Jenkins master slave poc

Use case:

1. In our current architecture, we're using multiple jenkins instances to build and deploy in multiple AWS accounts and multiple environment.
2. We're almost running 4-5 higher performance instances for the jenkins servers and the costs involved in running these instances is also higher.
3. These jenkins instances are running 24*7 whenever the jobs are running or the jenkins setup is idle without running any job.

Requirement:

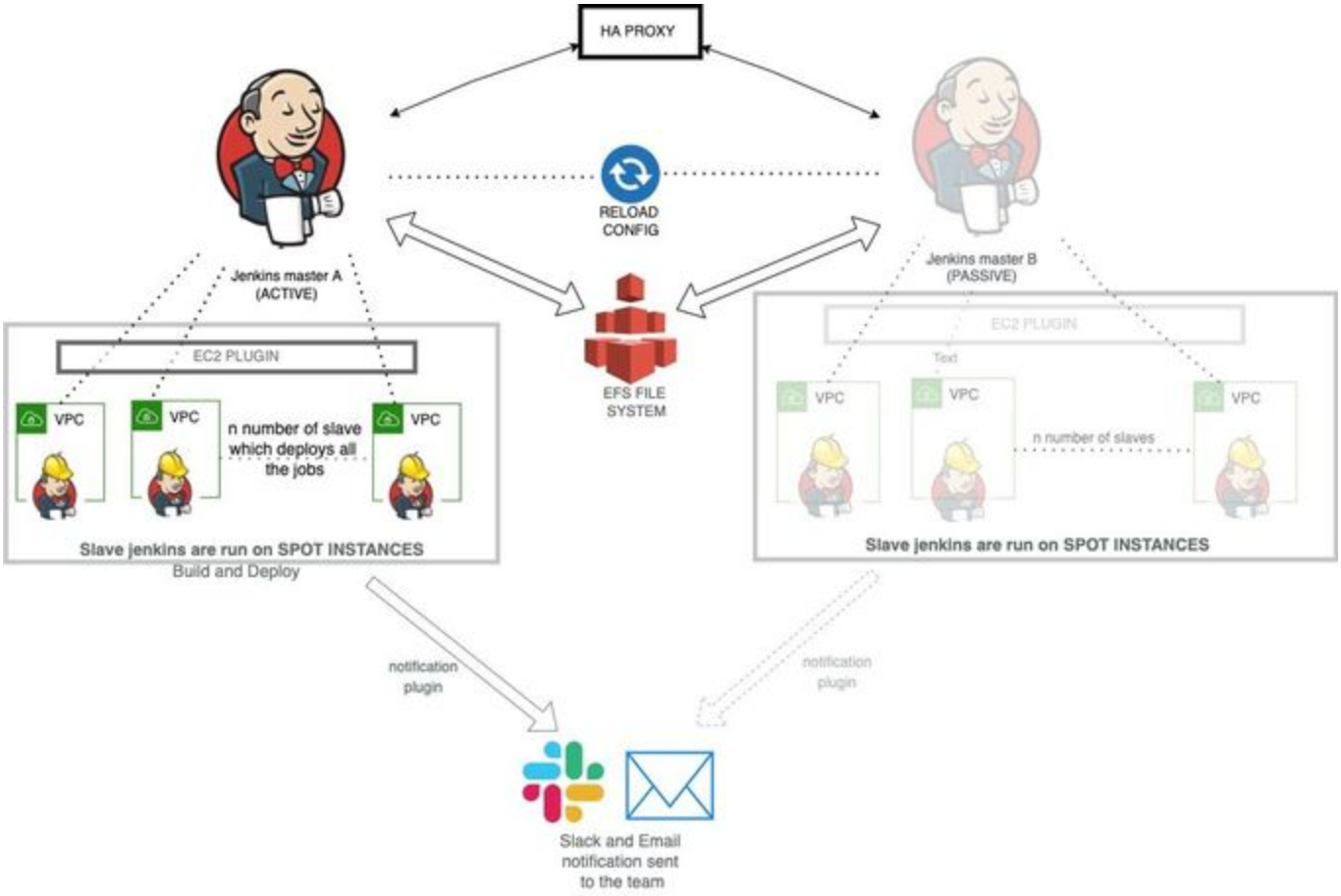
1. Making use of a single jenkins instance which can be used to run jobs in multiple AWS accounts.
2. To make sure we run on instances which has good performance and at the same time is stable.
3. Reducing the Costs involved in the jenkins setup.

Solution:

1. Using a Jenkins master node in one of the AWS accounts and connecting to multiple AWS accounts using the **EC2 plugin** and run the jobs in those particular accounts.
2. We'll be setting up 2 master jenkins (active and passive) with **HA proxy** setup which are high performance instances (t3a.large) and has a stability as well as a backup jenkins job.
3. We'll be using **SPOT instances** to run the jobs in multiple AWS accounts , and these instances will be up and running only when there's a job running which will terminate when automatically after a amount of time the jenkins job is idle.

How to Setup a Jenkins Master and Slave Architecture.

JENKINS-MASTER SLAVE ARCHITECTURE



Jenkins is an essential tool to run CI/CD pipelines for your projects.

However, if you have more and more pipelines running in single Jenkins, you may encounter some problems in availability and scalability.

Given that one Jenkins can cause a single point of failure problem and longer building time when executing the increasing number of CI/CD pipelines

STEPS FOLLOWED FOR THE IMPLEMENTATION:

1. Create a Jenkins master node.
2. Creating a AMI for slave node with dependencies.
3. Setup ec2 plugin for creating slave ec2 instance.
4. Jenkins job to test the Master-Slave working.
5. High Availability of the Master Jenkins.
6. Setup of reload configuration, HA proxy.
7. Setup of slack notification and email notification.

STEP 1: Setup of the Jenkins Master :

1. Login into your AWS console and create an ec2 instance .
    a. Login into the instance and install java JDK and jenkins on it.

```
sudo apt-get update;      -- updating the package in the
ubuntu machine
sudo apt-get install openjdk-11-jdk;     -- install java jdk

java -version;    -- checking the java version
```

    b. Allow the security groups to enable access to port 22 for ssh and port 8080 for jenkins.

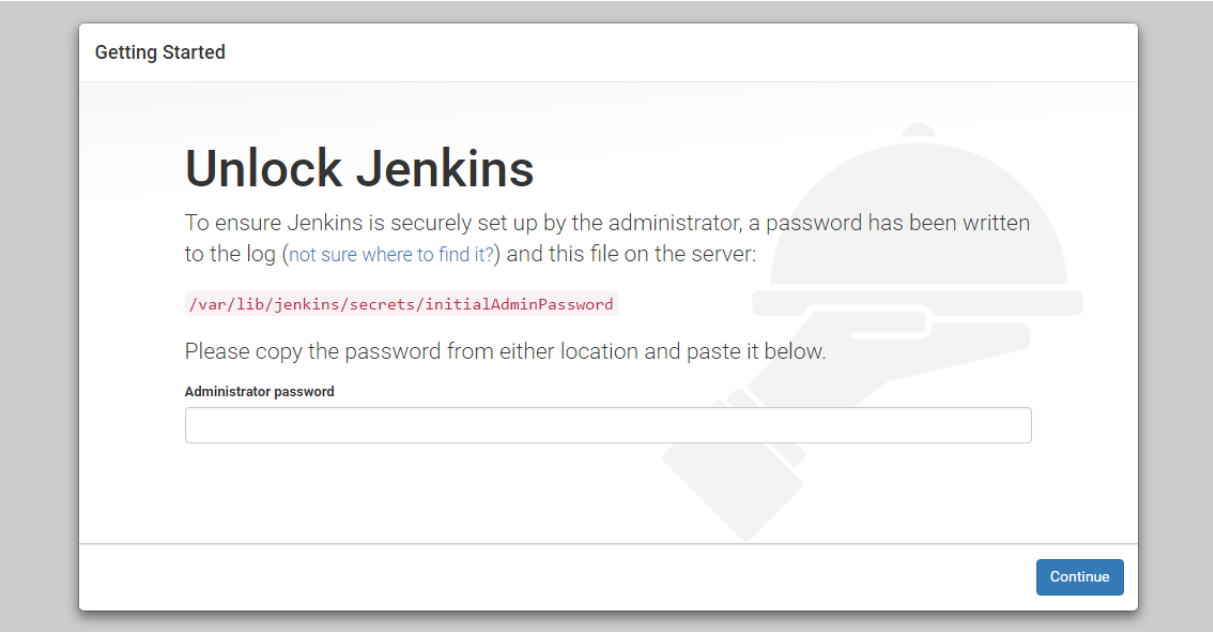
Security group rule ID	Type Info	Protocol Info	Port range Info	Source Info	
sgr-048d76499bf80e257	SSH	TCP	22	Custom	0.0.0.0/0
sgr-0daab860f89b57a22	Custom TCP	TCP	8080	Custom	::/0
sgr-05c670022922b35ea	Custom TCP	TCP	8080	Custom	0.0.0.0/0

c. install the Jenkins service.

```
-- (add the repository key to the system)
wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.
key | sudo apt-key add -

-- (append the Debian package repository)
sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable
binary/ > /etc/apt/sources.list.d/jenkins.list'

sudo apt update;

sudo apt install jenkins;  -- install jenkins


sudo systemctl start jenkins;  --start jenkins server

sudo systemclt status jenkins;  -- status of the jenkins server
```

```
● jenkins.service - Jenkins Continuous Integration Server
   Loaded: loaded (/usr/lib/systemd/system/jenkins.service; disabled; vendor preset: disabled)
   Active: active (running) since Fri 2022-06-10 09:33:26 UTC; 2 days ago
 Main PID: 12010 (java)
   CGroup: /system.slice/jenkins.service
           └─12010 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/jav...

Jun 13 07:35:14 ip-172-31-10-151.ap-south-1.compute.internal jenkins[12010]: ...
Jun 13 07:35:14 ip-172-31-10-151.ap-south-1.compute.internal jenkins[12010]: ...
Jun 13 07:35:14 ip-172-31-10-151.ap-south-1.compute.internal jenkins[12010]: ...
Jun 13 07:35:14 ip-172-31-10-151.ap-south-1.compute.internal jenkins[12010]: ...
Jun 13 07:35:14 ip-172-31-10-151.ap-south-1.compute.internal jenkins[12010]: ...
Jun 13 07:35:14 ip-172-31-10-151.ap-south-1.compute.internal jenkins[12010]: ...
Jun 13 07:35:14 ip-172-31-10-151.ap-south-1.compute.internal jenkins[12010]: ...
Jun 13 07:35:14 ip-172-31-10-151.ap-south-1.compute.internal jenkins[12010]: ...
Jun 13 07:35:14 ip-172-31-10-151.ap-south-1.compute.internal jenkins[12010]: ...
Jun 13 07:35:14 ip-172-31-10-151.ap-south-1.compute.internal jenkins[12010]: ...
Hint: Some lines were ellipsized, use -l to show in full.
```

2. Open the jenkins home page using the ip address of the instance. http://your instance ip address:8080
   a. Unlock your jenkins

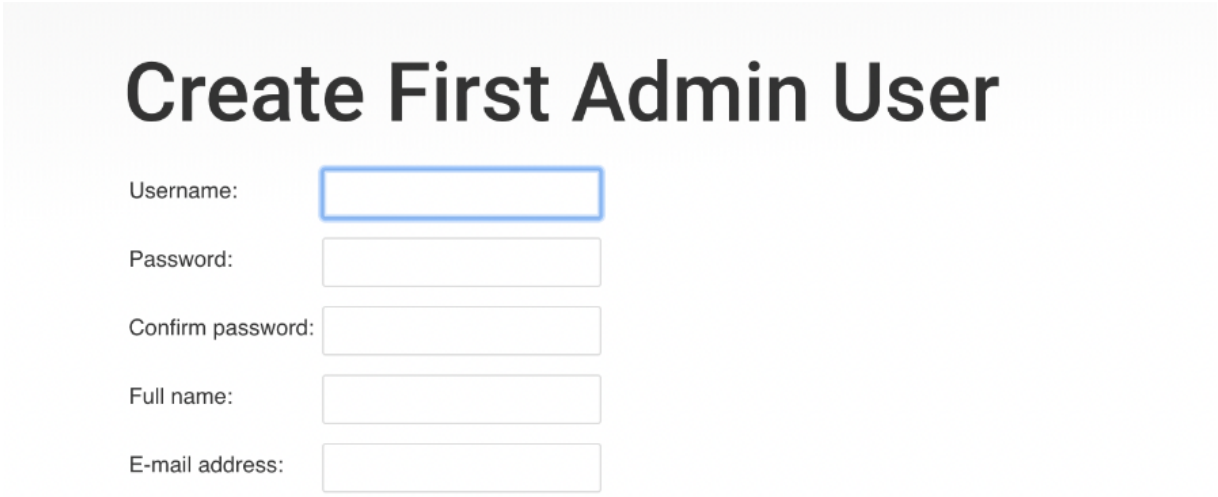


b. you'll find the admin password in the instance path or use this command below.

```
sudo cat  /var/lib/jenkins/secrets/initialAdminPassword
```

c. Next will be to install the suggested plugins .

d. Create your Admin user:



e. Then your jenkins is ready to work.

STEP 2: Creating a AMI for slave node.

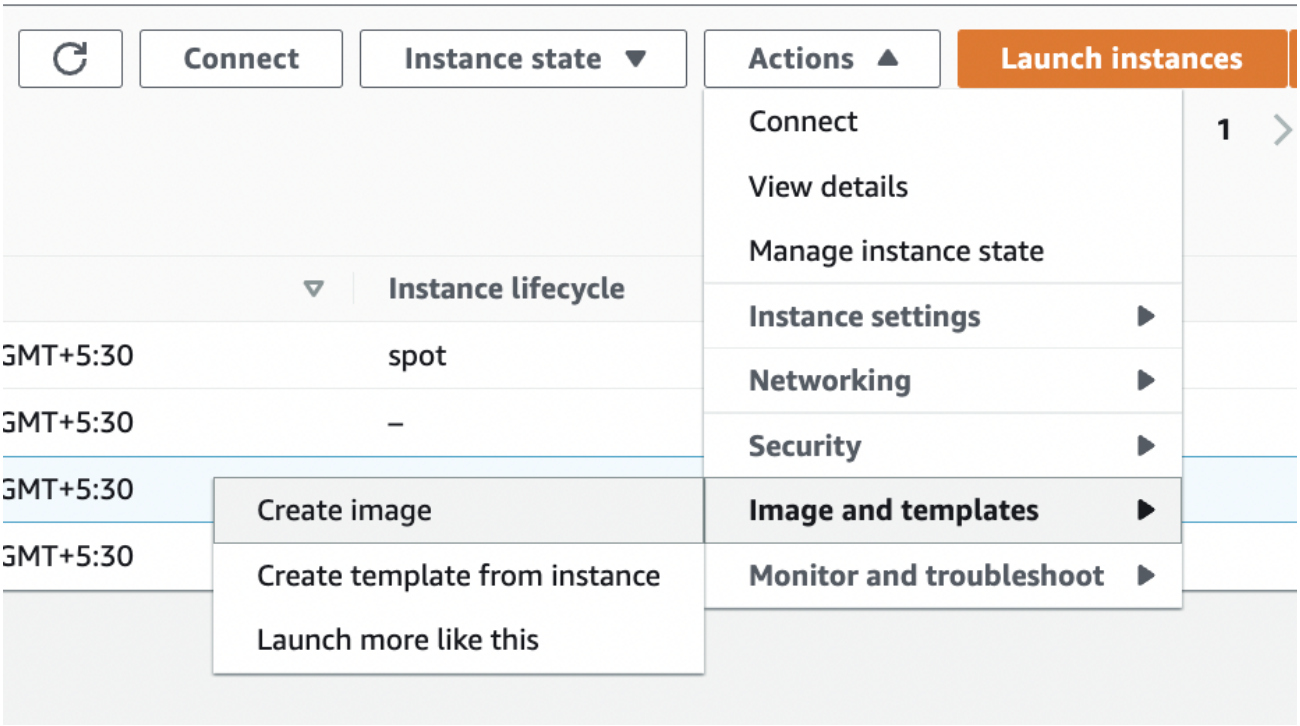
1. In the Aws ec2 console, create a ubuntu machine with 20.04 version.
2. Install java latest version.

```
sudo apt-get update;      -- updating the package in the ubuntu
machine
sudo apt-get install openjdk-11-jdk;     -- install java jdk

java -version;    -- checking the java version
```

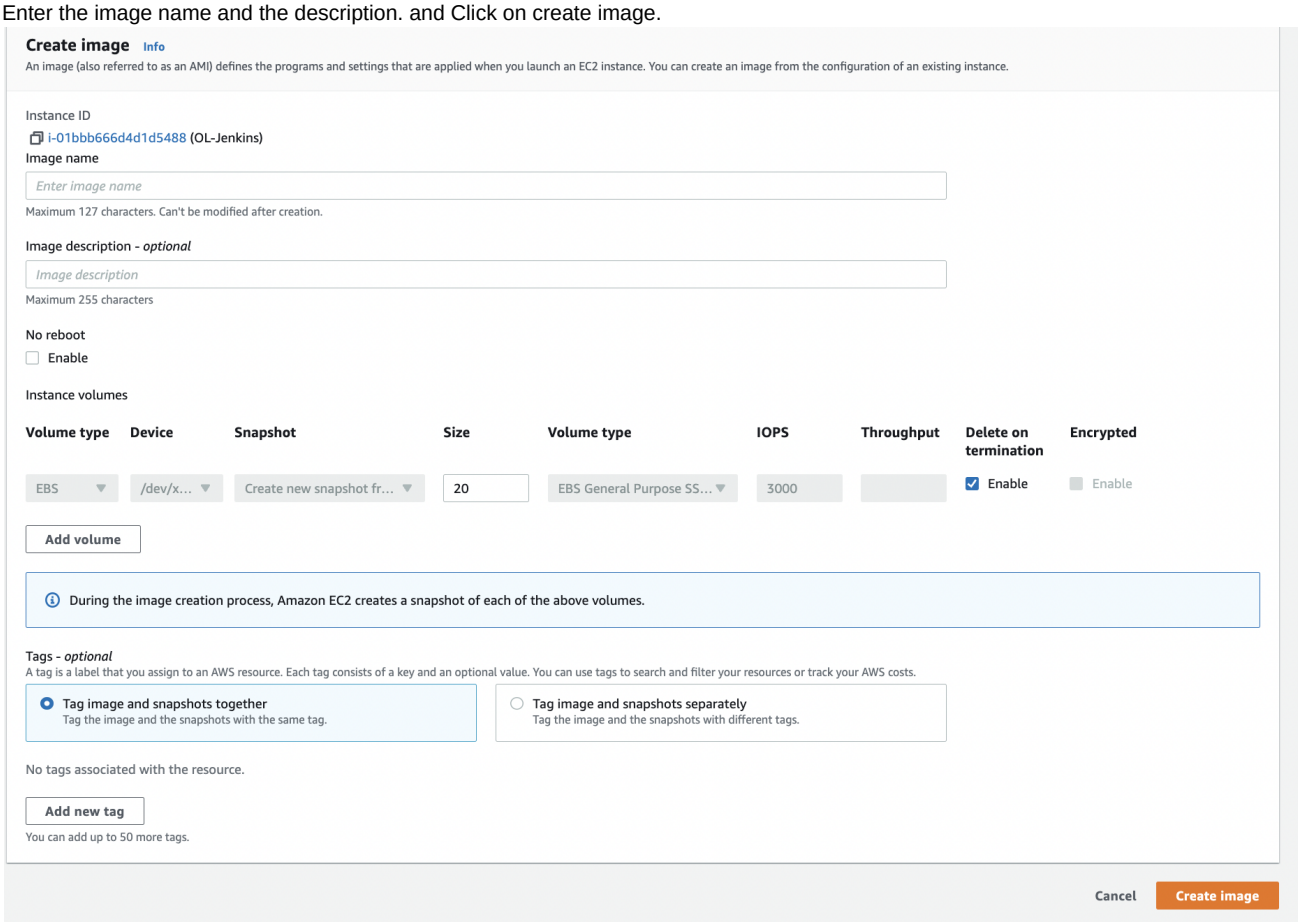
3. Install sonarqube.

```
wget https://binaries.sonarsource.com/Distribution/sonar-scanner-
cli/sonar-scanner-cli-4.7.0.2747-linux.zip
unzip sonar-scanner-cli-4.7.0.2747-linux.zip
mv sonar-scanner-4.7.0.2747-linux/ sonar-scanner
```

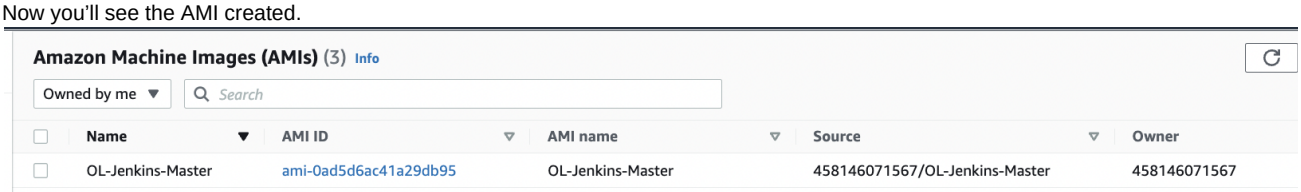
4. Go to the AWS ec2 console. and select the above instance click on Actions Image and template Create image.



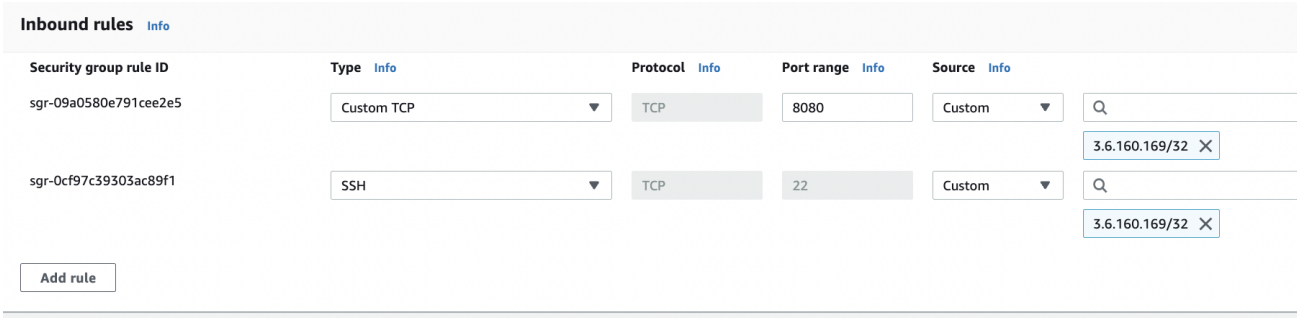
5. Enter the image name and the description. and Click on create image.



6. Now you'll see the AMI created.



7. Now we need to create a security group for the slave which has inbound access only from the jenkins master instance on port 22 (for ssh) and port 8080 (for jenkins) . So here we'll need to add the IP of the jenkins master for both these ports to access.



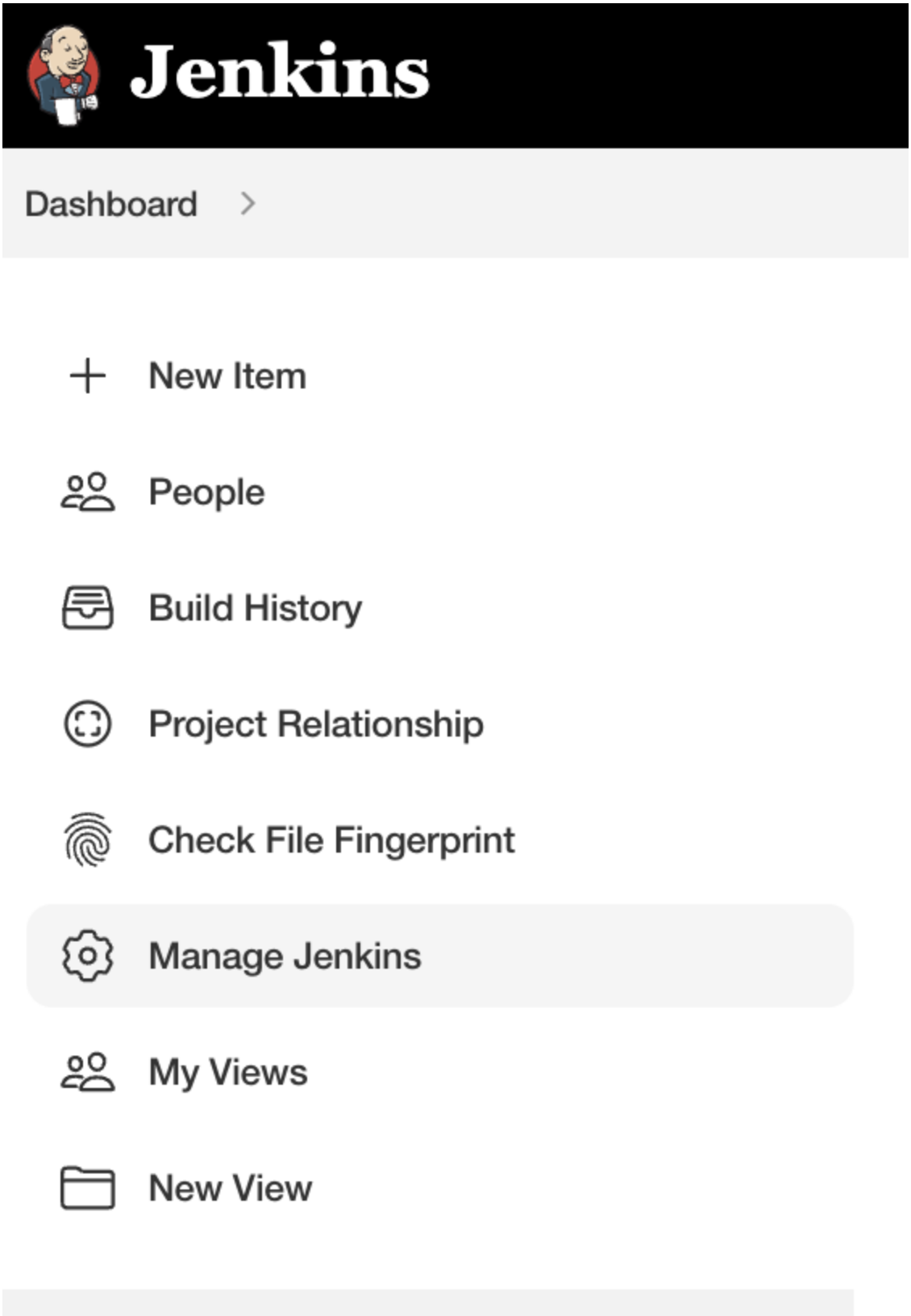
STEP 3: Setup ec2 plugin in jenkins to spin up slave instance.

To setup an ec2 plugin to spin up slave instances we need a Aws user to create slave instance and attach a role which supports the ec2 services.

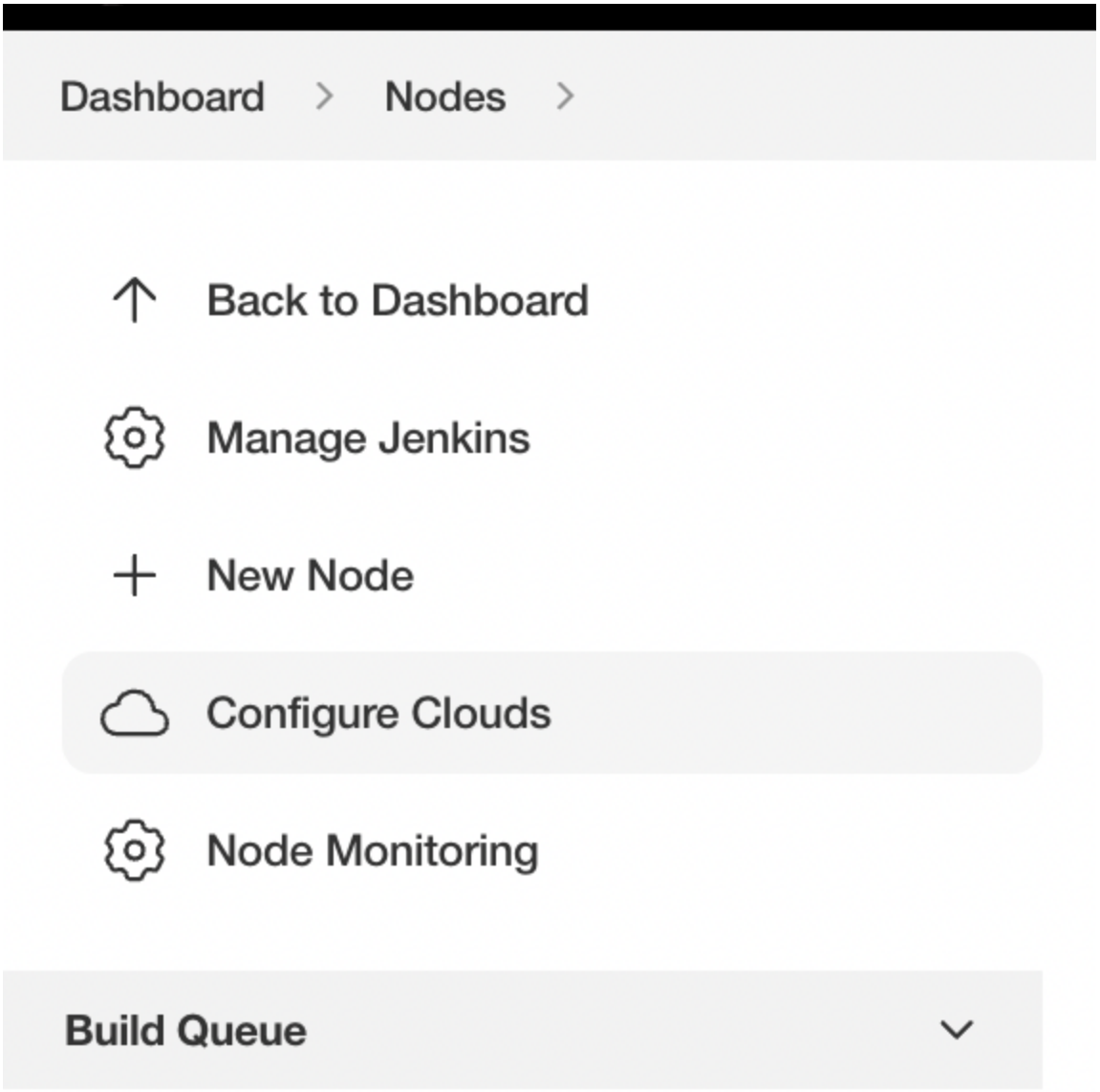
1. Create a programmatic access user with an access key and secret key. Name the instance as Jenkins-user.
2. Role to be attached to the user

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "ec2:*",
                "iam:ListInstanceProfiles",
                "iam:ListRoles",
                "iam:PassRole"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

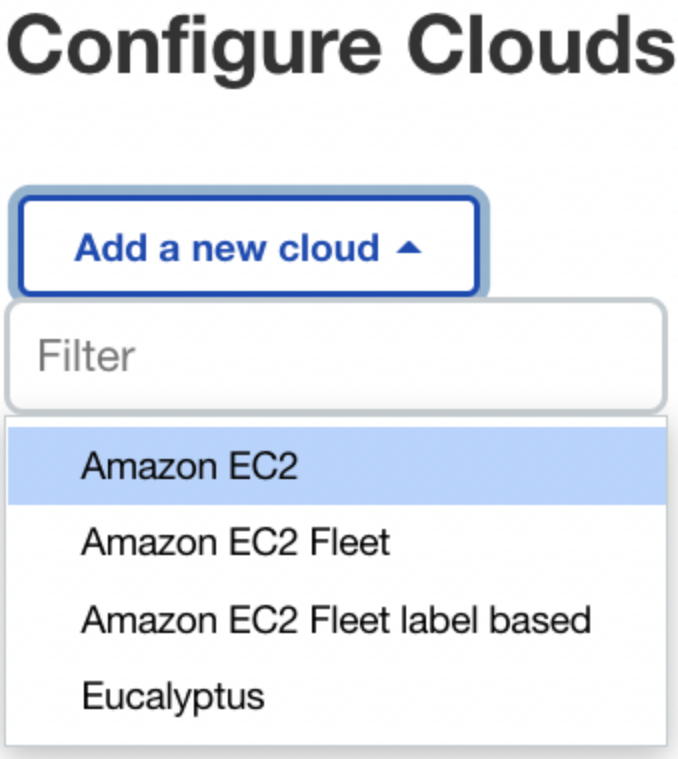
   Once the user and the role is adding we'll login into Jenkins to connect with this user created to check if we can spin up the jenkins-slave instance.
3. Login into Jenkins and go to manage plugin.



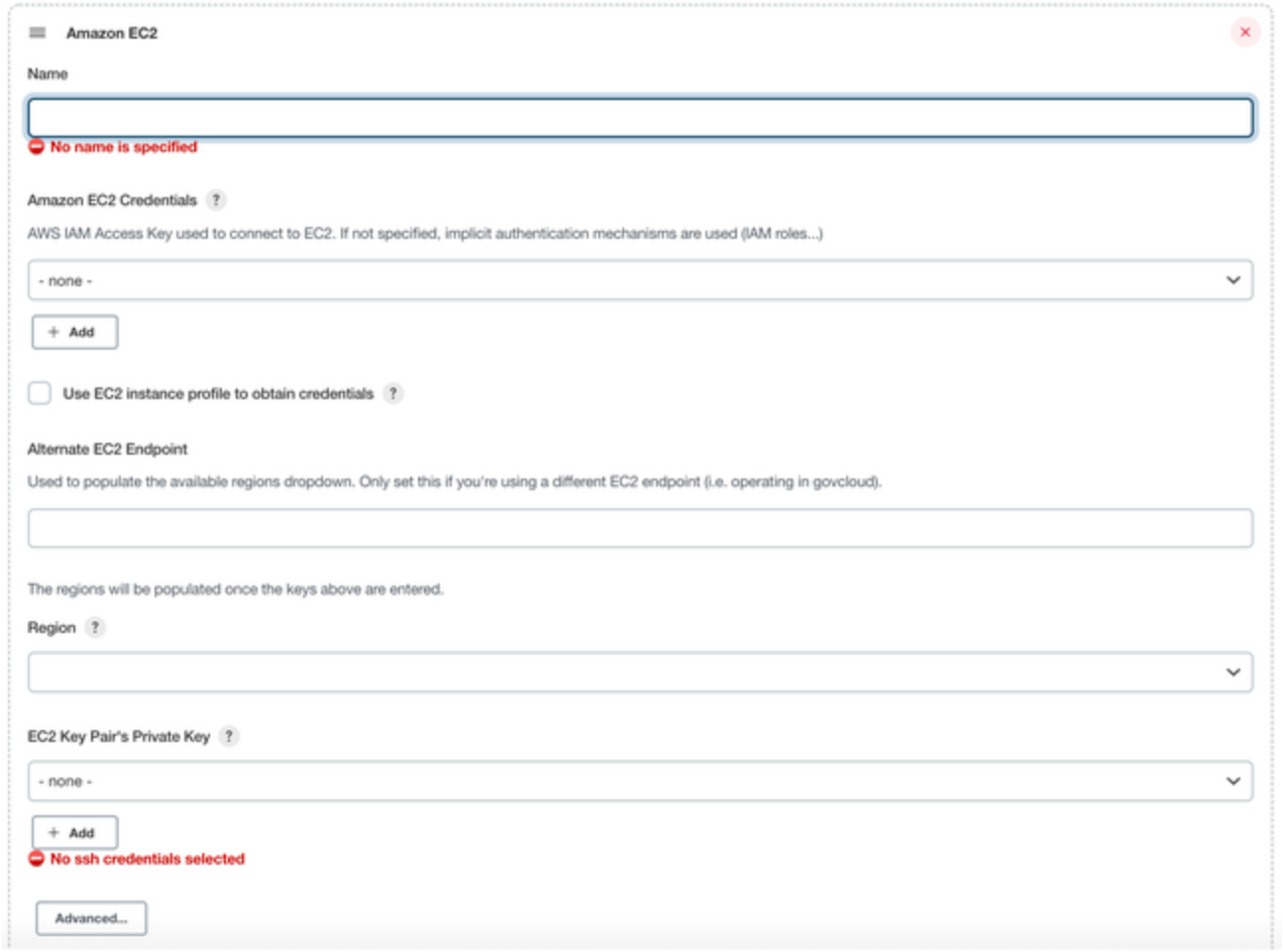
4. In the available tab , click on ec2 plugin download and install the ec2 plugin and restart jenkins once the installation is done.
5. Now go to jenkins Manage Jenkins Manage cloud and node Configure cloud.



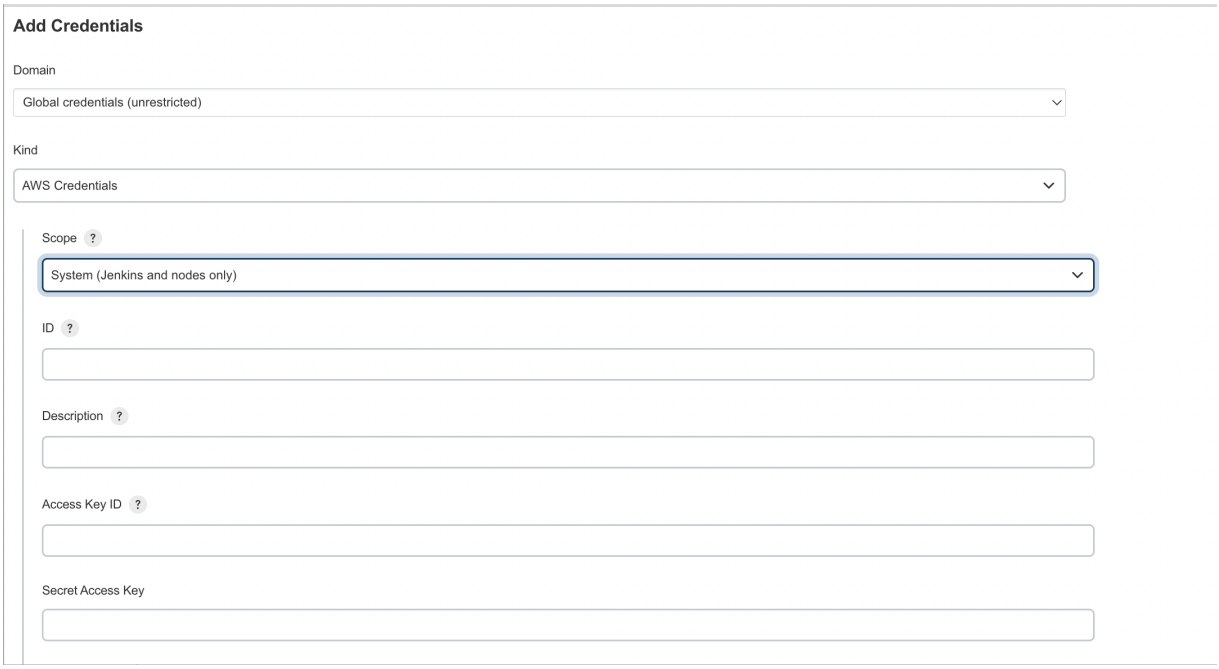
6. Under configure cloud click on Amazon Ec2.



7. Specify a name for the cloud: ex- Ol-jenkins-slave



a. For amazon ec2 credentials click on ADD.

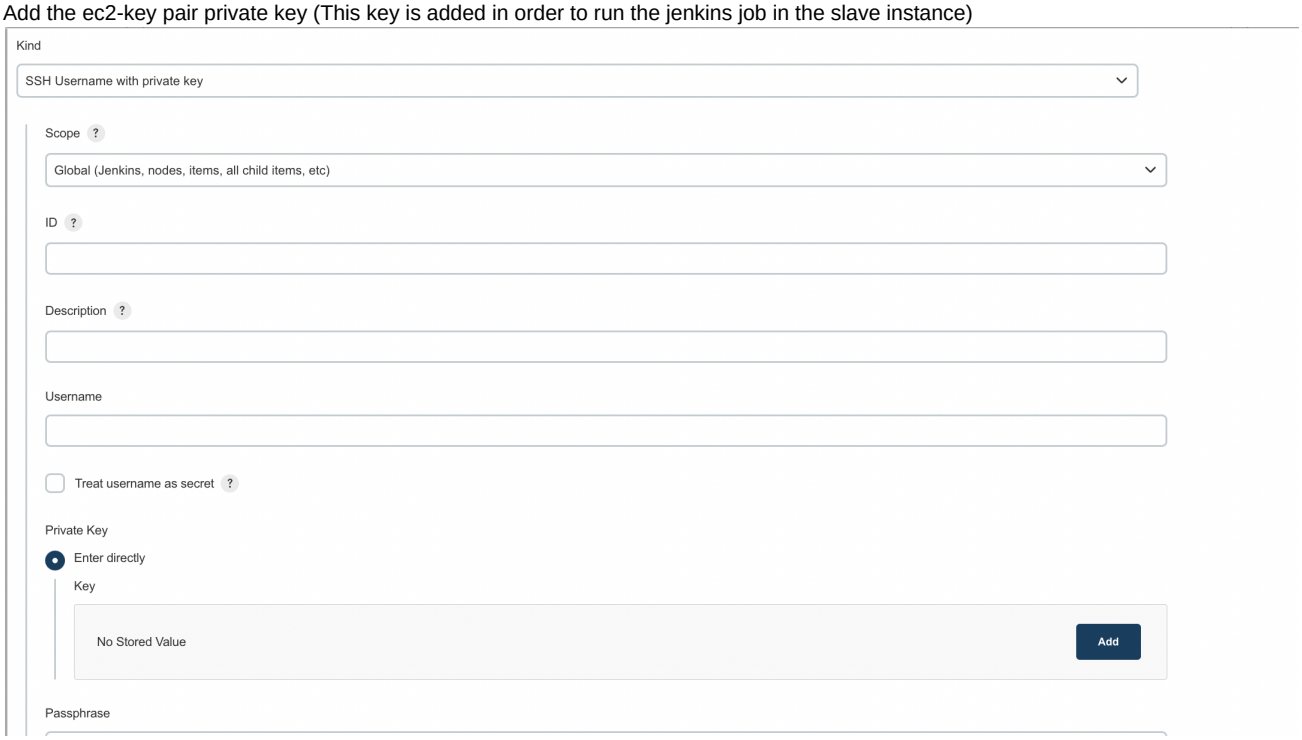


b. domain: global credentials
c. kind: AWS Credentials
d. Scope: system (jenkins and nodes only)
e. Create a Id for it: ol-jenkins-slave
f. Add the Access key Id and the Secret Access key of the user created.
g. Click on Add.

8. select the key added and select the region where we wanna spin up the ec2-instance.

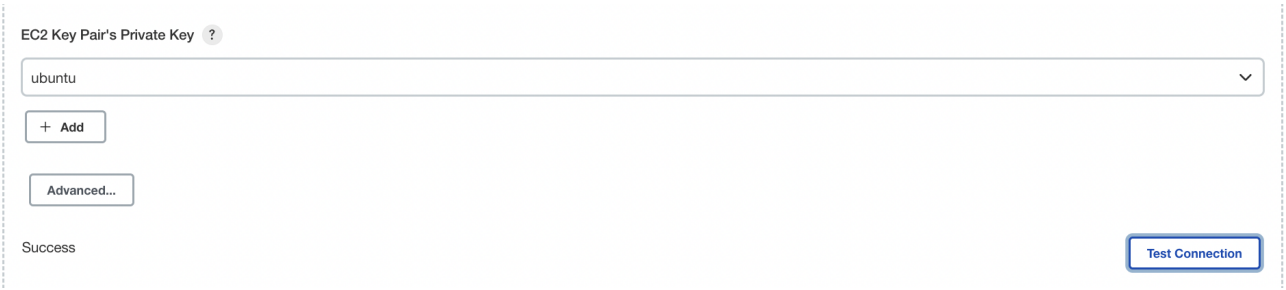
a. region: ap-south-1

9. Add the ec2-key pair private key (This key is added in order to run the jenkins job in the slave instance)



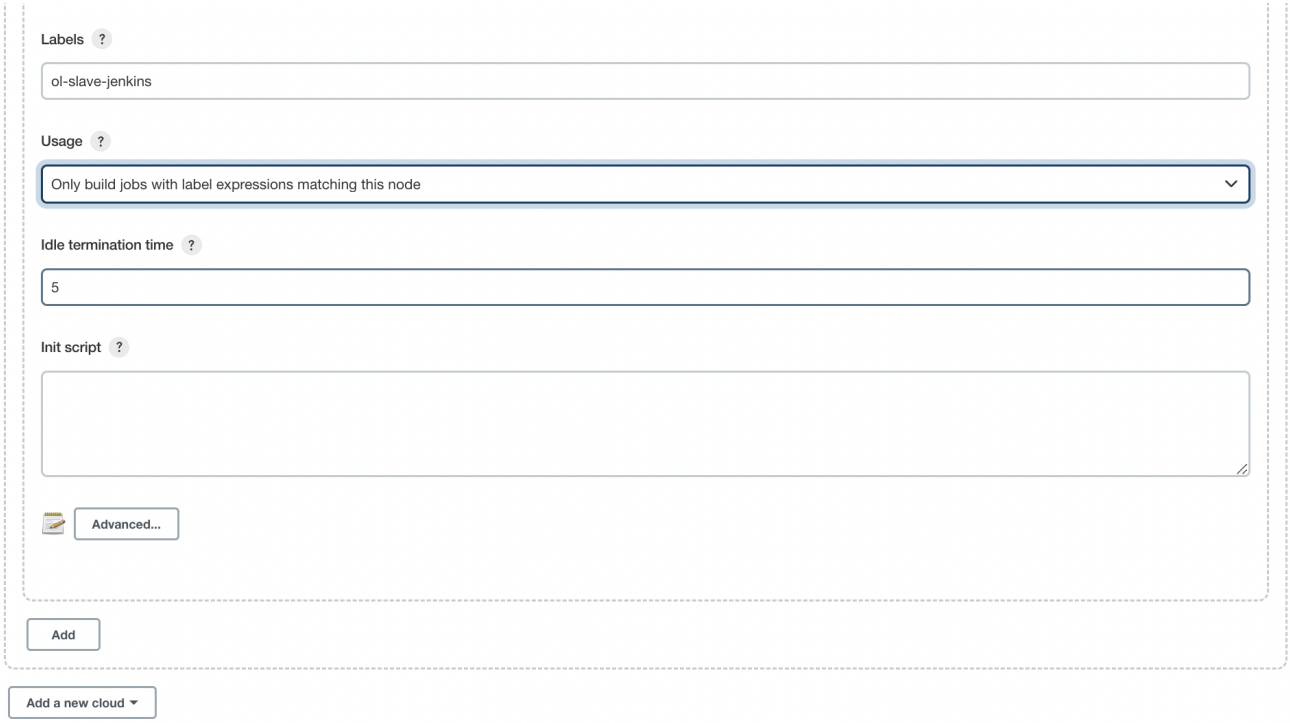
a. domain: global credentials
b. Kind: SSH username with private key
c. username: ubuntu (for ubuntu instance)(ec2-user for linux instance)
d. Private key: enter the private key directly
e. Click on add.

10. Now select the ubuntu user and click on test-connnection.



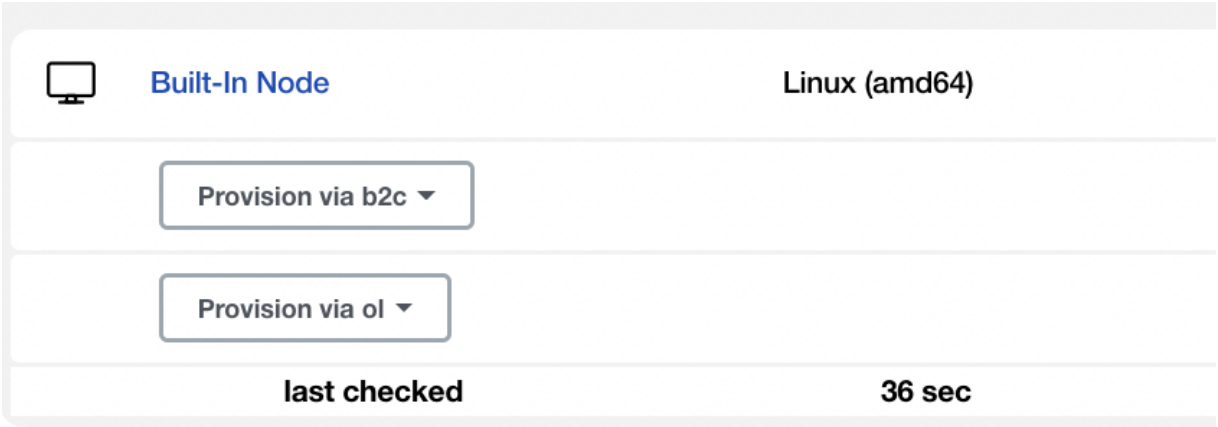
11. Select the AMI name of the jenkins which was created in STEP 2. Then give a description to the AMI and click on check AMI and you'll be able to see the AWS account id and the tag name of the AMI created **(accountid/ami-tag-name).**

12. Select the required instance type based on the number of jenkins jobs running. in our case we're using a t3a.micro instance for the POC instance.
13. Select the security group which was created for which the instance was created and name the security group.
14. Under the **Labels:**



Please mention a label for this particular cloud configuration created, since this label will be used for

a. When a jenkins deployment project is created we attach the label below to use this particular Ec2 instance type configuration.
b. Running jenkins on multiple Aws account can be don with the help of the labels.



c. **idle termination time:** it determine the time (in minutes) for how long the particular slave instance will be available or idle without performing any jobs after which the instance will get terminated automatically.

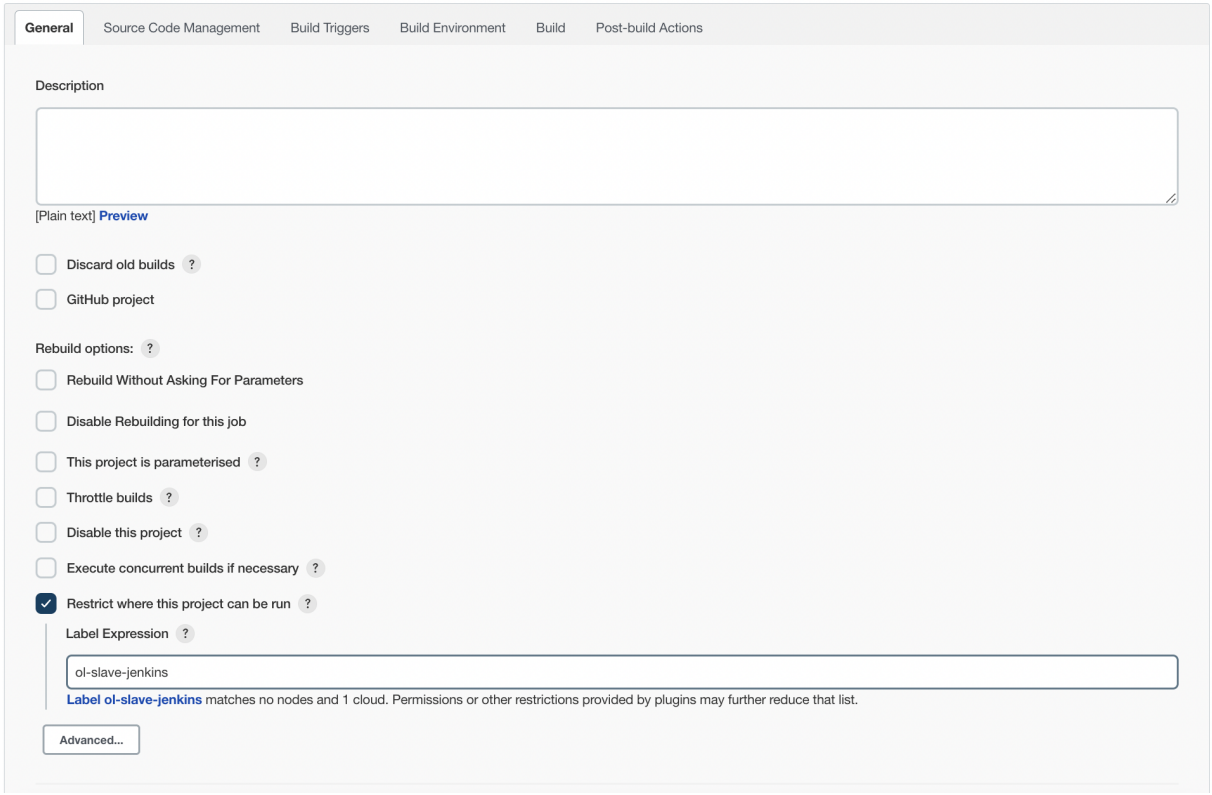
15. Click on save and Apply.

We can created a similar configure cloud for Another AWS account and follow the above steps and give a suitable label to identify both the AWS accounts in which the jenkins master will spin up slave instances to run the jenkins job.

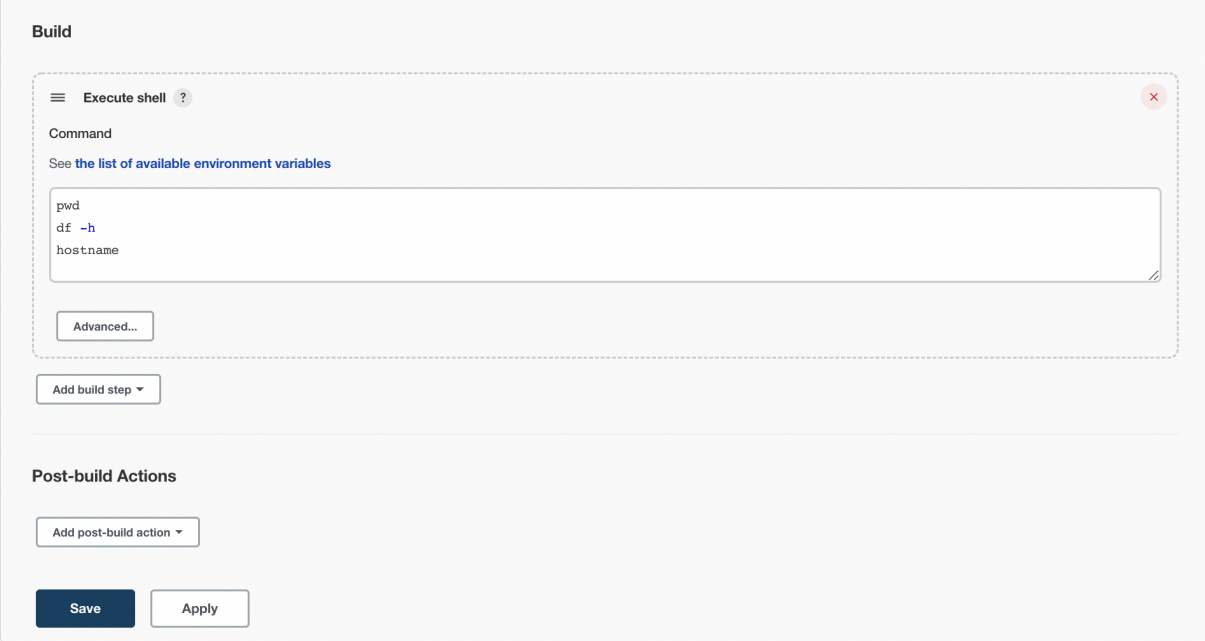
STEP 4: JENKINS JOB TO TEST THE MASTER-SLAVE WORKING:

1. In the Jenkins dashboard, click on a new item select Freestyle project enter a name for the project Click ok.

2. In the configuration page
    a. Add label of the cloud configure which we have configured for the ec2 plugin.



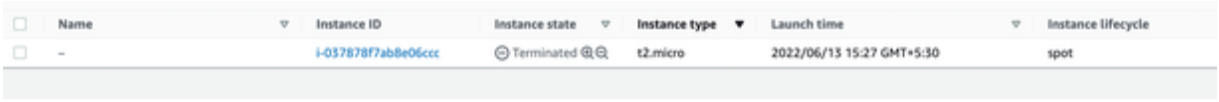
    b. Add a build step execute shell (adding basic commands when the job logs into the system)



c. Entering the basic shell command to test. and Click on Save Apply.
d. Now when we navigate to the console output of the jenkins job you can see that new instannce has been spinned up and the basic execute shell command outputs are visible at the console output.

```
Started by user OL-Jenkins
Running as SYSTEM
Building remotely on EC2 (ol) - master-slave-jenkins (i-037878f7ab8e06ccc) (ol-slave-jenkins) in workspace /home/ec2-user/workspace/Test-Ol-Jenkins
[Test-Ol-Jenkins] $ /bin/sh -xe /tmp/jenkins4618402285426130757.sh
+ pwd
/home/ec2-user/workspace/Test-Ol-Jenkins
+ df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        474M     0  474M   0% /dev
tmpfs           483M     0  483M   0% /dev/shm
tmpfs           483M  460K  483M   1% /run
tmpfs           483M     0  483M   0% /sys/fs/cgroup
/dev/xvda1      8.0G  1.9G  6.2G  23% /
tmpfs            97M     0   97M   0% /run/user/0
tmpfs            97M     0   97M   0% /run/user/1000
+ hostname
ip-172-31-12-73.ap-south-1.compute.internal
```

e. And we can see the instance initiated and terminated as well after the **idle termination time is set to 5 minutes.**



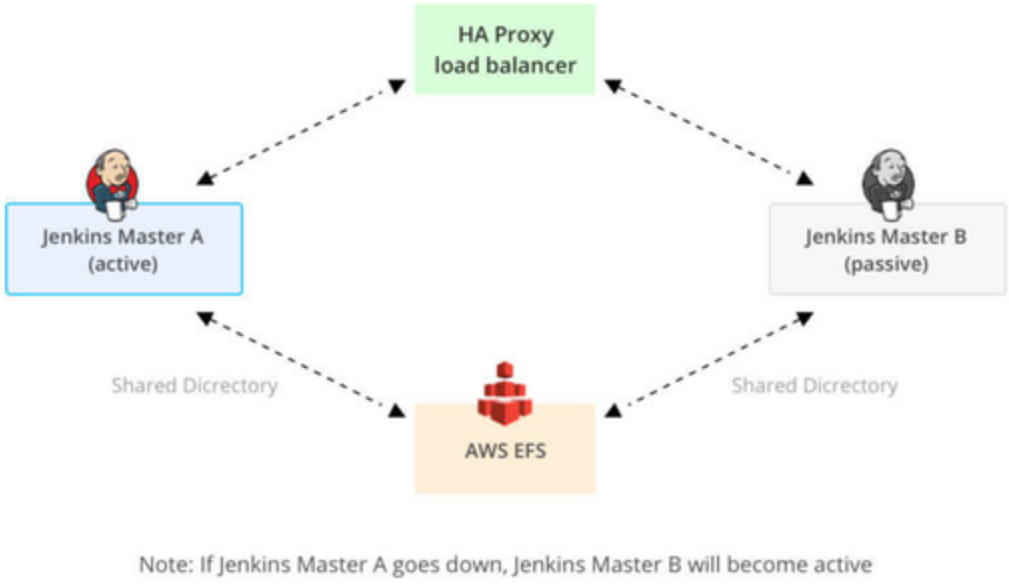
STEP 5: High Availability of the Master Jenkins.

What is High Availability (HA)?

When we say high availability, we are referring to systems that can operate continuously without failure for a long time. The term implies that the system has been tested thoroughly to stand any sort of failure. Jenkins is a crucial component of DevOps and its downtime may have adverse effects on the DevOps environment. To overcome these, we need a high availability setup for Jenkins

What is Elastic File System (EFS)?

The EFS service on AWS is similar to the NFS service on Unix systems wherein the mount point is created on network topology and can be shared with multiple systems at the same time. We are going to configure EFS with Jenkins master machines so that both masters can store and access Jenkins jobs from EFS.

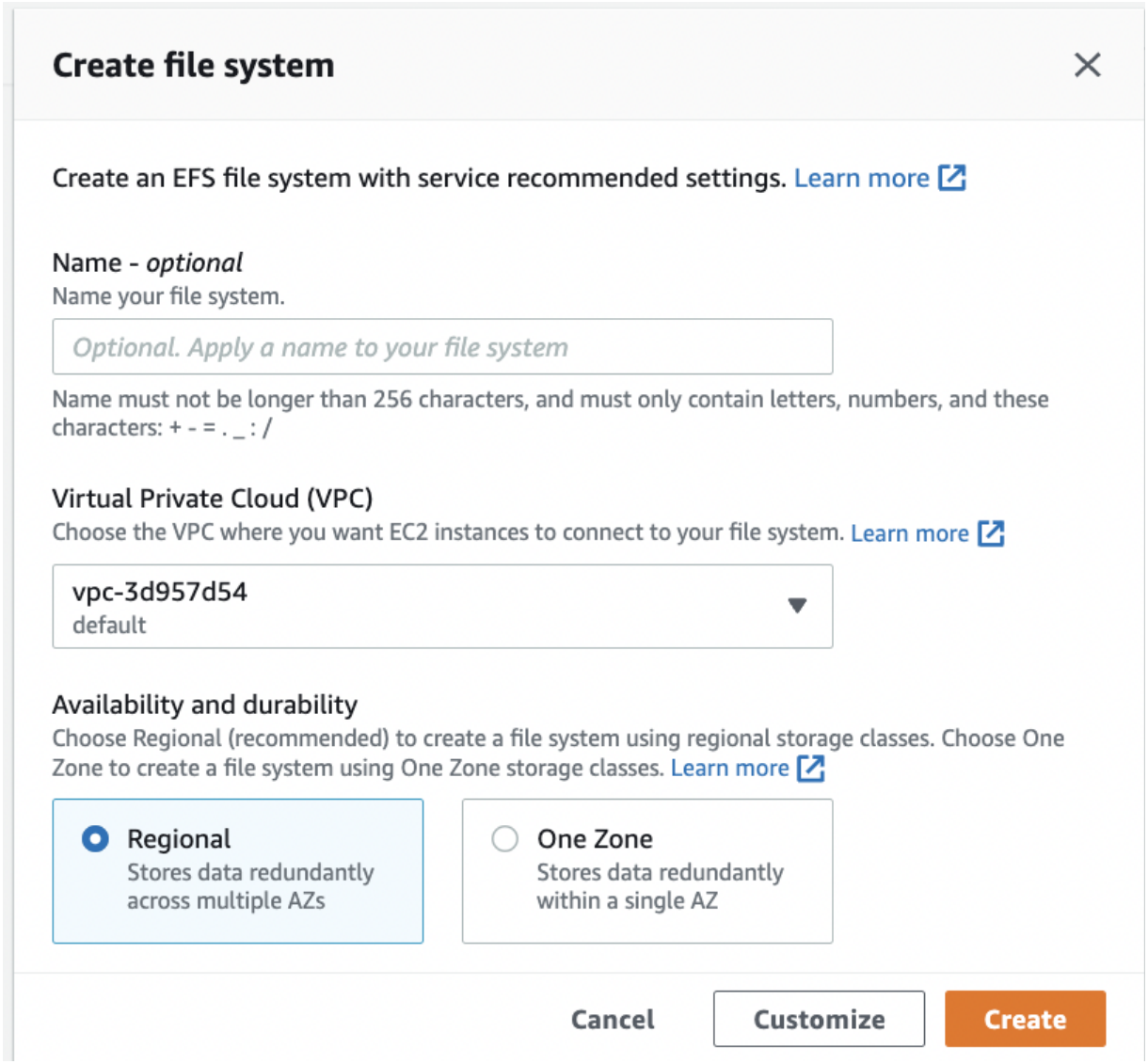


Note: If Jenkins Master A goes down, Jenkins Master B will become active

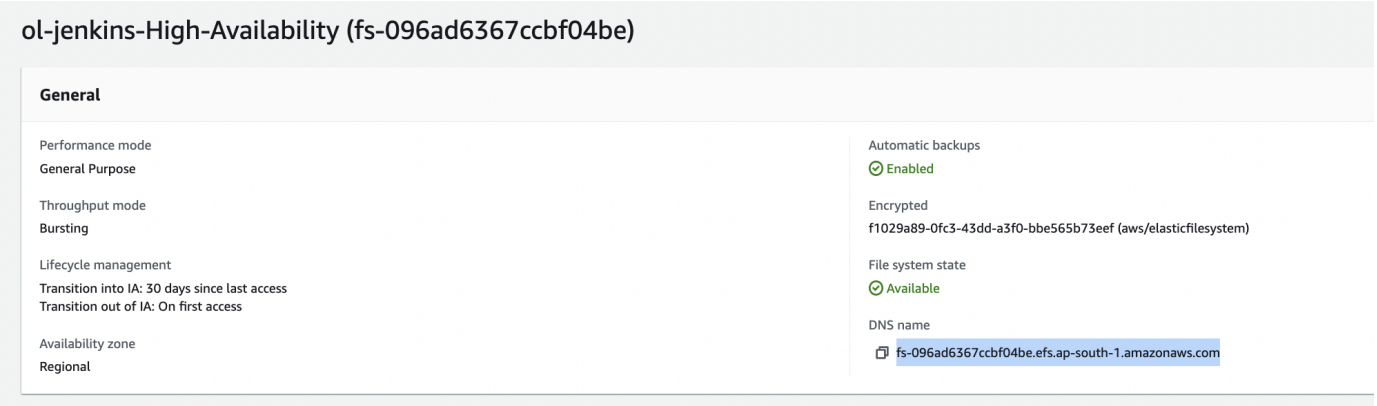
We need to create another Jenkins-Master-B i.e. passive node (follow Step 1:) or you can create a AMI of the Jenkins-master-A and launch a new instance.

We need to set up the EFS provisioning before configuring Jenkins. Here is how you can provision EFS on

1. Navigate to Amazon EFS from the Aws management console.
2. click on create File system and enter the following details
    a. Name of the file system: jenkins-High-Availability.
    b. Select the VPC ID (same VPC in which the master node is present)
    c. Select **Regional** in Availability and durability.
    d. Click on create



3. After the EFS Is created copy paste the DNS name of the EFS file system.



Once EFS is created, the next step will be to configure Jenkins instances to use the provisioned EFS. The configuration steps are pretty simple, all we need to do is ssh both Jenkins instances, i.e Jenkins A (Active) and Jenkins B (Passive) and add the entry given below in the /etc/fstab file.

```
<region>.fs-******.efs.us-east-2.amazonaws.com:/ /var/lib/jenkins/ nfs
defaults   1   1
```

fstab is a system configuration file on Linux and other Unix-like operating systems that contains information about major filesystems. It takes its name from the file systems table and it is located in the /etc directory.

Note :

- The first field is the DNS name with “root (/)” of EFS share.
- The second field is the path to be mounted **/var/lib/jenkins/.**
- The third field is a type of filesystem.

Once you complete the above changes in the fstab file, perform the following steps.

- Stop Jenkins Service using the command,

```
systemctl stop jenkins
```

- Mount EFS volume using the command,

```
mount -a
```

- Validate if the EFS is mounted,

```
mount: shows if /var/lib/jenkins/jobs mounted or not
```

- Once EFS is mounted, change the ownership to Jenkins using the following command,

```
chown -R jenkins:jenkins /var/lib/jenkins/jobs
```

- Start Jenkins Service using the command,

```
 systemctl start jenkins
```

once you finish the configuration on Jenkins A, ssh to Jenkins B instance and perform the same steps. Then, validate mounts using the “mount -a” command.

So to test the HA, create one test job in Jenkins A and now reload the configuration of jenkins B , you will be able to see the test job there as well.

you can write the reload script and run it every minute through cron so that the configuration will always be updated.

STEP 6: Setup of reloading configuration, HA proxy.

write a cron-job to reload the configuration.

The Jenkins Master should keep reloading the configuration being written to the disk (Shared Drive). This may be achieved by configuring a cron job to run a custom reload script on the Jenkins instance.

Install the jenkins cli file.

`wget http://jenkins-master-ip:8080/jnlpJars/jenkins-cli.jar`

Create one file in the root folder and paste this script.

```
#!/bin/bash
java -jar /root/jenkins-cli.jar -s http://jenkins-master-ip:8080 -
webSocket  -auth username:password  reload-configuration
```

Create a cronjob to reload the configuration after every 15 sec.

```
cd /etc/cron.d
sudo vi jenkins_reload
```

Paste the below code in jenkins_reload file

```
* * * * *  root /bin/bash /root/script
* * * * * sleep 15; root /bin/bash /root/script
* * * * * sleep 30; root /bin/bash /root/script
* * * * * sleep 45; root /bin/bash /root/script
```

This will run the script after every 15 sec and reload the jenkins configuration.

once you finish the configuration on Jenkins A, ssh to Jenkins B instance and perform the same steps.

Now, Jenkins A and Jenkins B have redundant data on both instances. But what if one Jenkins instance goes down? How can we achieve active-passive synchronization between Jenkins instances? The answer is a High Availability setup wherein both Instances are configured in the HA configuration file. So let's move to the next part configuring HAProxy setup on a separate instance which will be front-facing for both Jenkins instances. Let's see how we can achieve that -

HAProxy Setup

- Go to https://aws.amazon.com/ec2/
- Launch instance with the basic configuration required and download PEM key. Once the instance is in the ready state, log in to the instance.

```
sudo apt-get update
sudo apt install -y haproxy
```

edit file "**/etc/haproxy/haproxy.cfg**" as per the content below:

```
defaults
        log global
        maxconn 2000
        mode http
        option redispatch
        option forwardfor
        option http-server-close
        retries 3
        timeout http-request 10s
        timeout queue 1m
        timeout connect 10s
        timeout client 1m
        timeout server 1m
        timeout check 10s

    frontend ft_jenkins
        bind *:80
        default_backend bk_jenkins
        reqadd X-Forwarded-Proto:\ http

    backend bk_jenkins
        server jenkins1 <primary_jenkins_ip>:8080 check
        server jenkins2 <secondary_jenkins_ip>:8080 check backup
```

Now start the ha proxy server.

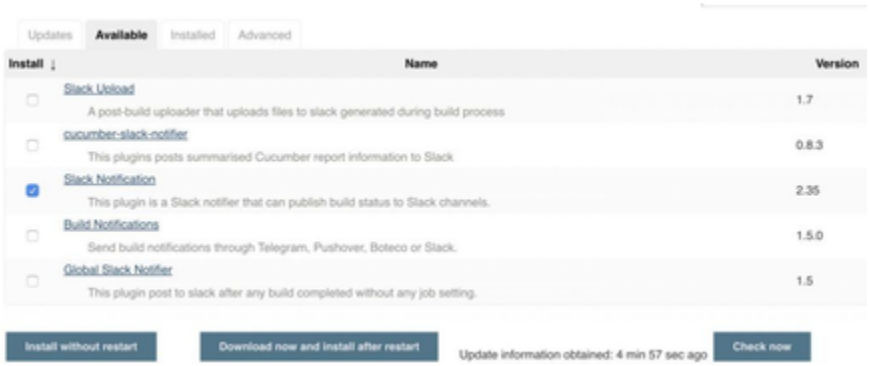
`systemctl start haproxy`

Once the high availability setup is done, we will have to test it. For this, we can stop Jenkins A and try to access it with the help of a domain name or IP address, which is common for both machines. If you are able to access the Jenkins console, all your Jenkins jobs/user and complex CI/CD pipelines created are accessible.

STEP 7: SETUP OF SLACK NOTIFICATION

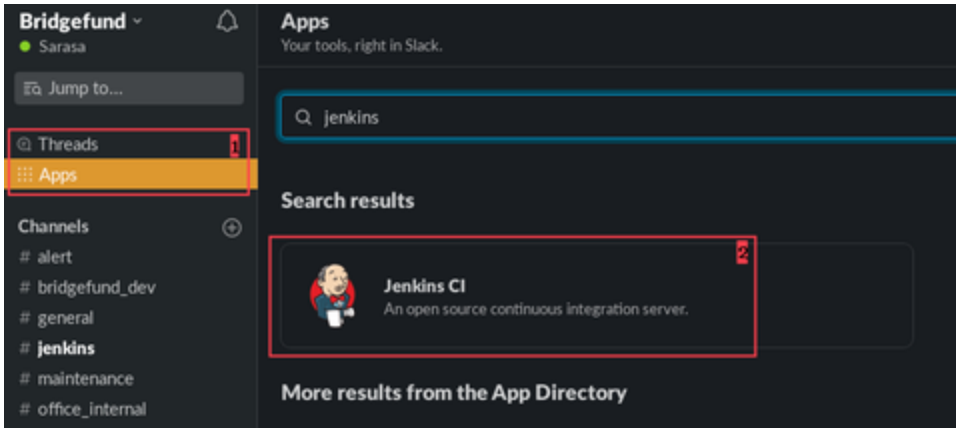
First, let's start by logging into Jenkins and navigating to *Manage Jenkins* > *Plugin Manager*.

Then, on the *Available* tab, we'll search for *Slack*:

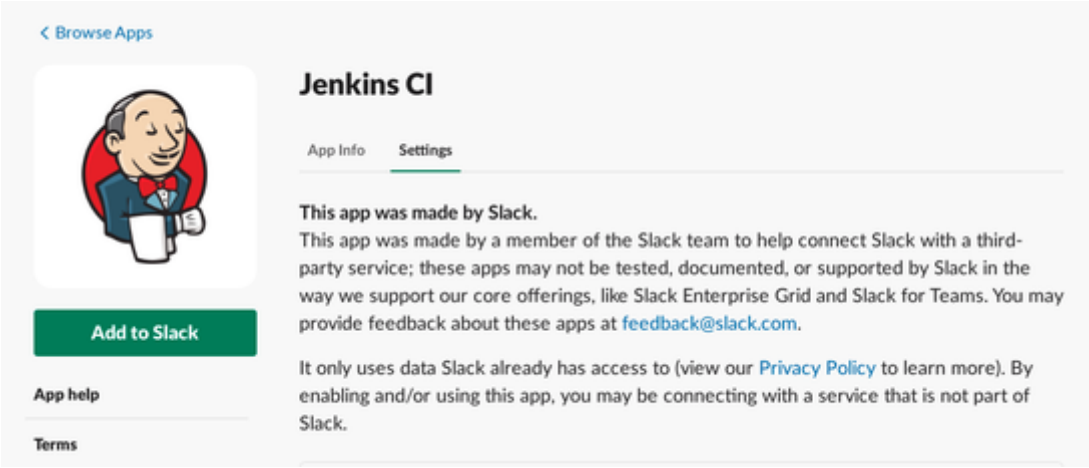


Let's select the checkbox for *Slack Notification* and click *Install without restart*.

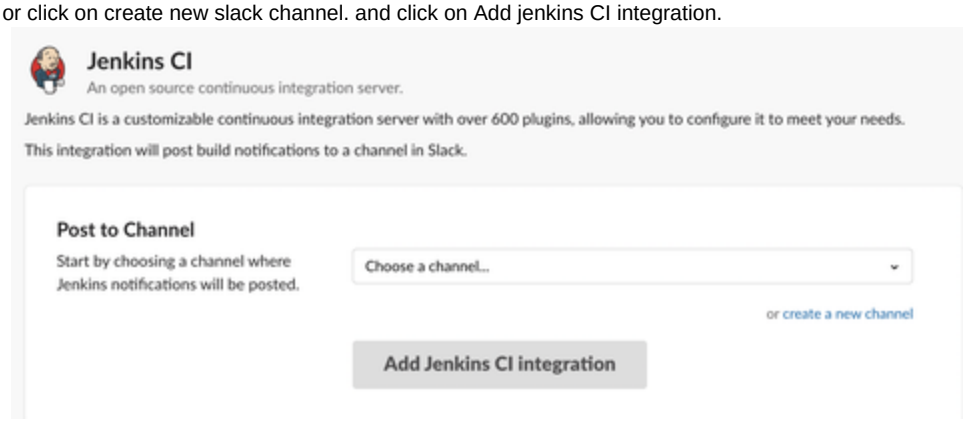
1. Go to the slack desktop and click on Add app.



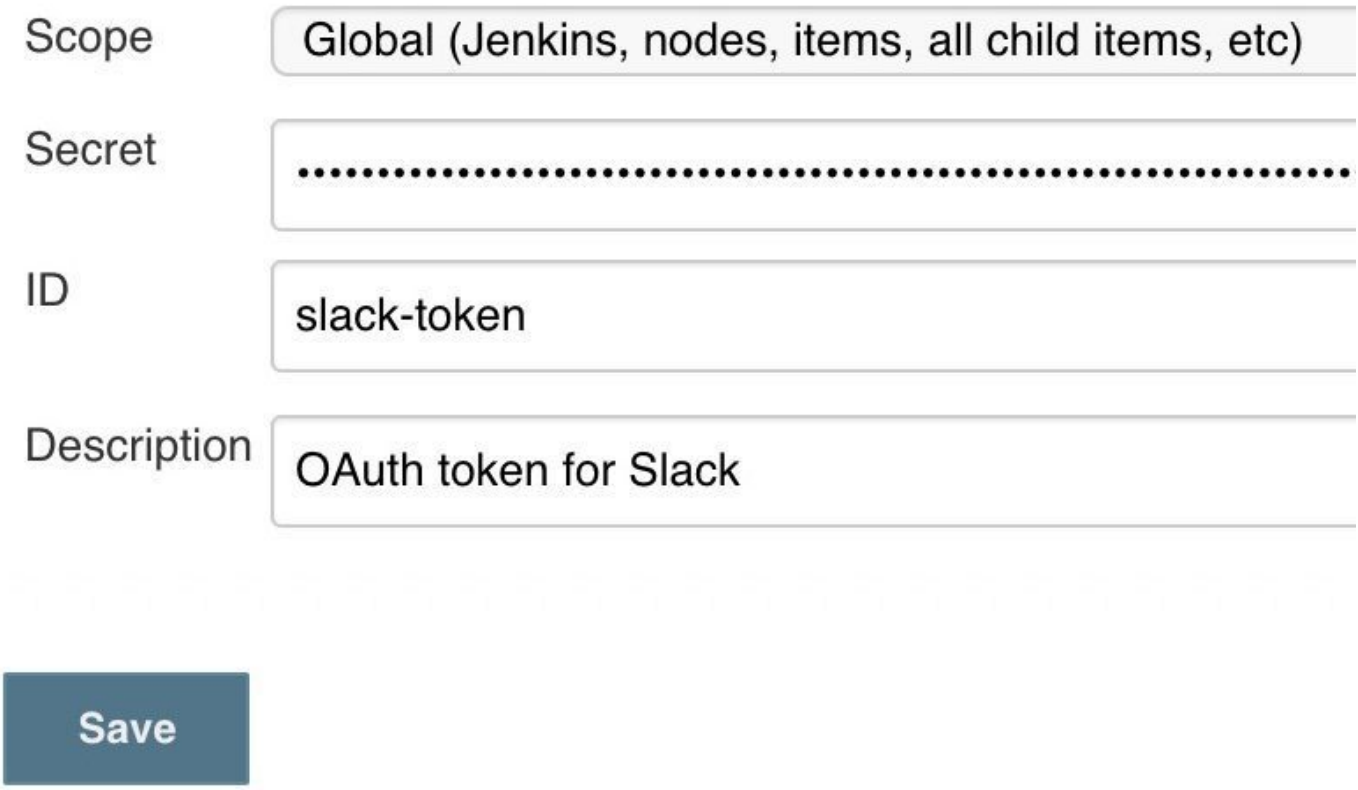
2. Click on Jenkins CI Configuration Add to slack.



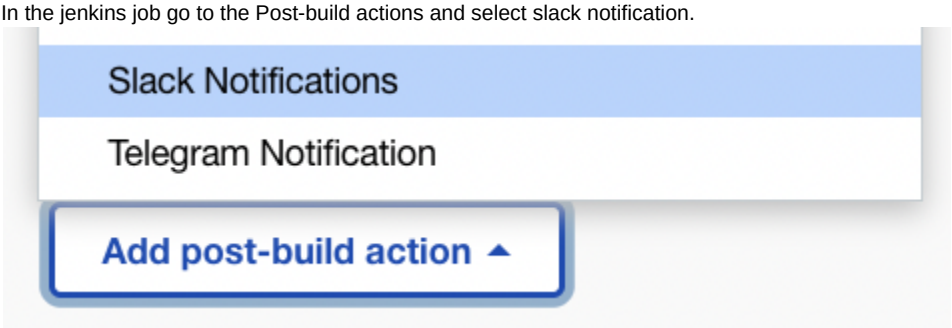
3. Select a channel or click on create new slack channel. and click on Add jenkins CI integration.



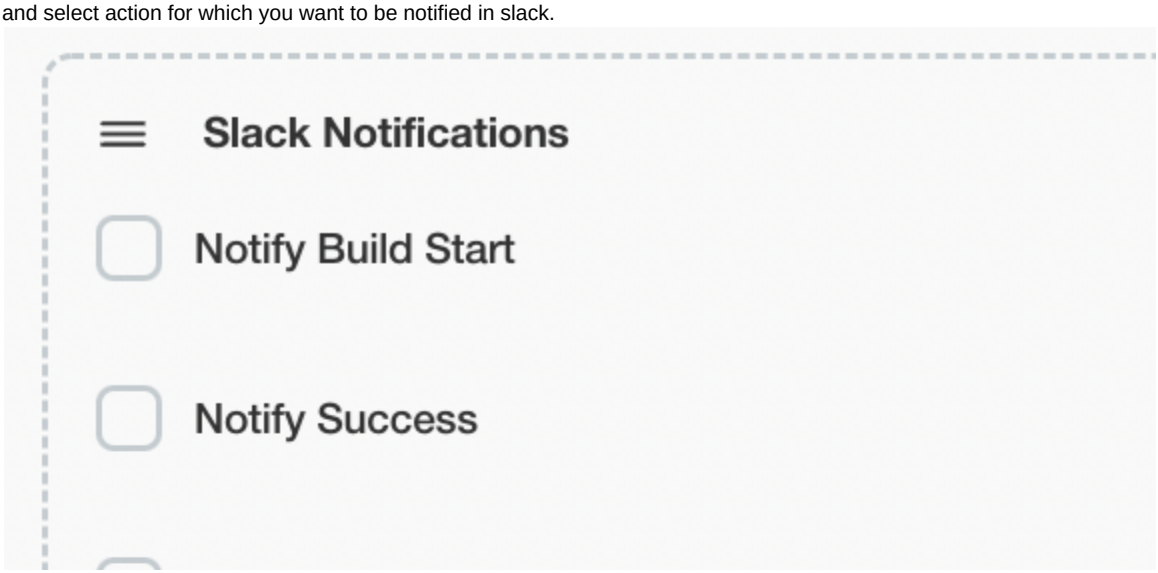
we need to configure new credentials. Let's navigate to *Jenkins* > *Credentials* > *System* > *Global Credentials* and add a new *Secret text* credential:

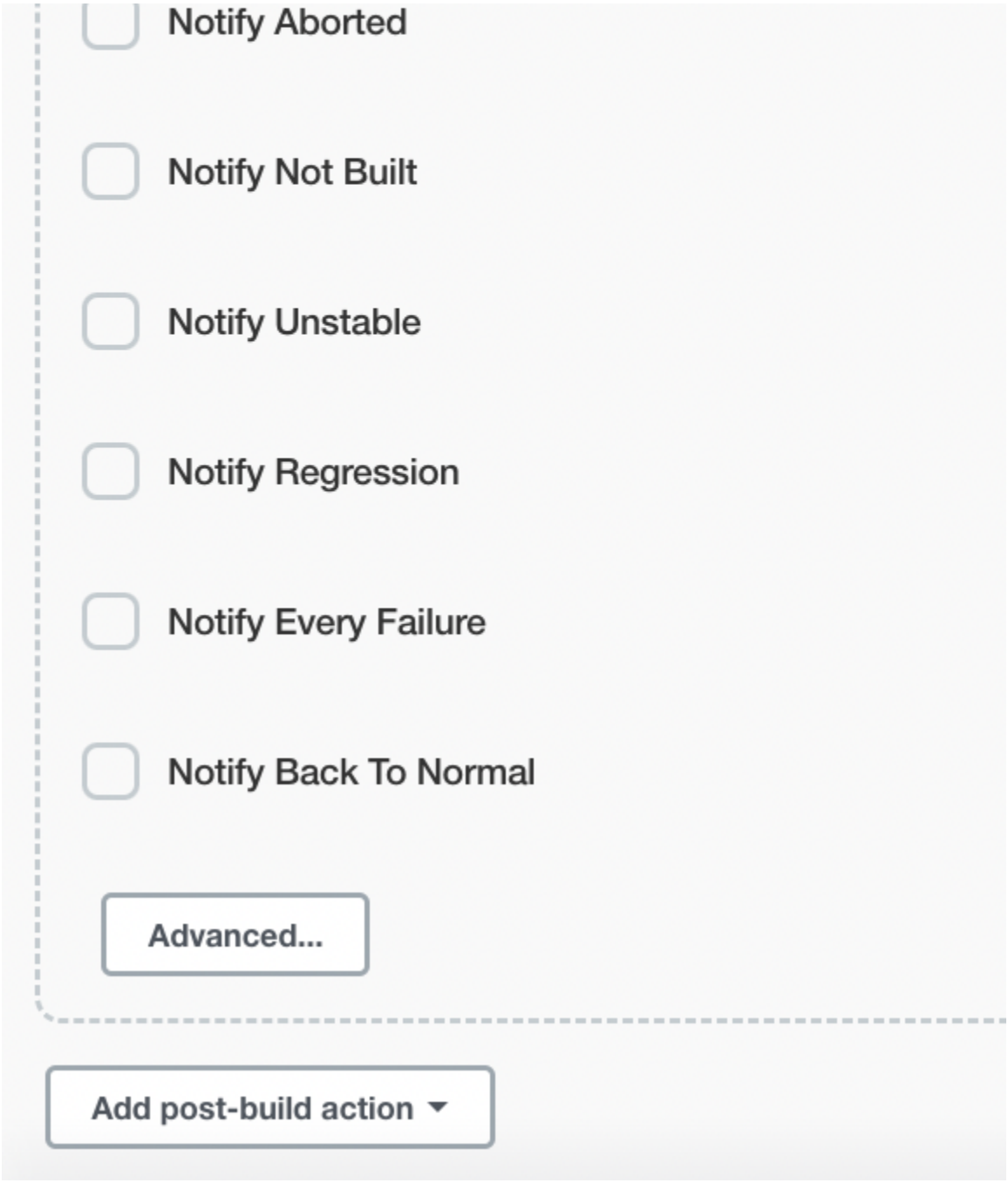


1. Now go to the Jenkins dashbaord Manage jenkins Configure System –> Search for **Global Slack Notifier Settings**
    a. Add Team subdomian: go-betterplace
    b. Integration token id:
    c. Save the credentials
2. In the jenkins job go to the Post-build actions and select slack notification.



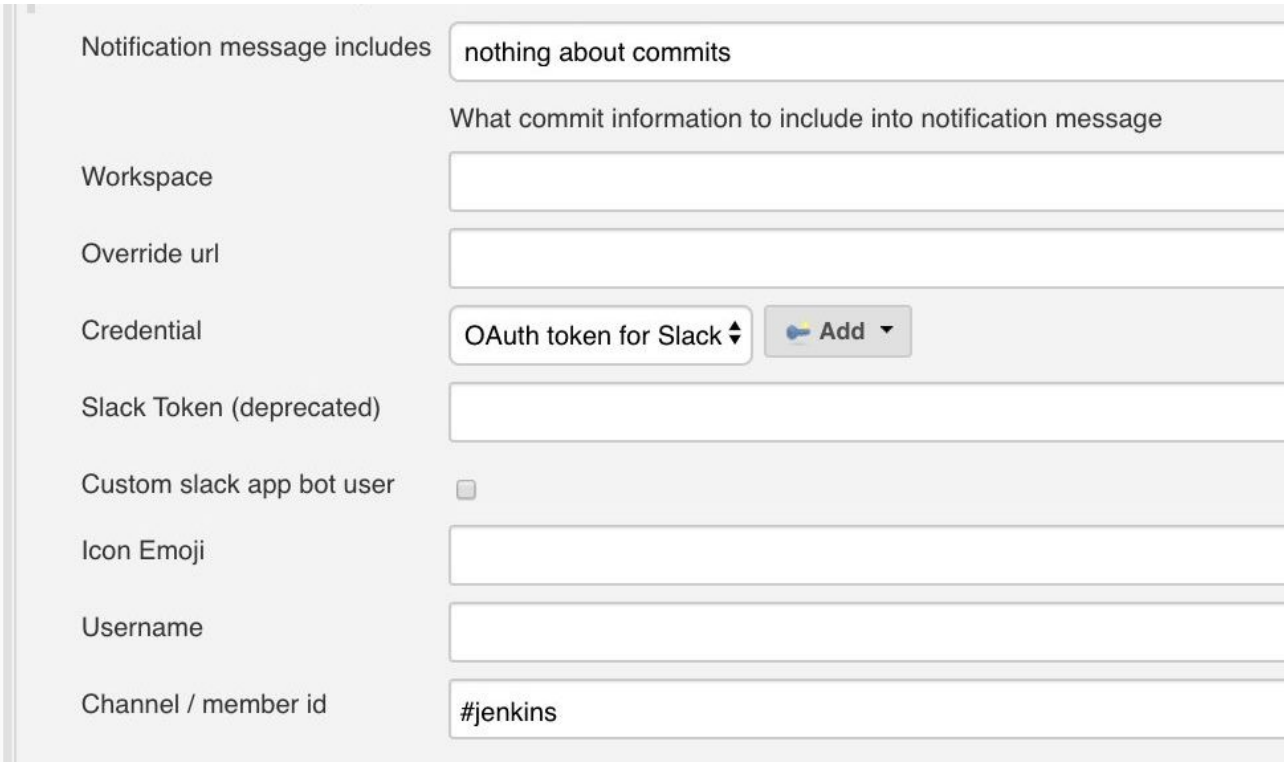
and select action for which you want to be notified in slack.





- Which build phases to send messages for (start, success, failure, etc)
- The name of the credentials to use – the ones we added previously
- The Slack channel name or member ID to send messages to

We can also specify additional fields if desired, such as commit information used for the Jenkins job, custom messages, custom bot icons, and more:



When setting things up via the UI, we can use the *Test Connection* button to ensure that Jenkins can reach Slack.

3. Click on Save and Apply.
4. And once the Build is built we'll get notification on slack for the same.