

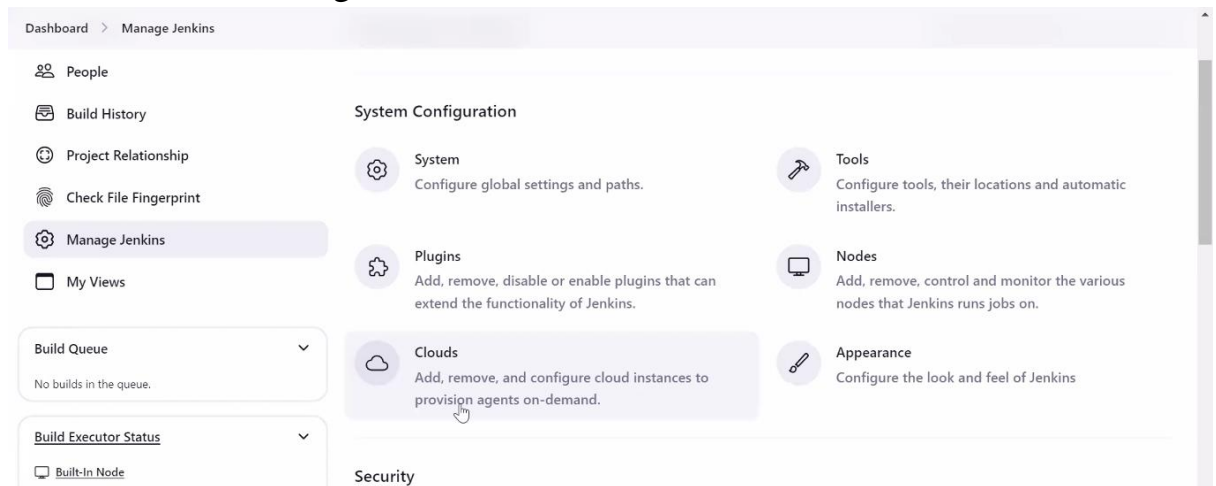


## Jenkins Session

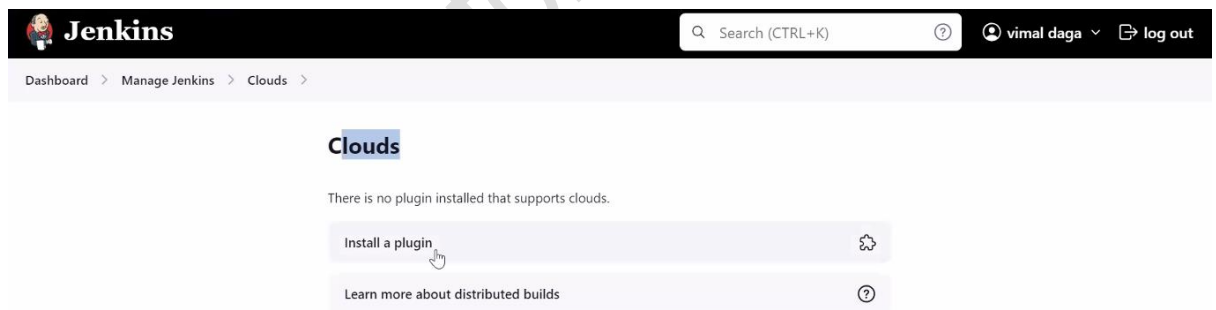
### Summary 03-03-2024

- In Jenkins multinode clusters with static nodes, a significant disadvantage is that if there are no jobs coming to Jenkins, the slave nodes will continue running, resulting in unnecessary resource wastage. To overcome this problem, we can utilize the concept of dynamic provisioning in Jenkins.
- Whenever any job come up, it is the duty of the master node to schedule the job to the slave nodes.
- In the slave node it is required to have the specific tools or softwares installed in order to perform the job.
- The power of dynamic provisioning is that, the master node will launch the slave whenever any job comeup and it will run till the job is running. Once the job is completed the slave will terminate automatically.
- By implementing dynamic provisioning, Jenkins clusters become more efficient, as resources are provisioned on-demand, reducing idle time and unnecessary resource consumption. This approach improves scalability and cost-effectiveness, making the cluster more responsive to varying workloads.
- For the Dynamic provisioning,
  - We need a platform where the master node will launch the slave automatically whenever any job come up to the master
  - Platform can be multi-cloud, Kubernetes or the docker platform.

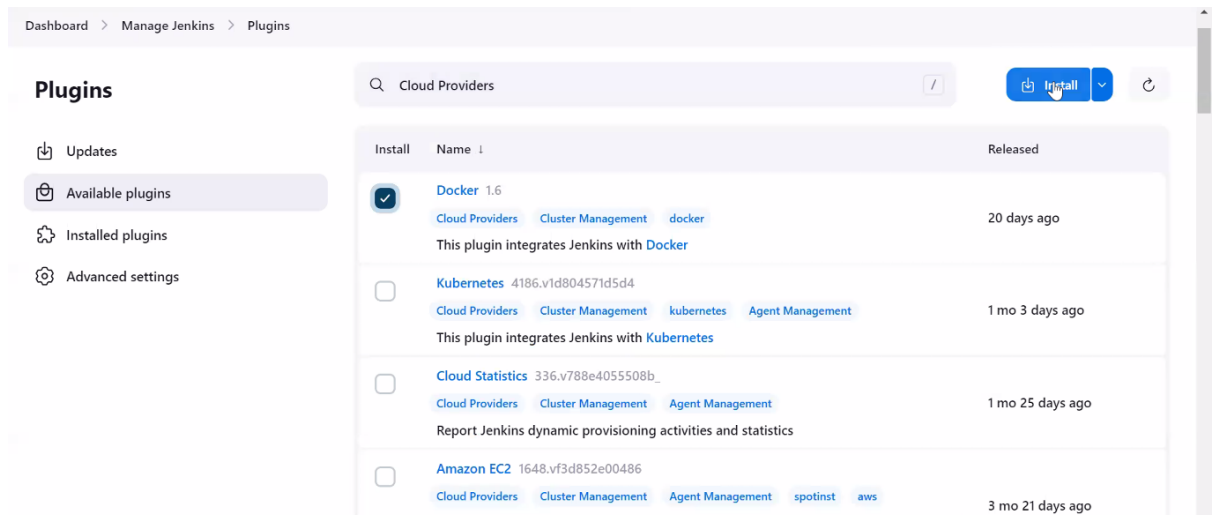
- We have several plugins which will help the master to contact with the platforms and launch the slave node there.
- To setup the dynamic provisioning architecture in the Jenkins follow the steps.
  - Go to the manage Jenkins and click on the cloud.



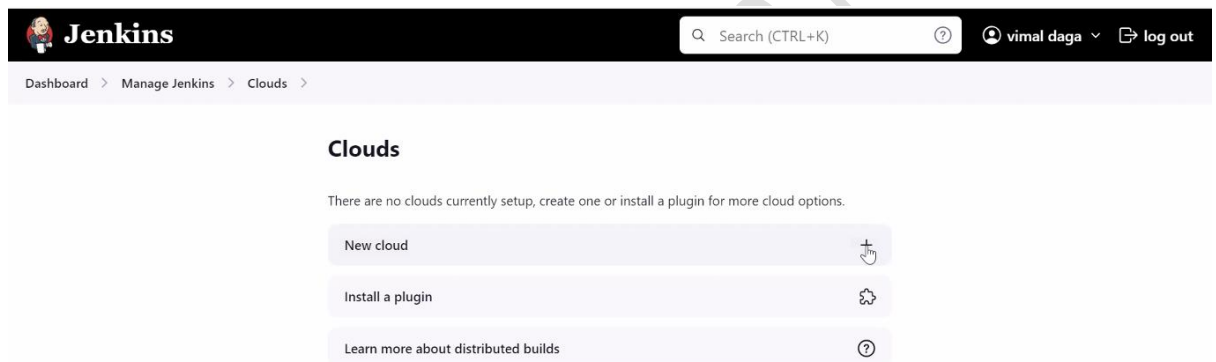
- Here we have to install the plugin which will help the Jenkins to connect with the platform where the slave nodes will be launched.



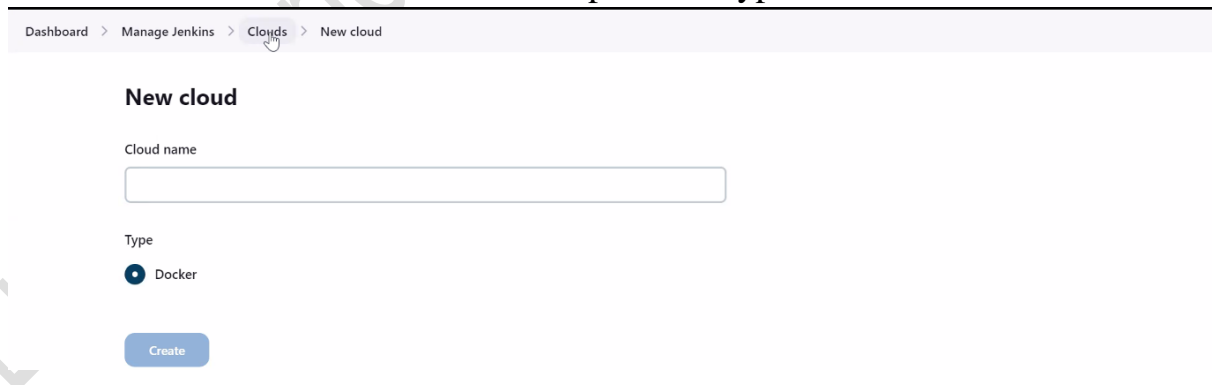
- We want to use docker as the platform so we will install the docker plugin for it.



- As soon as we install the plugin, we can see the new cloud option in the manage Jenkins section.

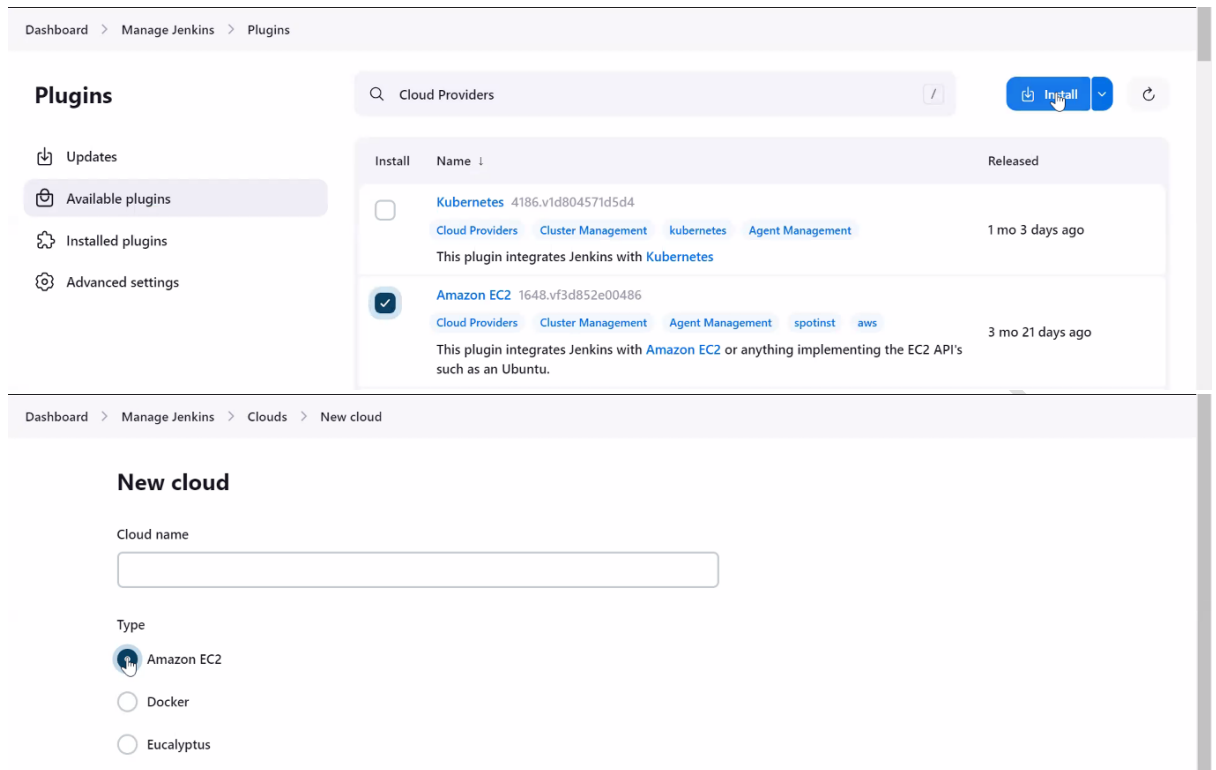


- In the new cloud we can see the platform type as docker.



- Similarly we can install amazon ec2 plugin if we want to use the ec2 as the platform type.

## [Mastering Jenkins]



The top screenshot shows the Jenkins 'Plugins' page. The breadcrumb trail is 'Dashboard > Manage Jenkins > Plugins'. A search bar contains 'Cloud Providers'. On the left, there are links for 'Updates', 'Available plugins', 'Installed plugins', and 'Advanced settings'. The main table lists installed plugins. The 'Amazon EC2' plugin is checked and highlighted. The bottom screenshot shows the 'New cloud' configuration page. The breadcrumb trail is 'Dashboard > Manage Jenkins > Clouds > New cloud'. The 'Cloud name' field is empty. Under the 'Type' section, 'Amazon EC2' is selected with a radio button, while 'Docker' and 'Eucalyptus' are unselected.

**Plugins**

Dashboard > Manage Jenkins > Plugins

Search: Cloud Providers

Install Name Released

Install	Name	Released
<input type="checkbox"/>	<b>Kubernetes</b> 4186.v1d804571d5d4 <a href="#">Cloud Providers</a> <a href="#">Cluster Management</a> <a href="#">kubernetes</a> <a href="#">Agent Management</a> This plugin integrates Jenkins with <a href="#">Kubernetes</a>	1 mo 3 days ago
<input checked="" type="checkbox"/>	<b>Amazon EC2</b> 1648.vf3d852e00486 <a href="#">Cloud Providers</a> <a href="#">Cluster Management</a> <a href="#">Agent Management</a> <a href="#">spotinst</a> <a href="#">aws</a> This plugin integrates Jenkins with <a href="#">Amazon EC2</a> or anything implementing the EC2 API's such as an Ubuntu.	3 mo 21 days ago

Dashboard > Manage Jenkins > Clouds > New cloud

**New cloud**

Cloud name

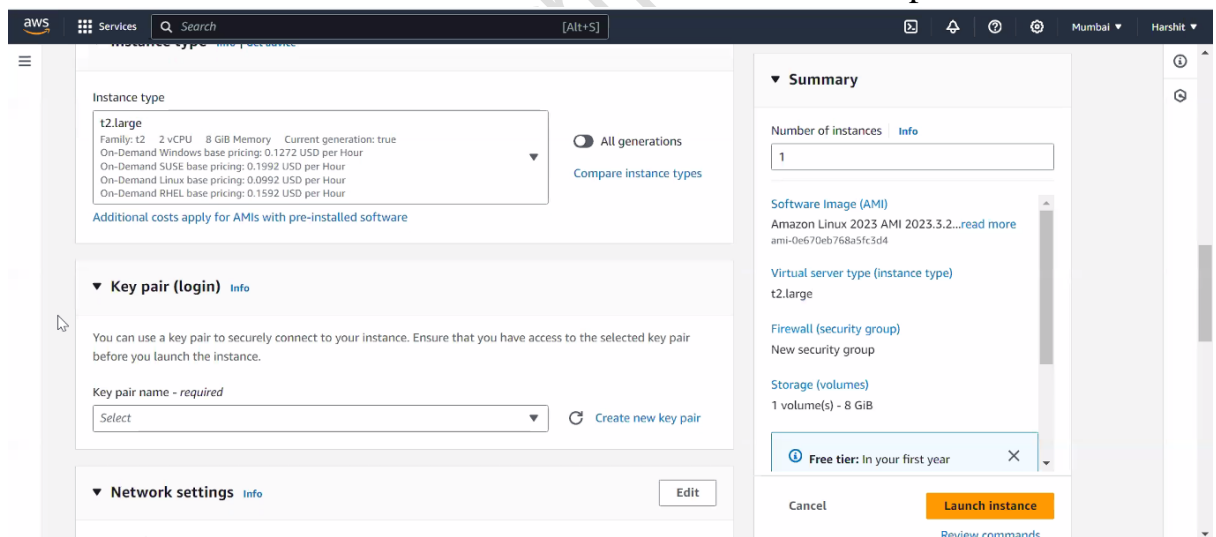
Type

☒ Amazon EC2

☐ Docker

☐ Eucalyptus

➤ Now we will launch the docker host to be used as the platform.



The screenshot shows the AWS Management Console 'Launch Instance' page. The breadcrumb trail is 'Services > EC2 > Launch Instance'. The 'Instance type' is set to 't2.large'. The 'Key pair (login)' section shows a dropdown for 'Key pair name - required' with a 'Create new key pair' button. The 'Network settings' section has an 'Edit' button. The 'Summary' section on the right shows 'Number of instances' as 1, 'Software Image (AMI)' as 'Amazon Linux 2023 AMI 2023.3.2...', 'Virtual server type (instance type)' as 't2.large', 'Firewall (security group)' as 'New security group', and 'Storage (volumes)' as '1 volume(s) - 8 GiB'. At the bottom, there is a 'Free tier: In your first year' banner and buttons for 'Cancel', 'Launch instance', and 'Review commands'.

aws Services Search [Alt+S]

Instance type

t2.large

Family: t2 2 vCPU 8 GiB Memory Current generation: true

On-Demand Windows base pricing: 0.1272 USD per Hour

On-Demand SUSE base pricing: 0.1992 USD per Hour

On-Demand Linux base pricing: 0.0992 USD per Hour

On-Demand RHEL base pricing: 0.1592 USD per Hour

Additional costs apply for AMIs with pre-installed software

▼ Key pair (login) Info

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - required

Select

Create new key pair

▼ Network settings Info

Edit

▼ Summary

Number of instances Info

1

Software Image (AMI)

Amazon Linux 2023 AMI 2023.3.2...read more

ami-0e670eb768a5fc3d4

Virtual server type (instance type)

t2.large

Firewall (security group)

New security group

Storage (volumes)

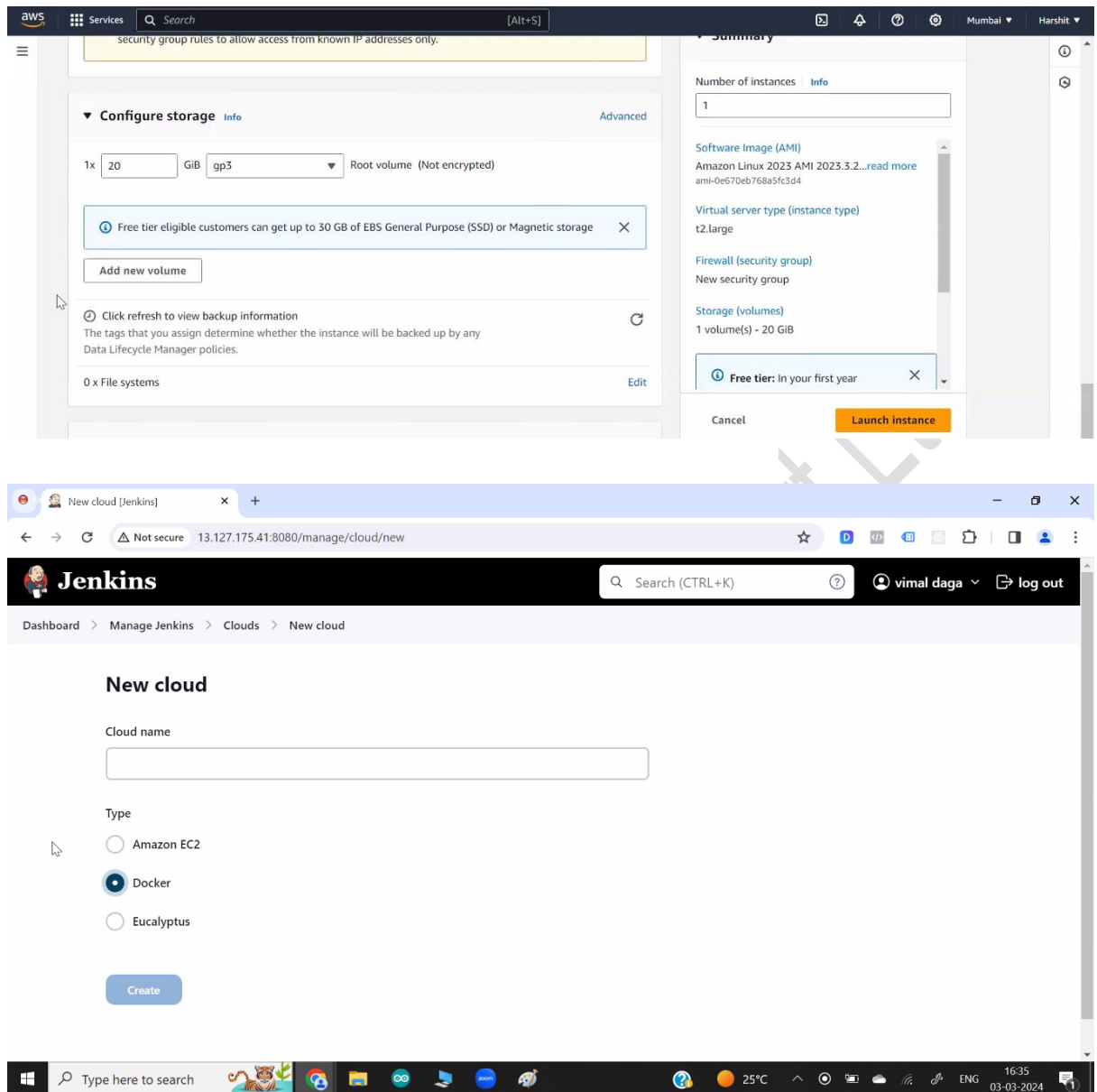
1 volume(s) - 8 GiB

Free tier: In your first year

Cancel Launch instance Review commands

➤ Configure the instance type for the docker host.

[Mastering Jenkins]



- In the docker host, install and enable the docker services.

```

#_
~\  ###
~\  #####\
~\  #####\
~\  \###|
~\  \|/
~\  V~'  ->
~\  /m/'
~\  /m/'

Amazon Linux 2023

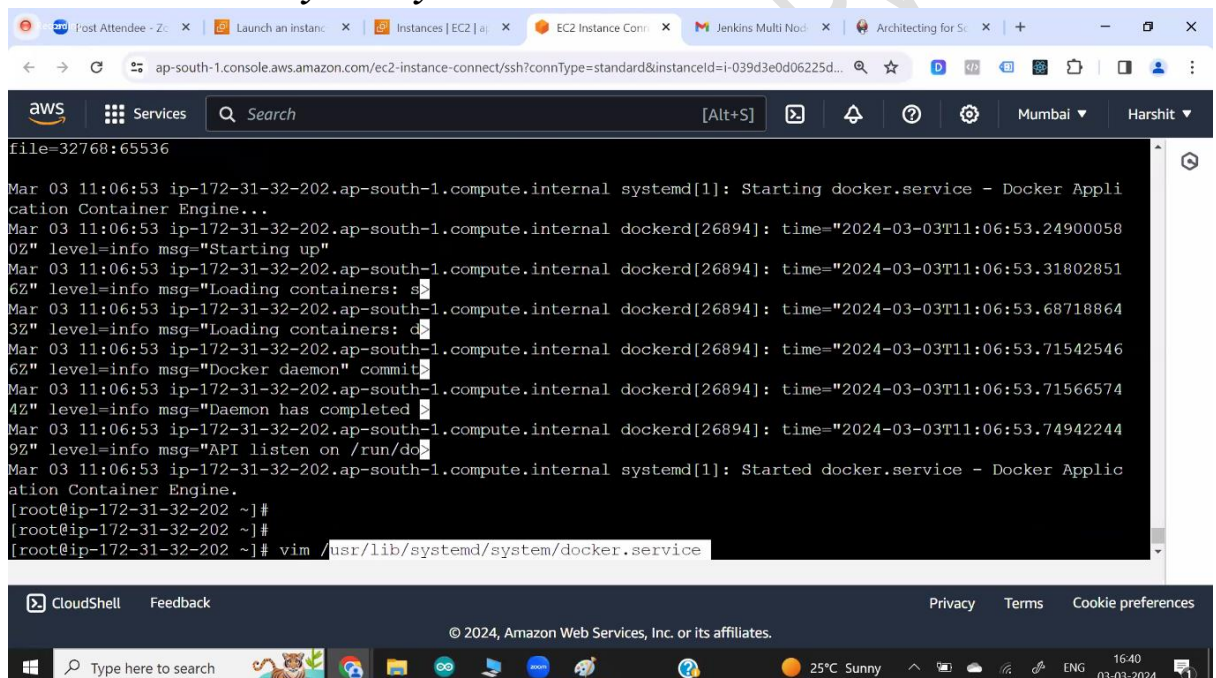
https://aws.amazon.com/linux/amazon-linux-2023

[ec2-user@ip-172-31-32-202 ~]$ sudo su -
[root@ip-172-31-32-202 ~]# yum install docker -y
Last metadata expiration check: 0:02:07 ago on Sun Mar 3 11:04:21 2024.
Dependencies resolved.

Complete!
[root@ip-172-31-32-202 ~]# systemctl start docker
[root@ip-172-31-32-202 ~]# systemctl enable docker
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service -> /usr/lib/systemd/system/docker.service.
[root@ip-172-31-32-202 ~]#
```

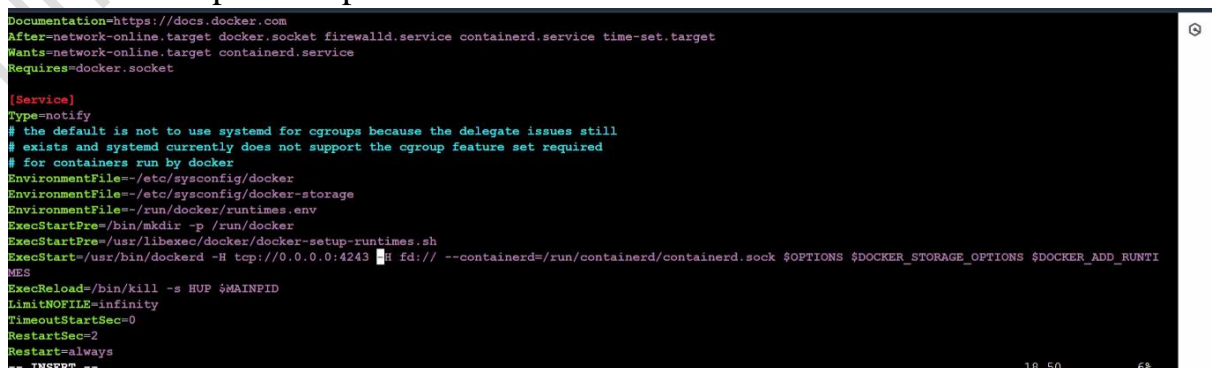
- Now this instance will be our docker host or the platform, only thing we have to do extra here is make this host reachable to the master node.
- Master node will only contact to the docker host.
- By default docker only supports local functionality means Docker can launch the containers by taking inputs locally from the machine but in this setup, we need to launch the container via network using TCP, so we need to enable settings in docker to support TCP services.
- To overcome this problem we have to make some changes in the docker service file.
- Open the service file of the docker.

***vim /usr/lib/systemd/system/docker.service***



```
file=32768:65536
Mar 03 11:06:53 ip-172-31-32-202.ap-south-1.compute.internal systemd[1]: Starting docker.service - Docker Application Container Engine...
Mar 03 11:06:53 ip-172-31-32-202.ap-south-1.compute.internal dockerd[26894]: time="2024-03-03T11:06:53.249000580Z" level=info msg="Starting up"
Mar 03 11:06:53 ip-172-31-32-202.ap-south-1.compute.internal dockerd[26894]: time="2024-03-03T11:06:53.318028516Z" level=info msg="Loading containers: start"
Mar 03 11:06:53 ip-172-31-32-202.ap-south-1.compute.internal dockerd[26894]: time="2024-03-03T11:06:53.687188643Z" level=info msg="Loading containers: done"
Mar 03 11:06:53 ip-172-31-32-202.ap-south-1.compute.internal dockerd[26894]: time="2024-03-03T11:06:53.715425466Z" level=info msg="Docker daemon" commit
Mar 03 11:06:53 ip-172-31-32-202.ap-south-1.compute.internal dockerd[26894]: time="2024-03-03T11:06:53.715665744Z" level=info msg="Daemon has completed"
Mar 03 11:06:53 ip-172-31-32-202.ap-south-1.compute.internal dockerd[26894]: time="2024-03-03T11:06:53.749422449Z" level=info msg="API listen on /run/docker.sock"
Mar 03 11:06:53 ip-172-31-32-202.ap-south-1.compute.internal systemd[1]: Started docker.service - Docker Application Container Engine.
[root@ip-172-31-32-202 ~]#
[root@ip-172-31-32-202 ~]#
[root@ip-172-31-32-202 ~]# vim /usr/lib/systemd/system/docker.service
```

- `fd://` — means it will support local functionality only. To support TCP we need to add the below line in this section, if the request come up on the port number 4243 then docker will listen to it.

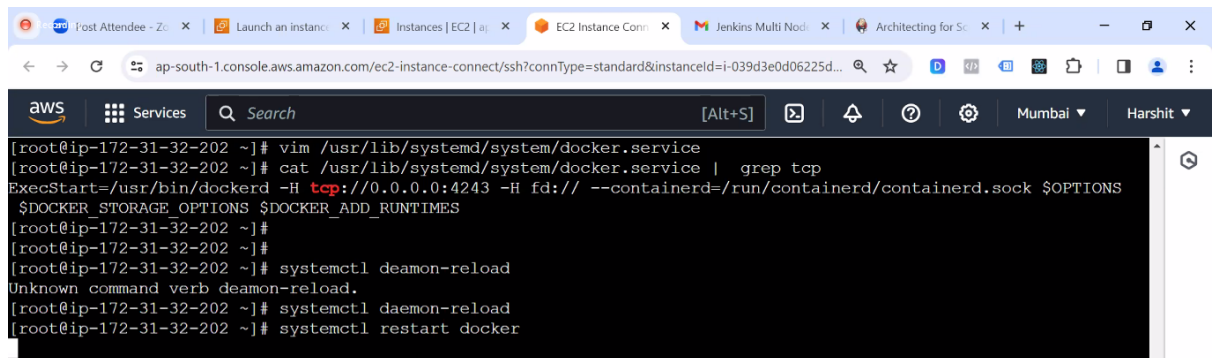


```
Documentation=https://docs.docker.com
After=network-online.target docker.socket firewalld.service containerd.service time-set.target
Wants=network-online.target containerd.service
Requires=docker.socket

[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
EnvironmentFile=/etc/sysconfig/docker
EnvironmentFile=/etc/sysconfig/docker-storage
EnvironmentFile=/run/docker/runtimes.env
ExecStartPre=/bin/mkdir -p /run/docker
ExecStartPre=/usr/libexec/docker/docker-setup-runtimes.sh
ExecStart=/usr/bin/dockerd -H tcp://0.0.0.0:4243 -H fd:// --containerd=/run/containerd/containerd.sock $OPTIONS $DOCKER_STORAGE_OPTIONS $DOCKER_ADDITIONAL_OPTIONS
ExecReload=/bin/kill -s HUP $MAINPID
LimitNOFILE=infinity
TimeoutStartSec=0
RestartSec=2
Restart=always
-- INSERT --
```

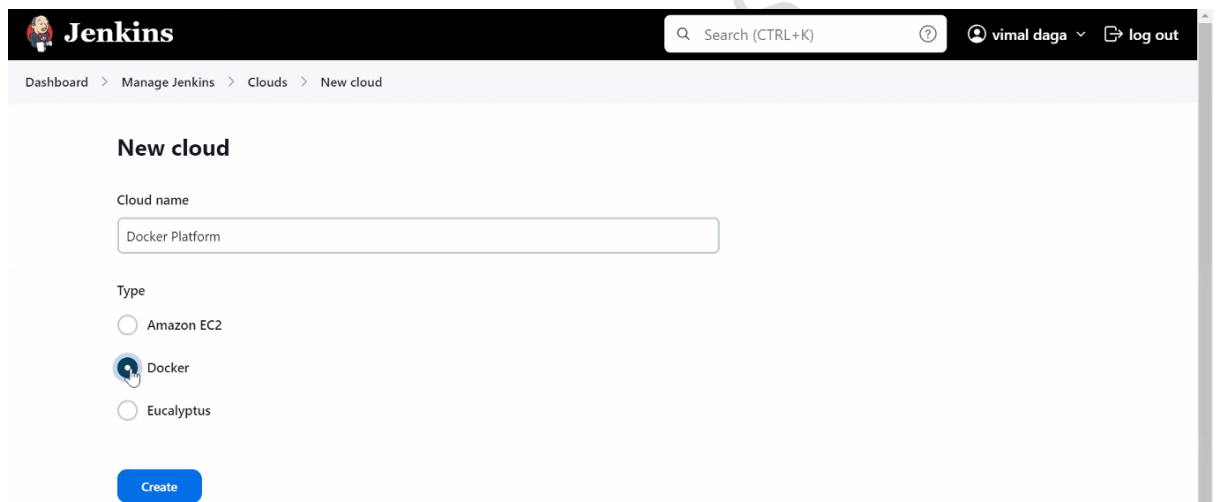
- This process is also known as socket binding.

- Then we need to reload systemd as we have updated systemd file and then restart the docker.



```
[root@ip-172-31-32-202 ~]# vim /usr/lib/systemd/system/docker.service
[root@ip-172-31-32-202 ~]# cat /usr/lib/systemd/system/docker.service | grep tcp
ExecStart=/usr/bin/dockerd -H tcp://0.0.0.0:4243 -H fd:// --containerd=/run/containerd/containerd.sock $OPTIONS
$DOCKER_STORAGE_OPTIONS $DOCKER_ADD_RUNTIMES
[root@ip-172-31-32-202 ~]#
[root@ip-172-31-32-202 ~]#
[root@ip-172-31-32-202 ~]# systemctl daemon-reload
Unknown command verb daemon-reload.
[root@ip-172-31-32-202 ~]# systemctl daemon-reload
[root@ip-172-31-32-202 ~]# systemctl restart docker
```

- Now this docker will take the instruction from the outside world.
- We have to connect the Jenkins master to the docker host now. For this go to the new cloud again and select the docker as the type.



**Jenkins** Search (CTRL+K) vimal daga log out

Dashboard > Manage Jenkins > Clouds > New cloud

### New cloud

Cloud name  
Docker Platform

Type

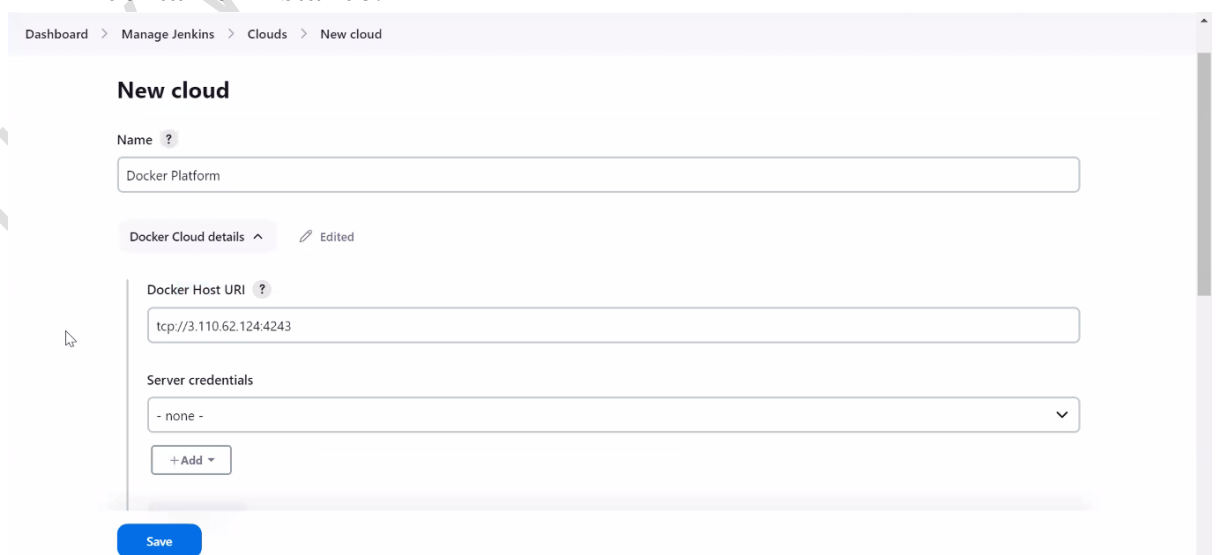
☐ Amazon EC2

☒ Docker

☐ Eucalyptus

Create

- Now, enter the docker-machine details that we set up in the container instance.



Dashboard > Manage Jenkins > Clouds > New cloud

### New cloud

Name ?  
Docker Platform

Docker Cloud details ^ Edited

Docker Host URI ?  
tcp://3.110.62.124:4243

Server credentials  
- none -

+ Add

Save



- Click on the test connection to test the connectivity to the docker host.

Dashboard > Manage Jenkins > Clouds > New cloud

Docker Host URI ?  
tcp://3.110.62.124:4243

Server credentials  
- none -  
+ Add

Advanced ▾  
Testing...

Test Connection

Enabled ?  
Note: Disabled.

Error Duration ?

Save

- This test might fail due to firewall issue of the aws instance. For this we have to edit the inbound rule of the aws instance.
- Go to the security group of that instance and edit the inbound rule.

EC2 Dashboard  
EC2 Global View  
Events

Instances  
Instances  
Instance Types  
Launch Templates  
Spot Requests  
Savings Plans  
Reserved Instances  
Dedicated Hosts  
Capacity Reservations  
New  
Images  
AMIs  
AMI Catalog

Instances (1/7) Info  
Find Instance by attribute or tag (case-sensitive)  
Any state

Name	Instance ID	Instance state	Instance type	Status check	Alarm
Docker platform	i-039d3e0d06225d53c	Running	t2.large	Initializing	View al
jenkins slave 02	i-0a98404a9d68e22cc	Running	t2.medium	2/2 checks passed	View al
jenkins slave 01	i-06133ad45d918b58a	Running	t2.medium	2/2 checks passed	View al

Instance: i-039d3e0d06225d53c (Docker platform)

Security groups  
sg-065bd9e24feda263f (launch-wizard-6)

Inbound rules

Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-0b33722ae4b96cfea	SSH	TCP	22	0.0.0.0/0	
-	All traffic	All	All	0.0.0.0/0	

Add rule

- Now again test the connection and we can see that the master Jenkins can successfully connect to the docker host.



Dashboard > Manage Jenkins > Clouds > New cloud

Docker Host URI ?  
tcp://3.110.62.124:4243

Server credentials  
- none -  
+ Add

Advanced

Version = 24.0.5, API Version = 1.43

Test Connection

☐ Enabled ?  
Note: Disabled.

Error Duration ?

Save

- We have to enable this platform now.

Version = 24.0.5, API Version = 1.43

Test Connection

☒ Enabled ?

Error Duration ?

Save

- So finally the Jenkins master is successfully connected with the docker host platform.

Jenkins

Search (CTRL+K)

vimal daga log out

Dashboard > Manage Jenkins > Clouds >

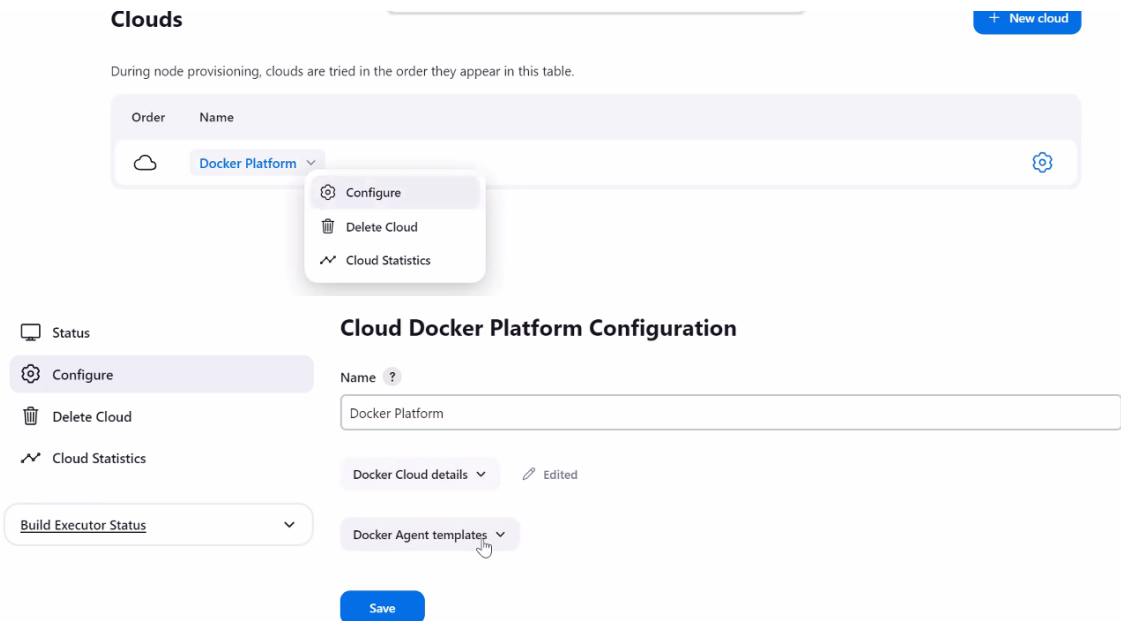
Clouds

+ New cloud

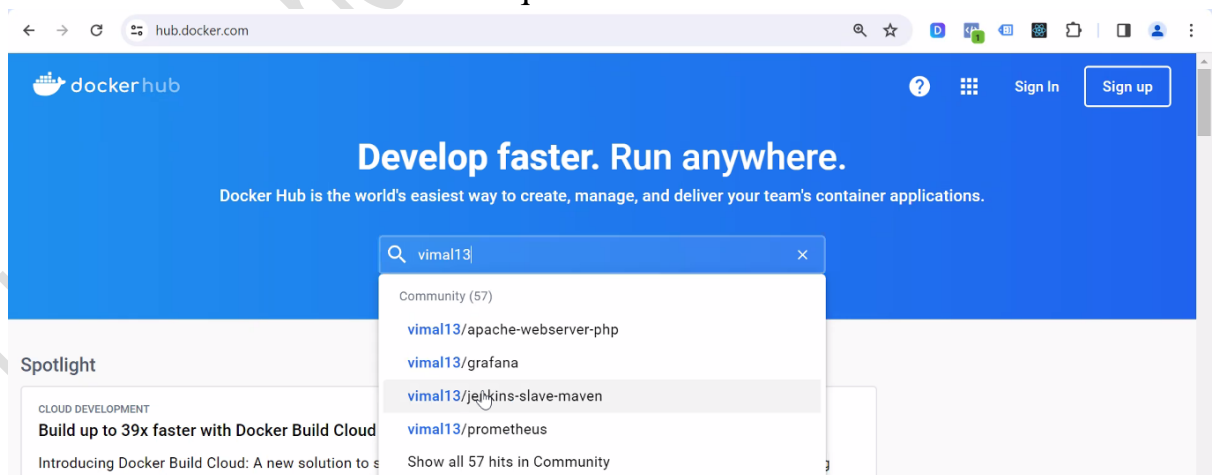
During node provisioning, clouds are tried in the order they appear in this table.

Order	Name
1	Docker Platform

- Now we have to set the template for this platform.



- Template means, whenever any job come to the master, it will launch the slave.
- Now there are two prerequisites for the slave, every job has its requirements so that tool kit need to be present in the slave, Slave should have ssh enabled and there should be java available in it.
- For this we have to use some image which will have the tool kit and the ssh, java enabled in it.
- As soon as the slave is launched, first the slave have to register itself to the master node so that the job can be assigned.
- We will use a pre created image otherwise we can create one also but it should meet all the requirements.



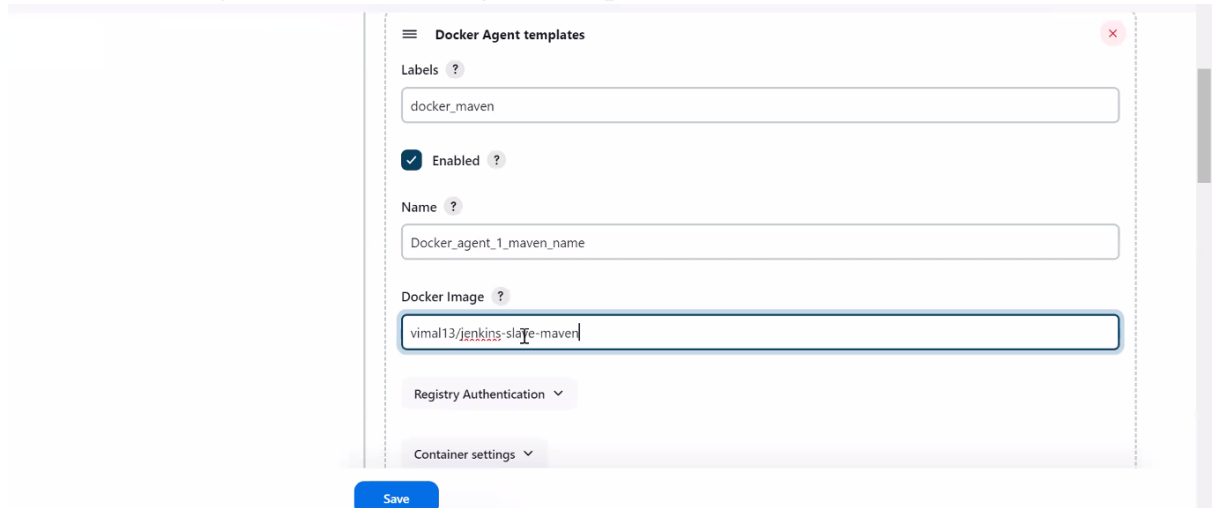
- Pull or download this image using the docker pull command.

```
[root@ip-172-31-32-202 ~]# docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
[root@ip-172-31-32-202 ~]#
[root@ip-172-31-32-202 ~]#
[root@ip-172-31-32-202 ~]# docker pull vimal13/jenkins-slave-maven
```

- We can check whether the ssh is enabled or not in this image.

```
[root@ip-172-31-32-202 ~]# ssh jenkins@172.17.0.2
The authenticity of host '172.17.0.2 (172.17.0.2)' can't be established.
ED25519 key fingerprint is SHA256:7PyU4Qw3tDIkwjXaj3a7OMN7tt0Aetxy4SF9o3e+ET4.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '172.17.0.2' (ED25519) to the list of known hosts.
jenkins@172.17.0.2's password:
```

- Jenkins should know which image is to be used so we have to give the image details in the agent template.



The screenshot shows the 'Docker Agent templates' configuration page in Jenkins. The 'Labels' field is set to 'docker\_maven'. The 'Enabled' checkbox is checked. The 'Name' field is set to 'Docker\_agent\_1\_maven\_name'. The 'Docker Image' field is set to 'vimal13/jenkins-slave-maven'. The 'Registry Authentication' dropdown is set to 'None'. The 'Container settings' dropdown is set to 'Default'. A 'Save' button is at the bottom.

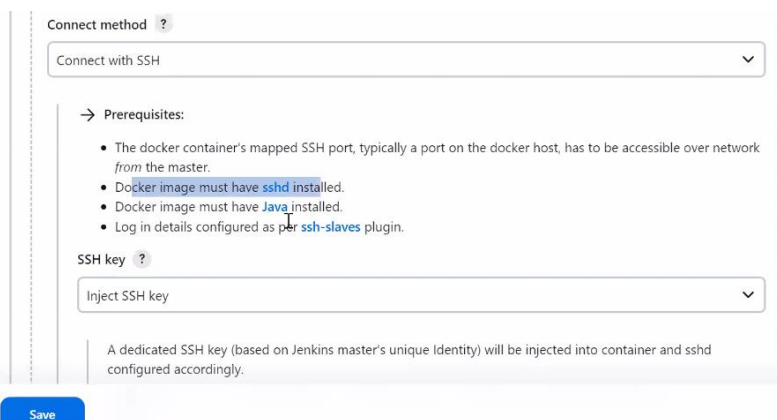
- We have to give the workspace details to the master Jenkins.

```
jenkins@bb26d603fb32:~$
jenkins@bb26d603fb32:~$
jenkins@bb26d603fb32:~$ whoami
jenkins
jenkins@bb26d603fb32:~$ pwd
/home/jenkins
jenkins@bb26d603fb32:~$ ls
```



The screenshot shows the workspace configuration page in Jenkins. The 'Instance Capacity' field is empty. The 'Remote File System Root' field is set to '/home/jenkins'.

- Now we have to select the connect method as the ssh and for this the docker image must have ssh and java installed in it then only this connect method will work.

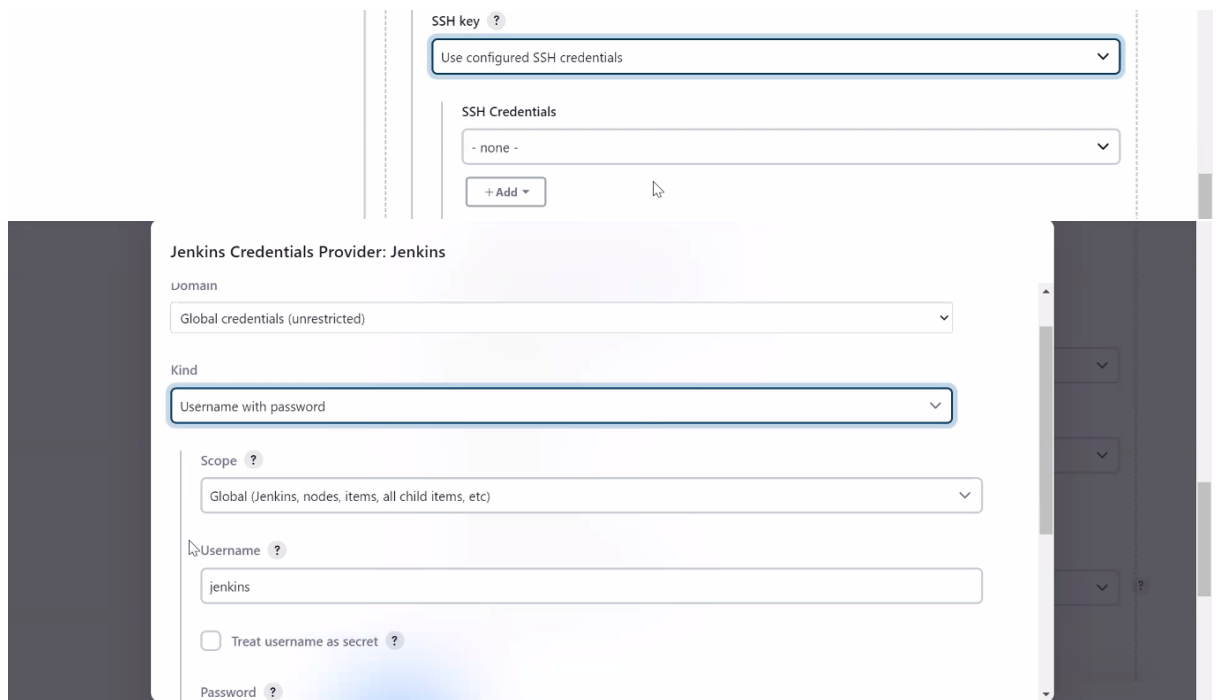


The screenshot shows the 'Connect method' configuration page in Jenkins. The 'Connect method' dropdown is set to 'Connect with SSH'. The 'Prerequisites' section lists the following requirements:

- The docker container's mapped SSH port, typically a port on the docker host, has to be accessible over network from the master.
- Docker image must have `sshd` installed.
- Docker image must have `Java` installed.
- Log in details configured as per `ssh-slaves` plugin.

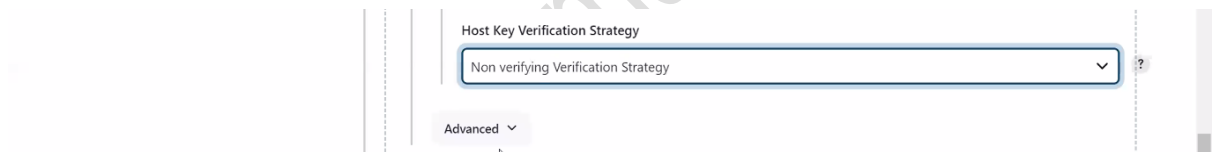
The 'SSH key' dropdown is set to 'Inject SSH key'. A note at the bottom states: 'A dedicated SSH key (based on Jenkins master's unique Identity) will be injected into container and sshd configured accordingly.' A 'Save' button is at the bottom.

- Add the ssh credentials here.



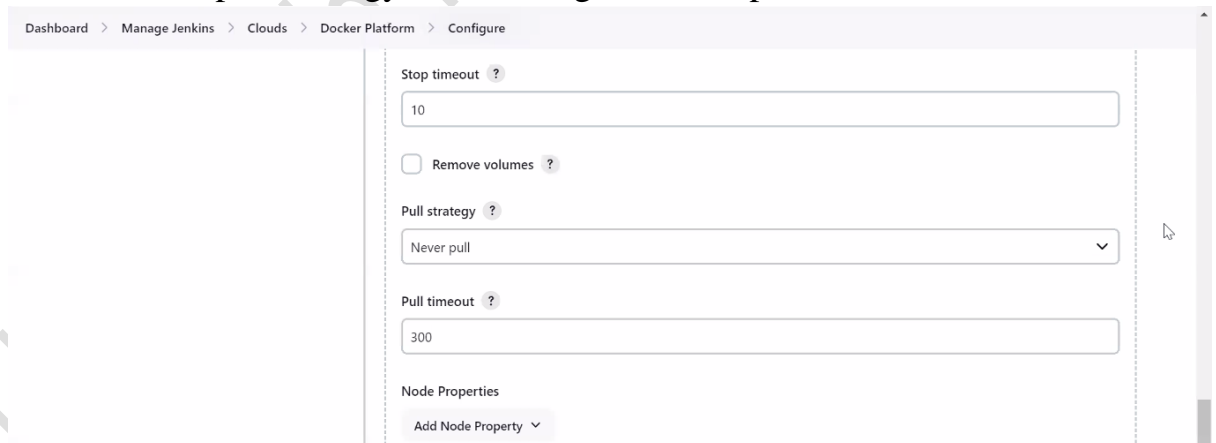
The screenshot shows the 'Jenkins Credentials Provider: Jenkins' configuration page. The 'Domain' is set to 'Global credentials (unrestricted)'. The 'Kind' is set to 'Username with password'. The 'Scope' is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'Username' is set to 'jenkins'. There is an unchecked checkbox for 'Treat username as secret'. The 'Password' field is empty. Above this, there is a section for 'SSH key' with a dropdown set to 'Use configured SSH credentials' and an 'SSH Credentials' dropdown set to '- none -' with an '+ Add' button below it.

- Select the non-verifying strategy for the host key verification.



The screenshot shows a 'Host Key Verification Strategy' dropdown menu. The selected option is 'Non verifying Verification Strategy'. There is an 'Advanced' dropdown button below it.

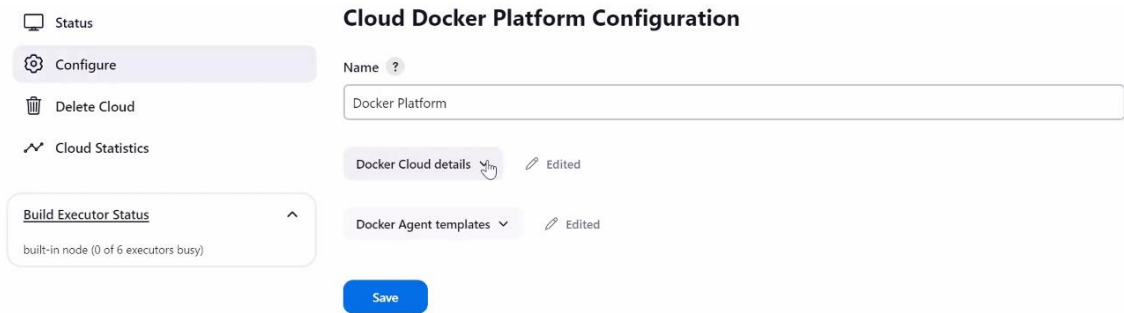
- Set the pull strategy for the image to never pull.



The screenshot shows the 'Configure' page for the 'Docker Platform'. The breadcrumb trail is 'Dashboard > Manage Jenkins > Clouds > Docker Platform > Configure'. The 'Stop timeout' is set to '10'. There is an unchecked checkbox for 'Remove volumes'. The 'Pull strategy' is set to 'Never pull'. The 'Pull timeout' is set to '300'. There is a section for 'Node Properties' with an 'Add Node Property' button.

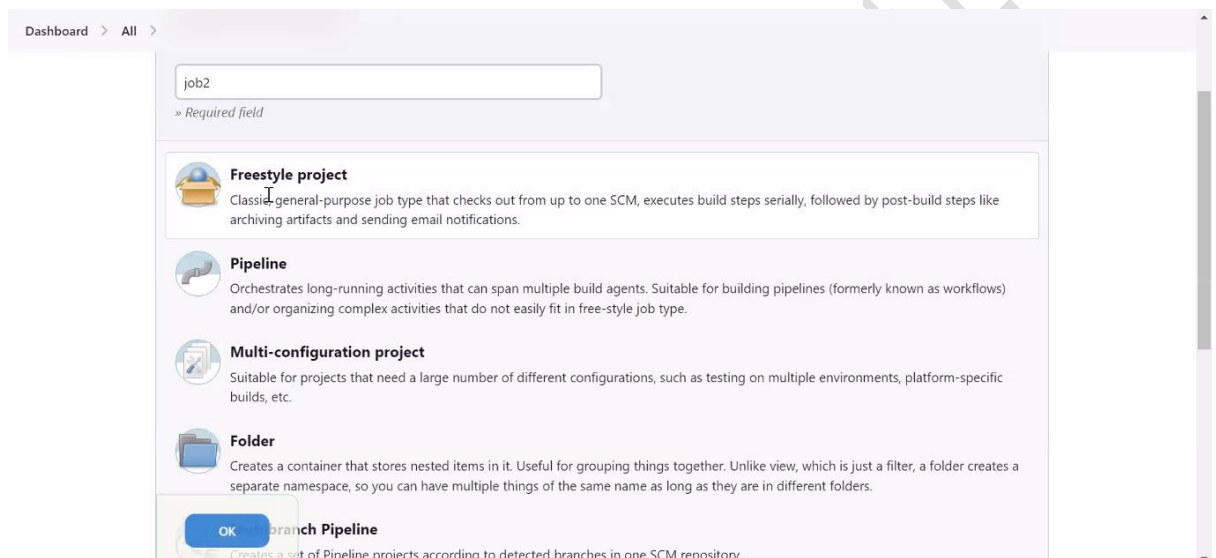
- Now we are done with the agent template part also.

## [Mastering Jenkins]



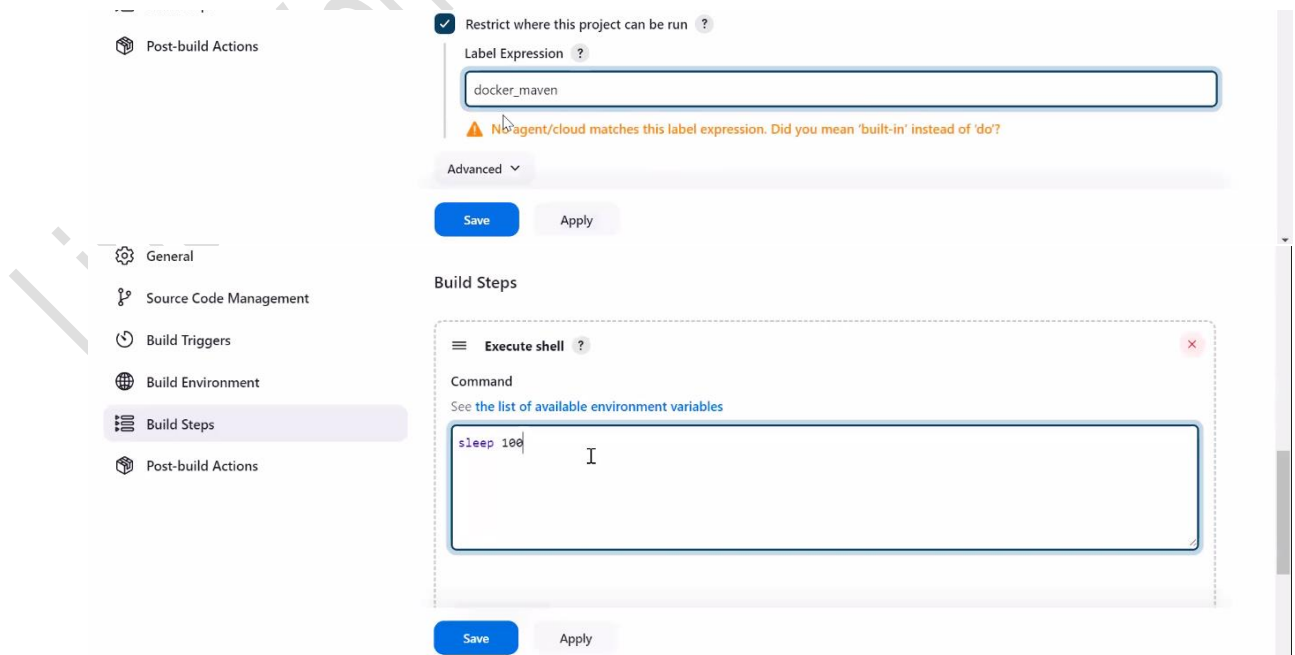
The screenshot shows the 'Cloud Docker Platform Configuration' page in Jenkins. On the left, there is a sidebar with links: 'Status', 'Configure' (highlighted), 'Delete Cloud', 'Cloud Statistics', and 'Build Executor Status' (showing 'built-in node (0 of 6 executors busy)'). The main area has a 'Name' field with the value 'Docker Platform'. Below it are two expandable sections: 'Docker Cloud details' and 'Docker Agent templates', both marked as 'Edited'. A blue 'Save' button is at the bottom.

- Will create a job to test the dynamic provisioning of the slave nodes.



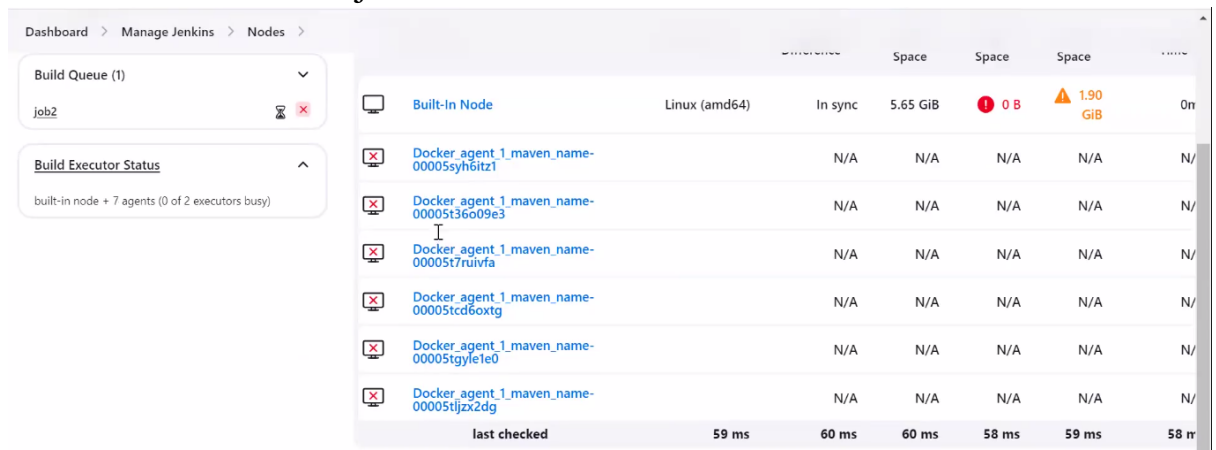
The screenshot shows the 'Job Configuration' page for a new job named 'job2'. The 'Job type' section is expanded, showing options: 'Freestyle project' (selected), 'Pipeline', 'Multi-configuration project', and 'Folder'. Each option has a brief description. At the bottom, there is a blue 'OK' button and a link to 'Branch Pipeline'.

- Restrict this job to run on the platform or the cloud we have created.



The screenshot shows the 'Restrict where this project can be run' section of the Jenkins job configuration. The 'Label Expression' field contains 'docker\_maven'. A warning message states: 'No agent/cloud matches this label expression. Did you mean 'built-in' instead of 'do'?'. Below this, there is an 'Advanced' section with 'Save' and 'Apply' buttons. The 'Build Steps' section is also visible, showing a step named 'Execute shell' with the command 'sleep 100'.

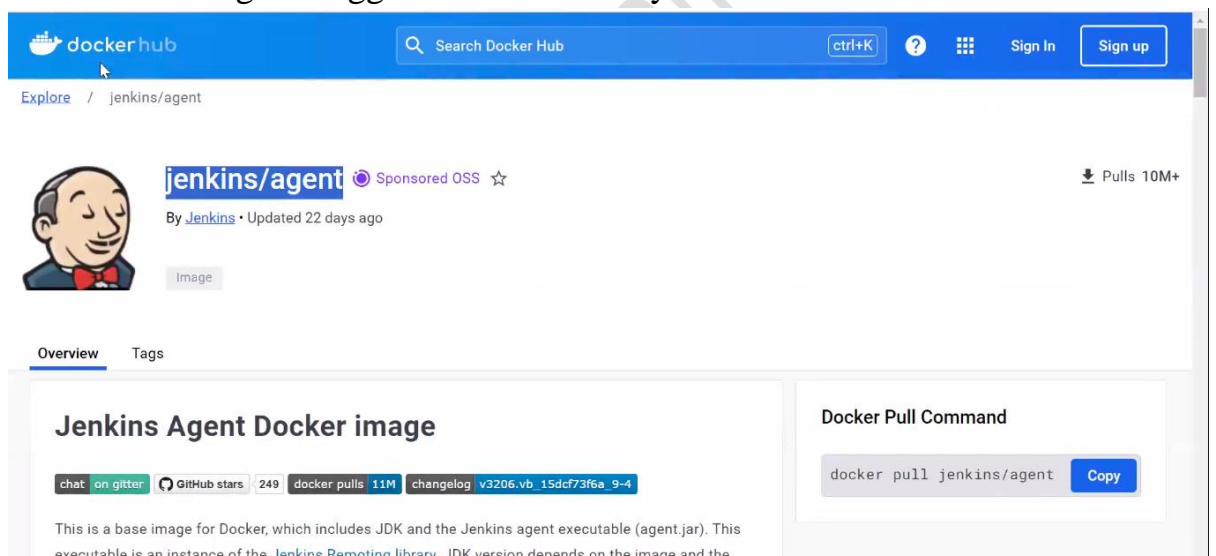
- Now as we run the job we can see that it will start launching some slaves to run the job.



The screenshot shows the Jenkins 'Nodes' page. On the left, there is a 'Build Queue (1)' section with a job named 'job2'. Below it, the 'Build Executor Status' section shows 'built-in node + 7 agents (0 of 2 executors busy)'. The main table lists the nodes:

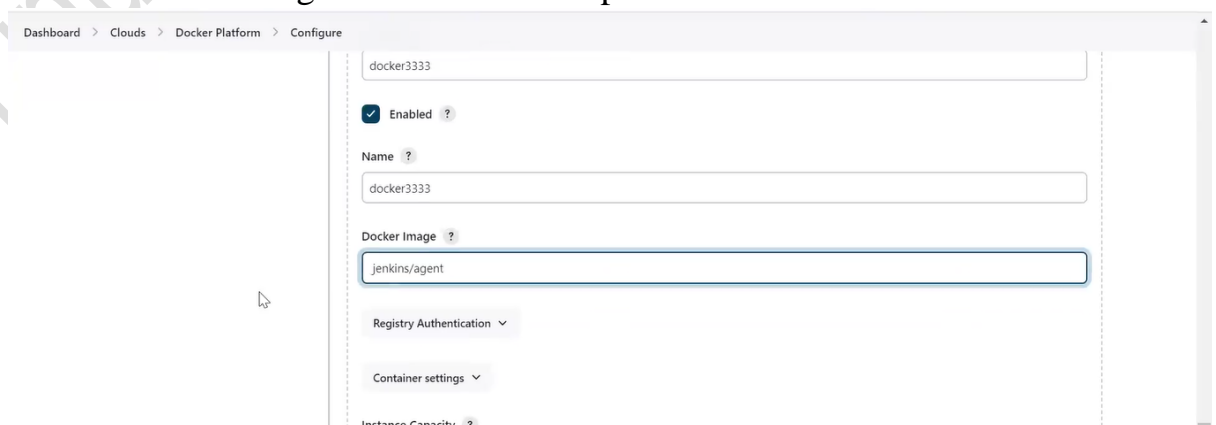
Node Name	Platform	Status	Space	Free Space	Used Space	Free Space
Built-In Node	Linux (amd64)	In sync	5.65 GiB	1.90 GiB	0 B	0 B
Docker_agent_1_maven_name-00005syh8itzi		N/A	N/A	N/A	N/A	N/A
Docker_agent_1_maven_name-00005t36o09e3		N/A	N/A	N/A	N/A	N/A
Docker_agent_1_maven_name-00005t7ruivfa		N/A	N/A	N/A	N/A	N/A
Docker_agent_1_maven_name-00005tcd6oxtg		N/A	N/A	N/A	N/A	N/A
Docker_agent_1_maven_name-00005tylete0		N/A	N/A	N/A	N/A	N/A
Docker_agent_1_maven_name-00005tljzx2dg		N/A	N/A	N/A	N/A	N/A

- It is failing to launch the slave because the image we are using have an older version of the java so the version conflict is arising here.
- So to overcome this problem we have to use another image.
- This image is suggested to be used by the Jenkins itself.



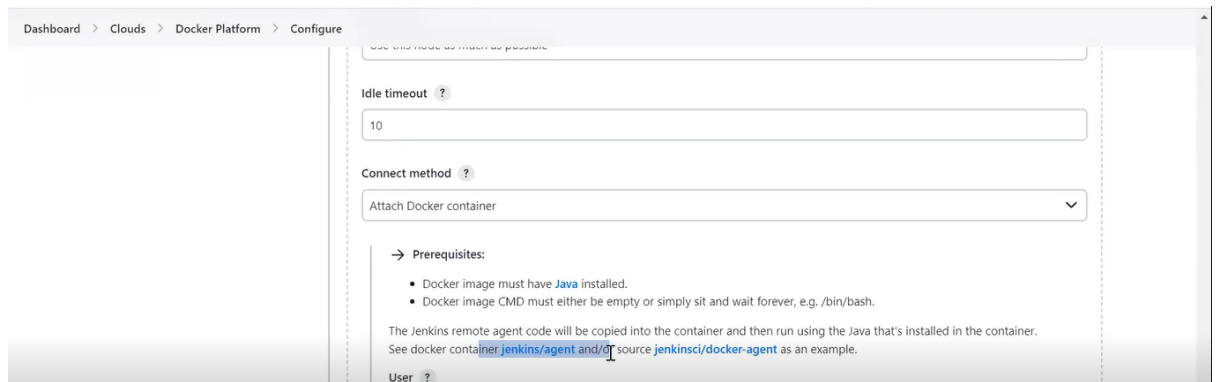
The screenshot shows the Docker Hub page for the 'jenkins/agent' image. The page includes the Docker Hub logo, a search bar, and navigation links. The main content area shows the 'jenkins/agent' image with a pull count of 10M+. Below this, there is a section titled 'Jenkins Agent Docker image' with a description: 'This is a base image for Docker, which includes JDK and the Jenkins agent executable (agent.jar). This executable is an instance of the Jenkins Remoting library. JDK version depends on the image and the...'. To the right, there is a 'Docker Pull Command' section with the command 'docker pull jenkins/agent' and a 'Copy' button.

- Add this image in the docker template.

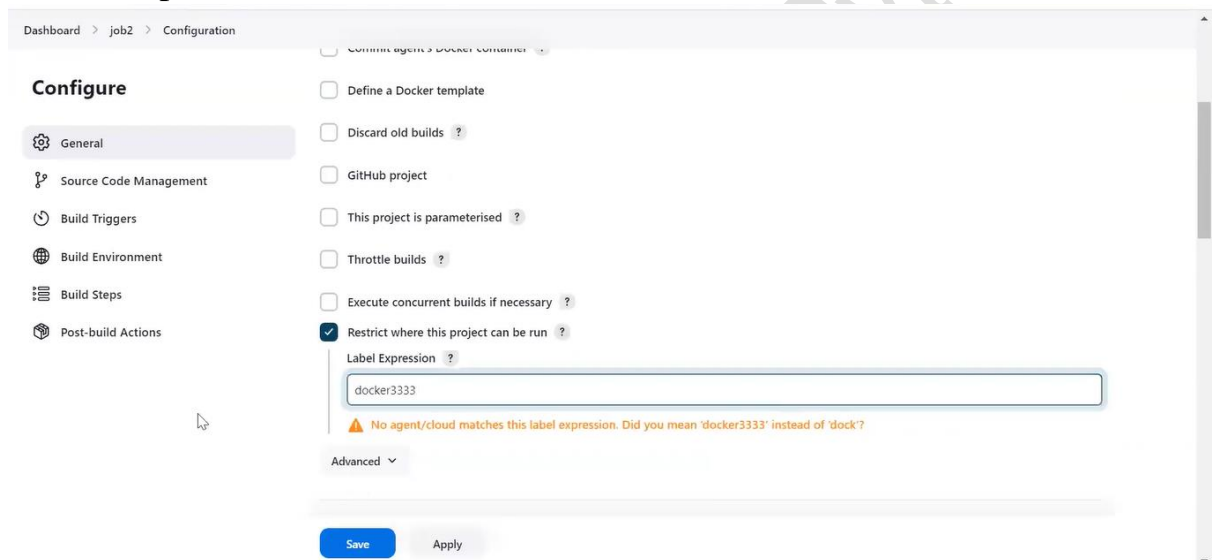


The screenshot shows the Jenkins 'Configure' page for the 'Docker Platform'. The page has a breadcrumb trail: 'Dashboard > Clouds > Docker Platform > Configure'. The main configuration area includes a text input field for 'docker3333', a checkbox for 'Enabled' which is checked, a text input field for 'Name' with the value 'docker3333', a text input field for 'Docker Image' with the value 'jenkins/agent', a dropdown menu for 'Registry Authentication', a dropdown menu for 'Container settings', and a text input field for 'Instance Capacity'.

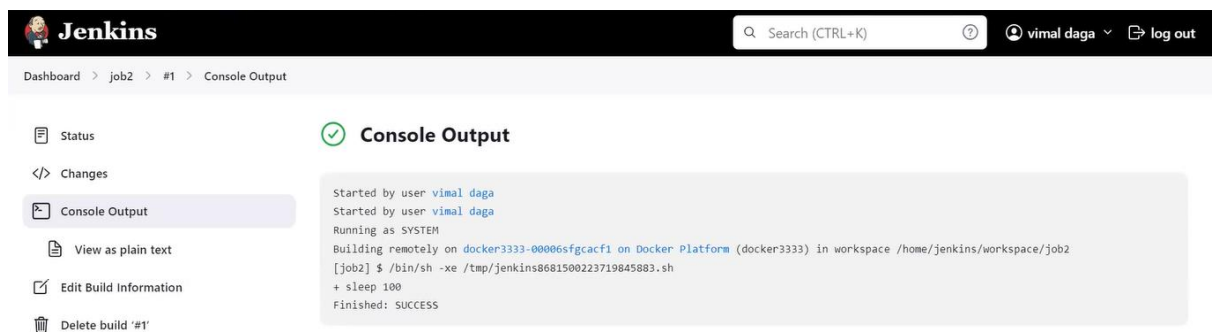
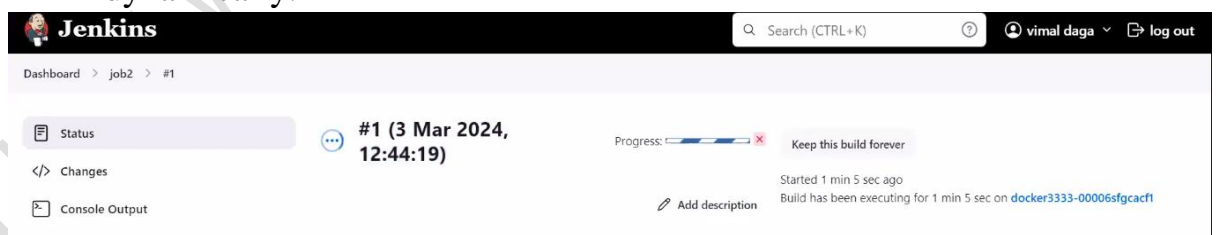
➤ Select the connect method.



➤ Configure the job also and restrict it to run on the new agent template.



➤ Now as we build or run this job we can see that job is being assigned to the slave node and the slave is provisioned dynamically.





- The life of the slave is the life of the job, as soon as the job completes the slave will be deleted automatically.

Linux World Informatics Pvt Ltd