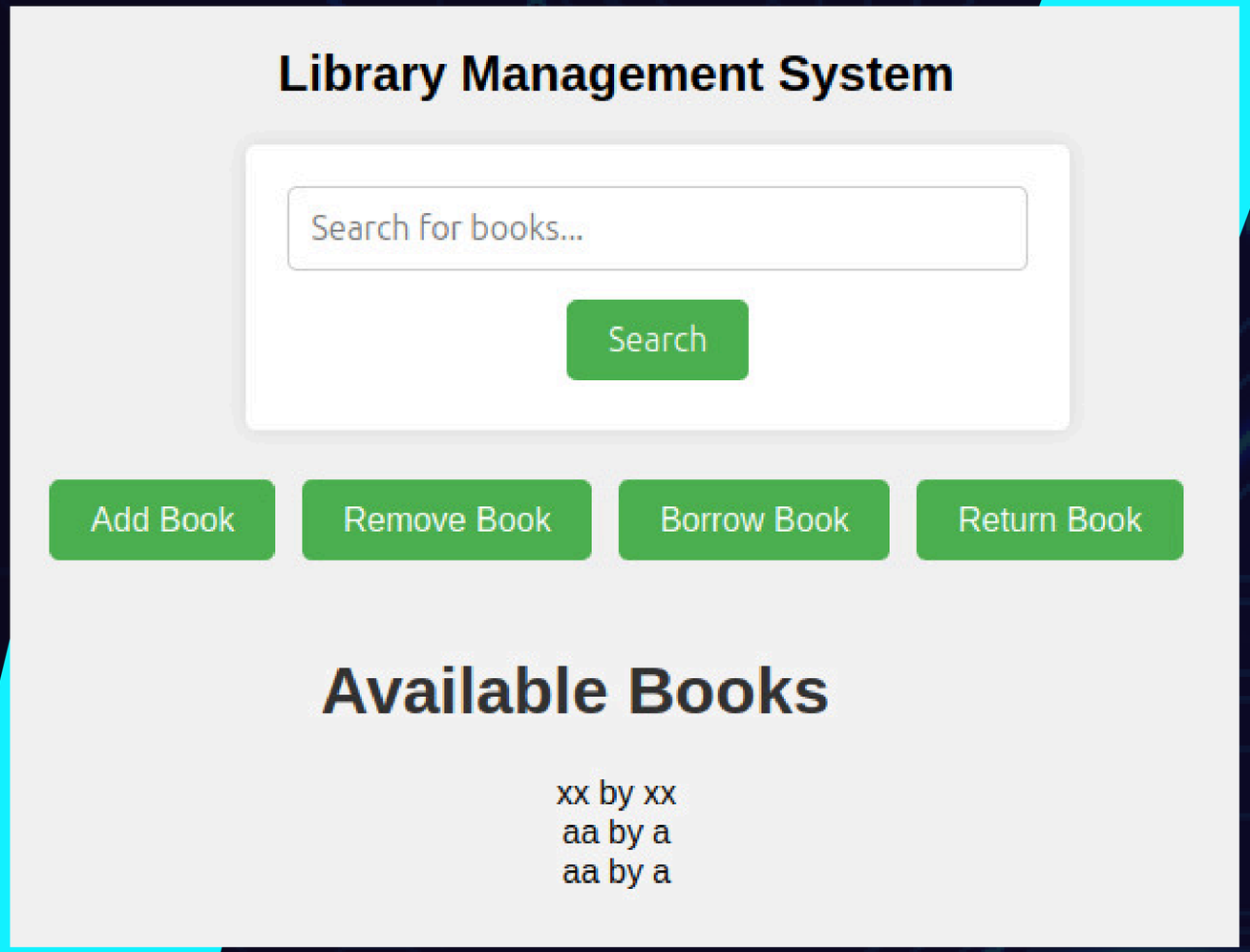
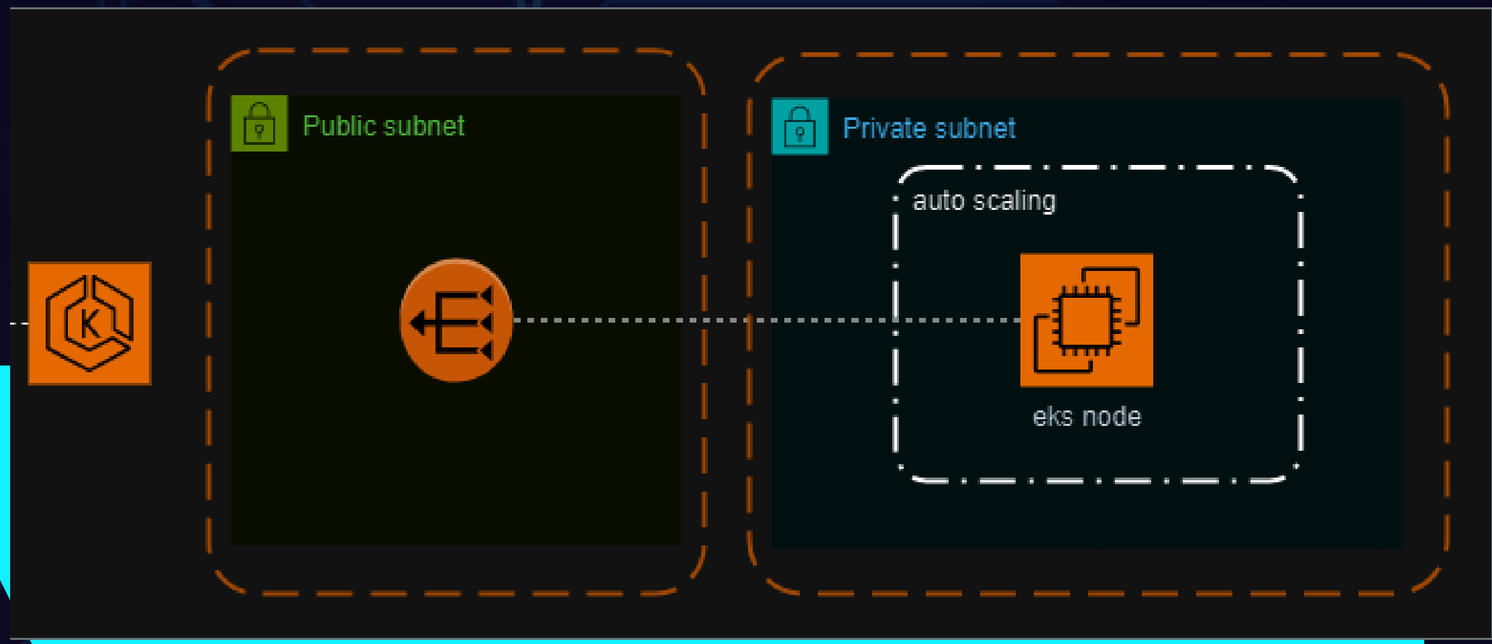
LIBRARY



APPLICATION DEVELOPMENT

INFRASTRUCTURE TERRAFORM FOR AWS EKS



01 EKS Cluster: The main Kubernetes control plane. EKS Node Group: Worker nodes that run the application workloads, auto-scaled based on demand.

02 Terraform provides the automation and infrastructure as code (IaC) capabilities, making the deployment process repeatable and manageable.

03 we achieve a robust, scalable, and secure Kubernetes environment on AWS, ready for deploying and managing containerized applications efficiently.

APP.YAML

This configuration sets up a Flask application with persistent storage, scalable deployment, and external access via a load balancer. The Persistent Volume and Persistent Volume Claim ensure data persistence, while the Deployment and Service manage application scalability and accessibility.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: new-flask-app-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: flask-app
  template:
    metadata:
      labels:
        app: flask-app
    spec:
      containers:
        - name: flask-app
          image: sarahassan11/myflask:latest
          ports:
            - containerPort: 5000
          volumeMounts:
            - mountPath: /data
              name: flask-app-storage
```

```yaml
# Persistent Volume
apiVersion: v1
kind: PersistentVolume
metadata:
  name: flask-app-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /data
---
```

```yaml
---
# Persistent Volume Claim
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: flask-app-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
---
# Deployment
```

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

CI/CD PIPELINE

01

Purpose

The CI/CD pipeline automates the build, push, and deployment processes for a Dockerized Flask application. This ensures efficient, error-free deployment, and continuous integration and continuous deployment (CI/CD) practices.

```
environment {
    registry = "veles3/library"
    registryCredential = 'dockerhub'
    dockerImage = ''
    kubeconfig = credentials('kubeconfig') // Jenkins credentials with your kubeconfig file
}
```

02

Key Components

1. **Environment Variables:**
   - registry: Specifies the Docker Hub registry name.
   - registryCredential: Jenkins credentials ID for Docker Hub authentication.
   - dockerImage: Holds the dynamically set Docker image name during the build stage.
   - kubeconfig: Jenkins credentials ID for the Kubernetes configuration file.
2. **Pipeline Stages:**
   - Build Docker Image: Builds the Docker image from the source code.
   - Push Docker Image: Pushes the built Docker image to Docker Hub.
   - Deploy to EKS: Deploys the Docker image to an AWS EKS (Elastic Kubernetes Service) cluster.
   - Cleanup: Cleans up local Docker environment by removing built images.
   - Post Actions: Ensures workspace cleanup after the pipeline run.

```
stages {
    stage('Build Docker Image') {
        steps {
            script {
                dockerImage = "${registry}:${env.BUILD_NUMBER}"
                sh "docker build -t ${dockerImage} ."
            }
        }
    }
    stage('Push Docker Image') {
        steps {
            script {
                withCredentials([usernamePassword(credentialsId: registryCredential, passwordVariable: 'DOCKER_PASSWORD', usernameVariable: 'DOCKER_USERNAME')]) {
                    sh "echo $DOCKER_PASSWORD | docker login -u $DOCKER_USERNAME --password-stdin"
                    sh "docker push ${dockerImage}"
                    sh "docker tag ${dockerImage} ${registry}:latest"
                    sh "docker push ${registry}:latest"
                }
            }
        }
    }
```

```
stage('Deploy to EKS') {
    steps {
        script {
            withCredentials([file(credentialsId: 'kubeconfig', variable: 'KUBECONFIG'),
                             [$class: 'AmazonWebServicesCredentialsBinding', credentialsId: 'aws-credentials']]) {
                // Echo the KUBECONFIG path for debugging
                sh "echo KUBECONFIG=$KUBECONFIG"
                // Update Kubernetes deployment with the new image
                sh """
                kubectl set image deployment/coredns coredns=${dockerImage} --namespace=kube-system --kubeconfig $KUBECONFIG
                """
            }
        }
    }
}
stage('Cleanup') {
    steps {
        script {
            sh "docker rmi ${dockerImage}"
            sh "docker rmi ${registry}:latest"
        }
    }
}
```

```
post {
    always {
        script {
            cleanWs()
        }
    }
}
```

THANK YOU