Deployment (`backend-deployment.yaml`)

Defines a backend pod in Kubernetes.

```
nano backend-deployment.yaml
```

Paste:

```
apiVersion: apps/v1 kind:
Deployment metadata:
  name: backend spec:
  replicas: 1
  selector:
    matchLabels: app:
      backend
  template: metadata:
      labels:
        app: backend spec:
      containers:
      - name: backend
        image: backend:latest ports:
        - containerPort: 5005
```

Frontend Deployment (`frontend-deployment.yaml`)

Defines a frontend pod in Kubernetes.

```
 nano frontend-deployment.yaml
```

Paste:

```
apiVersion: apps/v1 kind:
Deployment metadata:
  name: frontend spec:
  replicas: 1
  selector:
    matchLabels: app:
      frontend
  template: metadata:
      labels:
        app: frontend
    spec:
      containers:
      - name: frontend
        image: frontend:latest

        ports:
        - containerPort: 3000
```

Connecting Frontend & Backend (`service.yaml`)

Defines services for communication between frontend and backend.

```
nano service.yaml
```

Paste:

```
apiVersion: v1 kind:
Service metadata:
  name: backend-service
spec:
  selector:
    app: backend
  ports:
    - protocol: TCP port:
      5005
      targetPort: 5005
  type: ClusterIP
---
apiVersion: v1 kind:
Service metadata:
  name: frontend-service spec:
  selector:
    app: frontend
  ports:
    - protocol: TCP port:
      3000
      targetPort: 3000
  type: NodePort
```

ConfigMap (`configmap.yaml`)

Stores backend configuration values.

```
nano configmap.yaml
```

Paste:

```
apiVersion: v1 kind:
ConfigMap metadata:
  name: backend-config
data:
  DATABASE_FILE: "/backend/products.csv"
```

Install minikube

Minikube is a tool that allows you to run a Kubernetes cluster locally on our machine. It is designed for developers who want to test and experiment with Kubernetes without needing a full-scale cloud-based cluster

Download Minikube binary

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
```

Install Minikube

```
sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

Verify installation

```
minikube version
```

Install kubectl

kubectl is the command-line tool used to interact with a Kubernetes cluster. It allows you to deploy applications, inspect and manage cluster resources, and troubleshoot issues.

```
curl -LO https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl
```

Grant permission for kubectl

```
chmod +x kubectl
```

Move to kubectl to root

```
sudo mv kubectl /usr/local/bin/
```

Check the minikube and kubectl installed properly

```
kubectl version –client
```

Start Minikube

```
minikube start
```

Verify Minikube is running

```
kubectl get nodes
```

Load the image to the minikube

Befor loading images

Perform this commend: `eval $(minikube docker-env)`

```
minikube image load frontend:latest
```

```
minikube image load backend:latest
```

Commands are used to deploy your application components (backend and frontend), expose them through a service, and provide them with the necessary configuration via a ConfigMap.

```
kubectl apply -f k8s/backend-deployment.yaml kubectl
```

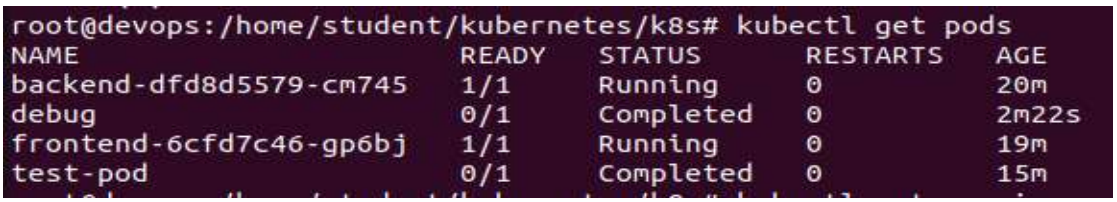
```
apply -f k8s/frontend-deployment.yaml kubectl apply -
```

```
f k8s/service.yaml
```

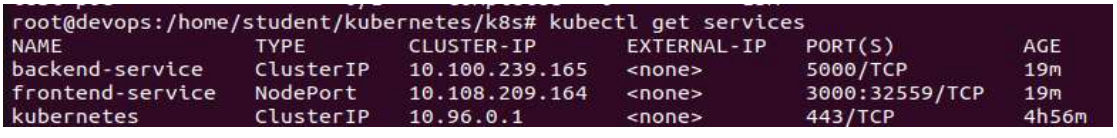
```
kubectl apply -f k8s/configmap.yaml
```

These commands are used to list and inspect the running resources in your Kubernetes cluster:

```
kubectl get pods
```




```
kubectl get svc
```



Welcome to Our Store

Loading...