Managing Security Across Multiple Environments with DevSecOps

PHASE 2- SOLUTION ARCHITECTURE

College Name: Shetty Institute of Technology Kalaburagi.

Group Members:

- **Name:** ABHISHEK S M

  CAN ID Number: CAN_33981242

  WORK: CI/CD PIPELINE DESIGN AND IMPLEMENTATION

- **Name:** BHAGYASHREE M HANDI

  CAN ID Number: CAN_33254590

- **Name:** SHARANU

  CAN ID Number: CAN_33489513

- **Name:** SHASHIKIRAN

  CAN ID Number: CAN_33287479

SOLUTION ARCHITECTURE

To streamline the deployment process for the project "Managing Security Across Multiple Environments with DevSecOps," we will establish version control, automate code commits, and set up a CI/CD pipeline. The solution architecture leverages tools like Jenkins, Docker, Kubernetes, SonarQube, OWASP ZAP, Hashi Corp Vault, and Snyk/Trivy.

In Windows Command Prompt, we can use the mkdir command to create directories for the application. Here’s how you can set up the project structure:

1. **Create the main project folder:**

```
mkdir devsecops-app

cd devsecops-app
```

2. **Create the security-tools folder for vulnerability scanning scripts:**

```
mkdir security-tools
```

3. **Create a folder for Jenkins pipeline scripts:**

```
 mkdir Jenkins
```

4. **Create a folder for Kubernetes configuration files:**

```
mkdir k8s
```

5. **Create additional necessary files in the project directory:**

```
echo. > Dockerfile
echo. > Jenkinsfile
echo. > README.md
```

After executing the above commands, your directory structure should look as follows:

```
devsecops-app/

├── security-tools/

├── jenkins/

├── k8s/

├── Dockerfile

├── Jenkinsfile

└── README.md
```

VERSION CONTROL SETUP

To ensure that the development team is working collaboratively and tracking changes efficiently, we will set up a **GitHub repository** for version control.

1. **Initialize Git in the project:**

```
git init
```

2. **Create a .gitignore file to avoid committing unnecessary files:**

```
echo node_modules/ > .gitignore

echo .env > .gitignore
```

3. **Add files to Git:**

```
git add .
```

4. **Commit the initial codebase:**

```
git commit -m "Initial commit of devsecops-app structure"
```

5. **Create a GitHub repository** (via GitHub's web interface).

6. **Push the local repository to GitHub:**

```
git remote add origin <repository_url>

git push -u origin master
```

CI/CD PIPELINE DESIGN AND IMPLEMENTATION

To automate the build, test, and deployment processes, we will design a CI/CD pipeline using Jenkins.

1. Jenkins Setup

- Install Jenkins on a local Windows machine.
- Configure necessary plugins like Docker, Git, SonarQube, Trivy, OWASP ZAP, and Kubernetes CLI.
- Set up a Jenkins job that triggers on code changes pushed to the GitHub repository.

2. Jenkins Pipeline Creation

- Create a Jenkinsfile in the root of the project, which will define the steps to build, test, and deploy the application.
- The Jenkinsfile will include the following stages:
  - **Checkout:** Pull the latest code from the GitHub repository.
  - **Build:** Build the Docker image for the application.
  - **Test:** Run static code analysis, vulnerability scans, and security testing using tools like SonarQube, Trivy, and OWASP ZAP.
  - **Deploy:** Deploy the updated Docker image locally or to a Kubernetes cluster.

3. Jenkinsfile Example:

```
pipeline {

   agent any

   environment {

      DOCKER_IMAGE = 'devsecops-app:latest'

   }

   stages {

      stage('Checkout') {
            steps {
                git 'https://github.com/<username>/devsecops-app.git'
            }
        }
        stage('Build Docker Image') {
            steps {
                script {
                    sh 'docker build -t $DOCKER_IMAGE .'
                }
            }
        }
        stage('Static Code Analysis') {
            steps {
                script {
                    sh 'sonar-scanner'
                }
            }
        }
        stage('Vulnerability Scan') {
            steps {
                script {
                    sh 'trivy image $DOCKER_IMAGE'
                }
            }
        }
        stage('Security Testing') {
            steps {
                script {
                    sh 'zap-cli quick-scan http://localhost:8080'
                }
            }
        }
        stage('Deploy') {
            steps {
                script {
                    sh 'docker run -d -p 8080:8080 $DOCKER_IMAGE'
                }
            }
        }
    }
    post {
        success {
            echo 'Pipeline executed successfully.'
        }
        failure {
            echo 'Pipeline failed. Please check the logs.'
        }
    }
```

FUTURE PLAN

1. **Container Image Management**
   - Utilize Docker Hub or a private container registry for storing and managing Docker images. This provides secure and centralized storage for container images, enabling streamlined deployments.
2. **Kubernetes Cluster Setup and Deployment**
   - Deploy the Dockerized application on a Kubernetes cluster using Minikube for local development and testing. This simulates a production-like environment, ensuring scalability and resilience.
3. **Enhanced Security with HashiCorp Vault**
   - Integrate HashiCorp Vault to securely store and manage sensitive information like API keys, passwords, and certificates.
4. **Vulnerability Scanning and Signing**
   - Implement tools like Snyk and OpenSSL for vulnerability scanning and signing Docker images to ensure only trusted versions are deployed, improving the security of the pipeline.
5. **CI/CD Pipeline Integration**
   - Automate the build, test, and deployment processes by integrating Jenkins with Kubernetes and other security tools, ensuring rapid and consistent updates.