CLasses:

**1. Jenkins Active Passive, Dedicated & On-Demand Slaves**

**2. Docker Containers as Jenkins Slaves. Used sreeharshav/pack-terr-jenkinsci-ssh-slave:v1 to perform terraform deployment using Jenkins Container Slave.**

**3. Jenkins RBAC, AD Integration, TomcatSnakes Artifact, S3 profile using AWS Role.**

# add PPA repo

sudo add-apt-repository ppa:openjdk-r/ppa

# Install OpenJDK

sudo apt-get update

sudo apt-get install -y openjdk-8-jdk nfs-common unzip jq

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

sudo unzip awscliv2.zip

sudo ./aws/install

# Switch to Java 8

sudo update-alternatives --config java

# Check Java installation

java -version

# Add PPA repo for Jenkins stable LTS

wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -

sudo sh -c "echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list"

sudo apt-get update

# Install Jenkins

sudo apt-get install jenkins

# Start Jenkins at boot time

sudo systemctl start jenkins.service

sudo systemctl enable jenkins.service

#Re-directing Port

sudo systemctl stop jenkins

nano /lib/systemd/system/jenkins.service

Environment="JENKINS\_PORT=80"

AmbientCapabilities=CAP\_NET\_BIND\_SERVICE

systemctl daemon-reload

sudo systemctl restart jenkins

For HTTPS you need to configure jks datastore with certificate and Environment="JENKINS\_HTTPS\_PORT=443"

#Environment="JENKINS\_HTTPS\_KEYSTORE=/path/to/keystore.jks"

#Environment="JENKINS\_HTTPS\_KEYSTORE\_PASSWORD=s3cR3tPa55w0rD"

Backup & Restore:

To Backup:

su - jenkins

tar -cvf jenkins\_backup.tar -C $HOME .

To-Restore:

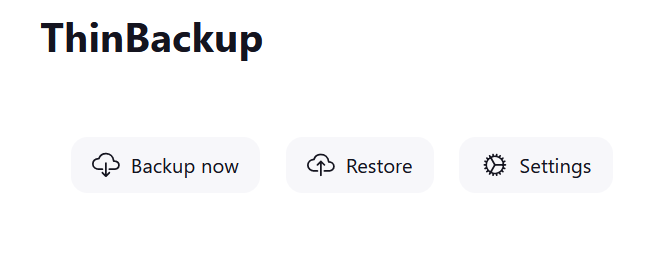
tar -xvf jenkins\_backup.tar

Backup & Restore from S3:

cd /var/lib/jenkins && aws s3 sync . s3://devsecopsb39jenkinsbackup

aws s3 sync s3://devsecopsb39jenkinsbackup /restore/

Backup & Restore using Thinbackup:



--------------------------------------------------------------------------------------------------------------------------------

# Pull official Docker image for Jenkins LTS

docker pull jenkins/jenkins:lts

# Start Docker container

docker run --rm -dit -p 80:8080 -p 50000:50000 --name jenkins --hostname jenkins -v jenkins\_home:/var/jenkins jenkins/jenkins:lts

docker run -p 8080:8080 -p 50000:50000 -v jenkins\_home:/var/jenkins jenkins/jenkins:lts

# Start Docker container in detached mode

docker run -p 8080:8080 -p 50000:50000 -v jenkins\_home:/var/jenkins -d jenkins/jenkins:lts

**Jenkins Docker Integration:**

**On Docker Host as below:**

Jenkins master must be installed with Docker engine . We need to edit the Docker host docker file to allow jenkins server to run the images.

nano /lib/systemd/system/docker.service and add following line.

ExecStart=/usr/bin/dockerd -H unix:// -H tcp://0.0.0.0:2375

systemctl daemon-reload

service docker restart

Run curl http://localhost:2375/images/json to confirm.

**On jenkins Server:**

sudo usermod -a -G docker jenkins

sudo service jenkins restart

*If above dont work:*

usermod -aG root jenkins

chmod 664 /var/run/docker.sock

chmod 777 /var/run/docker.sock

From the remote host run :

docker -H tcp://<DockerHost>:2375 ps

docker -H tcp://10.1.1.130:2375 ps

docker -H tcp://10.1.1.254:2375 run --rm -dit --name jenkins2 --hostname jenkins2 --network ansible\_nw sreeharshav/jenkins:v2

**Jenkins Build Script using the webhooks:**

#!/bin/bash

VERSION=$(date +%H-%M-%S)

docker build -t sreeharshav/testingbuild:${VERSION} .

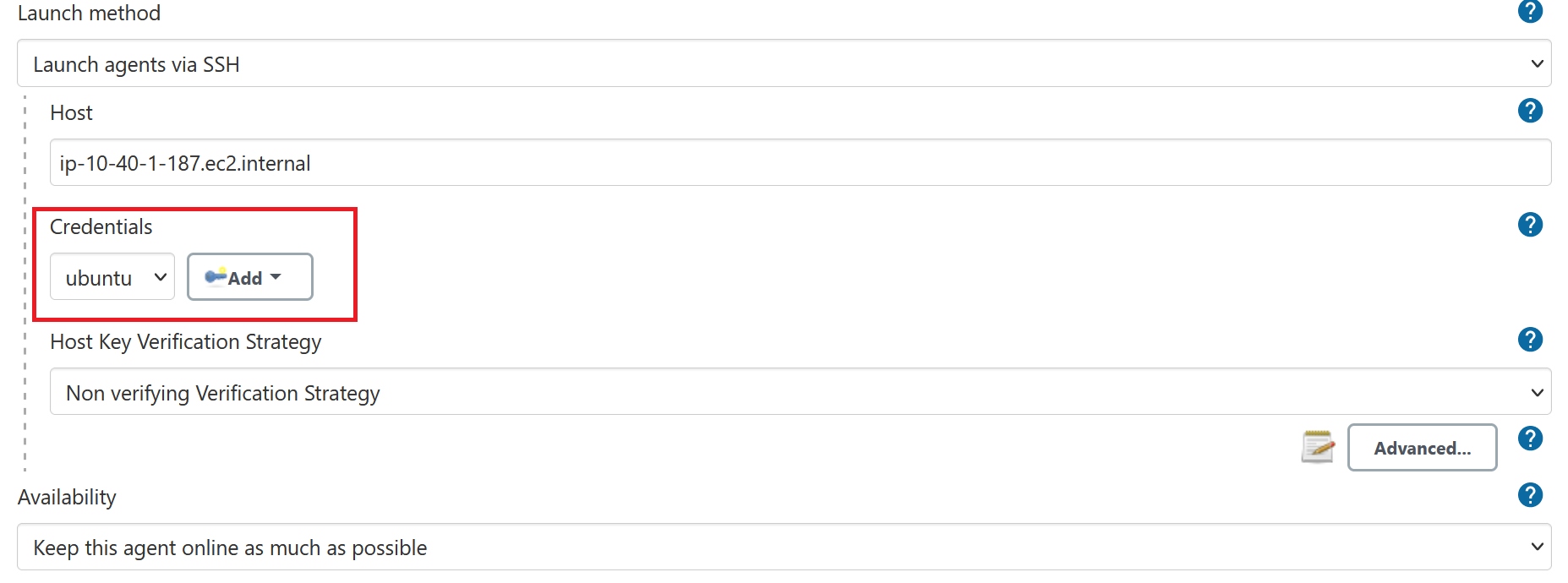
docker push sreeharshav/testingbuild:${VERSION}

docker -H tcp://10.1.1.200:2375 stop nginx

docker -H tcp://10.1.1.200:2375 run --rm -dit -p 8000:80 --name nginx --hostname nginx sreeharshav/testingbuild:${VERSION}



**For normal EC2 as Slave Machine running 24/7:**

****

**Slave Env Variables for Terraform & AWSCLI:**

AWS\_ACCESS\_KEY\_ID="anaccesskey"

AWS\_SECRET\_ACCESS\_KEY="asecretkey"

AWS\_DEFAULT\_REGION="us-west-2"

**For Azure:**

ARM\_CLIENT\_ID="00000000-0000-0000-0000-000000000000"

ARM\_CLIENT\_SECRET="00000000-0000-0000-0000-000000000000"

ARM\_SUBSCRIPTION\_ID="00000000-0000-0000-0000-000000000000"

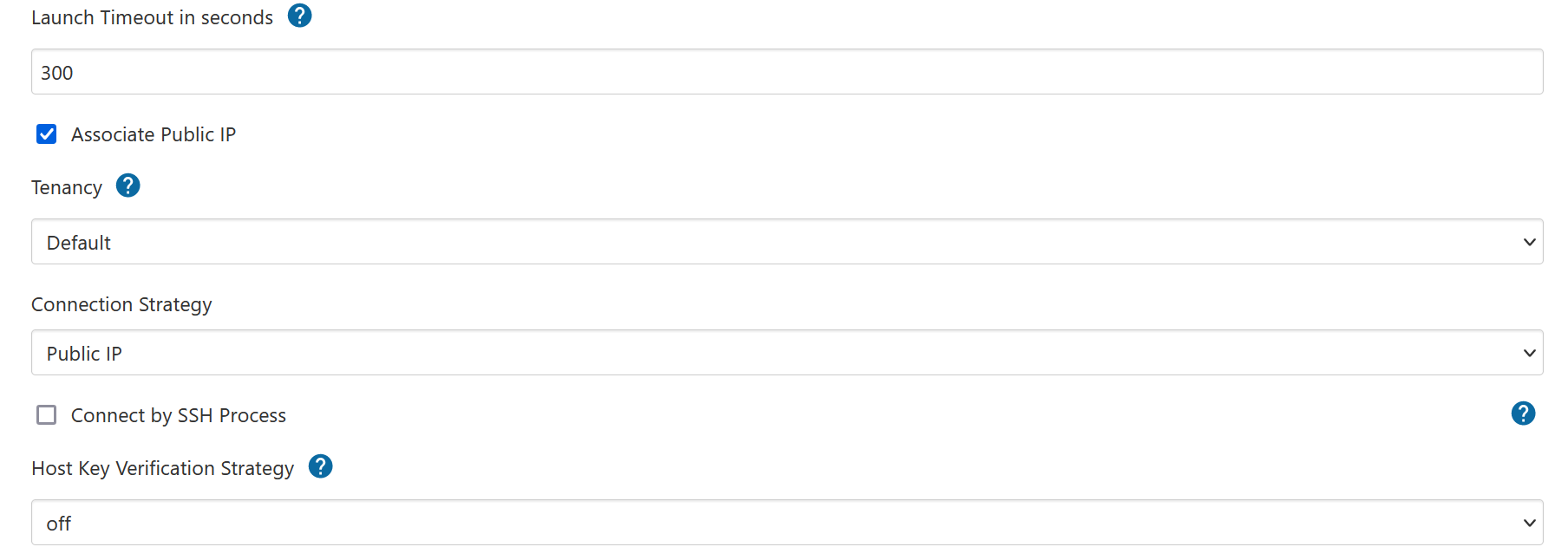
ARM\_TENANT\_ID="00000000-0000-0000-0000-000000000000"

Azure Storage Account Access Key Env Varible for Jenkins:

ARM\_ACCESS\_KEY=”AzLdN6Lx9Y4s9RZWhh4IxZpSQJs4dfC1Wp/F1DnxOEW2XhbTlDKc0fgRXgAVOV7ab5z1lrOPNZa50AdisyaasddadadasdasdMhFw==”

**Using AMAZON EC2 as on-demand jenkins slave:**

1. Create a AMI with JDK 1.8 and note down the AMI ID.-Mandatory
2. Create a users in AWS IAM with Access Key and Secret Key. This user must have permission to deploy EC2 machines.
3. Create a Jenkins Credentials using the above access and secret key.
4. Install Amazon EC2 plugin in Jenkins.
5. From Manage Jenkins -> Configure System, go down and select Amazon EC2 Cloud.
6. Enter the Access details and AMI Details and make sure proper labeling is done.



1. Create a job and provide the label and run the job.
2. This should deploy a new EC2 instance and run the job.
3. Based on the idle minutes selected, the EC2 machine will be terminated.

**Using Docker Container as jenkins slave:**

Use Ubuntu 20.04 as a Docker host for this lab.

1. We are going to use the Docker Image jenkinsci/ssh-slave:latest

sreeharshav/pack-terr-jenkinsci-ssh-slave:v1 container Terraform & Packer

Dockerfile for above is in the folder.

1. Install the Docker Plugin from Manage Plugins page.
2. Perform the following steps in the Docker Host. In our case Jenkins and Docker host are the same.

nano /lib/systemd/system/docker.service and add following line.

ExecStart=/usr/bin/dockerd -H unix:// -H tcp://0.0.0.0:2375

systemctl daemon-reload

service docker restart

curl http://localhost:2375/images/json

Run curl http://localhost:2375/images/json to confirm.

**On jenkins Server:**

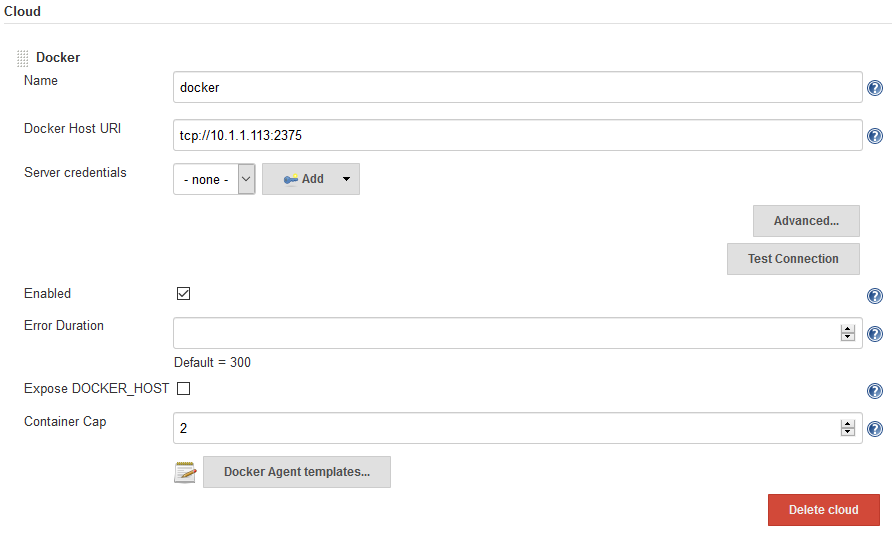
sudo usermod -a -G docker jenkins

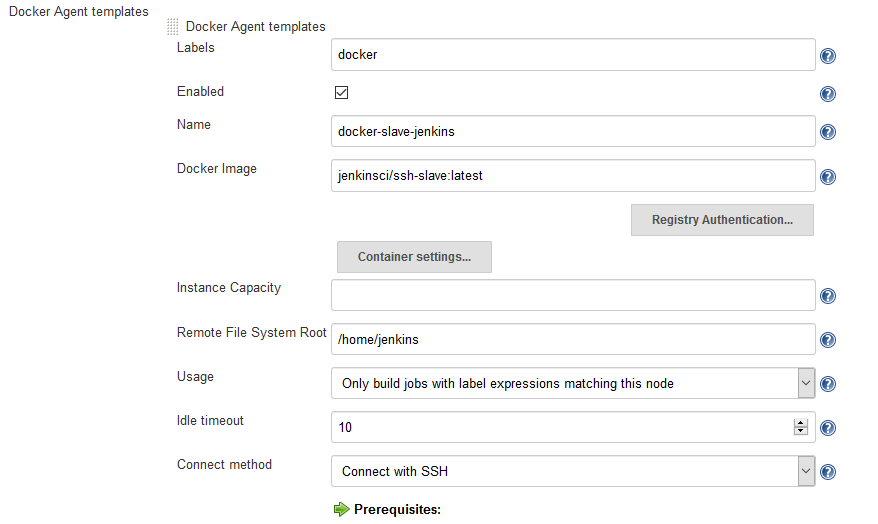
sudo usermod -a -G root jenkins

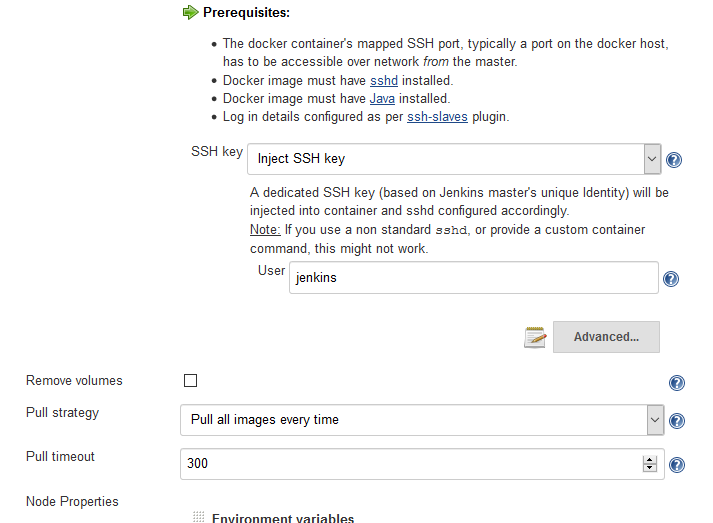
sudo service jenkins restart

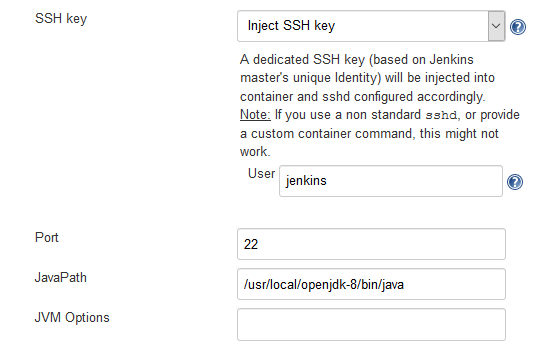
1. Similar to Amazon EC2 Cloud , we need to add the Docker Cloud. We also need to have a ssh key-pair.
2. Change user to jenkin and perform a ssh-keygen which generates public(id\_rsa.pub) and private(id\_rsa) key. Copy the contents of public key.
3. Add the docker cloud and enter all necessary details and make sure you give proper label. Dont forget to add the Enviroment Variable JENKINS\_SLAVE\_SSH\_PUBKEY.
4. Create a job with above labe and run it and it must deploy a container and run the job and once the job is done, container will be terminated.
5. /usr/local/openjdk-8/bin/java
6. http://ec2-44-200-162-110.compute-1.amazonaws.com:8080/log/all

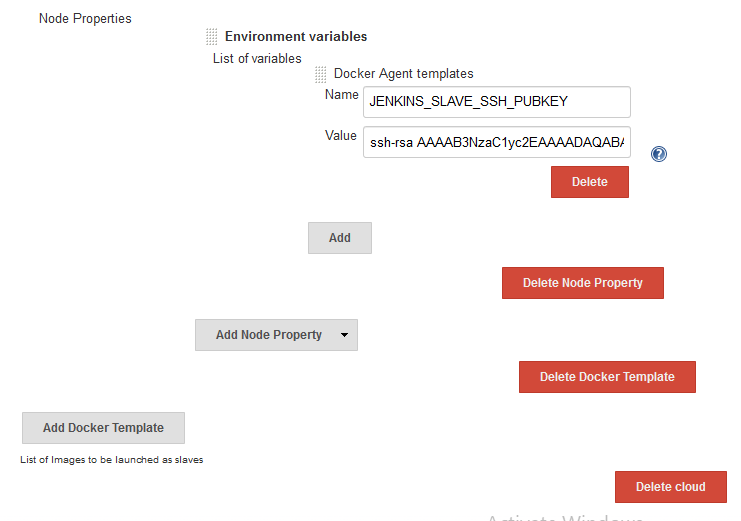
**FOLLOWING PIC IS VALID IF YOU USE SSH INJECT KEY:**

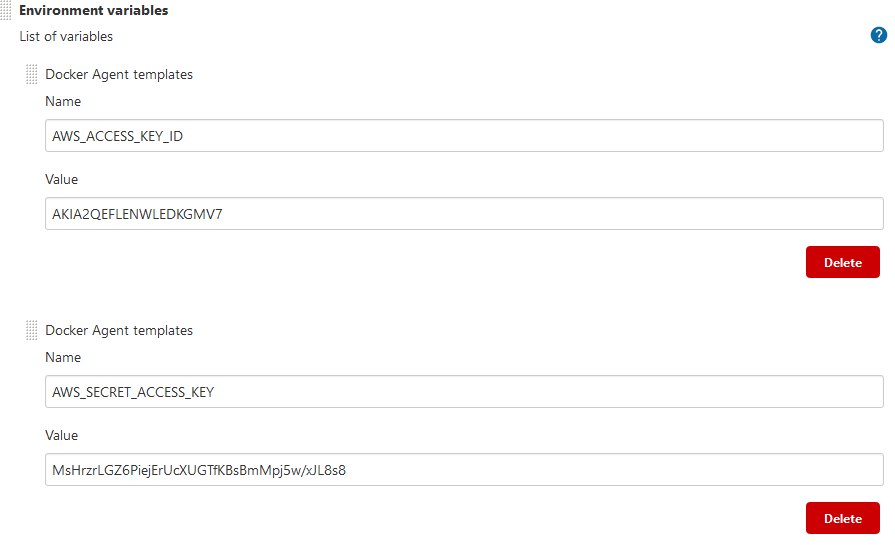




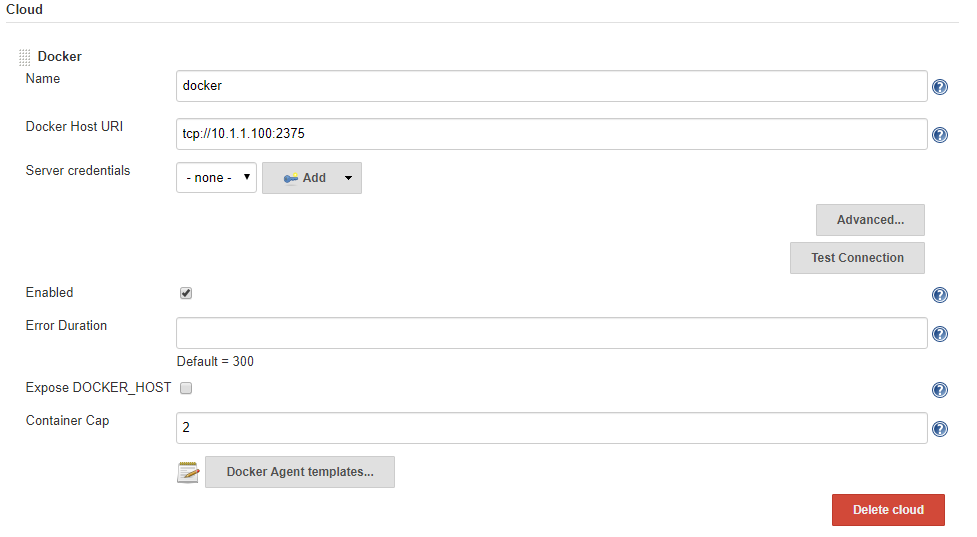


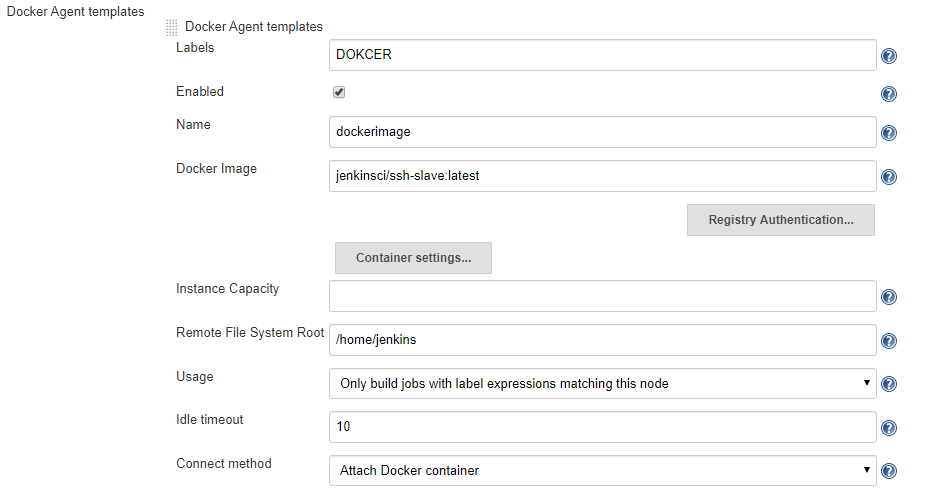


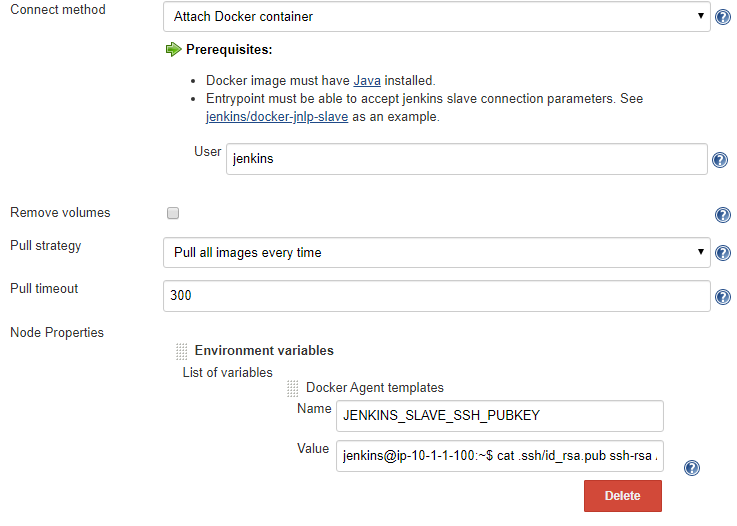




FOLLOWING PIC IS VALID IF YOU USE ATTACH CONTAINER (NOT SSH):







**Jenkins RBAC:**

1. Install [Role-based Authorization Strategy](https://wiki.jenkins.io/display/JENKINS/Role+Strategy+Plugin) plugin.
2. Got to Manage Jenkins - > Global Security-> Select “Role-Based Strategy”.
3. Check if the Manage and Assign Roles are visible.
4. Create two users, Allan and Bob. Allan works on Java and Bob works on Python.
5. We want Allan access only Java Projects and Bob Python projects only.
6. Create Global Roles with total read access and project roles for java and python.
7. Assign the Java role to Allan and Python role to Bob.
8. Create test projects with names java and python.
9. Login with Allan and Bob and check if they are able to run.

Project Role Pattern:

java.\*

python.\*

terraform.\*



**WHEN USING JENKINS WITH EFS USE THE FOLLOWING MOUNT:**

(Works with Amazon Linux and Ubuntu 16.04)

fs-fa0cc77a.efs.us-east-1.amazonaws.com:/ /var/lib/jenkins nfs defaults 0 0

**JENKINS HIGH-AVAILIBILITY:**

Create a user and login with that user and create the token which can be used to reload the config rather than restart the server. Username used for this example is reloadconfig

Make sure you have restarted the secondary jenkins service so that the user is visible to secondary.

Dont configure Secondary to Primary Reload Config because it will overwrite when you configure EC2 Slaves or Disable the Cron on the primary before configuring EC2 Slaves.

**wget** [**http://ec2-3-220-231-126.compute-1.amazonaws.com:8080/jnlpJars/jenkins-cli.jar**](http://ec2-3-220-231-126.compute-1.amazonaws.com:8080/jnlpJars/jenkins-cli.jar)

[**https://linuxbuff.wordpress.com/2020/07/29/howto-reload-jenkins-configuration-from-the-command-line/**](https://linuxbuff.wordpress.com/2020/07/29/howto-reload-jenkins-configuration-from-the-command-line/)

Download Jenkins-cli.jar to root folder and make sure you give absolute path in the cli.

java -jar /jenkins-cli.jar -s http://<secondary-jenkins-publicip>:8080 -auth reloadconfig:11753e581628002506468ce3e44a8a1076 reload-configuration

java -jar /jenkins-cli.jar -s http://ec2-3-220-231-126.compute-1.amazonaws.com:8080 -auth reloadconfig:11753e581628002506468ce3e44a8a1076 reload-configuration

<https://www.opcito.com/blogs/how-to-configure-jenkins-with-high-availability>

On the secondary create a cron scheduler as below:

Change user as jenkins using su - jenkins

Copy jenkins-cli.jar to jenkins home folder which is /var/lib/jenkins

crontab -e

java -jar /var/lib/jenkins/jenkins-cli.jar -s http://<secondary-fqdn>:8080 -auth reloadconfig:113b620a011af11b98d913003f1e190e69 reload-configuration

crontab -l

Same configuration needs to be performed on the primary as well incase if you want to sync from secondary to primary which is needed if you use a load balancer.

**Jenkins version upgrade:**

<https://www.thegeekstuff.com/2016/06/upgrade-jenkins-and-plugins/>

App.py

import os

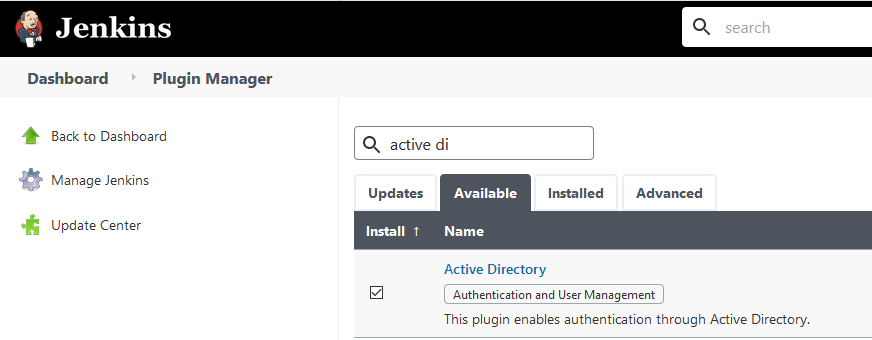
numcount = os.getenv("COUNT")

evenums = [ x for x in range(int(numcount)) if x%2 == 0 ]

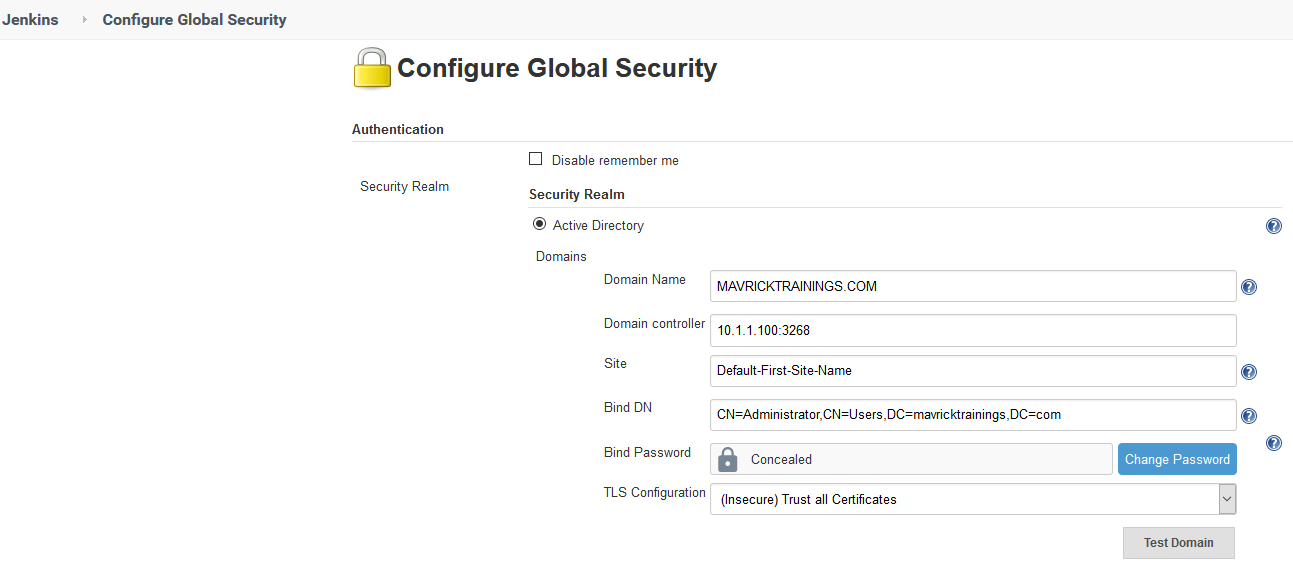
print(evenums)

AD Integration:

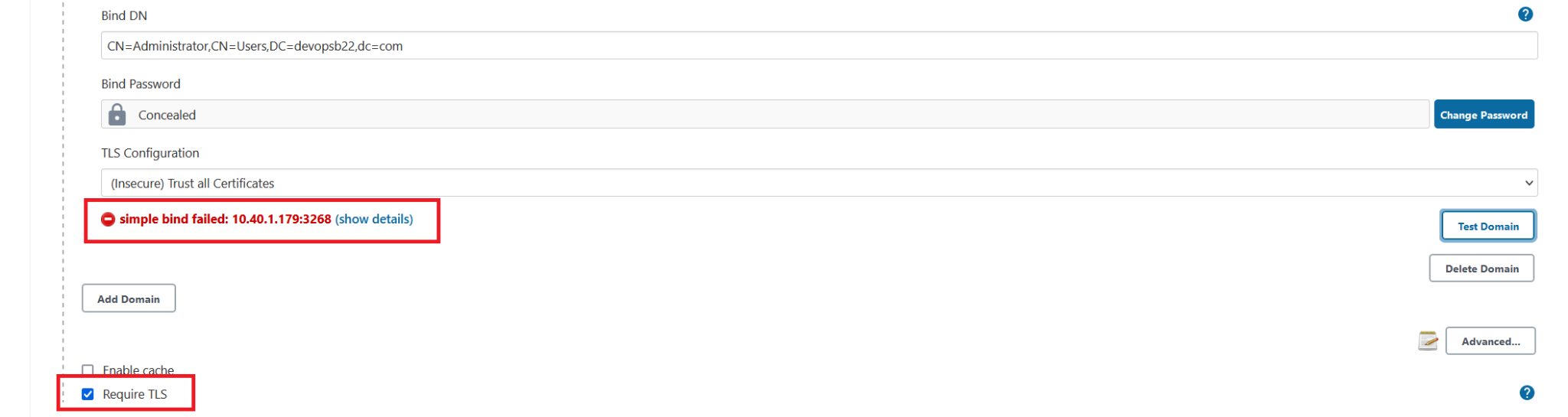
--------------------



CN=Administrator,CN=Users,DC=devopsb14,dc=xyz



Following issue will occur if “Requrire TLS” is selected.



**Docker Jenkins Slave Build Script:**

#!/bin/bash

ls -al

hostname

wget https://releases.hashicorp.com/packer/1.6.2/packer\_1.6.2\_linux\_amd64.zip > /dev/null 2>&1

wget https://releases.hashicorp.com/terraform/0.13.1/terraform\_0.13.1\_linux\_amd64.zip > /dev/null 2>&1

unzip packer\_1.6.2\_linux\_amd64.zip

unzip terraform\_0.13.1\_linux\_amd64.zip

ls -al

./terraform version

./packer version

**Removing AD config from jenkins:**

<https://mohitgoyal.co/2017/02/10/disable-security-in-jenkins/>

<https://support.cloudbees.com/hc/en-us/articles/206598218-How-do-I-login-to-Jenkins-when-I-locked-myself-out->

If you are lockedout, delete the following sections in /var/lib/jenkins/config.xml

<authorizationStrategy class="com.michelin.cio.hudson.plugins.rolestrategy.RoleBasedAuthorizationStrategy">

.....

</authorizationStrategy>

<securityRealm class="hudson.security.HudsonPrivateSecurityRealm">

...

</securityRealm>

-----------------------------------------------------------------------------------------------------------------------

**Deploying to Tomcat Server:**

1. Install Bitnami Tomcat Server
2. Add the jenkins public key to root ~/.ssh/authorized\_keys to the tomcat server.
3. scp -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null ROOT\*.war root@35.170.70.29:/opt/bitnami/tomcat/webapps

#!/bin/bash

ls -al

rm -rf ROOT\*

./build.sh

mv ROOT.war ROOT${BUILD\_ID}.war

scp -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null ROOT\*.war root@54.175.148.142:/opt/bitnami/tomcat/webapps/mywebapp.war

ls -al

---------------------------------------------------------------------------------------------------------------------------

JFROG CONFIG:

use t3.medium or t2.medium

docker run --rm --name artifactory -d -p 8081:8081 -p 8082:8082 docker.bintray.io/jfrog/artifactory-pro:latest ----> License Needed

docker run --rm --name artifactory -d -p 8081:8081 -p 8082:8082 docker.bintray.io/jfrog/artifactory-cpp-ce:latest

admin/password are default credentials

<https://github.com/jfrog/project-examples/tree/master/jenkins-examples/pipeline-examples/declarative-examples/jenkins-with-jfrog-pipelines>

Jenkins Backup & Restore:

https://medium.com/@\_oleksii\_/how-to-backup-and-restore-jenkins-complete-guide-62fc2f99b457

<https://blog.knoldus.com/backup-and-restoring-in-jenkins/>

**Maven Installation:**

<https://www.javatpoint.com/maven-example>

Use ubuntu-18.04 Installed with Jenkins

cd /usr/local/bin/

wget <https://mirrors.estointernet.in/apache/maven/maven-3/3.8.1/binaries/apache-maven-3.8.1-bin.tar.gz>

tar xzf apache-maven-3.8.1-bin.tar.gz

cd /usr/local/bin/apache-maven-3.8.1/bin

./mvn -version

PATH=$PATH:/usr/local/bin/apache-maven-3.8.1/bin

root@ip-10-1-1-150:~# mvn -version

Apache Maven 3.8.1 (05c21c65bdfed0f71a2f2ada8b84da59348c4c5d)

mvn archetype:generate -DgroupId=com.javatpoint -DartifactId=CubeGenerator -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

cd CubeGenerator/

mvn clean compile

cd target/classes/

java com.javatpoint.App

Packaging with Maven:

cd /tmp/CubeGenerator#

mvn package