### **Introduction to Jenkins**

Jenkins is an open-source automation server widely used for continuous integration and continuous delivery (CI/CD). It helps automate the parts of software development related to building, testing, and deploying, facilitating continuous integration and continuous delivery.

### **Jenkins Pipelines**

A Jenkins Pipeline is a suite of plugins that supports implementing and integrating continuous delivery pipelines into Jenkins. Pipelines are defined using a domain-specific language (DSL) in a text file called a Jenkinsfile.

### **Key Concepts**

* Stages: Logical blocks of a pipeline, such as Build, Test, and Deploy.
* Steps: Individual tasks within a stage.
* SCM (Source Code Management): Integration with version control systems like Git.
* Poll SCM: A trigger that checks the SCM at regular intervals for changes.

### **Jenkinsfile: Explained Stage by Stage**

Below is a detailed breakdown of the Jenkinsfile provided, with explanations for each stage and its purpose.

#### **Full Jenkinsfile that we will be using for understanding the pipelines in our hands-on: https://github.com/devopsmay24batch/demopythonapp/blob/main/Jenkinsfile**

pipeline {

agent any

environment {

FLASK\_APP = 'app.py'

FLASK\_ENV = 'development'

}

stages {

stage('Build') {

steps {

script {

echo 'Building Python Flask app...'

sh 'pip install -r requirements.txt'

}

}

}

stage('Test') {

steps {

script {

echo 'Running tests...'

// Add any test commands here if needed

}

}

}

stage('Approve via email') {

steps {

emailext(

to: 'devopsmay24batch@gmail.com',

mimeType: 'text/html',

subject: 'New build is waiting for your approval',

body: '<html><p>A new build has been deployed, please view it by visiting <a href="http://localhost:8000/">http://localhost:8080/</a>, please approve this deployment by clicking on <a href="http://localhost:8080/job/demopythonapp/${BUILD\_NUMBER}/input/">http://localhost:8080/</a></p></html>',

attachLog: true

)

timeout(time: 60, unit: 'MINUTES') {}

}

}

stage('Approval') {

steps {

timeout(time: 24, unit: 'HOURS') {

input message: 'Do you approve the deployment?', submitter: 'admin'

}

}

}

stage('Deploy') {

steps {

script {

echo 'Deploying Python Flask app...'

sh 'python app.py'

}

}

}

}

post {

success {

echo 'Pipeline succeeded!'

}

failure {

echo 'Pipeline failed!'

}

}

}

#### **Stage-by-Stage Breakdown**

1. Pipeline Declaration and Environment Setup

pipeline {

agent any

environment {

FLASK\_APP = 'app.py'

FLASK\_ENV = 'development'

}

* + agent any: This means the pipeline can run on any available agent.
  + environment: This block sets up environment variables for the pipeline, specifically for the Flask application.

1. Build Stage

stage('Build') {

steps {

script {

echo 'Building Python Flask app...'

sh 'pip install -r requirements.txt'

}

}

}

* + This stage installs the necessary Python packages listed in requirements.txt.

1. Test Stage

stage('Test') {

steps {

script {

echo 'Running tests...'

// Add any test commands here if needed

}

}

}

* + This stage is for running tests. You can add your test commands here to ensure your application works correctly.

1. Approve via Email Stage

stage('Approve via email') {

steps {

emailext(

to: '\*\*\*\*\*\*\*\*\*\*@gmail.com', //replace with valid emailaddress

mimeType: 'text/html',

subject: 'New build is waiting for your approval',

body: '<html><p>A new build has been deployed, please view it by visiting <a href="http://localhost:8000/">http://localhost:8080/</a>, please approve this deployment by clicking on <a href="http://localhost:8080/job/demopythonapp/${BUILD\_NUMBER}/input/">http://localhost:8080/</a></p></html>',

attachLog: true

)

timeout(time: 60, unit: 'MINUTES') {}

}

}

* + This stage sends an email to the specified recipient with a link to approve the deployment. The email contains a link to the Jenkins job and another to the input approval page.
  + timeout: Ensures the stage times out if not approved within 60 minutes.

1. Approval Stage

stage('Approval') {

steps {

timeout(time: 24, unit: 'HOURS') {

input message: 'Do you approve the deployment?', submitter: 'admin'

}

}

}

* + This stage waits for a manual approval before proceeding. It will timeout if no approval is given within 24 hours.

1. Deploy Stage

stage('Deploy') {

steps {

script {

echo 'Deploying Python Flask app...'

sh 'python app.py'

}

}

}

* + This stage deploys the Flask application by running python app.py.

1. Post Actions

post {

success {

echo 'Pipeline succeeded!'

}

failure {

echo 'Pipeline failed!'

}

}

* + These blocks define actions that will run after the pipeline completes. They provide notifications for both successful and failed pipeline executions.

### **Setting Up SCM and Poll SCM**

To integrate with your source code management (SCM) system like Git and enable Poll SCM, follow these steps:

1. SCM Integration
   * In the Jenkins job configuration, under the "Source Code Management" section, select "Git" and provide your repository URL and credentials.
2. Poll SCM
   * In the "Build Triggers" section, select "Poll SCM" and define the schedule using cron syntax (e.g., H/5 \* \* \* \* to poll every 5 minutes).

### **Conclusion**

This tutorial covers the basics of Jenkins Pipelines, including building, testing, sending email notifications, waiting for manual approval, and deploying a Python Flask application. By integrating SCM and using Poll SCM, you can automate the process of checking for changes in your repository and trigger builds accordingly. This setup is particularly useful for freshers to understand how to implement CI/CD pipelines in Jenkins.