**Jenkins:**

# How to identify [the type of Jenkins project](https://stackoverflow.com/questions/45064038/determining-the-type-of-jenkins-project)

There is no specific tag, but if you open a project, delete button have the type of project information available. Pipeline project will have "Delete Pipeline" button, maven project will have "Delete Maven Project" button and a freestyle project will only have "Delete Project" button.

1. Understanding the server provisioning workflow using terraform.

a. Configured terraform/terragrunt.

b. Familiarized configurations and modules.

c. Tried a test deployment against the dev-alpha platform without making config changes.

d. Tested with a test machine provisioning in dev-alpha env.

image-20200117-100948.png

e. Provisioning failed with ssh key error for my username. Not sure why is it connecting to the new machine using my id and whether user1 role is being called in the deployment. Checking the code base.

Error: timeout - last error: SSH authentication failed (hanita@10.20.5.111:22): ssh: handshake failed: ssh: unable to authenticate, attempted methods [none publickey], no supported methods remain

Jenkins job config:

Seed job setup:

1. Before proceeding, make sure to have the latest Job DSL plugin installed.

Workflow:

1. Developers create a runway/dsl.groovy file in their repository.
2. Developers modify their dsl.groovy and customize it for their specific needs.
3. Developers push their changes and a pre-receive git hook determines that the dsl.groovy file was changed.
4. If the dsl.groovy file was modified, the pre-receive git hook passes it to the Jenkins DSL seed job ([see tutorial](https://github.com/jenkinsci/job-dsl-plugin/wiki/Tutorial---Using-the-Jenkins-Job-DSL)) and regenerates the applications job.
5. The git hook monitors the status of the DSL seed job and reports back to the developer if the job fails or succeeds and rejects the push if the DSL is not properly generated.

<https://github.com/jenkinsci/job-dsl-plugin/wiki/Tutorial---Using-the-Jenkins-Job-DSL>

-----

One of the most helpful features of Jenkins CI is automatic job queuing. If there are more build jobs requested than there are resources to execute those jobs, Jenkins can queue your tests, executing them in the order they were requested as resources become available.

By assigning labels to nodes, you can specify the resources you want to use for specific jobs, and set up graceful queuing for your tests. This topic explains how queuing works in Jenkins, how to set up nodes and jobs, and how to use use labels. It assumes you have access to an admin user account in Jenkins, or the appropriate administrative rights.

Create a slave node:

Build the machine.

The current version of Jenkins does not support Java 10 (and Java 11) yet. If you have multiple versions of Java installed on your machine [make sure Java 8 is the default Java version](https://linuxize.com/post/install-java-on-ubuntu-18-04/).

root@staging-dev-processor07:/usr/lib/jvm/java-8-openjdk-amd64/bin# ll /usr/bin/java

lrwxrwxrwx 1 root root 22 Jan 21 15:25 /usr/bin/java -> /etc/alternatives/java\*

root@staging-dev-processor07:/home/jenkins# ll /etc/alternatives/java

lrwxrwxrwx 1 root root 42 Jan 28 11:57 /etc/alternatives/java -> /usr/lib/jvm/java-8-openjdk-amd64/bin/java\*

root@staging-dev-processor07:/home/jenkins#

Install git

Create a Jenkins user.

The Jenkins master will log into the agent as this user, and all build jobs will execute as this user.

sudo useradd -d /var/lib/jenkins jenkins

A screenshot of a cell phone

Description automatically generated

root@staging-dev-processor07:/home/jenkins# ls -ld /mnt/data1/jenkins/

drwxr-xr-x 3 jenkins jenkins 4096 Jan 28 12:08 /mnt/data1/jenkins/

root@staging-dev-processor07:/home/jenkins#

configure gitlab repo access rights:

### By using Token

Through API token you can also access the Git repo. To set API token in Jenkins for git repo access also includes two steps.

**A. Generate API token**  
**B. Configure API token in Jenkins**

<https://docs.gitlab.com/ee/user/profile/personal_access_tokens.html>

repo pluggin installation:

<https://wiki.jenkins.io/display/JENKINS/Repo+Plugin>

his plugin adds Repo (<http://code.google.com/p/git-repo/>) as an SCM provider for Jenkins. Projects can use this plugin to only run builds when changes are detected in any of the git repositories in the repo manifest, to list the changes between builds, and to re-create the project state across all repositories for any previous build using a static manifest.

Install repo command in Jenkins master and slave machines.

$ sudo apt-get install repo

<https://gerrit.googlesource.com/git-repo/>.

**Example:**

pipeline {

agent {

node {

label 'ansible'

}

}

//These params will be displayed for user input when running a build, They are also accepted by the API

parameters {

string defaultValue: 'hosts-staging', description: '''Inventory file to be used''', name: 'inventory', trim: false

string defaultValue: 'staging-haproxy01,staging-haproxy02', description: 'Specify the haproxy host group from inventory which will be used for the HA proxy config updates. ex: haproxy-uk', name: 'haproxy\_host\_group', trim: false

}

stages {

// stage('Build initiation notification') {

// steps {

// slackSend (color: '#D7DBDD', message: "STARTED: Job '${env.JOB\_NAME} [${env.BUILD\_NUMBER}]' (${env.BUILD\_URL})")

// }

// }

stage('SCM checkout deploy') {

steps {

dir("checkout\_dir") {

checkout([$class: 'RepoScm',jobs: 8,manifestPlatform: 'auto',manifestRepositoryUrl: 'git@deploy-git.company.net:devops/ansible.git',quiet: true])

}

}

}

stage ('ENV variables') {

steps {

sh 'printenv'

}

}

stage ('Workarounds') {

steps {

sh 'echo "Removed the existing workarounds"'

// sh 'find ./ -mindepth 1 -delete'

// sh 'find ./ -name .git -type d -exec git -C {\\}/.. clean -df \\;'

// sh 'mkdir -p group\_vars/all/ ; rm -f group\_vars/all/private.yml;ln -s $ANSIBLE\_PASSWORDS group\_vars/all/private.yml'

}

}

stage ('HA proxy config') {

steps {

dir("checkout\_dir") {

withCredentials([string(credentialsId: 'ANSIBLE\_VAULT\_PASSWORD', variable: 'ANSIBLE\_VAULT\_PASSWORD')]) {

withCredentials([string(credentialsId: 'SUDO\_PASSWORD', variable: 'SUDO\_PASSWORD')]) {

withCredentials([file(credentialsId: 'mspanakis-private-ssh-key', variable: 'ANSIBLE\_PRIVATE\_KEY')]) {

sh 'ansible-playbook roles/haproxy/haproxy.yml -i $inventory -t update-servers-list -b --limit $haproxy\_host\_group -u mspanakis --private-key=$ANSIBLE\_PRIVATE\_KEY'

}

}

}

}

}

}

}

}

+++++

Configuration.yml:

view:

name: Configuration

description: Jobs to apply configurations to servers

regex: '.\*config.\*'

jobs:

- name: ha-proxy-config-update

daysToKeep: 30 # Days to keep individual builds

description: Update HA proxy configuration

regular\_pipeline:

git:

url: git@deploy-git.company.net:di/jenkins-jobs.git

id: provision-multidev-environment-1

credentials: aa8c1dff-feab-4dc1-8468-7ac88501b378

# The safest way is to use the refs/heads/<branchName> syntax. This way the expected branch is unambiguous.

branches: 'refs/heads/master'

scriptPath: jenkinsfiles/Jenkinsfile\_HA\_Proxy\_config\_update

template: templates/pipeline.dsl.j2