

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

DEVOPS ENGINEER

PHASE 4

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

DEVOPS ENGINEER

PHASE 4

type: LoadBalancer

4.2 Deploy and Verify

kubectl apply -f Deployment.yml

kubectl apply -f Service.yml

kubectl get pods

kubectl get svc

```
root@ip-172-31-10-21: /home/ubuntu
* Restarting existing none bare metal machine for "minikube" ...
* OS release is Ubuntu 18.04.6 LTS
* Preparing Kubernetes v1.23.1 on Docker 20.10.12 ...
  - kubelet.housekeeping-interval=5m
  - kubelet.resolv-conf=/run/systemd/resolve/resolv.conf
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
* Configuring local host environment ...
*
! The 'none' driver is designed for experts who need to integrate with an existing VM
* Most users should use the newer 'docker' driver instead, which does not require root!
* For more information, see: https://minikube.sigs.k8s.io/docs/reference/drivers/none/
*
! kubectl and minikube configuration will be stored in /root
! To use kubectl or minikube commands as your own user, you may need to relocate them. For example, to overwrite your own
settings, run:
*
  - sudo mv /root/.kube /root/.minikube $HOME
  - sudo chown -R $USER $HOME/.kube $HOME/.minikube
*
* This can also be done automatically by setting the env var CHANGE_MINIKUBE_NONE_USER=true
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Enabled addons: default-storageclass, storage-provisioner
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
root@ip-172-31-10-21: /home/ubuntu# kubectl get all
NAME                TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
service/kubernetes  ClusterIP     10.96.0.1    <none>        443/TCP    8s
root@ip-172-31-10-21: /home/ubuntu# ls
Deployment.yml  Service.yml  docker.sh  kubectl.sha256  minikube.sh
Dockerfile     ansible.yml  kubectl    minikube-linux-amd64
root@ip-172-31-10-21: /home/ubuntu# v
```

```
aws  Services  Search  [Alt+S]  Mumbai  Infinity Cloud
Welcome to Ubuntu 22.04.1 LTS (GNU/Linux 5.15.0-1019-aws x86_64)

* Documentation: https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:       https://ubuntu.com/advantage

System information as of Sun Nov  6 17:58:02 UTC 2022

System load: 0.0          Processes: 114
Usage of /:  39.9% of 9.51GB  Users logged in: 1
Memory usage: 36%          IPv4 address for docker0: 172.17.0.1
Swap usage: 0%             IPv4 address for eth0: 172.31.34.70

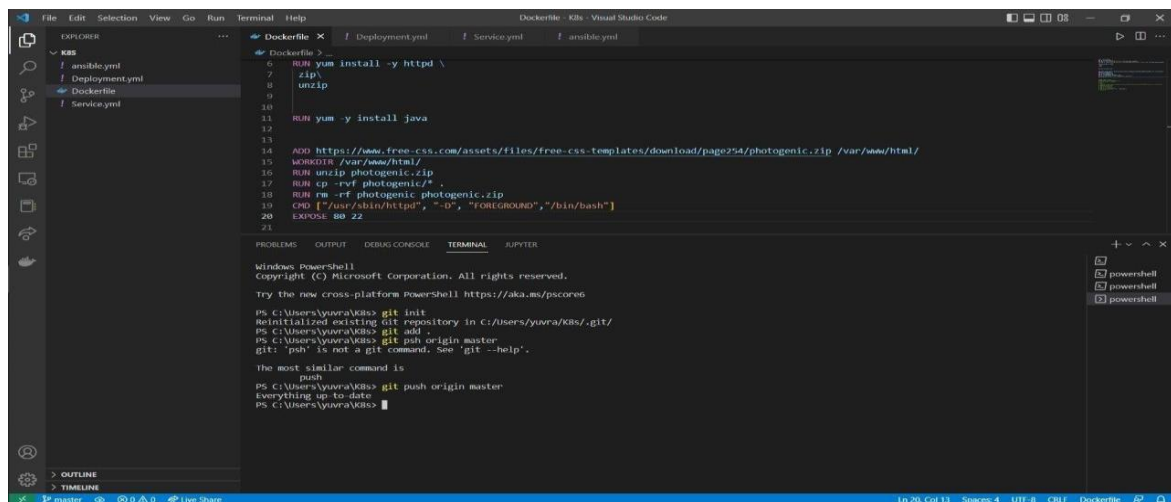
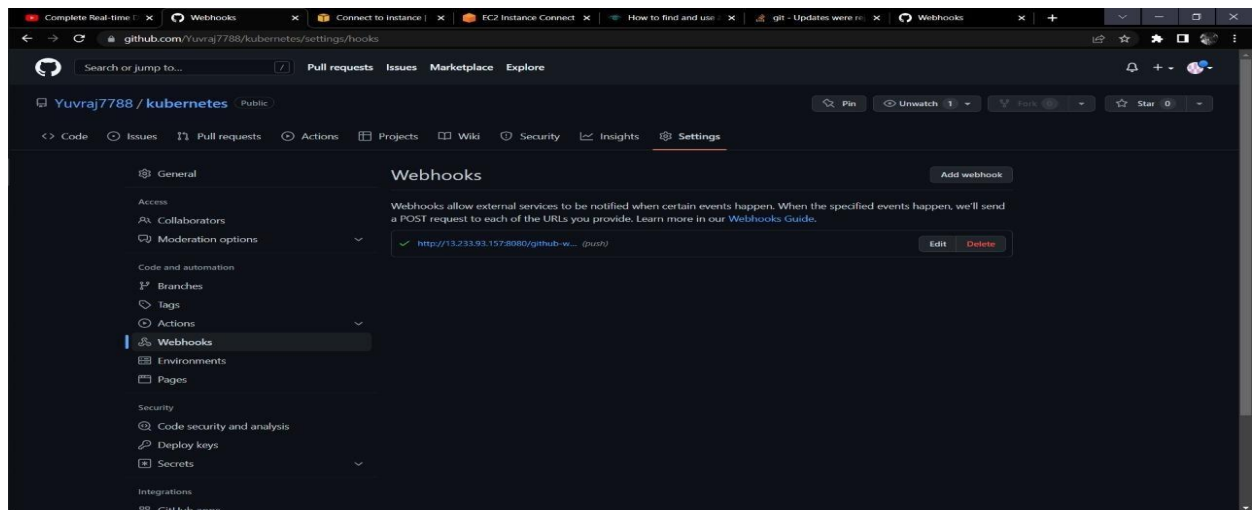
* Ubuntu Pro delivers the most comprehensive open source security and
compliance features.

https://ubuntu.com/aws/pro

83 updates can be applied immediately.
53 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Last login: Sun Nov  6 17:01:19 2022 from 13.233.177.3
ubuntu@ip-172-31-34-70:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
1d34d7803d84   ubuntu   "/bin/bash"   9 hours ago   Up 9 hours   ansible_master
ubuntu@ip-172-31-34-70:~$
```

i-0613314089937b61e (ANSIBLESERVER)
PublicIPs: 65.0.85.42 PrivateIPs: 172.31.34.70



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.

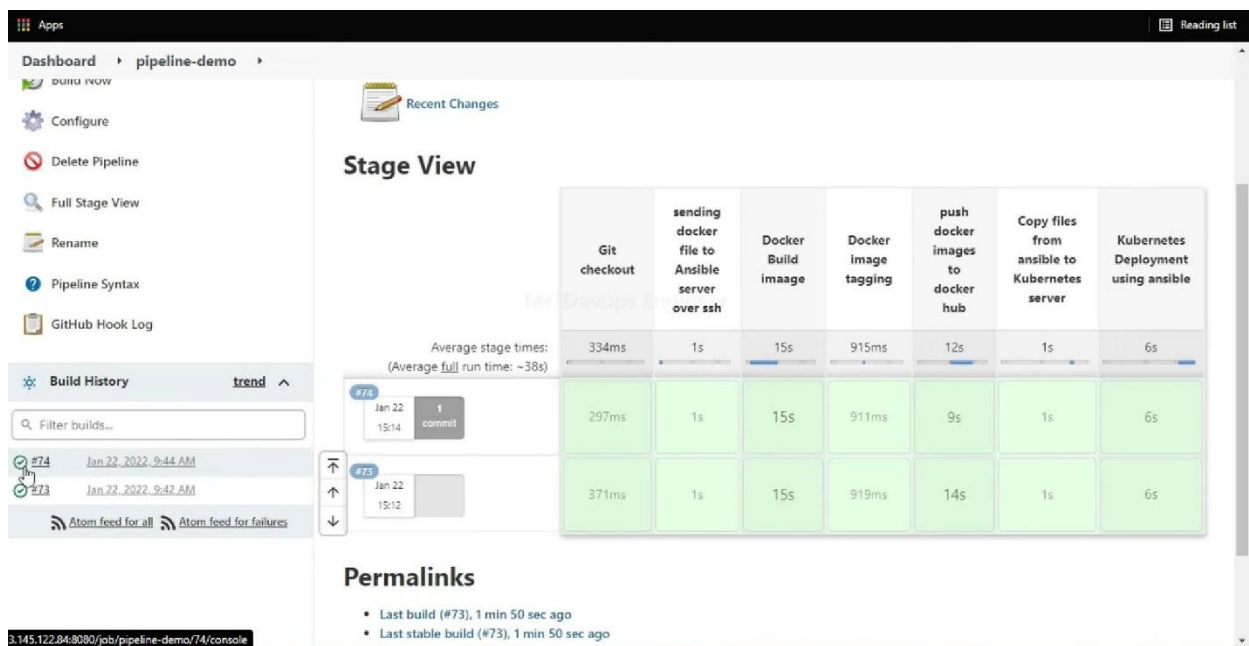
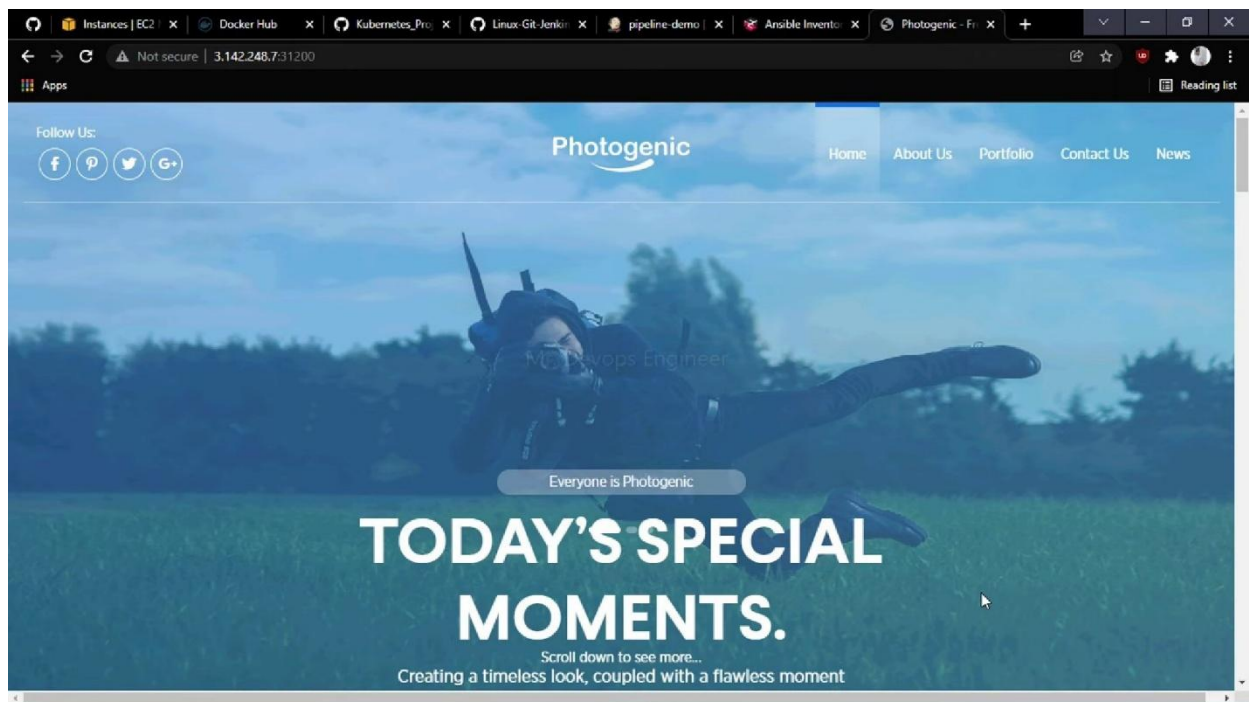
6. MONITORING AND MANAGEMENT

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.