DevOps Shack

Real-Time Corporate Azure DevOps CICD Pipeline

Click Here To Enrol To Batch-5 | DevOps & Cloud DevOps

Stage 0: Install Required Tools

In this stage, we'll ensure that all essential tools like JDK, Node.js, and Maven are set up on the build agent.

```
trigger:
  branches:
    include:
      - main  # Replace with your branch name

pool:
  vmImage: 'ubuntu-latest'

stages:
- stage: Setup
  jobs:
  - job: InstallRequiredTools
    displayName: 'Install Required Tools'
    steps:
    - script: |
        sudo apt-get update
        sudo apt-get install -y default-jdk nodejs npm maven
      displayName: 'Install JDK, Node.js, and Maven'
```

Explanation:

- The `trigger` section specifies the branch to trigger the pipeline.
- The `pool` section specifies the type of agent to use.
- The `stages` section defines the pipeline stages, starting with the setup stage.

- Inside the setup stage, a job named `InstallRequiredTools` is defined to install the required tools using apt-get commands.

Stage 1: Install Required Tools & Project Dependencies

In this stage, we'll install the required tools like JDK, Node.js, and Maven, and also install project-specific dependencies using npm or Maven.

```
trigger:
  branches:
    include:
      - main  # Replace with your branch name

pool:
  vmImage: 'ubuntu-latest'

stages:
- stage: Setup_and_Dependencies
  jobs:
  - job: InstallRequiredTools
    displayName: 'Install Required Tools'
    steps:
    - script: |
        sudo apt-get update
        sudo apt-get install -y default-jdk nodejs npm maven
      displayName: 'Install JDK, Node.js, and Maven'

  - job: InstallProjectDependencies
    displayName: 'Install Project Dependencies'
    steps:
    - script: |
        npm install   # For Node.js projects
        mvn clean install   # For Maven projects
      displayName: 'Install npm and Maven Dependencies'
```

Explanation:

- This stage includes two jobs: one for installing required tools and the other for installing project dependencies.
- Each job consists of steps to execute the necessary commands for installing tools and dependencies.
- `npm install` and `mvn clean install` commands are used to install project-specific dependencies based on the project type.

Stage 2: Execute Test Cases

In this stage, we'll run automated test cases to validate the functionality of the code.

```
- stage: Test
  jobs:
  - job: ExecuteTestCases
    displayName: 'Execute Test Cases'
    steps:
    - script: |
        npm test     # For Node.js projects
        mvn test     # For Maven projects
      displayName: 'Run Automated Tests'
```

Explanation:

- This stage includes a single job named `ExecuteTestCases` to run automated test cases.
- Depending on the project type, either `npm test` or `mvn test` command will be executed to run the tests.
- The `displayName` attribute provides a human-readable name for the job, making it easier to understand the purpose of the job in the pipeline.

Stage 3: Perform Security Scans

In this stage, we'll perform security scans on the project files and Docker images.

```
- stage: SecurityScans
  jobs:
  - job: PerformFileSystemScan
    displayName: 'Perform File System Scan'
    steps:
    - script: |
        trivy <path_to_project_directory>
      displayName: 'Scan Project Files for Vulnerabilities'

  - job: ScanDockerImage
    displayName: 'Scan Docker Image for Vulnerabilities'
    steps:
    - script: |
        trivy your_image_name:latest
      displayName: 'Scan Docker Image'
```

Explanation:

- This stage includes two jobs: one for performing a file system scan and the other for scanning a Docker image for vulnerabilities.
- Each job executes the appropriate security scanning tool (`trivy`) with the necessary parameters.
- The `displayName` attribute provides a descriptive name for each job in the pipeline.

Stage 4: Code Quality Analysis

In this stage, we'll evaluate the code quality using static analysis tools like SonarQube.

```
- stage: CodeQualityAnalysis
  jobs:
  - job: EvaluateCodeQuality
    displayName: 'Evaluate Code Quality'
    steps:
    - script: |
        sonar-scanner
      displayName: 'Run SonarQube Scanner'
```

Explanation:

- This stage includes a single job named `EvaluateCodeQuality` to run SonarQube's static analysis.
- The `sonar-scanner` command is executed to analyze the code quality and generate reports.
- The `displayName` attribute provides a clear name for the job in the pipeline.

Stage 5: Build Application Artifact

This stage involves compiling and packaging the application into an executable artifact.

```
- stage: BuildArtifact
  jobs:
  - job: CompileAndPackage
    displayName: 'Compile and Package Application'
    steps:
    - script: |
        mvn clean package
      displayName: 'Build Artifact'
    artifacts:
      paths:
        - target/*.jar   # Adjust based on artifact type (e.g., WAR, JAR)
```

Explanation:

- This stage contains a single job named `CompileAndPackage` responsible for building the application artifact.
- The `mvn clean package` command is used to compile and package the application using Maven.
- The `artifacts` section specifies the artifact paths to be published by the job for use in subsequent stages.

Stage 6: Publish Artifact to Nexus

In this stage, we'll upload the artifact to Nexus for version control.

```
- stage: PublishToNexus
  jobs:
  - job: UploadArtifact
    displayName: 'Upload Artifact to Nexus'
    steps:
    - script: |
        mvn deploy
      displayName: 'Publish Artifact'
```

Explanation:

- This stage contains a single job named `UploadArtifact` responsible for publishing the artifact to Nexus.
- The `mvn deploy` command is used to deploy the artifact to Nexus using Maven.

- This stage assumes that the Maven settings.xml file is properly configured with Nexus credentials for deployment.

Stage 7: Build Docker Image

This stage creates a Docker image for containerized deployment.

```
- stage: BuildDockerImage
  jobs:
  - job: BuildImage
    displayName: 'Build Docker Image'
    steps:
    - script: |
        docker build -t your_image_name:latest .
      displayName: 'Building Docker Image'
```

Explanation:

- This stage includes a single job named `BuildImage` responsible for building the Docker image.
- The `docker build` command is used to build the Docker image using the Dockerfile in the project directory.
- `your_image_name:latest` should be replaced with the appropriate image name and tag.

Stage 8: Scan Docker Image for Security Vulnerabilities

This stage checks the Docker image for security vulnerabilities using Trivy.

```
- stage: ScanDockerImage
  jobs:
  - job: SecurityScan
    displayName: 'Scan Docker Image for Vulnerabilities'
    steps:
    - script: |
        trivy your_image_name:latest
      displayName: 'Running Trivy Security Scan'
```

Explanation:

- This stage includes a single job named `SecurityScan` responsible for scanning the Docker image for vulnerabilities.
- The `trivy` command is used to run the security scan on the Docker image.
- `your_image_name:latest` should be replaced with the name and tag of your Docker image.

Stage 9: Push Docker Image to Docker Hub

In this stage, the Docker image is pushed to Docker Hub for distribution.

```
- stage: PushToDockerHub
  jobs:
```

```
  - job: PushImage
    displayName: 'Push Docker Image to Docker Hub'
    steps:
    - script: |
        docker login -u $DOCKER_USERNAME -p $DOCKER_PASSWORD
        docker push your_image_name:latest
      displayName: 'Pushing Docker Image'
```

Explanation:

- This stage includes a single job named `PushImage` responsible for pushing the Docker image to Docker Hub.
- The `docker login` command is used to authenticate with Docker Hub using the provided credentials.
- The `docker push` command is used to push the Docker image to Docker Hub.
- `your_image_name:latest` should be replaced with the name and tag of your Docker image.

Stage 10: Update Kubernetes Manifests

In this stage, Kubernetes manifest files are updated to reference the new Docker image.

```
- stage: UpdateKubernetesManifests
  jobs:
  - job: UpdateManifests
    displayName: 'Update Kubernetes Manifests'
    steps:
    - script: |
        sed -i 's|old_image_name|your_image_name|g'
path/to/kubernetes/*.yaml
      displayName: 'Updating Manifests'
```

Explanation:

- This stage includes a single job named `UpdateManifests` responsible for updating Kubernetes manifest files.
- The `sed` command is used to replace occurrences of the old image name with the new image name in the Kubernetes manifest files.
- `old_image_name` should be replaced with the previous image name, and `your_image_name` should be replaced with the new image name.

Stage 11: Deploy Application to Kubernetes Cluster

This stage deploys the application to a Kubernetes cluster for orchestration.

```
- stage: DeployToKubernetes
  jobs:
  - job: DeployApp
    displayName: 'Deploy Application to Kubernetes'
    steps:
    - script: |
        kubectl apply -f path/to/kubernetes/*.yaml
      displayName: 'Applying Kubernetes Manifests'
```

Explanation:

- This stage includes a single job named `DeployApp` responsible for deploying the application to a Kubernetes cluster.
- The `kubectl apply` command is used to apply the Kubernetes manifest files to the cluster.
- `path/to/kubernetes/*.yaml` should be replaced with the path to your Kubernetes manifest files.

Stage 12: Verify Deployment

This stage confirms the successful deployment of the application.

```
- stage: VerifyDeployment
  jobs:
  - job: VerifyApp
    displayName: 'Verify Deployment'
    steps:
    - script: |
        kubectl get pods --namespace your_namespace
        # Additional verification steps can be added here
      displayName: 'Check Pod Status'
```

Explanation:

- This stage includes a single job named `VerifyApp` responsible for verifying the deployment of the application.
- The `kubectl get pods` command is used to retrieve the status of pods in the specified namespace.
- Additional verification steps can be added as needed to ensure the application is running correctly.

Stage 13: Conduct Security Testing

In this stage, security testing is conducted using OWASP ZAP to identify vulnerabilities.

```
- stage: SecurityTesting
  jobs:
  - job: SecurityTest
    displayName: 'Security Testing with OWASP ZAP'
    steps:
    - script: |
        zap-cli --url <your_application_url> --spider --ajax --quick-scan -
-output <output_file>
      displayName: 'Running OWASP ZAP Security Test'
```

Explanation:

- This stage includes a single job named `SecurityTest` responsible for conducting security testing.

- The `zap-cli` command is used to run OWASP ZAP security testing on the provided application URL.
- The `--spider`, `--ajax`, and `--quick-scan` options are used to configure the security scan.
- The results of the security test are output to a specified file.

Stage 14: Send Email Notifications

This stage sends email notifications to stakeholders about the pipeline status and results.

```
- stage: SendEmailNotifications
  jobs:
  - job: SendEmail
    displayName: 'Send Email Notifications'
    steps:
    - script: |
        echo "Sending email notifications..."
        # Command to send email notifications using your email service
provider
      displayName: 'Send Email'
```

Explanation:

- This stage includes a single job named `SendEmail` responsible for sending email notifications.
- The `echo` command is used to indicate that email notifications are being sent.
- You can replace the comment with the actual command to send email notifications using your email service provider.

This completes the Azure DevOps CI/CD YAML pipeline, covering all the stages from setup to email notifications. Each stage performs specific tasks essential for the continuous integration and deployment process.

Here's the combined Azure DevOps CI/CD YAML pipeline with all the stages:

```
trigger:
  branches:
    include:
      - main  # Replace with your branch name

pool:
  vmImage: 'ubuntu-latest'

stages:
- stage: Setup_and_Dependencies
  jobs:
  - job: InstallRequiredTools
    displayName: 'Install Required Tools'
    steps:
    - script: |
        sudo apt-get update
        sudo apt-get install -y default-jdk nodejs npm maven
      displayName: 'Install JDK, Node.js, and Maven'

  - job: InstallProjectDependencies
    displayName: 'Install Project Dependencies'
    steps:
    - script: |
        npm install   # For Node.js projects
        mvn clean install   # For Maven projects
      displayName: 'Install npm and Maven Dependencies'

- stage: Test
  jobs:
  - job: ExecuteTestCases
    displayName: 'Execute Test Cases'
    steps:
    - script: |
        npm test     # For Node.js projects
        mvn test     # For Maven projects
      displayName: 'Run Automated Tests'

  - job: PerformFileSystemScan
    displayName: 'Perform File System Scan'
    steps:
    - script: |
        trivy <path_to_project_directory>
      displayName: 'Scan Project Files for Vulnerabilities'

- stage: CodeQualityAnalysis
  jobs:
  - job: EvaluateCodeQuality
    displayName: 'Evaluate Code Quality'
    steps:
    - script: |
        sonar-scanner
      displayName: 'Run SonarQube Scanner'

- stage: BuildArtifact
  jobs:
  - job: CompileAndPackage
    displayName: 'Compile and Package Application'
    steps:
    - script: |
        mvn clean package
      displayName: 'Build Artifact'
    artifacts:
      paths:
        - target/*.jar   # Adjust based on artifact type (e.g., WAR, JAR)

- stage: PublishToNexus
  jobs:
  - job: UploadArtifact
    displayName: 'Upload Artifact to Nexus'
    steps:
    - script: |
        mvn deploy
      displayName: 'Publish Artifact'

- stage: BuildDockerImage
  jobs:
  - job: BuildImage
    displayName: 'Build Docker Image'
    steps:
    - script: |
        docker build -t your_image_name:latest .
      displayName: 'Building Docker Image'

- stage: ScanDockerImage
  jobs:
  - job: SecurityScan
    displayName: 'Scan Docker Image for Vulnerabilities'
    steps:
    - script: |
        trivy your_image_name:latest
      displayName: 'Running Trivy Security Scan'

- stage: PushToDockerHub
  jobs:
  - job: PushImage
    displayName: 'Push Docker Image to Docker Hub'
    steps:
    - script: |
        docker login -u $DOCKER_USERNAME -p $DOCKER_PASSWORD
        docker push your_image_name:latest
      displayName: 'Pushing Docker Image'

- stage: UpdateKubernetesManifests
  jobs:
  - job: UpdateManifests
    displayName: 'Update Kubernetes Manifests'
    steps:
    - script: |
        sed -i 's|old_image_name|your_image_name|g'
path/to/kubernetes/*.yaml
      displayName: 'Updating Manifests'

- stage: DeployToKubernetes
  jobs:
  - job: DeployApp
    displayName: 'Deploy Application to Kubernetes'
    steps:
    - script: |
        kubectl apply -f path/to/kubernetes/*.yaml
      displayName: 'Applying Kubernetes Manifests'

- stage: VerifyDeployment
  jobs:
  - job: VerifyApp
    displayName: 'Verify Deployment'
    steps:
    - script: |
        kubectl get pods --namespace your_namespace
        # Additional verification steps can be added here
      displayName: 'Check Pod Status'

- stage: SecurityTesting
  jobs:
  - job: SecurityTest
    displayName: 'Security Testing with OWASP ZAP'
    steps:
    - script: |
        zap-cli --url <your_application_url> --spider --ajax --quick-scan -
-output <output_file>
      displayName: 'Running OWASP ZAP Security Test'

- stage: SendEmailNotifications
  jobs:
  - job: SendEmail
    displayName: 'Send Email Notifications'
    steps:
    - script: |
        echo "Sending email notifications..."
        # Command to send email notifications using your email service
provider
      displayName: 'Send Email'
```

Explanation:

- The pipeline consists of multiple stages, each containing one or more jobs.
- Jobs within each stage are executed sequentially.
- Each stage represents a specific phase of the CI/CD process, such as setup, testing, building, deployment, security scanning, and notification.
- You'll need to replace placeholders like `your_image_name`, `path/to/project_directory`, `old_image_name`, `your_namespace`, and `<your_application_url>` with your actual values.
- Ensure you have appropriate permissions and configurations for Docker Hub, Nexus, Kubernetes, and any other services used in the pipeline.