Leveraging Docker and Kubernetes for a Multi-Cloud Strategy

College Name: Vijaya Vittala Institute of Technology

Group Members:

- **Name: Narasimhamurthy D V**
  CAN ID: 33837191
- **Name: P P Jyothsna Sai**
  CAN ID: 34043056
- **Name: Mahalaksmi A**
  CAN ID: 33822962

SOLUTION ARCHITECTURE

To streamline the deployment process for a multi-cloud e-commerce application, we will leverage Docker and Kubernetes to ensure flexibility, scalability, and high availability across multiple cloud providers such as IBM Cloud, AWS, and Google Cloud Platform (GCP). The solution architecture will include a well-defined directory structure, CI/CD pipeline automation, and multi-cloud deployment strategies.

Application Structure

```
multi-cloud-app/
├── frontend/
|   ├── public/
|   |   ├── css/
|   |   |   └── style.css
|   |   ├── js/
|   |   |   └── app.js
|   |   └── index.html
|   ├── src/
|   |   ├── components/
|   |   └── App.js
|   └── package.json
├── backend/
|   ├── controllers/
```

```
|   |   └── apiController.js
|   ├── models/
|   |   └── dataModel.js
|   ├── routes/
|   |   └── apiRoutes.js
|   └── server.js
├── Dockerfile
├── docker-compose.yml
└── README.md
```

Version Control Setup

1. Initialize Git in the project: `git init`

2. Create a .gitignore file: `echo node_modules/ > .gitignore && echo .env > .gitignore`

3. Add files and commit: `git add . && git commit -m 'Initial commit of multi-cloud-app structure'`

4. Push to GitHub: `git remote add origin <repository_url> && git push -u origin master`

CI/CD Pipeline for Multi-Cloud Deployment

To automate deployment across multiple clouds, we will use Jenkins and GitHub Actions.

Jenkins Setup

Install Jenkins with necessary plugins (Docker, Git, Kubernetes CLI).

Set up a Jenkins job that triggers on code changes.

Jenkinsfile Example

```
pipeline {
    agent any
    environment {
        DOCKER_IMAGE = 'multi-cloud-app'
        AWS_REGISTRY = '<AWS_ECR_URL>'
        GCP_REGISTRY = '<GCP_GCR_URL>'
        IBM_REGISTRY = '<IBM_Container_Registry_URL>'
        CLUSTER_AWS = '<AWS_CLUSTER_NAME>'
        CLUSTER_GCP = '<GCP_CLUSTER_NAME>'
        CLUSTER_IBM = '<IBM_CLUSTER_NAME>'
    }
```

```
stages {
  stage('Checkout') {
    steps {
      git 'https://github.com/<username>/multi-cloud-app.git'
    }
  }
  stage('Build Docker Image') {
    steps {
      script {
        sh 'docker build -t $DOCKER_IMAGE .'
      }
    }
  }
  stage('Push Docker Image to Multi-Cloud Registries') {
    steps {
      script {
        sh 'docker tag $DOCKER_IMAGE $AWS_REGISTRY/$DOCKER_IMAGE'
        sh 'docker push $AWS_REGISTRY/$DOCKER_IMAGE'
        sh 'docker tag $DOCKER_IMAGE $GCP_REGISTRY/$DOCKER_IMAGE'
        sh 'docker push $GCP_REGISTRY/$DOCKER_IMAGE'
        sh 'docker tag $DOCKER_IMAGE $IBM_REGISTRY/$DOCKER_IMAGE'
        sh 'docker push $IBM_REGISTRY/$DOCKER_IMAGE'
      }
    }
  }
  stage('Deploy to Kubernetes Clusters') {
    steps {
      script {
        sh '''
        kubectl config use-context aws
        kubectl apply -f k8s/deployment.yaml
        kubectl config use-context gcp
        kubectl apply -f k8s/deployment.yaml
        kubectl config use-context ibm
        kubectl apply -f k8s/deployment.yaml
        '''
      }
    }
  }
}
post {
  success {
```

```
            echo 'Multi-cloud deployment successful.'
        }
        failure {
            echo 'Deployment failed. Please check logs.'
        }
    }
}
```

Future Plan

1. **Container Image Management**: Utilize AWS ECR, Google Container Registry, and IBM Cloud Container Registry for centralized image storage.

2. **Multi-Cluster Deployment**: Deploy the Dockerized application across AWS EKS, Google GKE, and IBM Kubernetes Service.

3. **Monitoring & Logging**: Implement Prometheus and Grafana for monitoring and centralized logging with ELK Stack.

4. **Security Enhancements**: Use OpenSSL and vulnerability scanning tools for securing container images.

5. **CI/CD Optimization**: Automate testing and deployments with GitHub Actions, Jenkins, and IBM Cloud Continuous Delivery.