Pipelines are made up of multiple steps that allow you to build, test and deploy applications. Jenkins Pipeline allows you to compose multiple steps in an easy way that can help you model any sort of automation process.

On linux, Mac Os uses sh step is used to execute shell commands in pipeline.

On windows we use bat to execute batch commands.

TimeOut and Retry:

There are some powerful steps that "wrap" other steps which can easily solve problems like retrying (retry) steps until successful or exiting if a step takes too long (timeout).

pipeline {

agent any

stages {

stage('Deploy') {

steps {

retry(3) {

sh './flakey-deploy.sh'

}

timeout(time: 3, unit: 'MINUTES') {

sh './health-check.sh'

}

}

}

}

}

We can compose these steps together.

pipeline {

agent any

stages {

stage('Deploy') {

steps {

timeout(time: 3, unit: 'MINUTES') {

retry(5) {

sh './flakey-deploy.sh'

}

}

}

}

}

}

Finishing up

When the Pipeline has finished executing, you may need to run clean-up steps or perform some actions based on the outcome of the Pipeline. These actions can be performed in the post section.

pipeline {

agent any

stages {

stage('Test') {

steps {

sh 'echo "Fail!"; exit 1'

}

}

}

post {

always {

echo 'This will always run'

}

success {

echo 'This will run only if successful'

}

failure {

echo 'This will run only if failed'

}

unstable {

echo 'This will run only if the run was marked as unstable'

}

changed {

echo 'This will run only if the state of the Pipeline has changed'

echo 'For example, if the Pipeline was previously failing but is now successful'

}

}

}

The agent directive tells Jenkins where and how to execute the Pipeline. There are a few things agent causes to happen:

* All the steps contained within the block are queued for execution by Jenkins. As soon as an [executor](https://jenkins.io/doc/book/glossary/#executor) is available, the steps will begin to execute.
* A [workspace](https://jenkins.io/doc/book/glossary/#workspace) is allocated which will contain files checked out from source control as well as any additional working files for the Pipeline.

Environment variables:

Environment variables can be set globally or per stage. When we set variables per stage they are only be available to the stage iin which they are defined.

An example for global variable:

pipeline {

agent {

label '!windows'

}

environment {

DISABLE\_AUTH = 'true'

DB\_ENGINE = 'sqlite'

}

stages {

stage('Build') {

steps {

echo "Database engine is ${DB\_ENGINE}"

echo "DISABLE\_AUTH is ${DISABLE\_AUTH}"

sh 'printenv'

}

}

}

}

Jenkins can record and aggregate test results so long as your test runner can output test result files. To collect our test results and artifacts, we will use the post section. Jenkins typically comes bundled with the junit step, but if your test runner cannot output JUnit-style XML reports, then there are plugins which can give you the results in the desired format.

post {

always {

junit 'build/reports/\*\*/\*.xml'

}

}

This will grab the test results and lets Jenkins track the results.

A pipeline which has failing tests will be marked as “unstable” . When there are test failures Jenkins grabs built artifacts(files generated during execution of pipeline) for analysis by its built-in support for storing artifacts. Recording tests and artifacts in Jenkins is useful for quickly and easily surfacing information to various members of the team.

We can add some notification or other steps to perform finalization, notification, or other end-of-Pipeline tasks in the post section as it runs after the pipeline is executed. There are plenty of ways to send notifications, below are a few snippets demonstrating how to send notifications about a Pipeline to an email orr to a team mail. For example:

post {

failure {

mail to: 'team@example.com',

subject: "Failed Pipeline: ${currentBuild.fullDisplayName}",

body: "Something is wrong with ${env.BUILD\_URL}"

}

}

The most basic continuous delivery pipeline will have, at minimum, three stages which should be defined in a Jenkinsfile: Build, Test, and Deploy. Stable build and test stages important before deployment process.

pipeline{

agent any

tools{

maven 'Maven 3'

}

stages{

stage('Source'){

steps{

git 'https://github.com/Diptimb/Maven-Project.git'}

}

stage('Clean-Package-create-war-file'){

steps{

sh 'mvn clean package'

}

}

Stage(‘Deploy’){

deploy adapters: [tomcat8(path: '', url: 'http://40.70.14.67:8081/')], contextPath: null, war: '\*\*/\*.war'Steps{

}}

}

post {

always {

mail to: 'deepubebarta@gmail.com',

subject: "Status of pipeline: ${currentBuild.fullDisplayName}",

body: "${env.BUILD\_URL} has result ${currentBuild.result}"

}

}

}

pipeline{

agent any

tools{

maven 'Maven 3'

}

stages{

stage('Source'){

steps{

git 'https://github.com/Diptimb/Maven-Project.git'}

}

stage('Build'){

steps{

sh 'mvn clean package'

}

}

stage('artifact'){

steps{

archive 'target/.\*war'

}

}

stage('deploy'){

steps{deploy adapters: [tomcat8(path: '/manager/text', url: 'http://40.70.14.67:8081/')], contextPath: null, war: '\*\*/\*.war'

}}

}

post {

always {

mail to: 'deepubebarta@gmail.com',

subject: "Status of pipeline: ${currentBuild.fullDisplayName}",

body: "${env.BUILD\_URL} has result ${currentBuild.result}"

}

}

}