This project involves building a Java application, creating a Docker image, pushing it to AWS ECR, and updating an AWS ECS task definition.

# Jenkins Pipeline Runbook

## Project Overview:

This project involves automating the build and deployment of a Java application using Jenkins pipelines. The pipeline performs the following steps:

1. Clones the Java application code from a GitHub repository.

2. Builds a Docker image from the Java application.

3. Pushes the Docker image to AWS Elastic Container Registry (ECR).

4. Updates an AWS ECS task definition with the latest Docker image.

### Prerequisites:

1. Jenkins Server:

* A Jenkins server is set up and running.
* Jenkins plugins installed: Git, AWS, Docker, Pipeline.

1. GitHub Repository:

* A GitHub repository containing the Java application code.

1. AWS Credentials in Jenkins:

* AWS credentials configured in Jenkins with permissions to push Docker images to ECR and update ECS task definitions.

1. GitHub Personal Access Token:

* A GitHub Personal Access Token stored as a Jenkins credential for accessing the GitHub repository.

### Jenkins Pipeline Configuration:

1. Create a New Jenkins Job:

* Open Jenkins and create a new Pipeline job.

1. Configure Pipeline from SCM:

* In the job configuration, choose "Pipeline script from SCM" as the Pipeline definition.
* Select Git as the SCM, and enter the GitHub repository URL.

1. Set Up GitHub Webhook:

* In the GitHub repository settings, add a webhook pointing to the Jenkins server's GitHub webhook endpoint (<http://your-jenkins-server/github-webhook/>).
* Configure the webhook to trigger push events.

#### Jenkinsfile:

Ensure that there is a `Jenkinsfile` at the root of your GitHub repository. The Jenkinsfile defines the stages and steps of the pipeline.

```groovy

pipeline {

agent any

environment {

GIT\_REPO\_URL = 'https://github.com/MadanrajM/docker-jenkins-demo.git'

GIT\_BRANCH = 'main'

ECR\_REPO = '676546158846.dkr.ecr.us-east-1.amazonaws.com/docker-demo'

ECS\_TASK\_DEFINITION = 'docker-demo'

}

stages {

stage('Clone Repository') {

steps {

script {

// Clean workspace before cloning

deleteDir()

// Clone the Git repository

git branch: 'main',

credentialsId: 'github-pat',

url: 'https://github.com/MadanrajM/docker-jenkins-demo.git'

}

}

}

stage('Build Docker Image') {

steps {

script {

// Assuming Maven is installed on the Jenkins agent

sh '''

mvn clean package

docker build -t my-java-app:${BUILD\_NUMBER} .

'''

}

}

}

stage('Run Docker Image') {

steps {

script {

// Assuming Maven is installed on the Jenkins agent

sh 'docker run my-java-app'

}

}

}

stage('Push to AWS ECR') {

steps {

script {

withAWS(credentials: 'aws-creds', region: 'us-east-1') {

sh '''

LOGIN\_PASSWOD=$(aws ecr get-login-password --region us-east-1)

docker login -u AWS -p $LOGIN\_PASSWOD $ECR\_REPO

docker tag my-java-app:${BUILD\_NUMBER} $ECR\_REPO:${BUILD\_NUMBER}

docker push $ECR\_REPO:${BUILD\_NUMBER}

'''

}

}

}

}

stage('Update ECS Task Definition') {

steps {

script {

withAWS(credentials: 'aws-creds', region: 'us-east-1') {

sh "aws ecs register-task-definition --family $ECS\_TASK\_DEFINITION --memory 2048 --container-definitions '[{\"name\":\"my-java-app\",\"image\":\"$ECR\_REPO:${BUILD\_NUMBER}\"}]'"

}

}

}

}

}

}

```

## Runbook Steps:

1. Trigger Pipeline:

* Push changes to your GitHub repository to trigger the Jenkins pipeline automatically.

1. View Pipeline Progress:

* Monitor the Jenkins console or the Blue Ocean interface to view the progress of the pipeline.

1. Troubleshooting:

* If any stage fails, check the console output for error messages.
* Inspect the Jenkins logs for detailed information.

1. Manual Trigger:

* Optionally, trigger the pipeline manually through the Jenkins UI.

1. Credentials Management:

* Periodically review and update credentials stored in Jenkins.
* Rotate GitHub Personal Access Token and AWS credentials as needed.

1. Pipeline Maintenance:

* Periodically review and update the Jenkinsfile as the project evolves.
* Keep Jenkins and plugins up to date.

1. Logging and Monitoring:

* Implement logging and monitoring for the Jenkins server and AWS resources to detect issues early.

1. Documentation:

* Keep the runbook and other documentation up to date.
* Document any changes made to the Jenkins pipeline configuration.

1. Security Best Practices:

* Follow security best practices for Jenkins, AWS, and Docker.
* Regularly review and update security configurations.