SETTING UP A CI/CD PIPELINE FOR AUTOMATED DEPLOYMENT

PHASE-3

College Name:Cambridge Institute Of Technology

Group Members:

1. **Name:** Keerthi
   CAN_ID: CAN_33856988

2. **Name:** Pallavi V
   CAN_ID: CAN_33854717

3. **Name:** Pranitha S
   CAN_ID: CAN_33856510

4. **Name:** Priya V
   CAN_ID: CAN_33853241

—---

SOLUTION DEVELOPMENT:

Step 1: Create IBM Cloud Account and Set Up the Environment

1. **Create IBM Cloud Account:**
   - Navigate to IBM Cloud.
   - Sign up for an account or log in if you already have one.
   - Ensure you have a billing account set up to access IBM Cloud services.
2. **Install Required Tools Locally:**
   - **Minikube:** Follow the Minikube installation guide.
   - **kubectl:** Download and install kubectl for Kubernetes CLI interaction.
   - **Docker:** Set up Docker for building and managing container images using the Docker installation guide.
3. **Set Up IBM Cloud Container Registry:**
   - From the IBM Cloud dashboard, search for "Container Registry."
   - Create a namespace for your container images:

     ```
     ibmcloud cr namespace-add <namespace_name>
     ```

4. **Enable image vulnerability scanning:**

   ```
   ibmcloud cr policy-update --scan-on-push true
   ```

Implementing Containerization and Pushing to IBM Cloud Container Registry

1. **Dockerize the Application:**

   Frontend Dockerfile (for React.js):
   Dockerfile

```
FROM node:16-alpine

WORKDIR /app

COPY . .

RUN npm install

EXPOSE 3000

CMD ["npm", "start"]
```

2. **Backend Dockerfile** (for Node.js, Express.js, OpenCV):

```
FROM node:16-alpine

WORKDIR /app

COPY . .

RUN npm install

EXPOSE 5000

CMD ["node", "server.js"]
```

3. **Build Docker Images:**

```
docker build -t frontend-app:1.0 ./frontend

docker build -t backend-app:1.0 ./backend
```

4. Push Docker Images to IBM Cloud Container Registry:

Tag the images:

```
docker tag frontend-app:1.0
<region>.icr.io/<namespace>/frontend-app:1.0

docker tag backend-app:1.0
<region>.icr.io/<namespace>/backend-app:1.0
```

- Log in to IBM Cloud Container Registry:
  ```
  ibmcloud cr login
  ```
- Push the images:
  ```
  docker push <region>.icr.io/<namespace>/frontend-app:1.0

  docker push <region>.icr.io/<namespace>/backend-app:1.0
  ```

TESTING THE SOLUTION

Step 1: Set Up Minikube and Deploy Applications

1. **Start Minikube:**`minikube start`
2. **Create Kubernetes Deployment and Service YAML Files:**

   Frontend Deployment (frontend-deployment.yaml):
   yaml
   ```
   apiVersion: apps/v1

   kind: Deployment

   metadata:

     name: frontend-deployment

   spec:

     replicas: 3

     selector:

       matchLabels:

         app: frontend

     template:

       metadata:

         labels:

           app: frontend

       spec:
   ```

```yaml
      containers:
      - name: frontend
        image: <region>.icr.io/<namespace>/frontend-app:1.0
        ports:
        - containerPort: 3000
---
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
spec:
  type: NodePort
  selector:
    app: frontend
  ports:
    - port: 3000
      targetPort: 3000
```

a. **Backend Deployment (backend-deployment.yaml):**
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: <region>.icr.io/<namespace>/backend-app:1.0
        ports:
        - containerPort: 5000
---
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  type: NodePort
  selector:
    app: backend
  ports:
    - port: 5000
```

```
      targetPort: 5000
```

b. **Apply YAML Files:**
   bash
   Copy code
   ```
   kubectl apply -f frontend-deployment.yaml
   ```

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

3. **Step 2: Verify Deployments**

   Check running pods:
   ```
   kubectl get pods
   ```

1. **Check services:**
   ```
   kubectl get svc
   ```
2. **Access applications:**

   Use Minikube's service IP or tunnel to access services:
   ```
   minikube service frontend-service --url

   minikube service backend-service --url
   ```

Step 3: Testing CI/CD Integration

1. **Set Up GitHub Actions:**
   ○ Create a `.github/workflows/ci-cd-pipeline.yml` file for automating the build, test, and deployment process.

   Example CI/CD pipeline YAML:
   yaml
   ```
   name: CI/CD Pipeline



   on:

     push:

       branches:

         - main



   jobs:

     build:
   ```

```
    runs-on: ubuntu-latest


    steps:
      - uses: actions/checkout@v2
      - name: Set up Node.js
        uses: actions/setup-node@v2
        with:
          node-version: '16'
      - name: Install dependencies
        run: npm install
      - name: Run tests
        run: npm test
      - name: Dockerize and push to IBM Cloud
        run: |
          docker build -t frontend-app:latest ./frontend
          docker tag frontend-app:latest
<region>.icr.io/<namespace>/frontend-app:latest
          docker push
<region>.icr.io/<namespace>/frontend-app:latest
          docker build -t backend-app:latest ./backend
          docker tag backend-app:latest
<region>.icr.io/<namespace>/backend-app:latest
          docker push
<region>.icr.io/<namespace>/backend-app:latest
```

2. **Step 4: Conduct Stress and Load Testing**
1. **Use tools like Apache JMeter or Postman** to simulate requests and measure response times.

2. **Monitor performance** using Minikube’s dashboard or tools like **Grafana** integrated with Prometheus.

FUTURE IMPROVEMENTS

1. **Enable Autoscaling for Minikube Clusters:**
    - Use **Kubernetes Horizontal Pod Autoscaler** to automatically scale pods based on load.
2. **Integrate Advanced CI/CD Pipelines:**
    - Use **Jenkins** or **Tekton Pipelines** to enhance automation and streamline the development pipeline.
3. **Add Automated Vulnerability Scanning:**
    - Integrate automated image vulnerability scanning during the CI/CD process to improve security.
4. **Implement Security Best Practices:**
    - Use **OpenSSL** for secure image management and encryption, signing Docker images to ensure only trusted versions are deployed.
