Assignment: Building a Multi-Service Application with In-Place Upgrades on Kubernetes

Duration: 4 Hours
Objective:

- Deploy a multi-service application using Kubernetes.
- Containerize at least two different services (e.g., a FastAPI backend and a Node.js frontend).
- Explore Kubernetes concepts such as deployments, services, networking, scaling, and especially in-place upgrades via rolling updates.

Overview of the Multi-Service Application

Your application will consist of at least two services running in separate containers:

Backend API Service (FastAPI):
Provides REST endpoints (e.g., `/, /status`).
Containerized using a Dockerfile.
Upgrade Scenario: Later in the assignment, you will modify the API to display a new version message, then perform an in-place (rolling) update without downtime.

Frontend Service (Node.js):
A simple web interface that calls the backend API.
Containerized separately.

Part 1: Set Up Your Kubernetes Environment

Install and Start Minikube:
Follow instructions at Minikube Start.

```
# run:
minikube start

# verify your nodes
minikube get nodes
```

Part 2: Create and Containerize Your Services

Service A: Backend API (FastAPI)

Create the API Code (`backend/app.py`):

```python
from fastapi import FastAPI
import os

app = FastAPI()

@app.get("/")
def read_root():
    return {"message": "Hello from FastAPI v1 on Kubernetes"}

@app.get("/status")
def status():
    return {"status": "API Running", "host": os.getenv("HOSTNAME")}
```

Create `backend/equirements.txt` for FastAPI:

```
fastapi
uvicorn
```

Write a Dockerfile for the API (`backend/Dockerfile`):

```
FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
CMD ["uvicorn", "app:app", "--host", "0.0.0.0", "--port", "8000"]
```

Build the Backend Image:
If using Minikube's Docker daemon, run:

```
# sh
eval $(minikube docker-env)
cd backend
docker build -t my-fastapi:v1 .
cd ..
```

Service B: Frontend (Node.js)

Create the Frontend Code (`frontend/server.js`):

```
const express = require('express');
const axios = require('axios');
const app = express();
const port = 3000;

app.get('/', async (req, res) => {
  try {
    // Call the backend API using the Kubernetes DNS name (e.g., "backend-service")
    const response = await axios.get('http://backend-service:8000/');
    res.send(`<h1>Frontend</h1><p>Response from API: ${JSON.stringify(response.data)}</p>`);
  } catch (error) {
    res.send(`<h1>Error</h1><p>${error.message}</p>`);
  }
});

app.listen(port, () => {
  console.log(`Frontend app listening on port ${port}`);
});
```

Create a `frontend/package.json` for the Frontend:

```
{
  "name": "frontend",
  "version": "1.0.0",
  "main": "server.js",
  "dependencies": {
    "axios": "^0.21.1",
    "express": "^4.17.1"
  },
  "scripts": {
    "start": "node server.js"
  }
}
```

Write a Dockerfile for the Frontend (`frontend/Dockerfile`):

```
FROM node:14
WORKDIR /app
COPY package.json .
RUN npm install
COPY . .
CMD ["npm", "start"]
```

Build the Frontend Image:

```
# sh
cd frontend
docker build -t my-frontend:latest .
cd ..
```

Part 3: Deploy Services to Kubernetes

1. Create Kubernetes Deployment YAMLs

Backend Deployment (`backend-deployment.yaml`):

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
      - name: backend
        image: my-fastapi:v1
        ports:
        - containerPort: 8000
```

Frontend Deployment (`frontend-deployment.yaml`):

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
      - name: frontend
        image: my-frontend:latest
        ports:
        - containerPort: 3000
```

2. Create Kubernetes Services

Backend Service (`backend-service.yaml`):

```yaml
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  selector:
    app: backend
  ports:
    - protocol: TCP
      port: 8000
      targetPort: 8000
  type: ClusterIP
```

Frontend Service (`frontend-service.yaml`):

```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
spec:
  selector:
    app: frontend
  ports:
    - protocol: TCP
      port: 80
      targetPort: 3000
  type: LoadBalancer
```

3. Deploy Everything

Run the following commands:

```sh
#sh
kubectl apply -f backend-deployment.yaml
kubectl apply -f backend-service.yaml

kubectl apply -f frontend-deployment.yaml
kubectl apply -f frontend-service.yaml
```

4. Access Your Application

For the frontend, run:

```
minikube service frontend-service --url
```

Open the URL in your browser. The frontend should display data fetched from the backend API.

Part 4: Demonstrating In-Place Upgrades (Rolling Update)

After verifying that the application is running correctly:
1. Modify the Backend API Code for an Upgrade

Edit `backend/app.py` so that the API returns a different message (for example, indicating version 2):

```python
from fastapi import FastAPI
import os

app = FastAPI()

@app.get("/")
def read_root():
    return {"message": "Hello from FastAPI v2 – Upgraded!"}

@app.get("/status")
def status():
    return {"status": "API Running", "host": os.getenv("HOSTNAME")}
```

2. Build the New Docker Image

Update the Docker image version tag to v2:

```
# sh
cd backend
docker build -t my-fastapi:v2 .
cd ..
```

3. Perform a Rolling Update in Kubernetes

Use `kubectl set image` to update the deployment:

```
kubectl set image deployment/backend-deployment backend=my-fastapi:v2
```

Monitor the rolling update process:

```
kubectl rollout status deployment/backend-deployment
```

Verify that the new version is serving traffic without downtime by refreshing the frontend page or using:

```
curl http://<backend-service-ip>:8000/
```

4. Rollback (Optional)

If desired, demonstrate a rollback:

```
kubectl rollout undo deployment/backend-deployment
```

Confirm the rollback is successful.

Submission Requirements

GitHub Repository including:
- Source code for the backend (`backend/`) and frontend (`frontend/`) folders.
- All Dockerfiles and Kubernetes YAML manifests.
- A README with detailed deployment instructions and upgrade procedures.

Screenshots of:
- `kubectl get pods` showing running pods before and after the upgrade.
- The frontend displaying data from the upgraded API.
- Output of `kubectl rollout status deployment/backend-deployment`