The Mightywomble Follow
thoughts about the world I'm surrounded by, tech, politics, opinion..
Jan 31 · 5 min read

Jenkins—Pipeline (Beginners guide)

The following document is designed to explain what Jenkins Pipelining is, and provide examples of a Jenkins pipeline which runs a basic script on a Jenkins slave.

I spent a while searching around google and couldn't find a good simple concise guide, so I help this helps someone get up and going.

The Management platform run a more advanced version of this to create machines using Terraform.

Step-by-step guide

What is Jenkins

Jenkins is an open source automation server written in Java. **Jenkins** helps to automate the non-human part of software development process, with continuous integration and facilitating technical aspects of continuous delivery. It is a server-based system that runs in servlet containers such as Apache Tomcat. It can execute Apache Ant, Apache Maven and sbt based projects as well as arbitrary shell scripts and Windows batch commands.

What is a Jenkins Pipeline

According to Jenkins it's..

Jenkins Pipeline (or simply "Pipeline" with a capital "P") is a suite of plugins which supports implementing and integrating *continuous delivery pipelines* into Jenkins.

A *continuous delivery pipeline* is an automated expression of your process for getting software from version control right through to your users and customers. Every change to your software (committed in source control) goes through a complex process on its way to being released. This process involves building the software in a reliable and repeatable manner, as well as the progression of the built software (called a "build") through multiple stages of testing and deployment.

In real English,

A pipeline is a script which gives Jenkins a series of jobs to do in a pipeline like manner (one after the other) these jobs can be as Jenkins code, scripts, build languages etc.

The Jenkinsfile

The core of the pipeline is the file called Jenkinsfile. It's the list of jobs which the pipeline will perform and can be held on the Jenkins server itself as part of the pipeline or at the root of a linked git/Bitbucket repository

An example Jenkinsfile looks like this:

```
pipeline {

environment {

BUILD_SCRIPTS_GIT="http://10.100.100.10:7990/scm/~myname/myp
ipeline.git"

BUILD_SCRIPTS='mypipeline'

BUILD_HOME='/var/lib/jenkins/workspace'

}

agent any

stages {

stage('Checkout: Code') {

steps {

sh "mkdir -p $WORKSPACE/repo;\

git config --global user.email 'email@address.com';\

git config --global user.name 'myname';\

git config --global push.default simple;\

git clone $BUILD_SCRIPTS_GIT repo/$BUILD_SCRIPTS"

sh "chmod -R +x $WORKSPACE/repo/$BUILD_SCRIPTS"

}

}

stage('Yum: Updates') {

steps {

sh "sudo chmod +x
$WORKSPACE/repo/$BUILD_SCRIPTS/scripts/update.sh"

sh "sudo $WORKSPACE/repo/$BUILD_SCRIPTS/scripts/update.sh"
}

}

}

post {

always {

cleanWs()

}

}

}
```

This pipeline file

- sets up environment variables
- pulls data down from a git repo
- sets it up in a Jenkins workspace
- runs a script under scripts/
- once completes by cleaning up the workspace (successful or not)

In this example we break this information down to the following chunks with an explanation

Pipeline

We define the pipeline using the **pipeline{}** section

Our first stage is an option but recommended one where we setup environment variables using an **environment{}** section

In this example I'm not restricting the agent the pipleine can run on, so i've inclued **agent any**

```
pipeline {

environment {

BUILD_SCRIPTS_GIT="http://10.100.100.10:7990/scm/~myhome/myp
ipeline.git"

BUILD_SCRIPTS='mypipeline'

BUILD_HOME='/var/lib/jenkins/workspace'

}

agent any
```

Stages

Checkout: Code

The core of pipelines are stages{} and steps{}, stages can have multiple steps in them

In this example our first stage is called Checkout: code, this label will also be shown as we will see later in the Jenkins interface so making it relevent is useful

Our steps are to create a workspace on the Jenkins slave and then setup and pull down the data from the Jenkins repo we linked the pipeline to when we set it up (see later)

```
stage('Checkout: Code') {

steps {

sh "mkdir -p $WORKSPACE/repo;\

git config --global user.email 'email@address.com';\

git config --global user.name 'myname';\

git config --global push.default simple;\
git clone $BUILD_SCRIPTS_GIT repo/$BUILD_SCRIPTS"

sh "chmod -R +x $WORKSPACE/repo/$BUILD_SCRIPTS"

}

}
```

Yum: updates

Our second stage is an example of using the data pulled down from the git repo, which includes a script (in this instance it just runs yum install -y wget nano nmap traceroute)

You can see we have to do some linux maintenance by making the script executable and running as sudo, (I've got a user setup on the Jenkins client with sudo nopasswd for this example)

```
stage('Yum: Updates') {

steps {

sh "sudo chmod +x
$WORKSPACE/repo/$BUILD_SCRIPTS/scripts/update.sh"

sh "sudo $WORKSPACE/repo/$BUILD_SCRIPTS/scripts/update.sh"

}

}
```

Post

The post section comes after the stages and will execute even if the stages failed. In this example I've used it to clean up the workspace folder; if you don't do this Jenkins will fail the next time the pipeline is run

```
post {
```

```
always {

cleanWs()

}

}
```

Note about brackets

Every open bracket { must have a close } and vice versa, if there are too many of either bracket then the pipeline will fail with various errors.

The Jenkinsfile in a git/Bitbucket project

The above Jenkinsfile along with the scripts/update.sh folder are held in a git/Bitbucket repository. The Jenkinsfile should be held in the root of the repository; if it is it will be automatically picked up when you setup the git repo in the Jenkins pipeline.

The Pipeline plugin

The pipeline needs to be added to Jenkins for this to work. This isn't always added by default and can be added via the Manage Jenkins interface.

Creating a new pipeline

With our Jenkinsfile and script in git/Bitbucket the pipeline needs to be setup in Jenkins. We do this using the Jenkins server web interface.

Login to Jenkins

Click on New Job

We need to create a pipeline, so we

- Give the pipeline a name
- Select pipeline in the list
- Click on OK

Next we scroll down to Advanced Project Options.

- Select—Pipeline script from SCM (if you don't see this option on your Jenkins server see the plugins note below)
- Add your git/Bitbucket repo link
- Add the credentials for connecting to your git repo

Note: because we have put the Jenkinsfile in the root of the Git repo path, it will be picked up and used by Jenkins automatically.

Note Additional Jenkins plugins

On a default installation of Jenkins most of the plugins you needed are installed, however to check we should have at least the following plugins:

Manage Jenkins -> Manage Plugins

- Bitbucket Plugin
- Pipeline
- Git
- Pipeline SCM Step

Running a Pipeline build

Once the pipeline has been created we can build the plugin. On the first build you will see a screen which can see a breakdown of the pipeline

Note that the heading we used in our stages of the above Jenkinsfile are now displayed within the pipeline display of Jenkins

As each stage of the pipeline runs through, the Jenkins interface will display a simple Red failed, Green worked status for the stage.

Error checking

If the pipeline fails (or even works in the interface but has unexpected results) the Jenkins interface provides logs for each stage of the pipeline

Clicking on the Logfile will explain what happened during the run of that stage

These instructions were written using a docker image of Bitbucket as the git repo and of Jenkins.

Related articles

https://jenkins.io/solutions/pipeline/

https://jenkins.io/doc/book/pipeline/jenkinsfile/

https://wiki.jenkins.io/display/JENKINS/Build+Pipeline+Plugin

Jenkins—Pipeline (Beginners guide)