ADOPTING A MULTI-CLOUD STRATEGY WITH DOCKER AND KUBERNETES

PHASE 4 - SOLUTION ARCHITECTURE

College Name: K S Institute of Technology

Group Members:

- **Name: Challa Deepika**

  CAN ID Number: CAN-33689518

- **Name: Vidheesha T M**

  CAN ID Number: CAN-33689420

- **Name: Arun A**

  CAN ID Number: CAN-33672150

- **Name: Chandrasekar A**

  CAN ID Number: CAN-33671259

---

1. OVERVIEW OF MULTI-CLOUD APPLICATION DEPLOYMENT

This phase focuses on deploying containerized applications across IBM Cloud, AWS, and Azure using Kubernetes, Ansible, and CI/CD automation. The goal is to establish a scalable, resilient, and automated pipeline for multi-cloud application deployment.

Key Components:

- Containerization: Package applications into portable containers.
- Multi-Cloud Registries: Store and manage images across IBM Cloud, AWS, and Azure.
- Kubernetes Clusters: Deploy and scale workloads with Kubernetes.
- CI/CD Automation: Streamline build, push, and deployment pipelines with Ansible.

2. CONFIGURING MULTI-CLOUD KUBERNETES AND CONTAINER REGISTRIES

2.1 Steps to Set Up Kubernetes Clusters Across Clouds

1. Create Cloud Accounts & Log In
    - Register and log in to IBM Cloud, AWS, and Azure.

2. Provision Kubernetes Clusters
    - Create Kubernetes clusters using:
3. ibmcloud ks cluster config --cluster <cluster-name>
4. aws eks update-kubeconfig --region <region> --name <cluster-name>
5. az aks get-credentials --resource-group <resource-group> --name <cluster-name>
6. Integrate Cloud Container Registries
    - Set up private registries for each cloud:
7. ibmcloud cr namespace-add <namespace_name>
8. aws ecr create-repository --repository-name <repository_name>
9. az acr create --name <registry_name> --resource-group <resource_group> --sku Basic

3. CONTAINERIZING APPLICATIONS WITH DOCKER

3.1 Create Dockerfiles

Example for a Python Flask app:

FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt /app/requirements.txt

RUN pip install -r requirements.txt

COPY . /app

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

3.2 Build and Push Images

docker build -t myapp:latest .

docker tag myapp:latest <REGISTRY_URL>/<namespace>/myapp:latest

docker push <REGISTRY_URL>/<namespace>/myapp:latest

4. DEPLOYING CONTAINERS ACROSS MULTI-CLOUD KUBERNETES CLUSTERS

4.1 Create Kubernetes Deployment and Service Files

Deployment YAML

apiVersion: apps/v1

```
kind: Deployment
metadata:
  name: myapp-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp
        image: <REGISTRY_URL>/<namespace>/myapp:latest
        ports:
        - containerPort: 5000
```

Service YAML

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

type: LoadBalancer

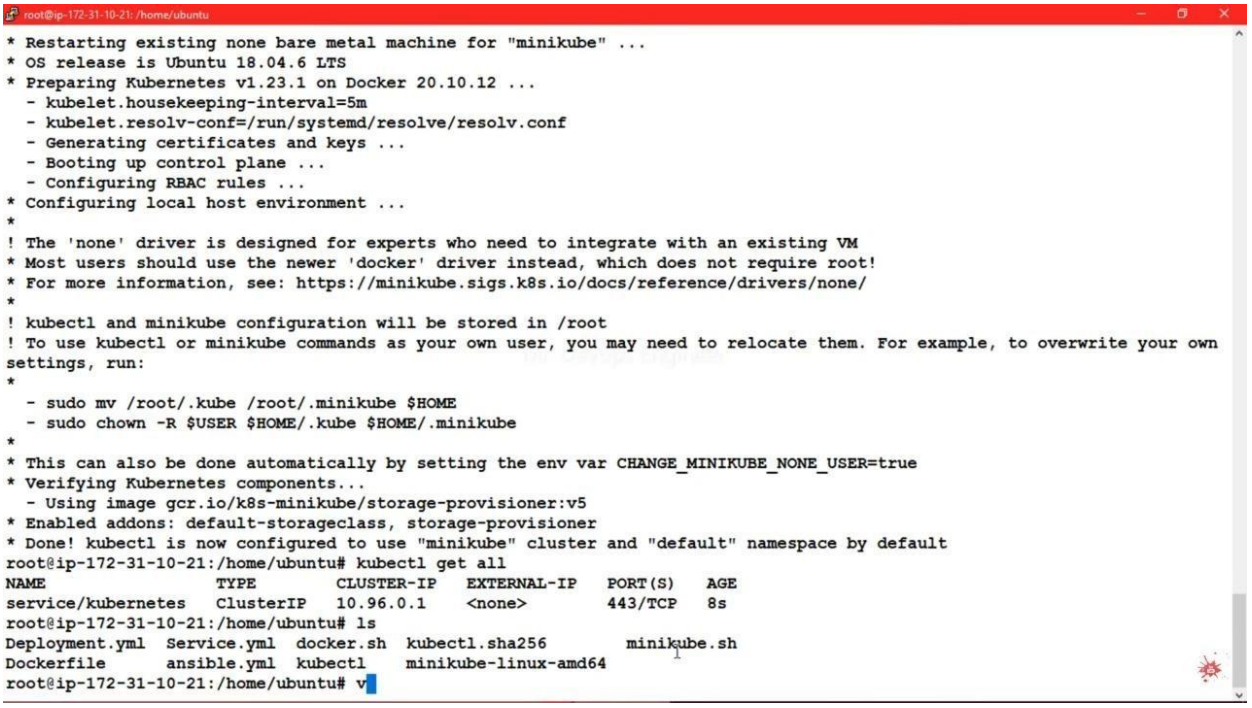
4.2 Deploy and Verify

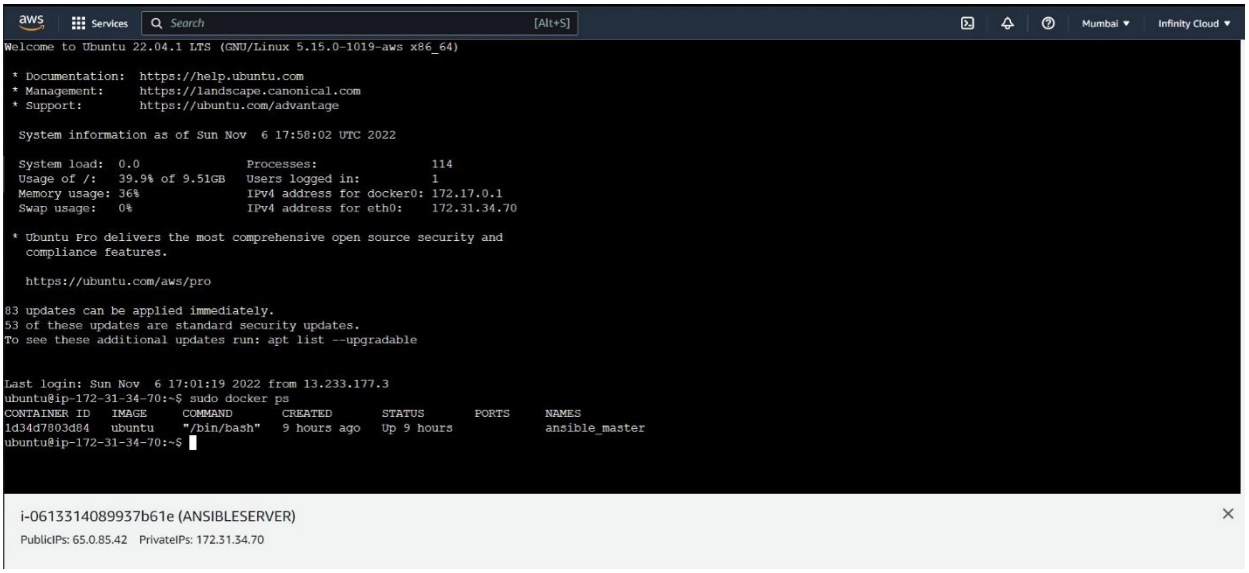
kubectl apply -f Deployment.yml

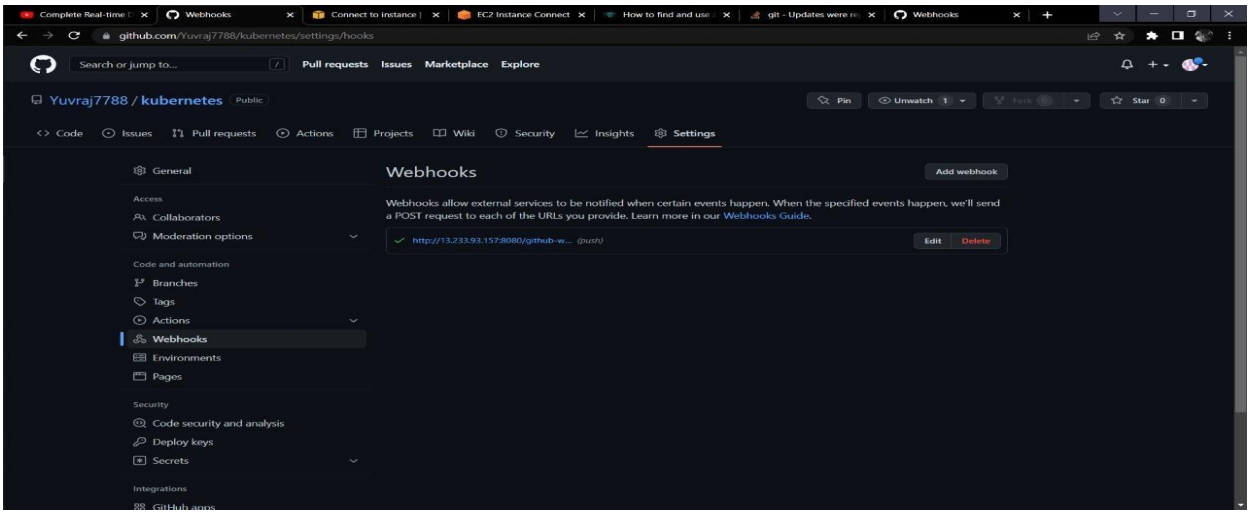
kubectl apply -f Service.yml

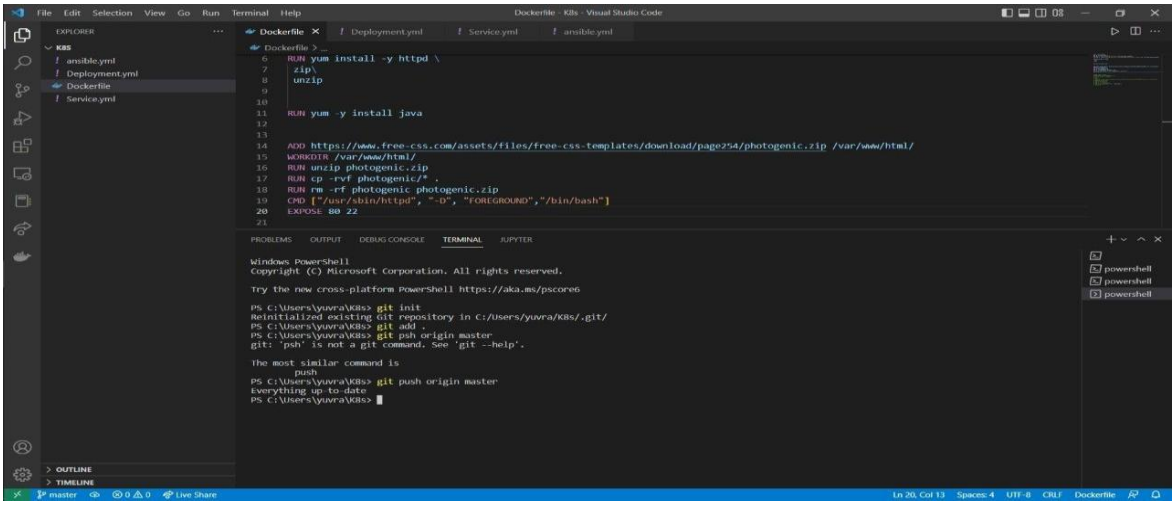
kubectl get pods

kubectl get svc









5. AUTOMATING MULTI-CLOUD DEPLOYMENT WITH CI/CD

5.1 Multi-Cloud CI/CD Integration

Create CI/CD Pipelines

- Use Jenkins, GitHub Actions, or IBM Continuous Delivery.
- Implement Ansible-based Kubernetes deployment:

```
- hosts: all
  become: true
  tasks:
    - name: create new deployment
      command: kubectl apply -f /home/ubuntu/Deployment.yml
    - name: create new service
```

command: kubectl apply -f /home/ubuntu/Service.yml

Pipeline Stages:

1. Build Stage: Build Docker images.
2. Push Stage: Push images to cloud registries.
3. Deploy Stage: Deploy to Kubernetes clusters using Ansible.

Trigger Pipelines

- Automate CI/CD pipelines on code commits using webhooks.

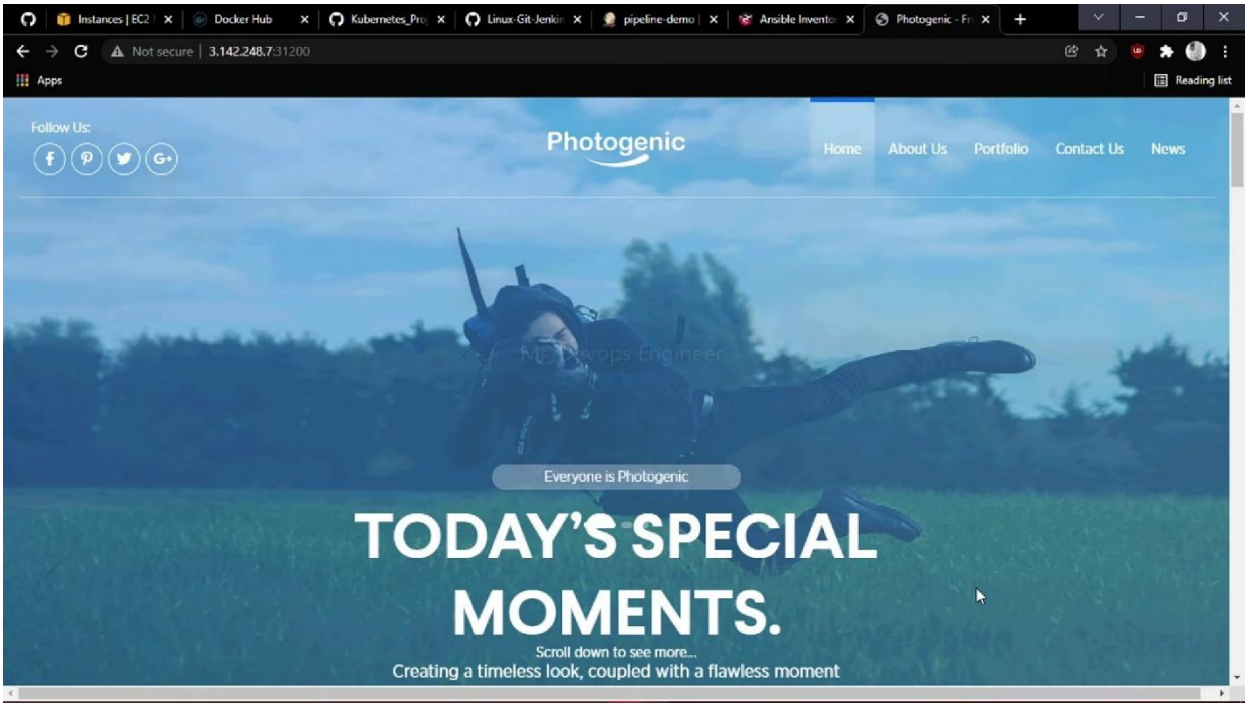
6. MONITORING AND MANAGEMENT

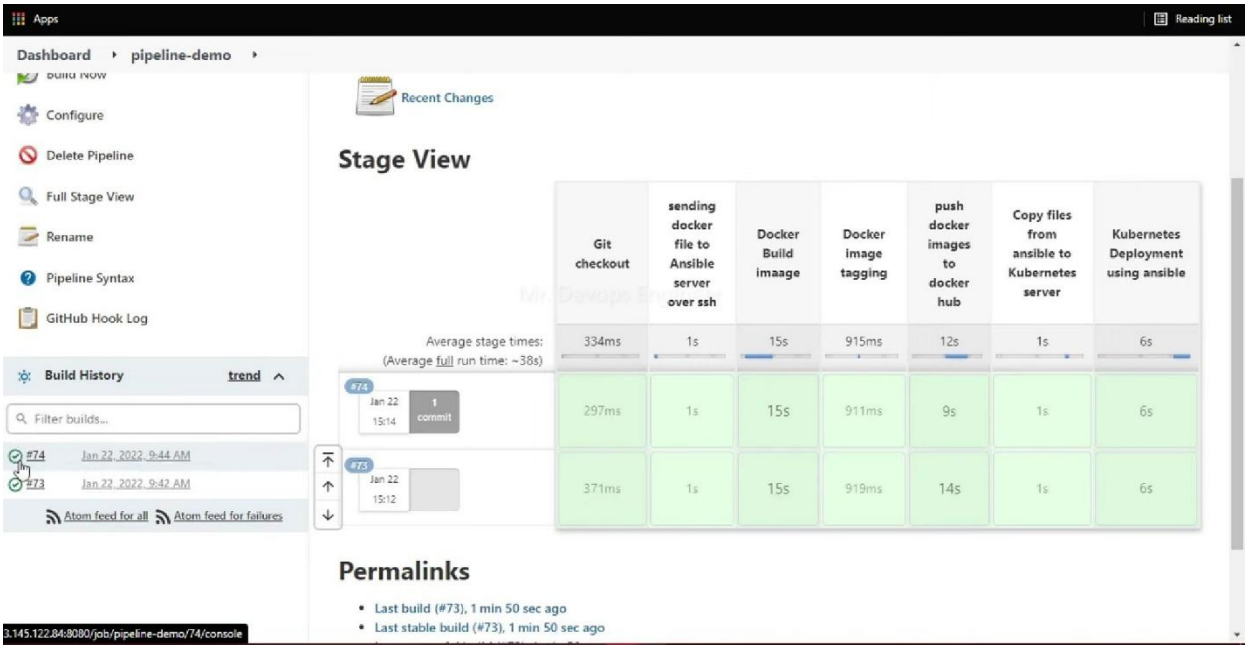
6.1 Tools for Monitoring Multi-Cloud Deployments

- IBM Cloud Monitoring: Monitor Kubernetes cluster health and resource usage.
- Prometheus & Grafana: Create dashboards for tracking application performance.

7. MULTI-CLOUD FEATURES AND CONSIDERATIONS

Feature	**Benefits**	**Best Practices**
Scalability	Handles fluctuating workloads	Enable auto-scaling & monitor usage
Security	Protects sensitive data	Use IAM, MFA, and encryption
Monitoring	Tracks app performance	Set alerts & use dashboards
Cost Efficiency	Optimizes resource usage	Analyze patterns & adjust scaling
Feature	Benefits	Best Practices





8. CONCLUSION

By adopting a multi-cloud strategy with Docker, Kubernetes, and Ansible, organizations can achieve scalable, automated, and resilient deployments across cloud environments.