STREAMLINING CONTAINERIZED APPLICATION DEPLOYMENT ON IBM CLOUD KUBERNETES USING CONTAINER REGISTRY

PHASE 4- FINAL PHASE DOCUMENT

College Name: S G Balekundri Institute of Technology

Group Members:

- **Name: Gousiya Ibrahim Desai**
  CAN_ID: CAN_33846252
  Contribution: Website design and Docker Containerization

- **Name: Bibiraheema Nadaf**
  CAN_ID: CAN_33317104
  Contribution: Website Development and Github Setup

- **Name: Galshed A Momin**
  CAN_ID: CAN_33836869
  Contribution: Code Testing

- **Name: Mohammadsadik Momin**
  CAN_ID: CAN_33845454
  Contribution: Deployment and orchestration using Kubernetes

---

1. Overview of Containerized Application Deployment Key Components:

- **Containerization:** Package the image uploader application into containers for consistent, portable deployments.
- **IBM Cloud Container Registry:** Store and manage container images with Kubernetes.
- **Automation & CI/CD:** Automate the build, push, and deployment pipeline for rapid and reliable application deployment.

2. Configuring IBM Cloud Kubernetes and Container Registry

2.1 Steps to Set Up IBM Cloud Kubernetes Service (IKS)

1. **Create an IBM Cloud Account & Log In**
2. **Provision the Kubernetes Cluster**
3. **Integrate IBM Cloud Container Registry (ICR)**
4. **Create a Dockerfile**

Create a Dockerfile in the root directory of your image uploader project. Below is an example for a Python Flask-based image uploader application:

```
# Use an official Python runtime as the base image

FROM python:3.9-slim


# Set the working directory in the container

WORKDIR /app


# Copy the requirements.txt file

COPY requirements.txt /app/


# Install dependencies

RUN pip install --no-cache-dir -r requirements.txt


# Copy the entire project into the container

COPY . /app/


# Expose the port Flask will run on

EXPOSE 5000


# Default command to run the Flask application

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

5. **Build the Docker Image**

docker build -t username/image-uploader:latest .

6. **Tag the Image for IBM Cloud Container Registry:**

docker tag username/image-uploader:latest us.icr.io/username/image-uploader:latest

7. **Push the Docker Image to IBM Cloud Container Registry:**

ibmcloud cr login

docker push us.icr.io/username/image-uploader:latest

3. Deploying Containers on IBM Kubernetes

3.1 Create Kubernetes Deployment and Service

1. **Deployment YAML:** Create a deployment.yaml file for Kubernetes deployment.

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

```
    labels:
      app: image-uploader
  spec:
    containers:
    - name: image-uploader
      image: us.icr.io/username/image-uploader:latest
      ports:
      - containerPort: 5000
```

2. Service YAML: Create a service.yaml file to expose the application.

```
apiVersion: v1
kind: Service
metadata:
  name: image-uploader-service
spec:
  selector:
    app: image-uploader
  ports:
    - protocol: TCP
      port: 80
      targetPort: 5000
  type: LoadBalancer
```

3. Deploy to Kubernetes:

kubectl apply -f deployment.yaml

kubectl apply -f service.yaml

4. Verify Deployment:

kubectl get pods

kubectl get svc

4. Automating the Deployment with CI/CD Pipeline

4.1 Integrating with IBM Cloud Continuous Delivery

1. **Create a Delivery Pipeline:**
   - Navigate to **IBM Cloud Dashboard > Continuous Delivery**.
   - Create a new pipeline to automatically build and deploy the application upon code changes.
2. **Pipeline Configuration:**
   - **Step 1:** Add a Build stage to build the Docker image.
   - **Step 2:** Add a Deploy stage to deploy the image to IBM Kubernetes cluster using kubectl.
   -
3. **Trigger the Pipeline:**
   - Set up GitHub webhooks to trigger the pipeline on code commits.

5. User Interface Development for Monitoring and Management

To monitor Kubernetes deployments, integrate tools like IBM Cloud Monitoring or Prometheus with Grafana.

1. **IBM Cloud Monitoring:**
   - Track the health of Kubernetes clusters and deployed applications.
   - Set up alerts for failures, resource exhaustion, or scaling issues.
2. **Prometheus and Grafana:**
   - Install Prometheus and Grafana on the Kubernetes cluster to monitor application performance and visualize key metrics.

6. IBM Cloud Platform Features and Considerations

Feature	Benefits	Best Practices
Scalability	Handles fluctuating workloads and large datasets.	Enable auto-scaling and monitor cluster usage.
Security	Protects sensitive data with IAM roles and encryption.	Use least privilege policies, enable encryption, and enforce MFA for admin roles.
Monitoring	Tracks containerized application performance and health.	Set alerts for critical metrics and use dashboards for proactive issue resolution.
Cost Efficiency	Optimizes resource allocation and storage.	Analyze usage patterns and adjust scaling and storage policies accordingly.

7. Conclusion

This project enables scalable and efficient deployment of the **Image Uploader Application** using **IBM Cloud Kubernetes Service (IKS) and IBM Cloud Container Registry (ICR)**. By leveraging automation, monitoring, and security best practices, the system ensures reliable and cost-effective operations.

8. Further Enhancements

- **Automated Rollbacks:** Implement rollback mechanisms in case of failed deployments.
- **Multi-Cluster Management:** Extend the deployment to manage multiple Kubernetes clusters.
- **Cost Optimization:** Use cost management tools to optimize cloud resource allocation.

GitHub Repositor: https://github.com/Gousiyadesai/Cloud-Kubernets-.git