STREAMLINING CONTAINERIZED APPLICATION DEPLOYMENT ON IBM CLOUD KUBERNETES USING CONTAINER REGISTRY

PHASE 3- SOLUTION DEVELOPMENT AND TESTING

College Name: DDD

Group Members:

- **Name:** AAA
  CAN ID Number: BBB
- **Name:** XXX
  CAN ID Number: YYY

SOLUTION DEVELOPMENT:

Setting up IBM Cloud Environment and Configuring Necessary Tools

Step 1: Create an IBM Cloud Account

1. Navigate to IBM Cloud.
2. Sign up for an account (or log in if you already have one).
3. Ensure you have a billing account set up to access IBM Cloud services.

Step 2: Install Required Tools Locally

1. **Install Minikube**:
   - Follow the instructions from the Minikube installation guide.
2. **Install kubectl**:
   - Download and set up the kubectl CLI using the official guide.
3. **Install Docker**:
   - Set up Docker for building and managing container images (Docker installation guide).

Step 3: Set Up IBM Cloud Container Registry

1. From the IBM Cloud dashboard, search for "Container Registry."

2. Create a namespace for your container images:

ibmcloud cr namespace-add <namespace_name>

3. Enable image vulnerability scanning:

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

Implementing Containerization and Pushing to IBM Cloud Container Registry

Step 1: Dockerize the Application

1. **Create Dockerfiles**:
    - **Frontend Dockerfile** (/public/Dockerfile):

FROM node:16-alpine

WORKDIR /app

COPY . .

RUN npm install

EXPOSE 3000

CMD ["npm", "start"]

    - **Backend Dockerfile** (/server/Dockerfile):

FROM node:16-alpine

WORKDIR /app

COPY . .

RUN npm install

EXPOSE 5000

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

2. **Build Docker Images**:

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

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

Step 2: Push Docker Images to IBM Cloud Container Registry

1. 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

2. Log in to IBM Cloud Container Registry:

ibmcloud cr login

3. Push the images:

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

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

SECTION 2: 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):

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
```

```
    labels:
      app: frontend
  spec:
    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
```

- **Backend Deployment** (backend-deployment.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
```

3. **Apply YAML Files**:

kubectl apply -f frontend-deployment.yaml

kubectl apply -f backend-deployment.yaml

Step 2: Verify Deployments

1. Check running pods:

kubectl get pods

2. Check services:

kubectl get svc

3. Access applications:
    - Use the Minikube service IP or tunnel to expose services.

Step 3: Testing CI/CD Integration

1. Set up GitHub Actions with a CI/CD pipeline YAML file.
2. Automate build, test, and deployment stages using Minikube and IBM Cloud CLI.

Step 4: Conduct Stress and Load Testing

1. Use tools like Apache JMeter or Postman.
2. Monitor performance using Minikube's dashboard or tools like Grafana.

SECTION 3: FUTURE IMPROVEMENTS

1. Enable autoscaling for Minikube clusters (e.g., using Kubernetes Horizontal Pod Autoscaler).
2. Integrate advanced CI/CD pipelines with Jenkins or Tekton Pipelines.
3. Add automated vulnerability scanning during the CI/CD process.