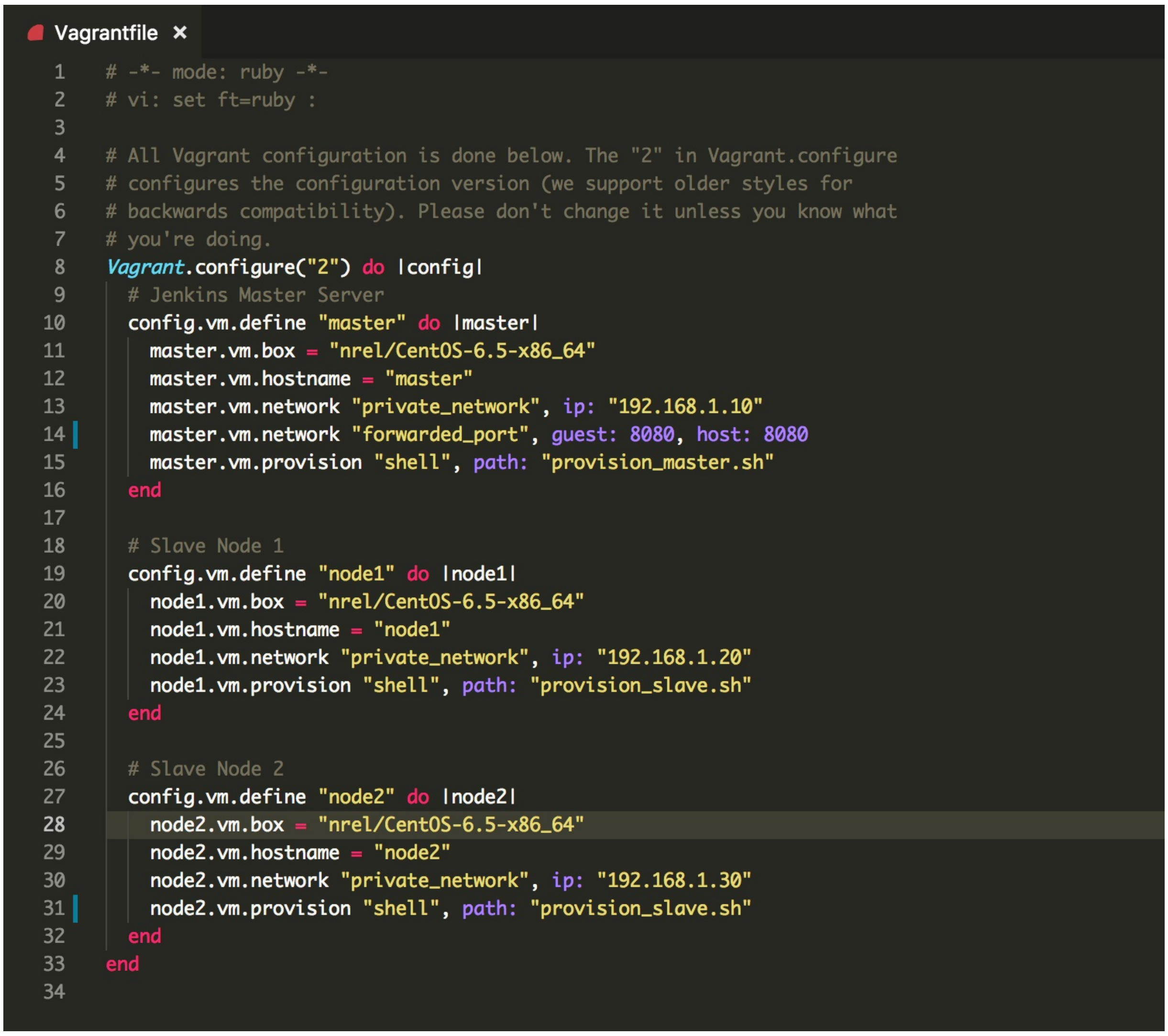


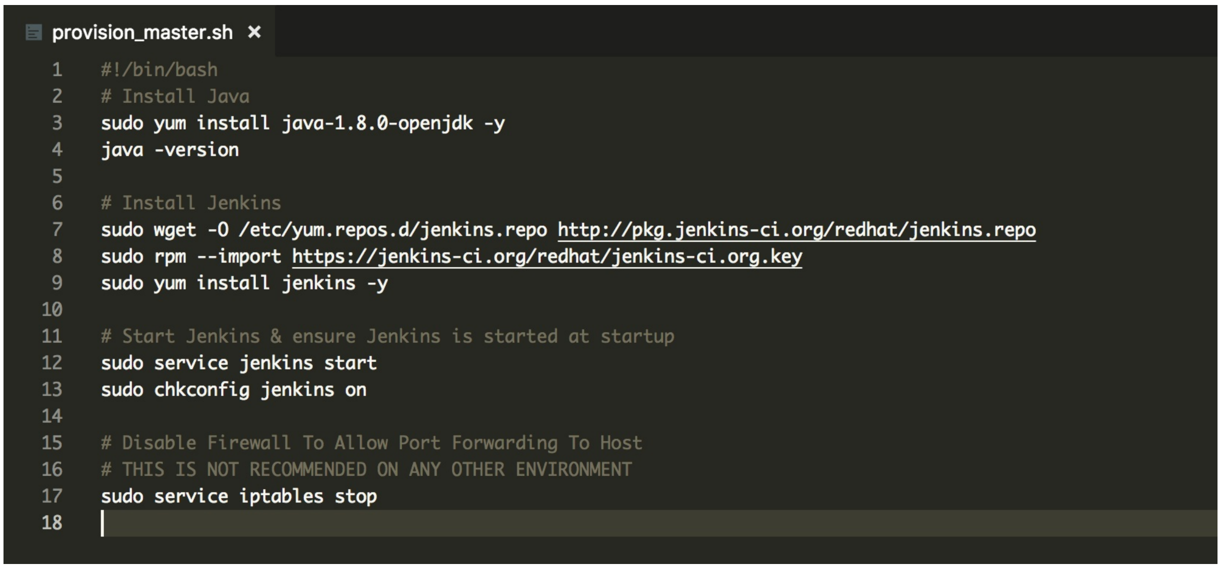
```ruby
# -*- mode: ruby -*-
# vi: set ft=ruby :

# All Vagrant configuration is done below. The "2" in Vagrant.configure
# configures the configuration version (we support older styles for
# backwards compatibility). Please don't change it unless you know what
# you're doing.
Vagrant.configure("2") do |config|
  # Jenkins Master Server
  config.vm.define "master" do |master|
    master.vm.box = "nrel/CentOS-6.5-x86_64"
    master.vm.hostname = "master"
    master.vm.network "private_network", ip: "192.168.1.10"
    master.vm.network "forwarded_port", guest: 8080, host: 8080
    master.vm.provision "shell", path: "provision_master.sh"
  end

  # Slave Node 1
  config.vm.define "node1" do |node1|
    node1.vm.box = "nrel/CentOS-6.5-x86_64"
    node1.vm.hostname = "node1"
    node1.vm.network "private_network", ip: "192.168.1.20"
    node1.vm.provision "shell", path: "provision_slave.sh"
  end

  # Slave Node 2
  config.vm.define "node2" do |node2|
    node2.vm.box = "nrel/CentOS-6.5-x86_64"
    node2.vm.hostname = "node2"
    node2.vm.network "private_network", ip: "192.168.1.30"
    node2.vm.provision "shell", path: "provision_slave.sh"
  end
end

```

The Vagrantfile is defined in Ruby. Without diving too deep into the syntax, we will break down the significance of the lines in the file. You will notice a lot of repetition for the three nodes in this file, which are `master`, `node1`, and `node2`. Notice in the file that we are defining the machine's context using `config.vm.define` and then using the defined name for the first instance. The name we are going to use is `master`. The three machines have four similar directives: `box`, `hostname`, `network`, and `provision`. Let's see what each of these do.

`box`	Configures what box the machine will be brought up against. The value here should be the name of an installed box or a shorthand name of a box on HashiCorp's Vagrant Cloud. This is quite similar to Docker Hub for Docker images.

`hostname`	Configures the hostname that the machine should be assigned.
`network`	Configures networks on the machine. The `private_ network` value allows you to access the machine using a private IP address (not accessible from the internet). On the master, we also used the `forwarded_port` setting to forward the Jenkins port inside the guest to our host machine. This will allow us to access Jenkins on our browser via `http://localhost 8080`.
`provision`	Configures provisions on the machine so that software can be automatically installed and configured when the machine is created. We are using a bash script to provision our nodes.

The master provisioning script is as follows:

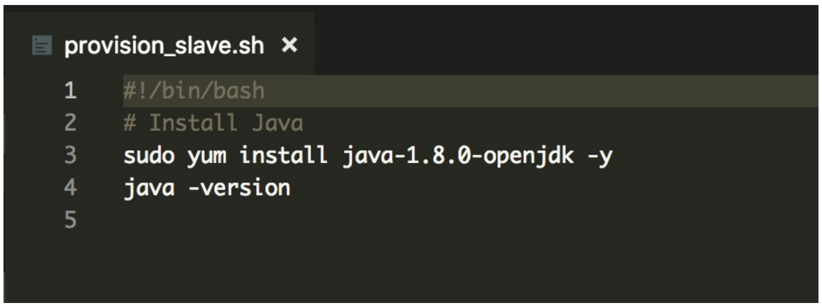


provision_master.sh

```
#!/bin/bash
# Install Java
sudo yum install java-1.8.0-openjdk -y
java -version

# Install Jenkins
sudo wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat/jenkins.repo
sudo rpm --import https://jenkins-ci.org/redhat/jenkins-ci.org.key
sudo yum install jenkins -y

# Start Jenkins & ensure Jenkins is started at startup
sudo service jenkins start
sudo chkconfig jenkins on

# Disable Firewall To Allow Port Forwarding To Host
# THIS IS NOT RECOMMENDED ON ANY OTHER ENVIRONMENT
sudo service iptables stop

```

The script installs Java, which Jenkins depends on, and then installs Jenkins and starts up the Jenkins service. Finally, it disables the firewall on the guest to allow for port forwarding. This should not be done in any production environment. Ideally, you should just allow traffic through port 8080 on your firewall to allow access to Jenkins.

The slave provisioning script is as follows:


provision_slave.sh

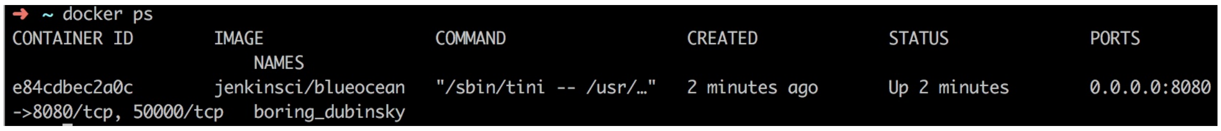
```
#!/bin/bash
# Install Java
sudo yum install java-1.8.0-openjdk -y
java -version

```

On the slaves, we don't need to do much other than ensure that Java is installed. The same version that's installed on the master should be installed on the slaves.

Stopping the Docker Container

If you have been working with Docker up till this point, you need to make sure that you stop all containers to free up any ports that might be in use, especially port 8080, to allow Vagrant to work. To stop Docker containers from working, follow these steps:

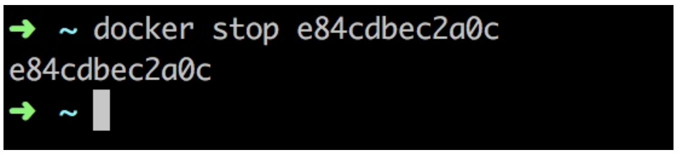
1. First, view the running containers by running `docker ps`.



```
➜ ~ docker ps
CONTAINER ID        IMAGE                   COMMAND                  CREATED             STATUS              PORTS
        NAMES
e84cdbec2a0c        jenkinsci/blueocean     "/sbin/tini -- /usr/…"   2 minutes ago       Up 2 minutes        0.0.0.0:8080
->8080/tcp, 50000/tcp    boring_dubinsky
```

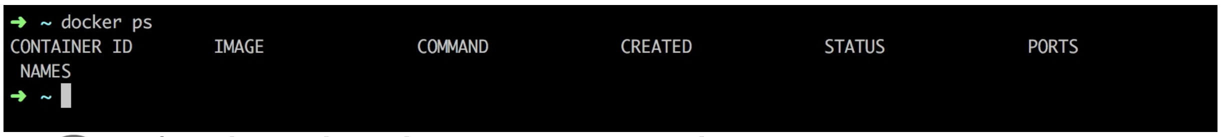
   From the output, we can see that the `jenkinsci/blueocean image` is running with a container ID of `e84cdbec2a0c`.

2. Copy the container ID and pass it to the docker `stop` command as shown:



```
➜ ~ docker stop e84cdbec2a0c
e84cdbec2a0c
➜ ~ 
```

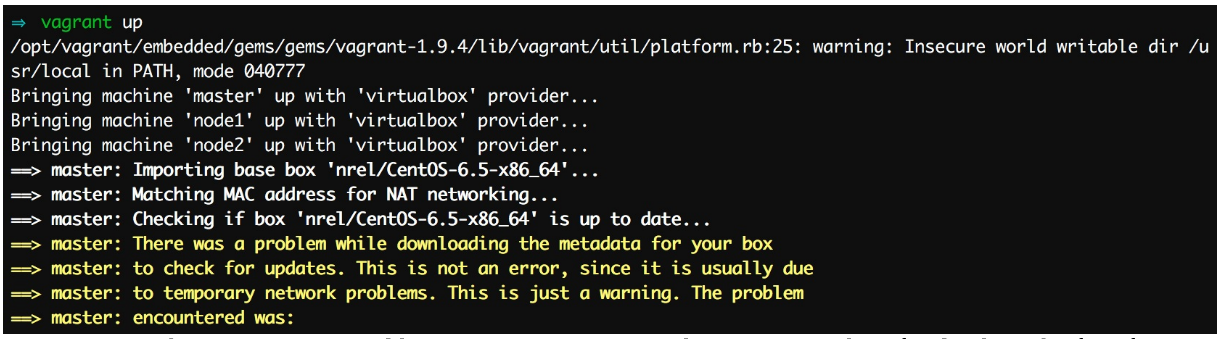
   If we run `docker ps` again, we can see that there is no container running and the ports that were mapped are now freed:



```
➜ ~ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
 NAMES
➜ ~ 
```

Refer to the complete code at `https://bit.ly/2utxZa4` *and* `https://bit.ly/2Jre4xh`.

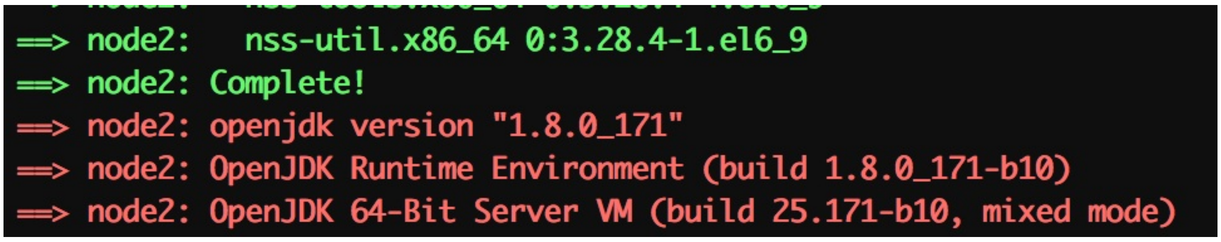
3. To spin up your environment, navigate to the `Lesson6` folder in the repository on your terminal after cloning it to your PC. This is where files such as `Vagrantfile`, `provision_master.sh`, and `provision_slave.sh` are located. Inside this directory, run `vagrant up`:



```
⇒ vagrant up
/opt/vagrant/embedded/gems/gems/vagrant-1.9.4/lib/vagrant/util/platform.rb:25: warning: Insecure world writable dir /u
sr/local in PATH, mode 040777
Bringing machine 'master' up with 'virtualbox' provider...
Bringing machine 'node1' up with 'virtualbox' provider...
Bringing machine 'node2' up with 'virtualbox' provider...
==> master: Importing base box 'nrel/CentOS-6.5-x86_64'...
==> master: Matching MAC address for NAT networking...
==> master: Checking if box 'nrel/CentOS-6.5-x86_64' is up to date...
==> master: There was a problem while downloading the metadata for your box
==> master: to check for updates. This is not an error, since it is usually due
==> master: to temporary network problems. This is just a warning. The problem
==> master: encountered was:
```

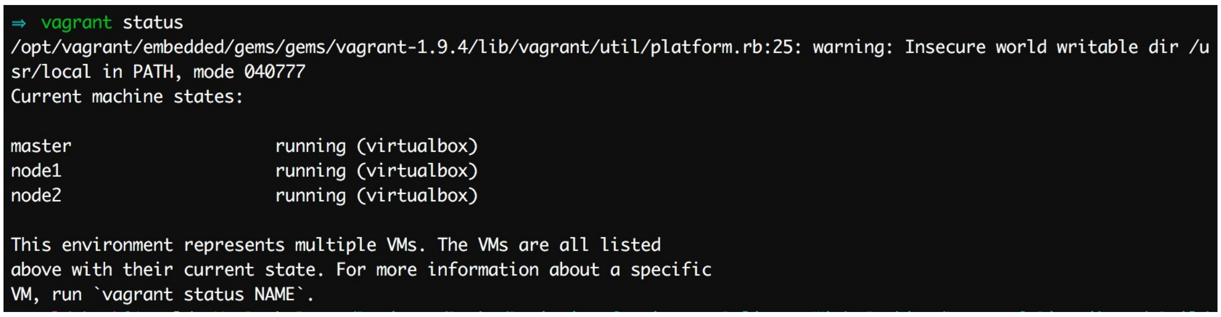
The output generated by running `vagrant up` is a lot to capture, but if it displays the first few lines similar to the ones depicted in the preceding screenshot, you are off to a great start. Vagrant will basically pull the defined box image and run the steps outlined in the Vagrantfile that we discussed earlier. This includes assigning hostnames and IP addresses, port forwarding, and finally, running the provisioning scripts. You will also notice that the output is helpful enough.

Let this process run to completion until you get control of your terminal back. The final line of output before you get control of your terminal should look something similar to the following:



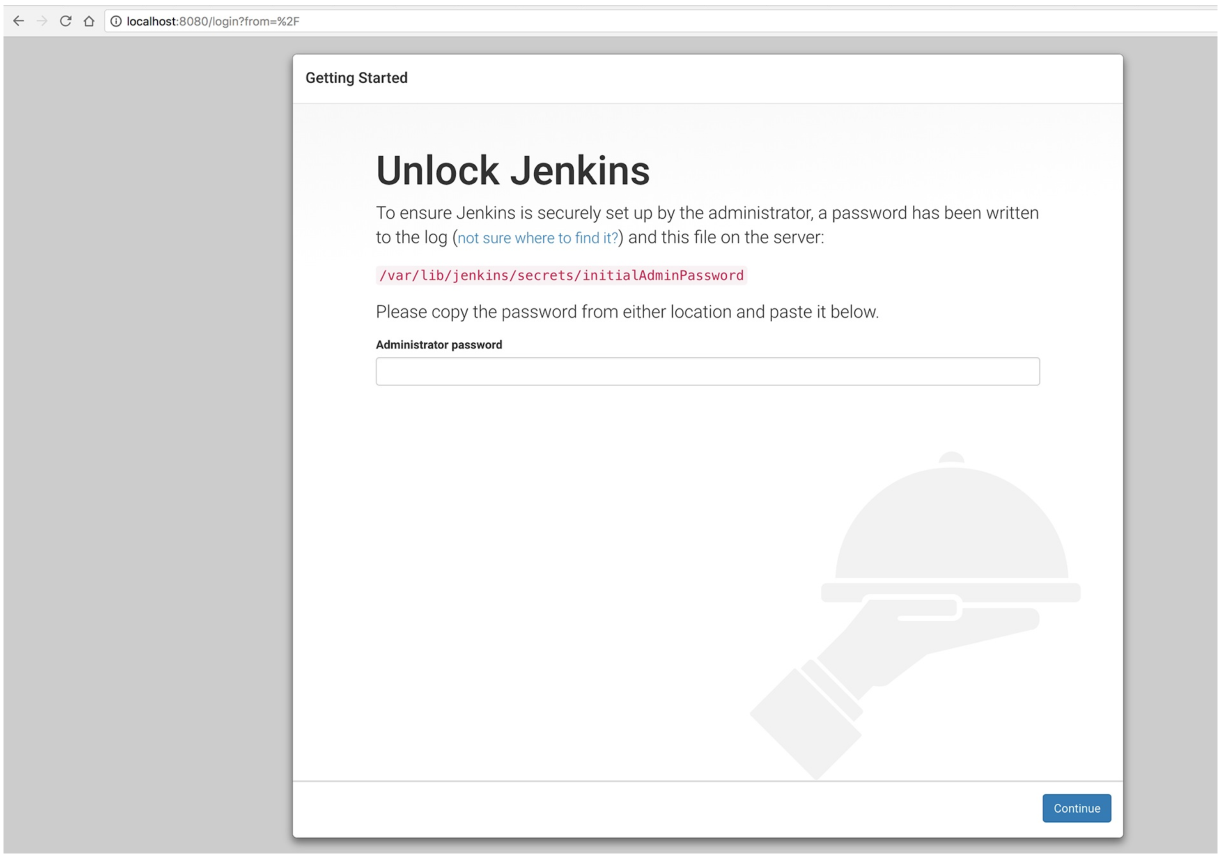
```
==> node2:   nss-util.x86_64 0:3.28.4-1.el6_9
==> node2: Complete!
==> node2: openjdk version "1.8.0_171"
==> node2: OpenJDK Runtime Environment (build 1.8.0_171-b10)
==> node2: OpenJDK 64-Bit Server VM (build 25.171-b10, mixed mode)
```

4. To ensure that everything works as expected, run the `vagrant status` command on the same directory:



```
⇒ vagrant status
/opt/vagrant/embedded/gems/gems/vagrant-1.9.4/lib/vagrant/util/platform.rb:25: warning: Insecure world writable dir /u
sr/local in PATH, mode 040777
Current machine states:

master                    running (virtualbox)
node1                     running (virtualbox)
node2                     running (virtualbox)

This environment represents multiple VMs. The VMs are all listed
above with their current state. For more information about a specific
VM, run `vagrant status NAME`.
```

From the output, we can see that our three nodes are running, and these are `master`, `node1`, and `node2`. After verifying that our nodes are running, go to your browser and open the URL `http://localhost:8080`. You should be presented with the Jenkins Getting Started page as follows:


localhost:8080/login?from=%2F

Getting Started

Unlock Jenkins

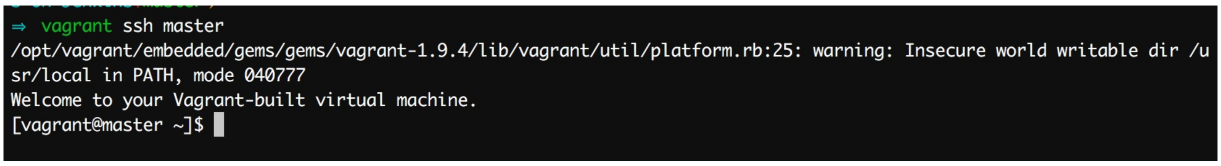
To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

`/var/lib/jenkins/secrets/initialAdminPassword`

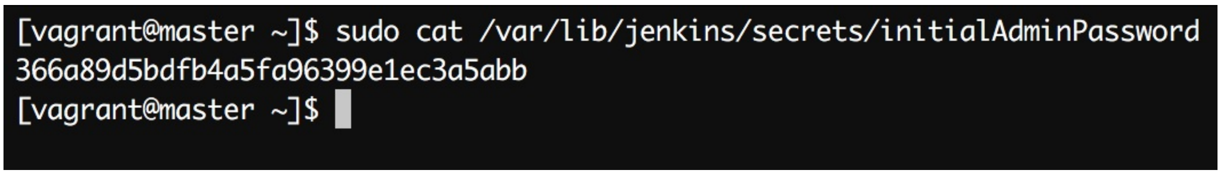
Please copy the password from either location and paste it below.

Administrator password

The page requires us to provide the default administrator password. To access this password, we will connect to the master node. On the same directory that you ran the previous vagrant commands, run `vagrant ssh master`. This, as you might have guessed, will connect to the master node:

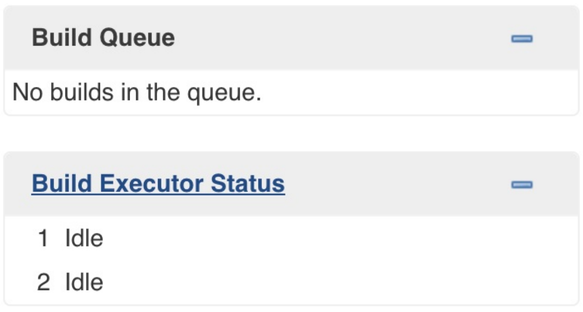

```
⇒ vagrant ssh master
/opt/vagrant/embedded/gems/gems/vagrant-1.9.4/lib/vagrant/util/platform.rb:25: warning: Insecure world writable dir /u
sr/local in PATH, mode 040777
Welcome to your Vagrant-built virtual machine.
[vagrant@master ~]$ 
```

This lands us in the terminal of the master node as the default Vagrant user. We want to display the default password in the `/var/lib/jenkins/secrets/initialAdminPassword` file, so run `cat` on this file as `sudo` to display the password:


```
[vagrant@master ~]$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
366a89d5bdfb4a5fa96399e1ec3a5abb
[vagrant@master ~]$ 
```

Copy this over to the text field on your browser and continue with the setup. The setup process is similar to what we covered in the first chapter; thus it should be familiar to you. Complete the Jenkins setup, create an admin user, and log in to your installation.

While on the dashboard, you will notice something in the lower-left part of the screen:

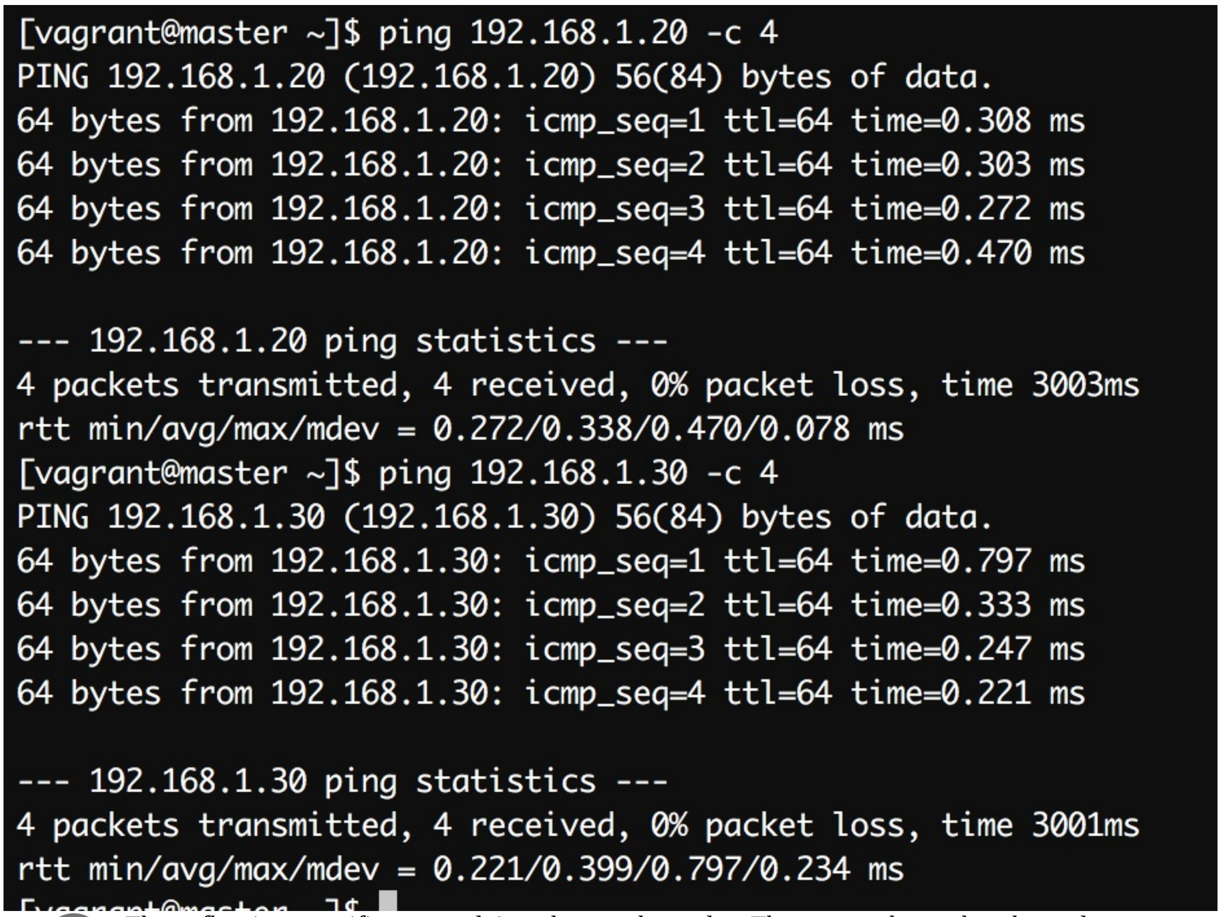


We talked about the build queue and build executors earlier in this section. This is where the status of both of these is displayed. When there is a build running, Jenkins displays on which executor it's running and if there are any builds waiting for the availability of an executor, they are displayed on Build Queue.

Verifying Node Connectivity

We are going to verify that our nodes can communicate with each other, first via the ping command and then via SSH. Follow the steps given below to verify node connectivity in Jenkins:

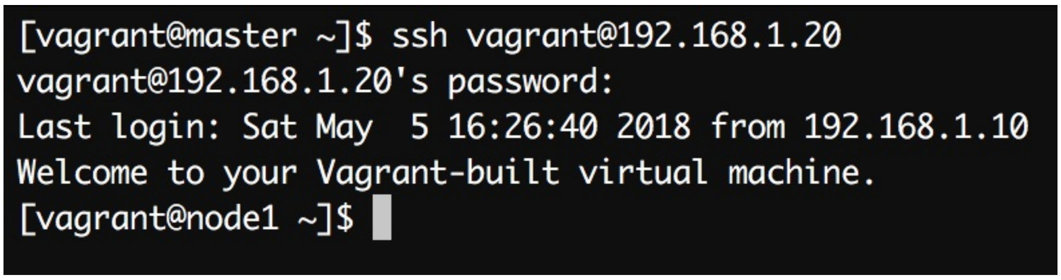
1. Connect to the master node using `vagrant SSH master` and ping both the IP addresses of `node1` and `node2` as follows:



```
[vagrant@master ~]$ ping 192.168.1.20 -c 4
PING 192.168.1.20 (192.168.1.20) 56(84) bytes of data.
64 bytes from 192.168.1.20: icmp_seq=1 ttl=64 time=0.308 ms
64 bytes from 192.168.1.20: icmp_seq=2 ttl=64 time=0.303 ms
64 bytes from 192.168.1.20: icmp_seq=3 ttl=64 time=0.272 ms
64 bytes from 192.168.1.20: icmp_seq=4 ttl=64 time=0.470 ms

--- 192.168.1.20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 0.272/0.338/0.470/0.078 ms
[vagrant@master ~]$ ping 192.168.1.30 -c 4
PING 192.168.1.30 (192.168.1.30) 56(84) bytes of data.
64 bytes from 192.168.1.30: icmp_seq=1 ttl=64 time=0.797 ms
64 bytes from 192.168.1.30: icmp_seq=2 ttl=64 time=0.333 ms
64 bytes from 192.168.1.30: icmp_seq=3 ttl=64 time=0.247 ms
64 bytes from 192.168.1.30: icmp_seq=4 ttl=64 time=0.221 ms

--- 192.168.1.30 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3001ms
rtt min/avg/max/mdev = 0.221/0.399/0.797/0.234 ms
[vagrant@master ~]$
```

The `-c` flag just specifies to send 4 packets to the nodes. The output shows that the nodes can communicate as there is 0% packet loss.

2. Connect to the two nodes from our master using SSH. Run `ssh vagrant@192.168.1.20` to connect to `node1` as follows:



```
[vagrant@master ~]$ ssh vagrant@192.168.1.20
vagrant@192.168.1.20's password:
Last login: Sat May  5 16:26:40 2018 from 192.168.1.10
Welcome to your Vagrant-built virtual machine.
[vagrant@node1 ~]$ 
```

3. Enter `yes` if a prompt asking you about the authenticity of the host is displayed and use `vagrant` as the password.
4. Note that, as you type in the password, it will not be displayed.

When Vagrant provisions VMs, you do not want to specify any SSH settings. You can find out more at `https://www.vagrantup.com/docs/vagrantfile/ssh_settings.html`. This will create a user with the username `vagrant` and password `vagrant` by default. We will use this default user to access both our nodes since we did not specify any users in our Vagrantfile.

5. Repeat this process to connect to `node2`. You can do this from either the `master` or `node2`. Note The connection will work as long as you provide the correct credentials, since all nodes are on the same network.

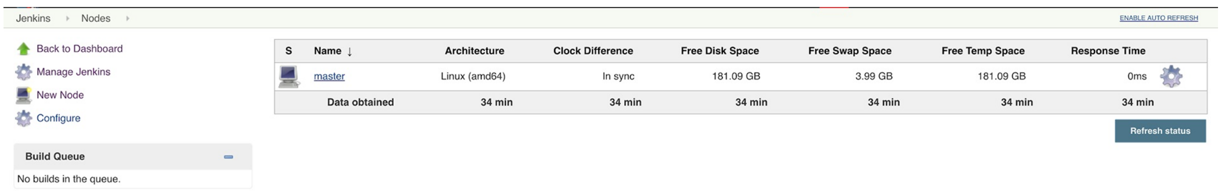
With that, you have successfully verified that the nodes can communicate with each other; thus, the next steps we will take in this chapter should work without any problems.

Securely Connecting to the Slaves

In the previous section, we provisioned our environment with a master that has Jenkins installed on it, and also added two nodes to our network. We are going to connect to our nodes via SSH with a process similar to what we did in the previous section when verifying connectivity, but this time via the Jenkins user interface.

Adding Slave Nodes

Currently, on our Jenkins setup, there is only one node and that is the master with two build executors, as we mentioned earlier. From the main dashboard, we can verify this by navigating to Manage Jenkins -> Manage Nodes:


Jenkins › Nodes ›

ENABLE AUTO REFRESH

Back to Dashboard
Manage Jenkins
New Node
Configure

Build Queue

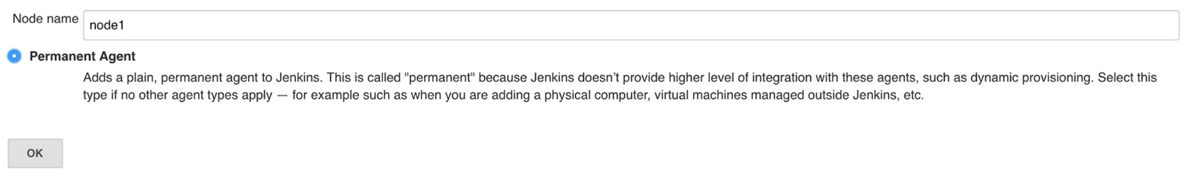
No builds in the queue.

S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	master	Linux (amd64)	In sync	181.09 GB	3.99 GB	181.09 GB	0ms
	Data obtained	34 min	34 min	34 min	34 min	34 min	34 min

Refresh status

We can see that there is only one node, which is the `master`. This label is assigned by default to the node on which Jenkins is installed. Now, let's add a slave node `node1` using the following steps:

1. On the left-hand side of the same view, we can see New Node on the menu.
2. Click on this and you will be presented with the following view:

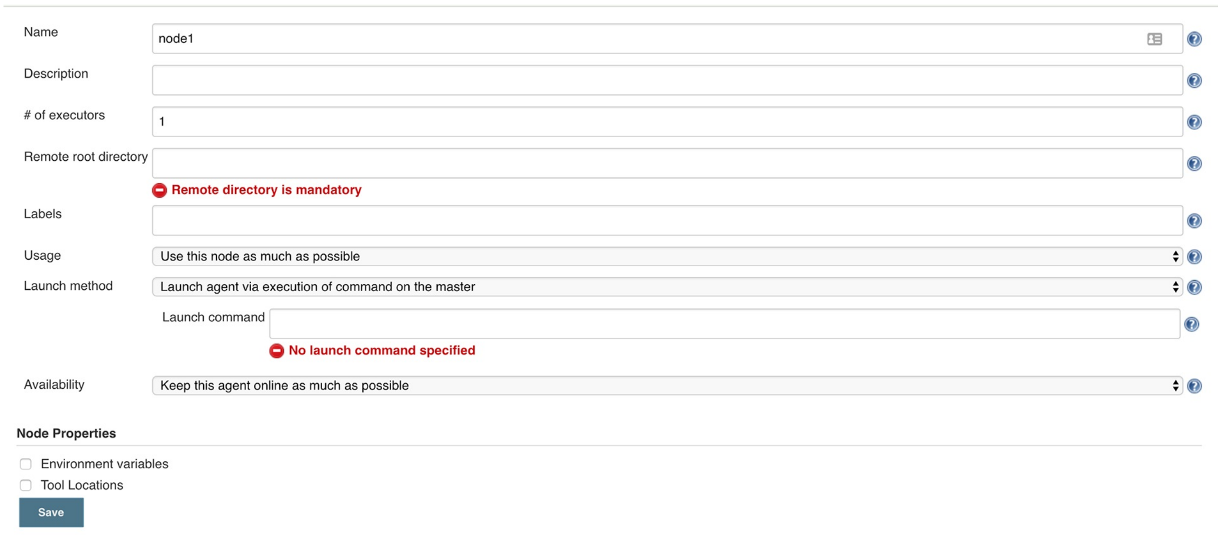

Node name: node1

● Permanent Agent

Adds a plain, permanent agent to Jenkins. This is called "permanent" because Jenkins doesn't provide higher level of integration with these agents, such as dynamic provisioning. Select this type if no other agent types apply — for example such as when you are adding a physical computer, virtual machines managed outside Jenkins, etc.

OK

3. Configure the preceding form and press OK. You will then be presented with the following view for further configuration of the node:

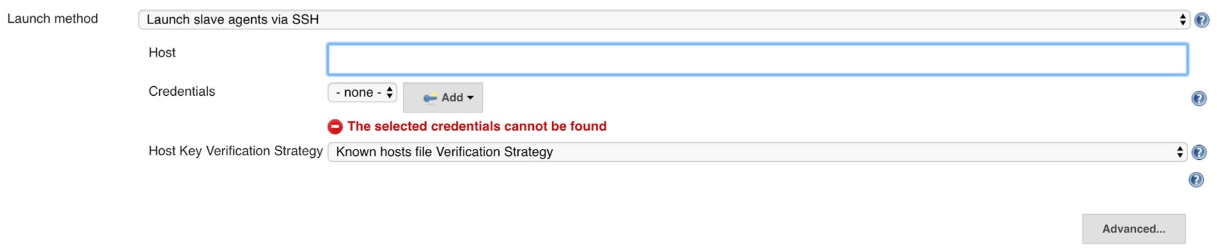


Name	node1
Description	
# of executors	1
Remote root directory	Remote directory is mandatory
Labels	
Usage	Use this node as much as possible
Launch method	Launch agent via execution of command on the master
Launch command	No launch command specified
Availability	Keep this agent online as much as possible

Node Properties

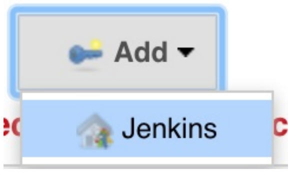
- [ ] Environment variables
- [ ] Tool Locations

4. For the configuration, fill in the first six form fields as follows:
   - Name: `node1`
   - Description: Our First Node (optional)
   - # of executors: 2
   - Remote root directory: `/tmp/`
   - Labels: `node1`
   - Usage: Use this node as much as possible
5. For the Launch method, select the drop-down and choose Launch slave agents via SSH. This will present us with another form, as follows:

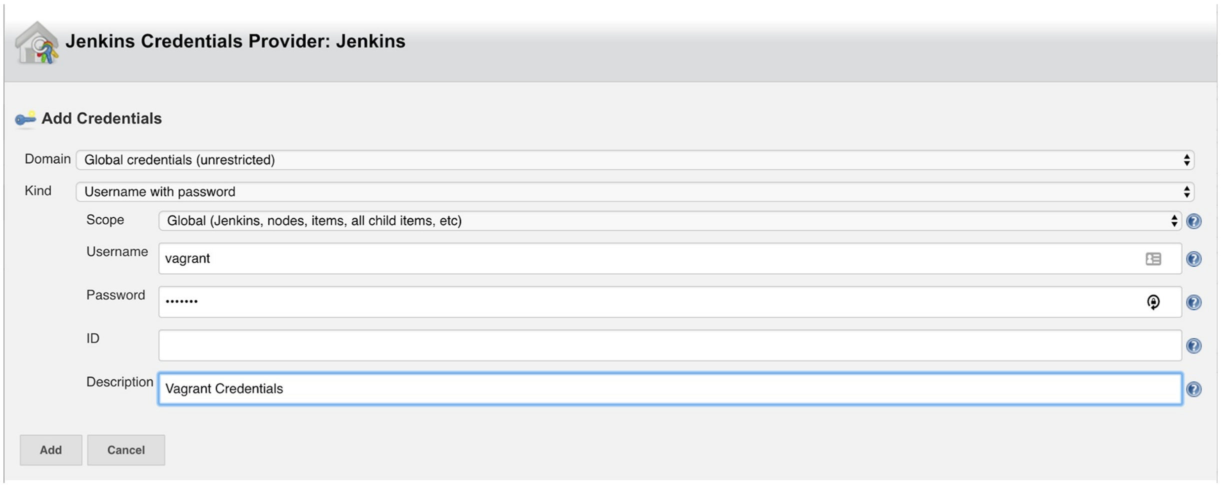


Launch method	Launch slave agents via SSH
Host	
Credentials	- none - ; The selected credentials cannot be found
Host Key Verification Strategy	Known hosts file Verification Strategy

For the host, enter the IP address as `node1`, as defined in the `Vagrantfile`: `192.168.1.20`. We will then create credentials to allow our master to connect to `node1`. Click on the Add drop-down next to the credentials field and select Jenkins.



This will display a form, which we will configure. For the password field, enter `vagrant` (these are the same SSH credentials we used earlier when verifying host connectivity):

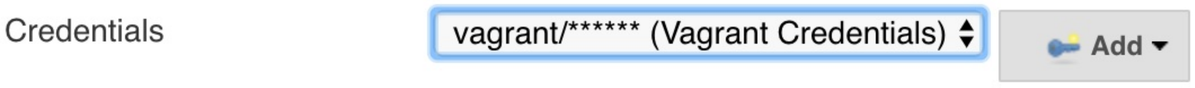


Jenkins Credentials Provider: Jenkins

Add Credentials

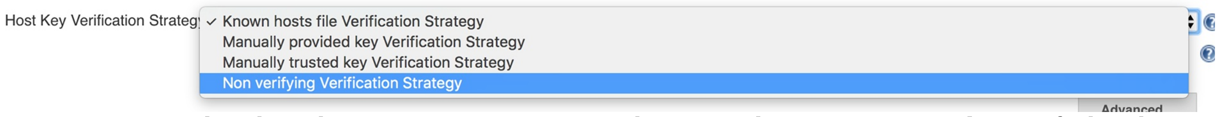
Domain	Global credentials (unrestricted)
Kind	Username with password
Scope	Global (Jenkins, nodes, items, all child items, etc)
Username	vagrant
Password	•••••••
ID	
Description	Vagrant Credentials

6. Click Add and select the credentials you just created on the previous view:



Credentials: vagrant/****** (Vagrant Credentials)

7. For the Host Key Verification Strategy, we are going to select Non verifying Verification Strategy, as follows:



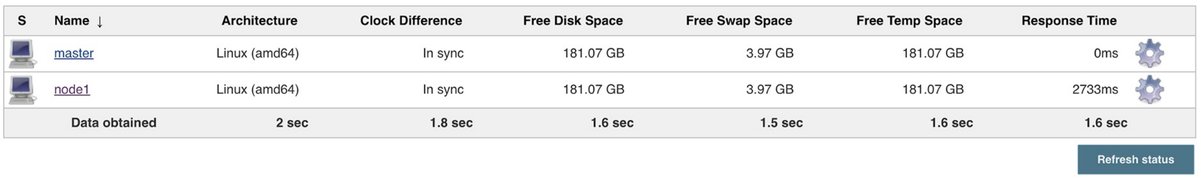
Host Key Verification Strategy
- ✓ Known hosts file Verification Strategy
- Manually provided key Verification Strategy
- Manually trusted key Verification Strategy
- Non verifying Verification Strategy

Remember that when connecting via SSH on the terminal, you are prompted to verify that the host is authentic, especially if you haven't connected to the host before. This will prevent the connection from failing and reduces the administrative tasks we have to perform if we want to verify our hosts. This is not recommended on production environments. Always ensure that you verify the hosts you are connecting to.

8. Finally, for Availability, select Keep this agent online as much as possible, as follows:

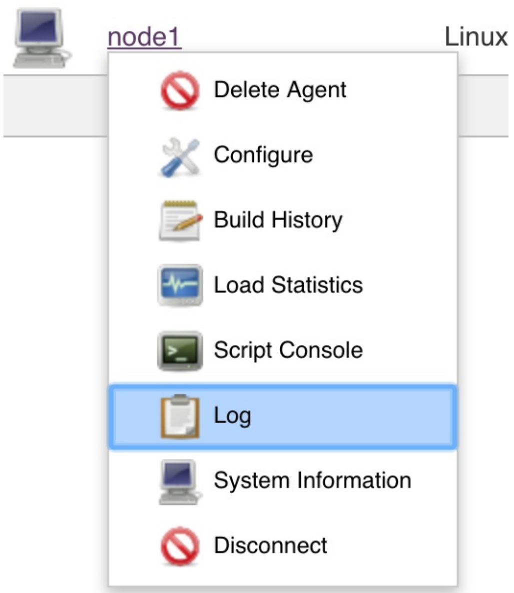
Availability: Keep this agent online as much as possible

9. Finally, press Save to add the node. Wait a few seconds and your node will go online if you configured everything correctly:

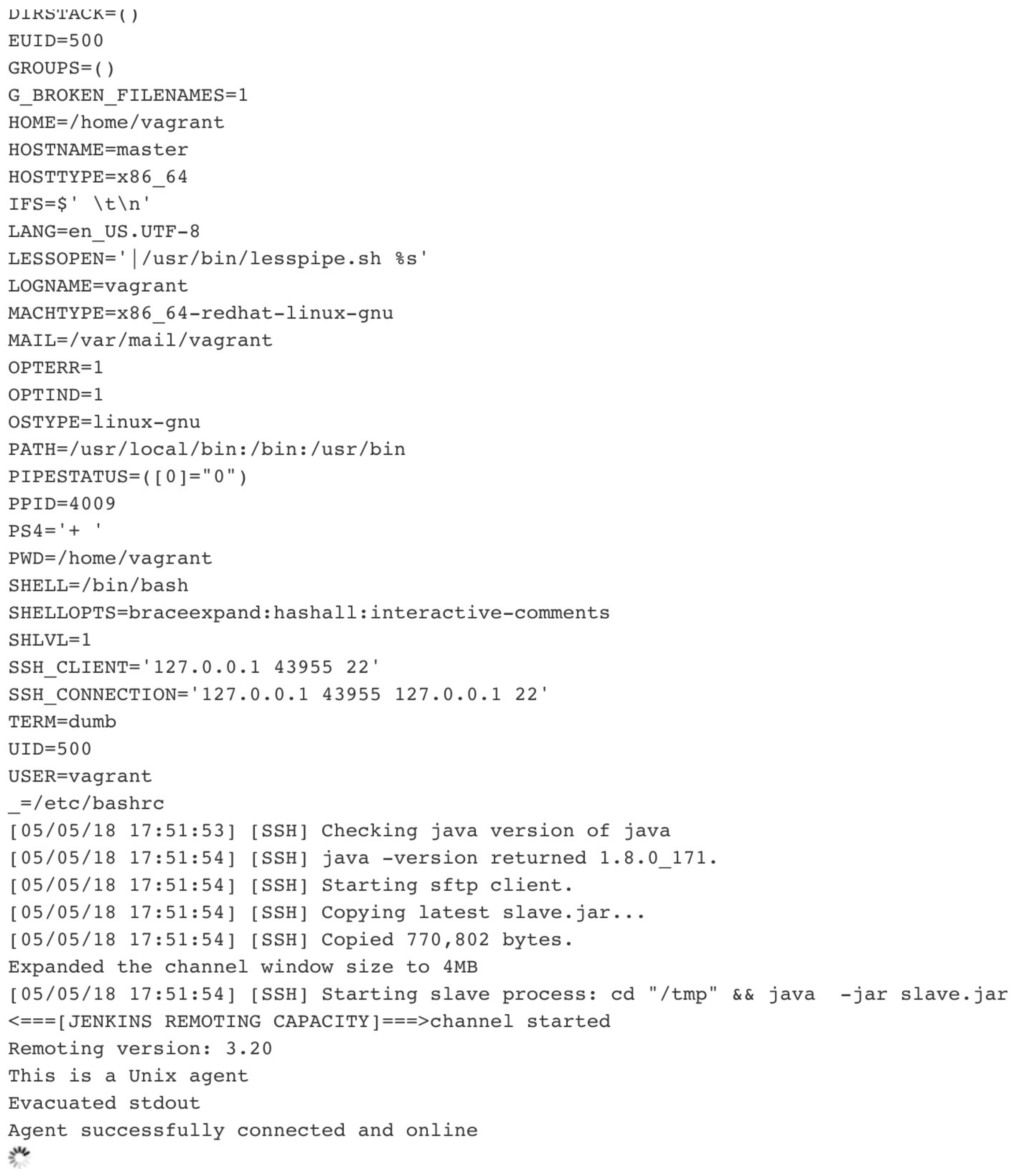


S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time	
	master	Linux (amd64)	In sync	181.07 GB	3.97 GB	181.07 GB	0ms	
	node1	Linux (amd64)	In sync	181.07 GB	3.97 GB	181.07 GB	2733ms	
	Data obtained	2 sec	1.8 sec	1.6 sec	1.5 sec	1.6 sec	1.6 sec	

10. If your node does not go online, click on the drop-down on the node name and select Log, as follows:

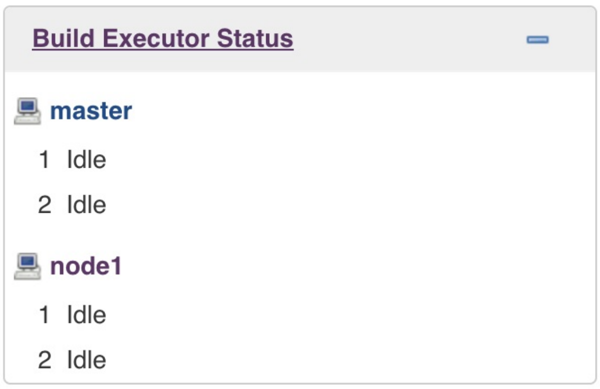


This will show you the connection attempt logs, and if anything goes wrong, you will be able to find out from this view:



```
DIRSTACK=()
EUID=500
GROUPS=()
G_BROKEN_FILENAMES=1
HOME=/home/vagrant
HOSTNAME=master
HOSTTYPE=x86_64
IFS=$' \t\n'
LANG=en_US.UTF-8
LESSOPEN='|/usr/bin/lesspipe.sh %s'
LOGNAME=vagrant
MACHTYPE=x86_64-redhat-linux-gnu
MAIL=/var/mail/vagrant
OPTERR=1
OPTIND=1
OSTYPE=linux-gnu
PATH=/usr/local/bin:/bin:/usr/bin
PIPESTATUS=([0]="0")
PPID=4009
PS4='+ '
PWD=/home/vagrant
SHELL=/bin/bash
SHELLOPTS=braceexpand:hashall:interactive-comments
SHLVL=1
SSH_CLIENT='127.0.0.1 43955 22'
SSH_CONNECTION='127.0.0.1 43955 127.0.0.1 22'
TERM=dumb
UID=500
USER=vagrant
_=/etc/bashrc
[05/05/18 17:51:53] [SSH] Checking java version of java
[05/05/18 17:51:54] [SSH] java -version returned 1.8.0_171.
[05/05/18 17:51:54] [SSH] Starting sftp client.
[05/05/18 17:51:54] [SSH] Copying latest slave.jar...
[05/05/18 17:51:54] [SSH] Copied 770,802 bytes.
Expanded the channel window size to 4MB
[05/05/18 17:51:54] [SSH] Starting slave process: cd "/tmp" && java  -jar slave.jar
<===[JENKINS REMOTING CAPACITY]===>channel started
Remoting version: 3.20
This is a Unix agent
Evacuated stdout
Agent successfully connected and online
```

11. Under the Build Executor Status section, we can see that we now have two nodes (`master` and `node1`) and four build executors in total:



Build Executor Status

master

1 Idle

2 Idle

node1

1 Idle

2 Idle