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.

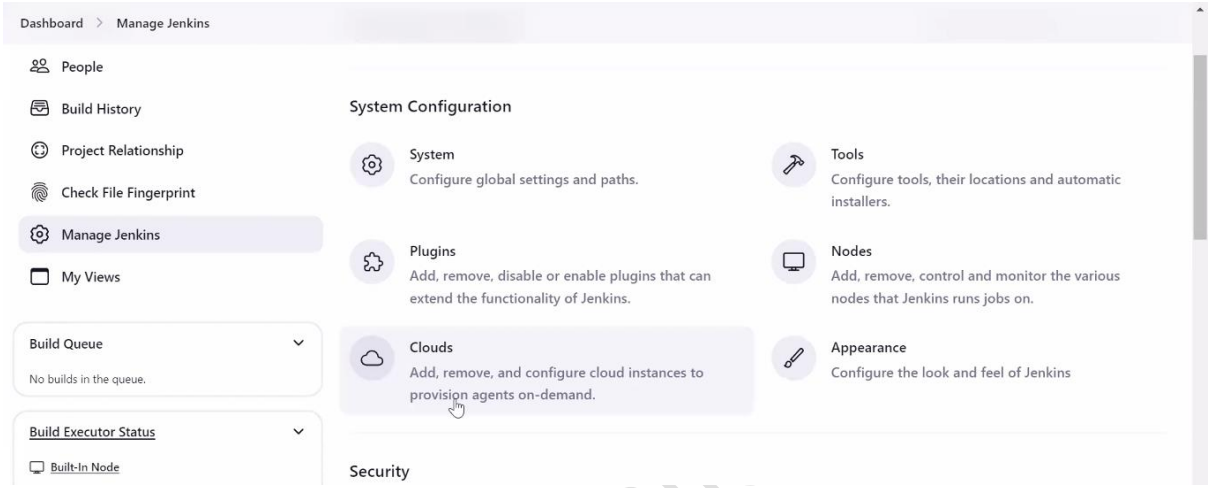
- By implementing dynamic provisioning, Jenkins clusters become more efficient, as resources are provisioned on-demand, reducing idle time and unnecessary resource consumption. This approach improves scalability and cost-effectiveness, making the cluster more responsive to varying workloads.

- For the Dynamic provisioning,

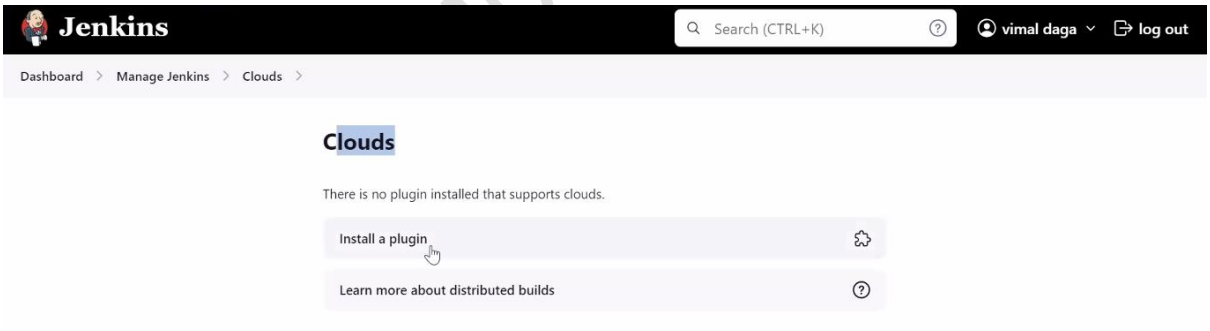
    - We need a platform where the master node will launch the slave automatically whenever any job come up to the master
    - Platform can be multi-cloud, Kubernetes or the docker platform.

- We have several plugins which will help the master to contact with the platforms and launch the slave node there.

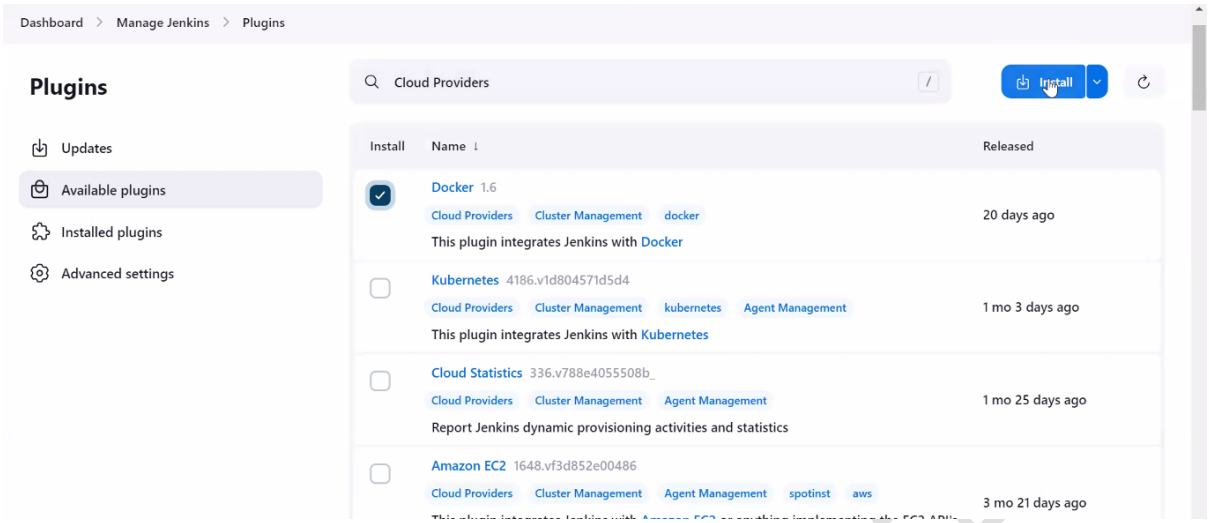
- To setup the dynamic provisioning architecture in the Jenkins follow the steps.
  - Go to the manage Jenkins and click on the cloud.



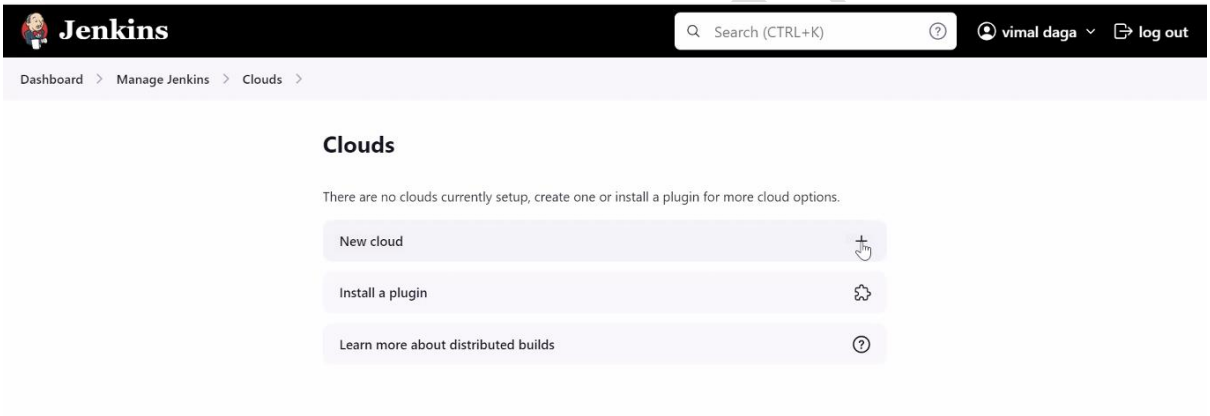
  - Here we have to install the plugin which will help the Jenkins to connect with the platform where the slave nodes will be launched.



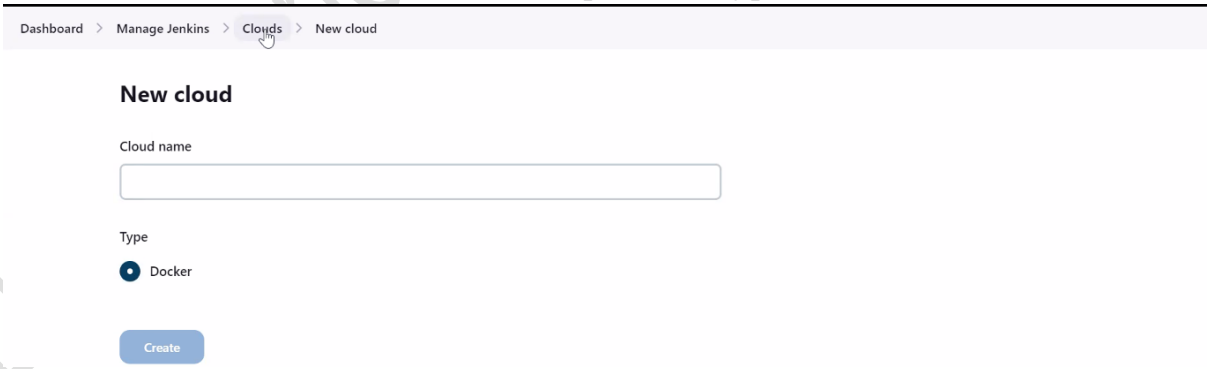
  - We want to use docker as the platform so we will install the docker plugin for it.



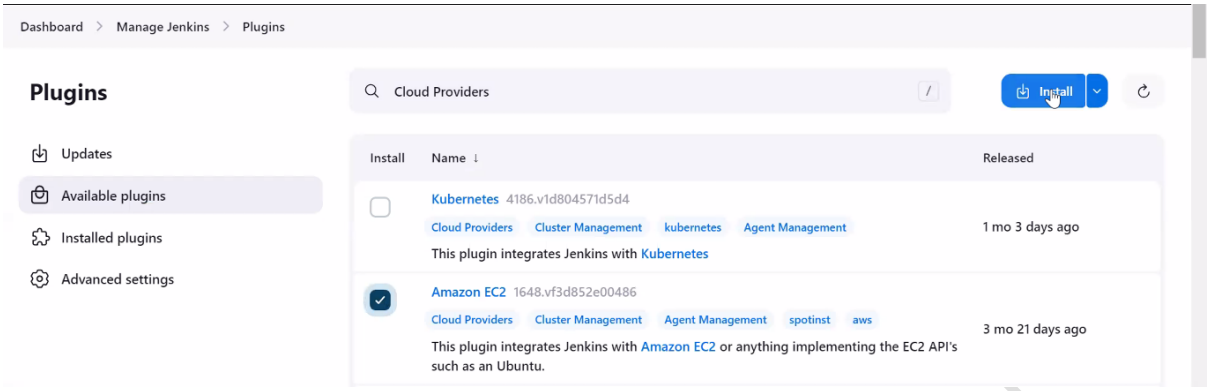
- As soon as we install the plugin, we can see the new cloud option in the manage Jenkins section.

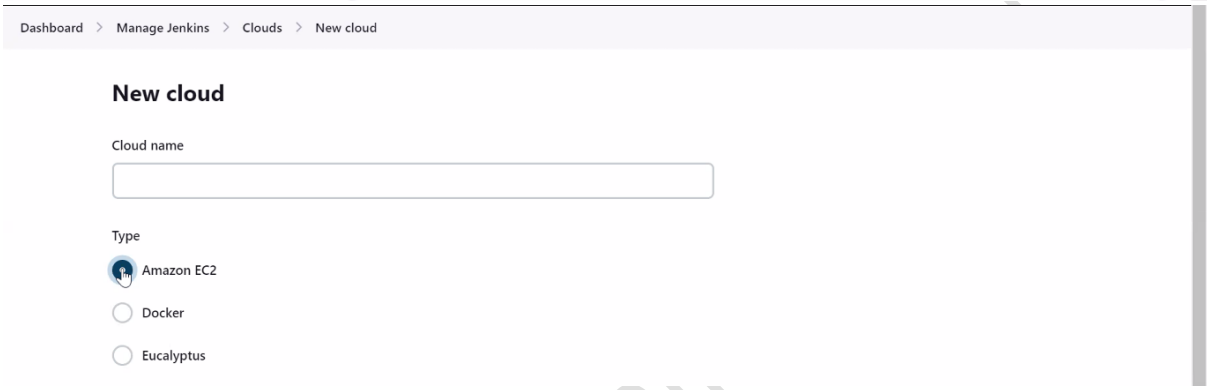


- In the new cloud we can see the platform type as docker.

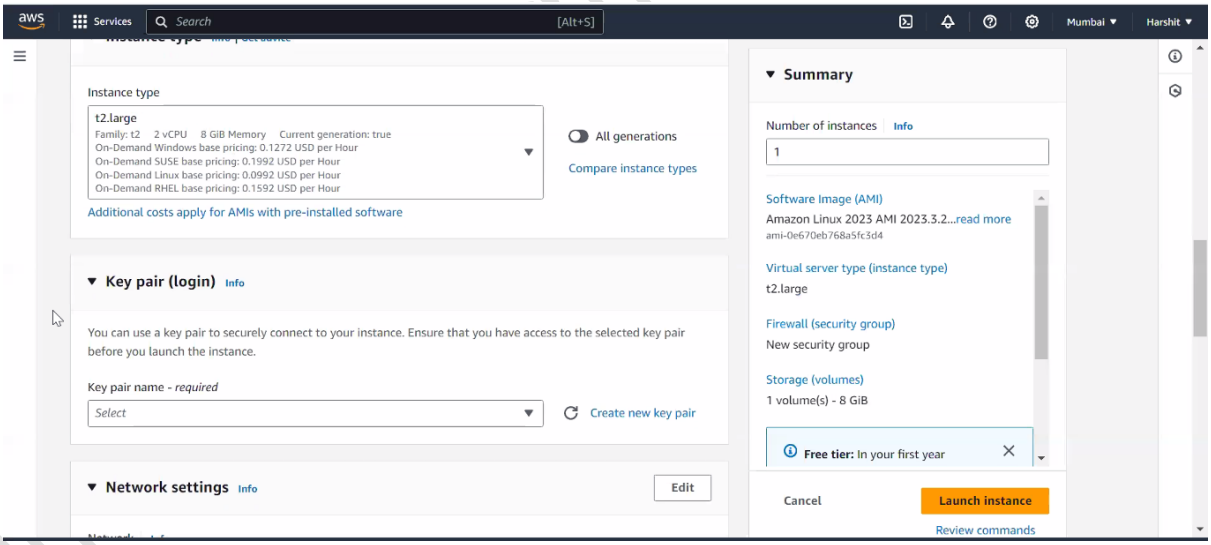


- Similarly we can install amazon ec2 plugin if we want to use the ec2 as the platform type.

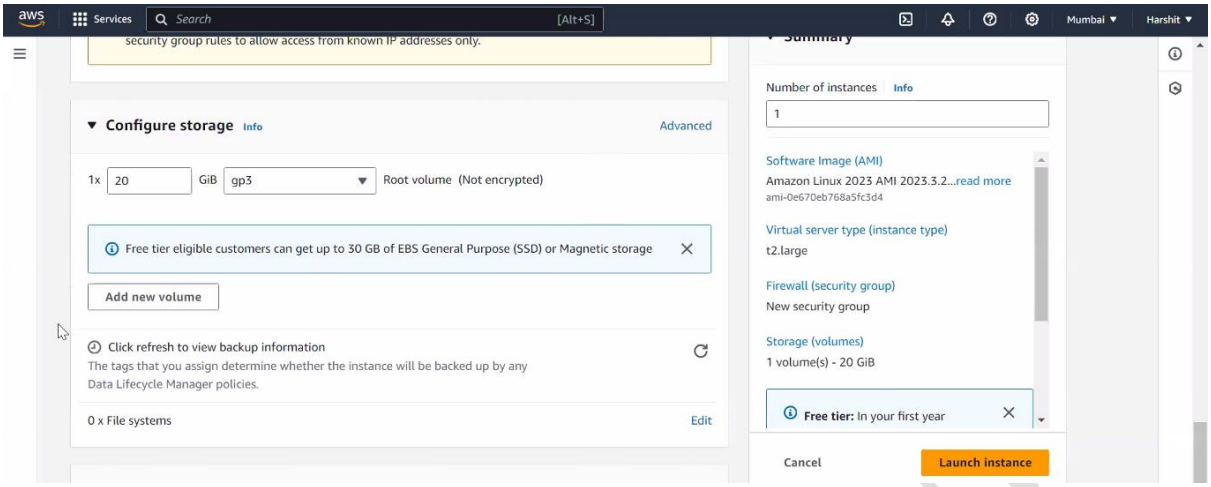


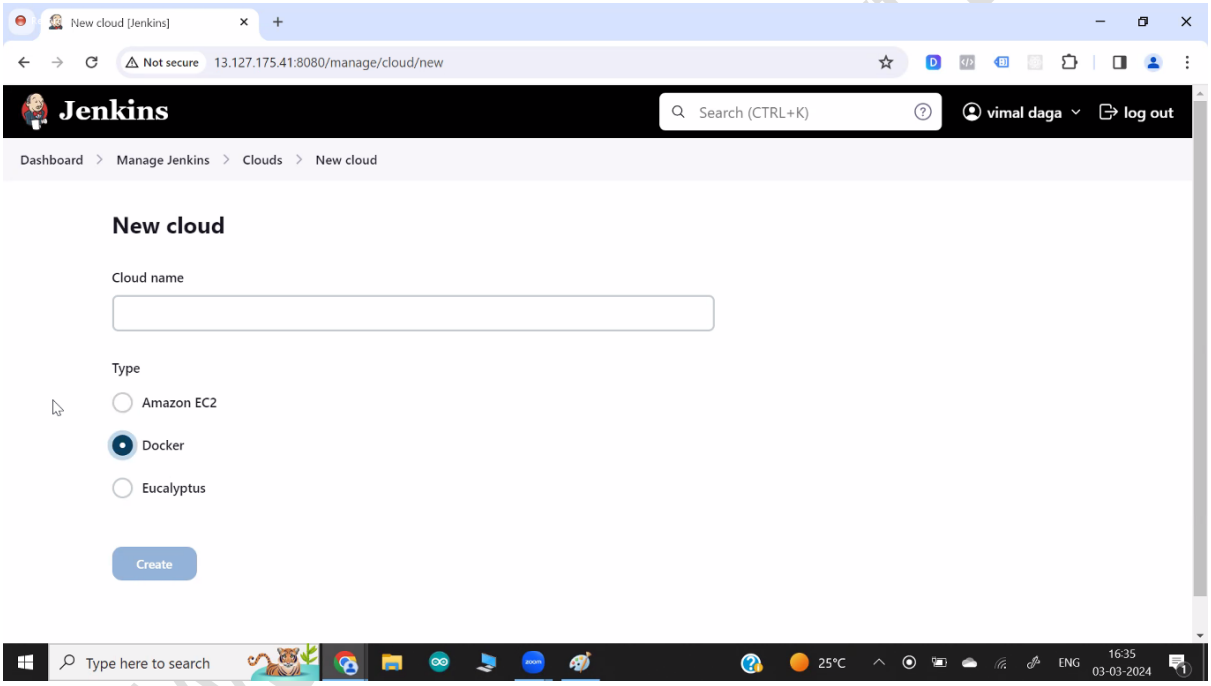


- Now we will launch the docker host to be used as the platform.

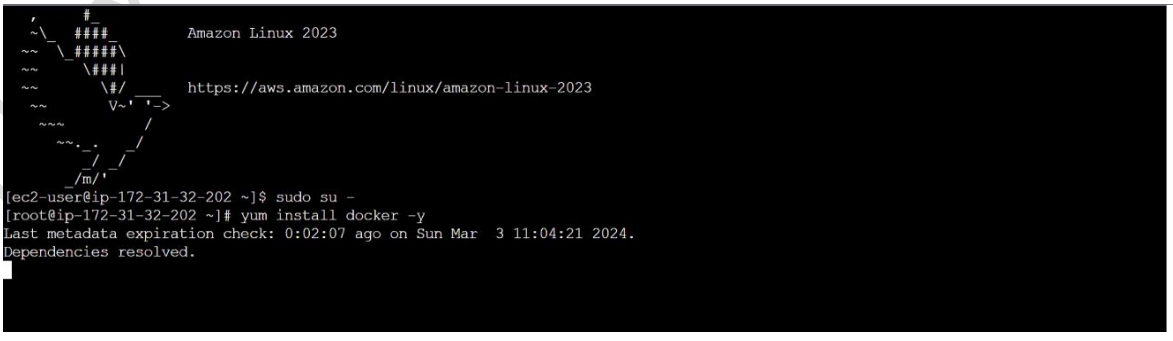


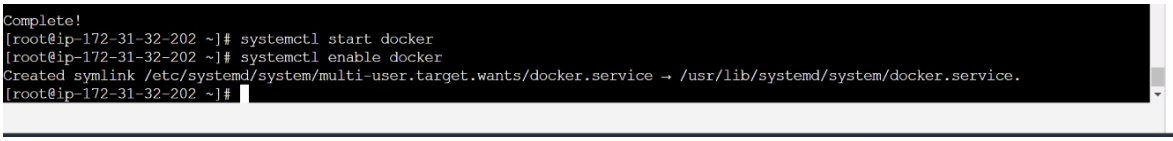
- Configure the instance type for the docker host.



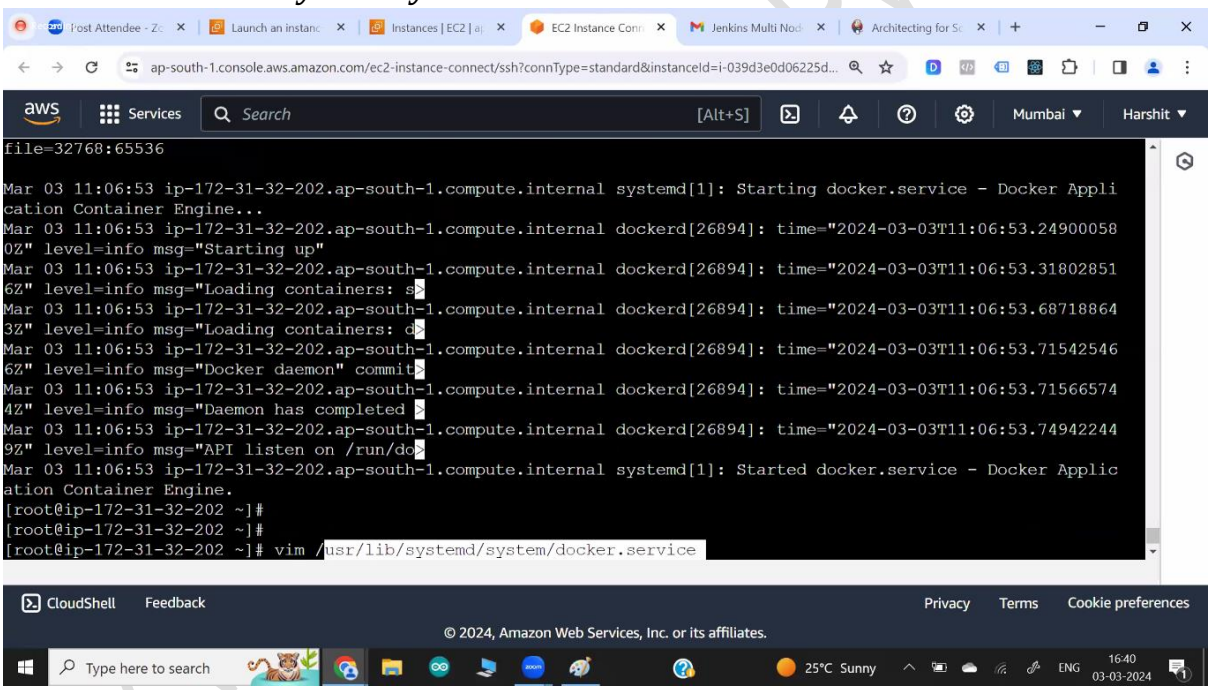


- In the docker host, install and enable the docker services.

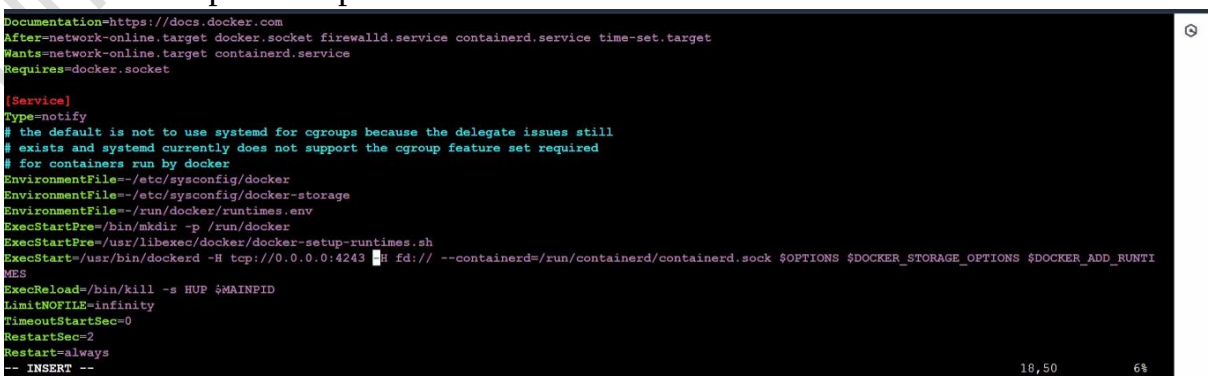




- Now this instance will be our docker host or the platform, only thing we have to do extra here is make this host reachable to the master node.
- Master node will only contact to the docker host.
- By default docker only supports local functionality means Docker can launch the containers by taking inputs 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/system/system/docker.service

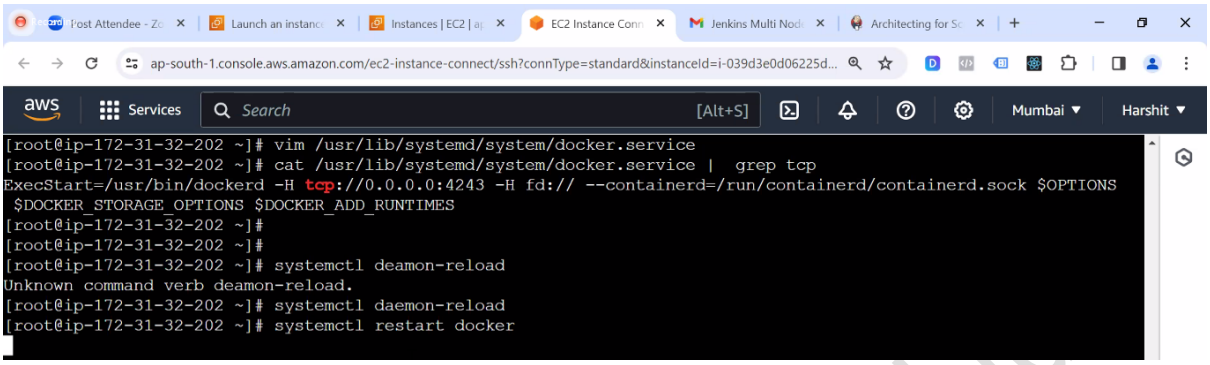


- fd: // — means it will support local functionality only. To support TCP we need to add the below line in this section, if the request come up on the port number 4243 then docker will listen to it.



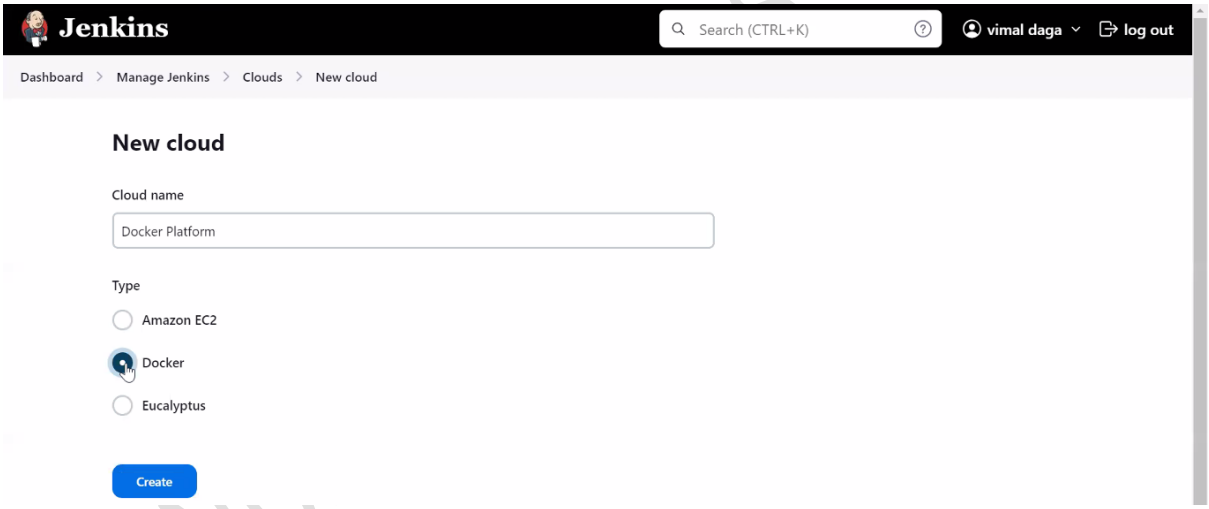
- This process is also known as socket binding.

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

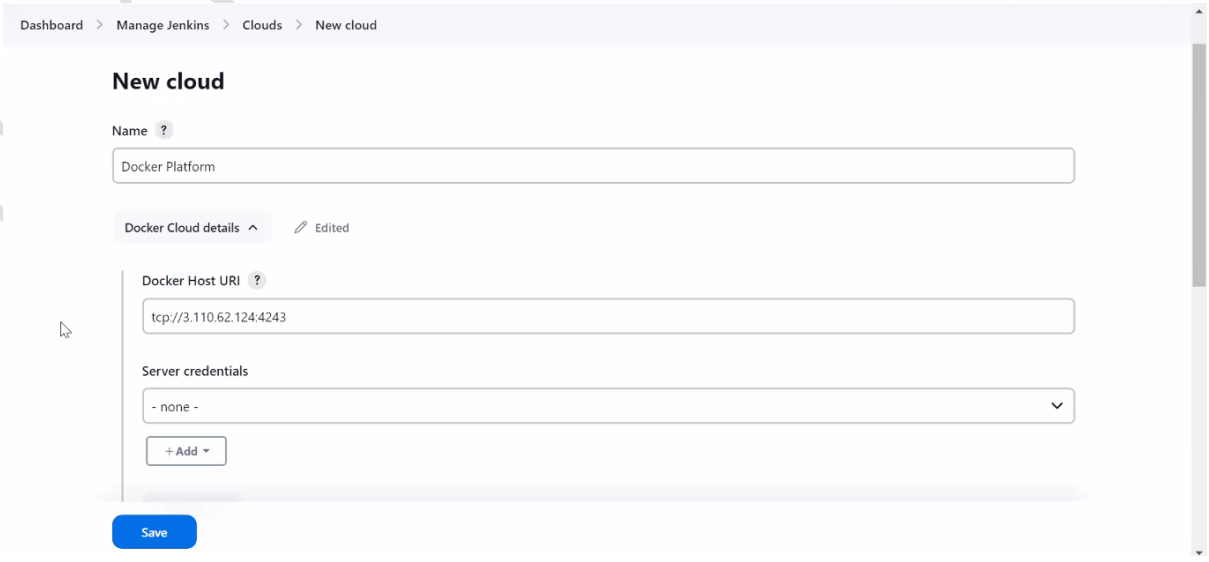


- Now this docker will take the instruction from the outside world.

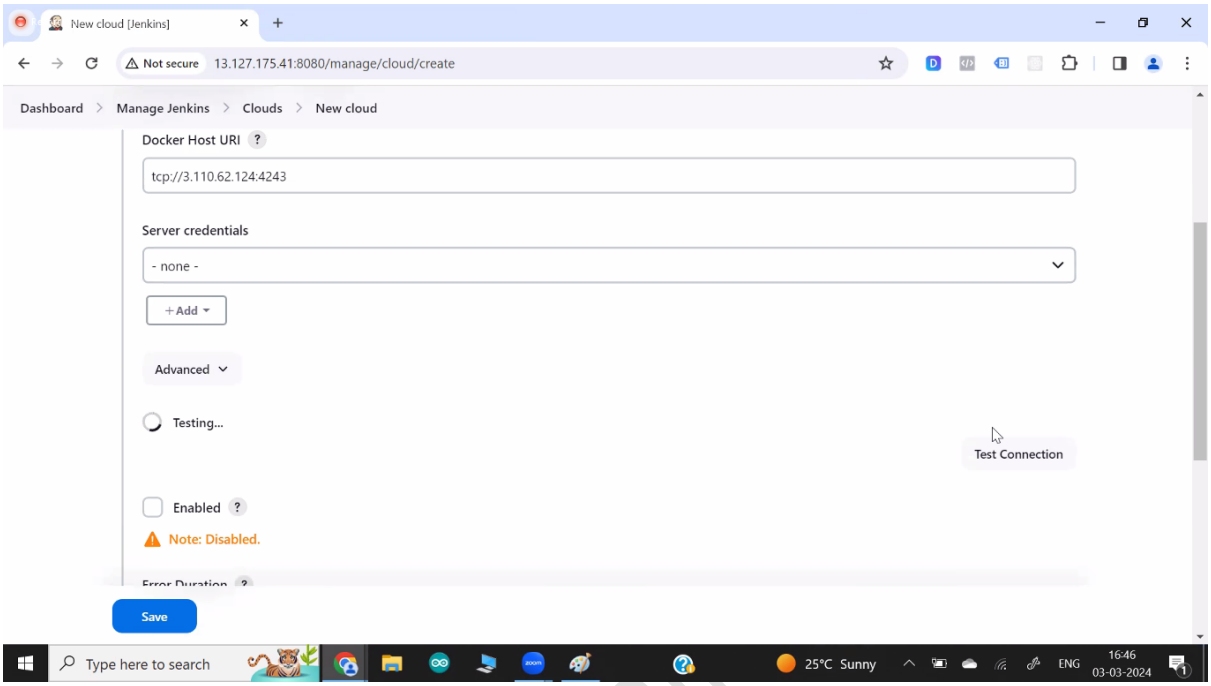
- We have to connect the Jenkins master to the docker host now. For this go to the new cloud again and select the docker as the type.



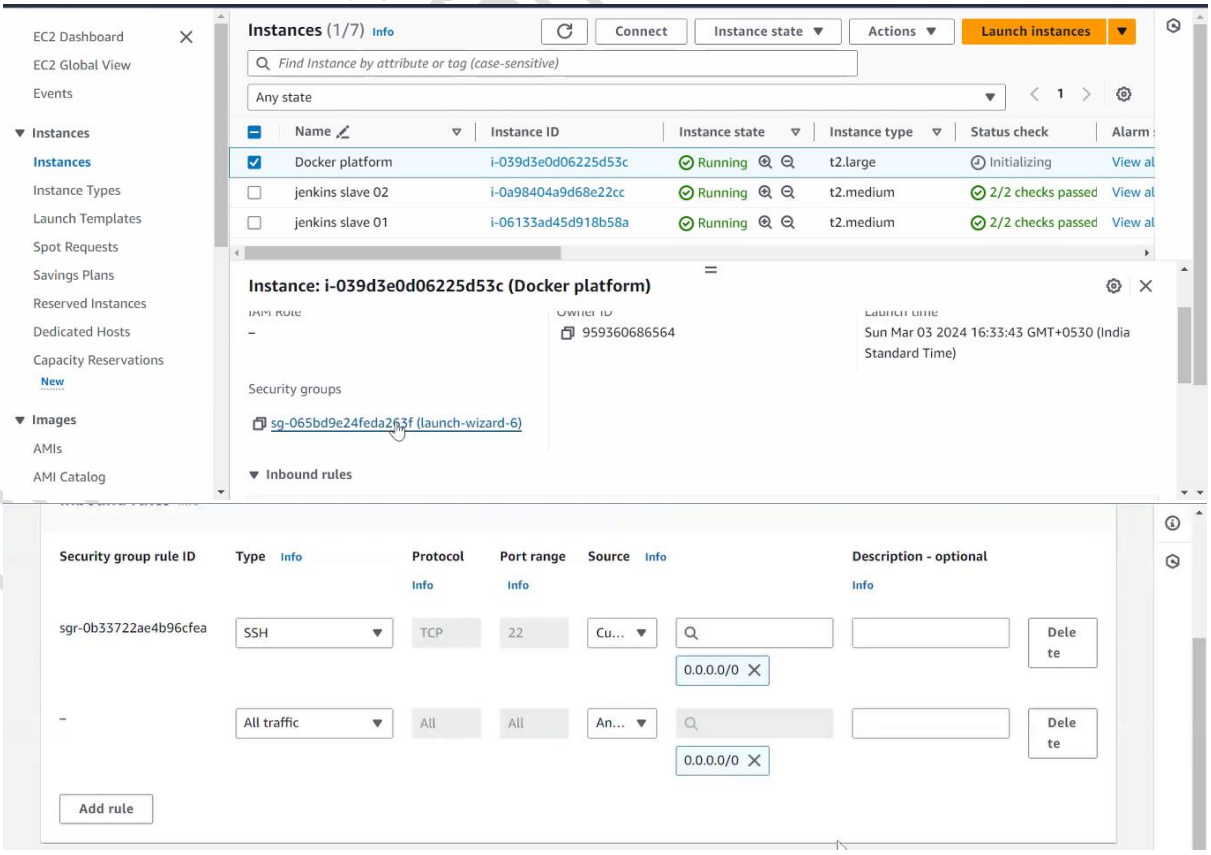
- Now, enter the docker-machine details that we set up in the container instance.



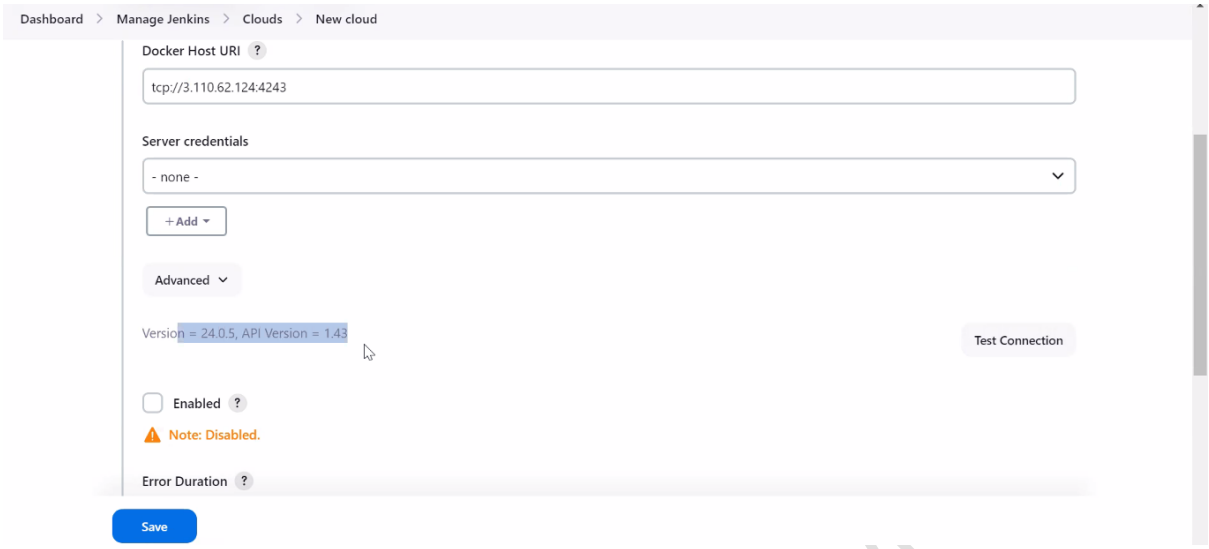
- Click on the test connection to test the connectivity to the docker host.



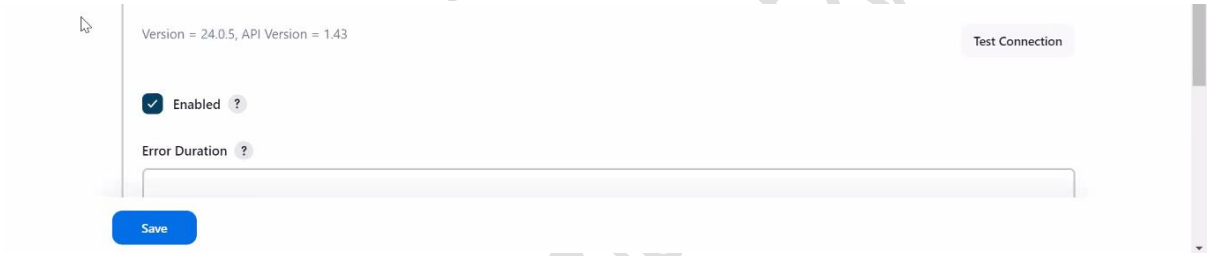
- This test might fail due to firewall issue of the aws instance. For this we have to edit the inbound rule of the aws instance.
- Go to the security group of that instance and edit the inbound rule.



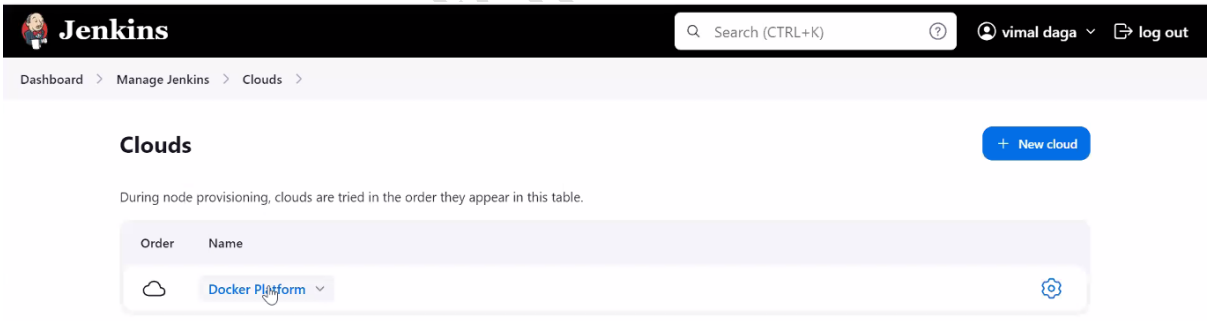
- Now again test the connection and we can see that the master Jenkins can successfully connect to the docker host.



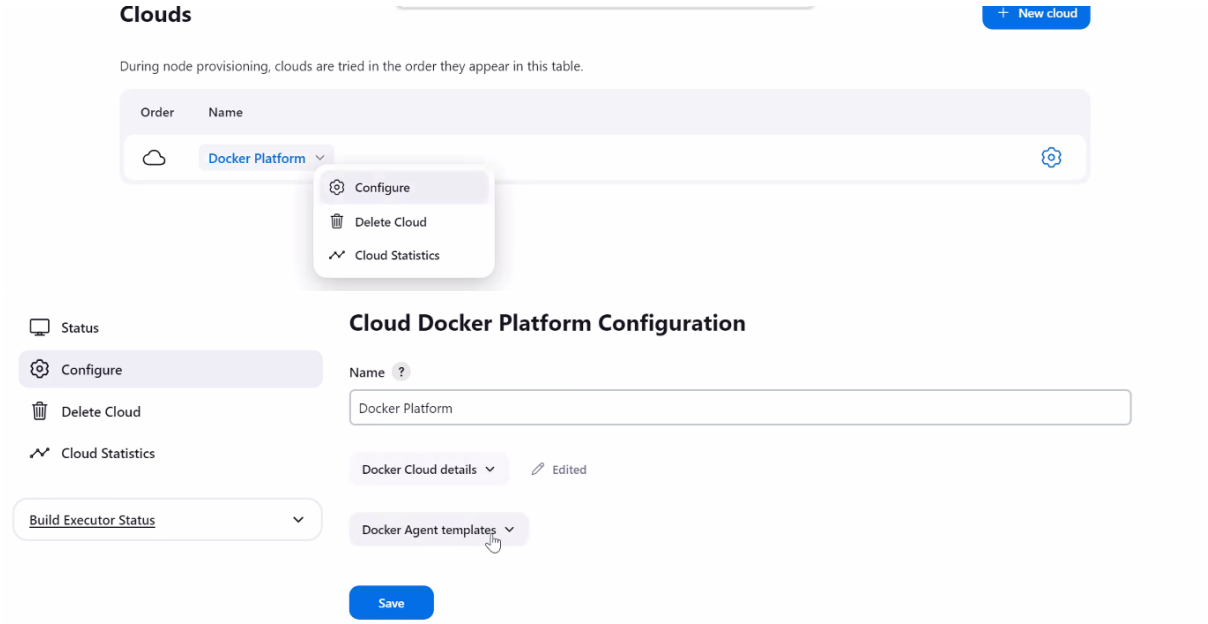
- We have to enable this platform now.



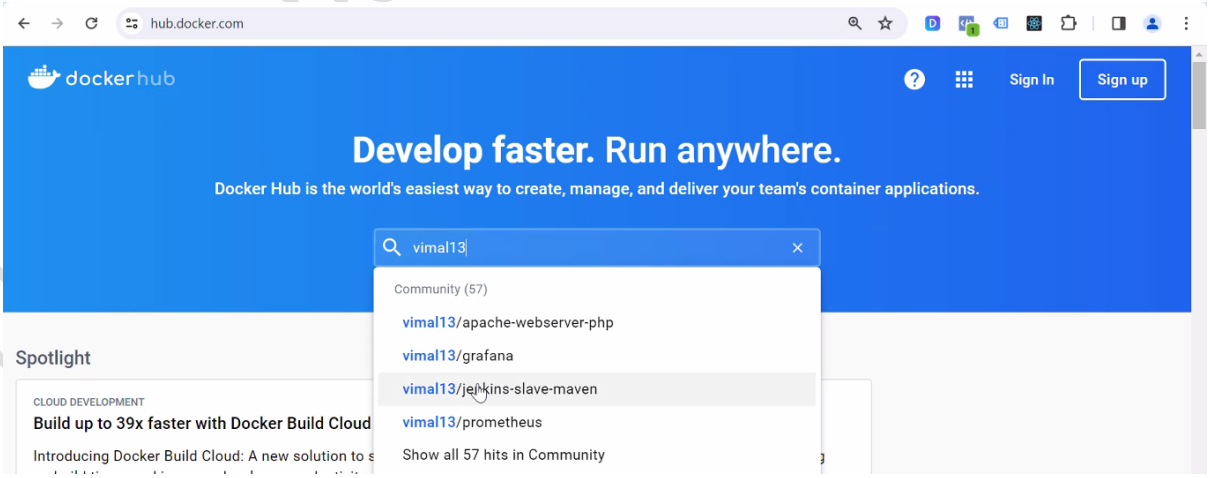
- So finally the Jenkins master is successfully connected with the docker host platform.



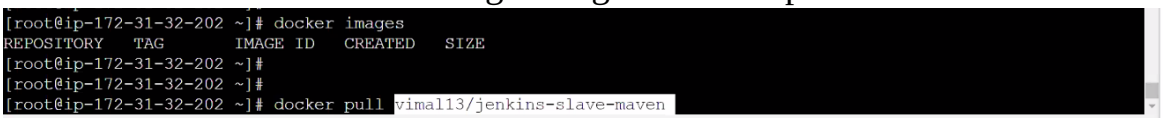
- Now we have to set the template for this platform.



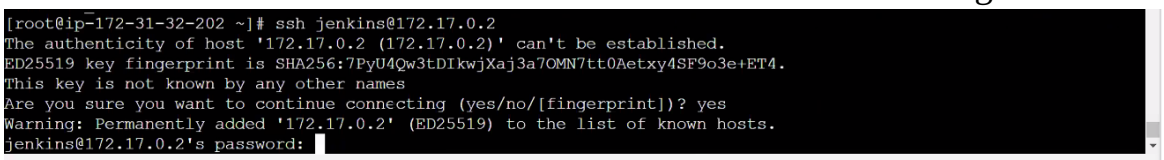
- Template means, whenever any job come to the master, it will launch the slave.
- Now there are two prerequisites for the slave, every job has its requirements so that tool kit need to be present in the slave, Slave should have ssh enabled and there should be java available in it.
- For this we have 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.



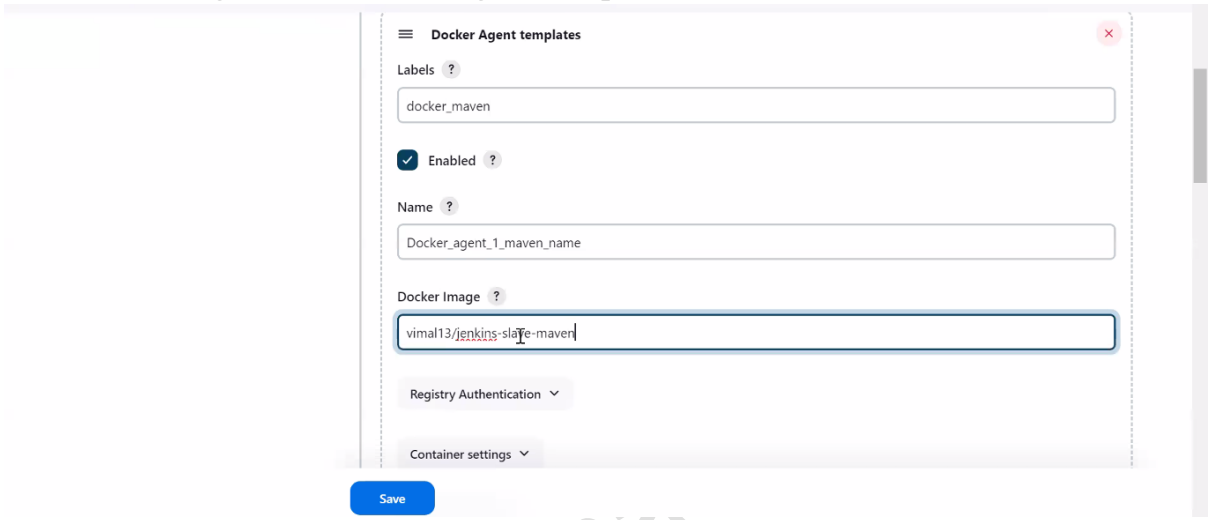
- Pull or download this image using the docker pull command.



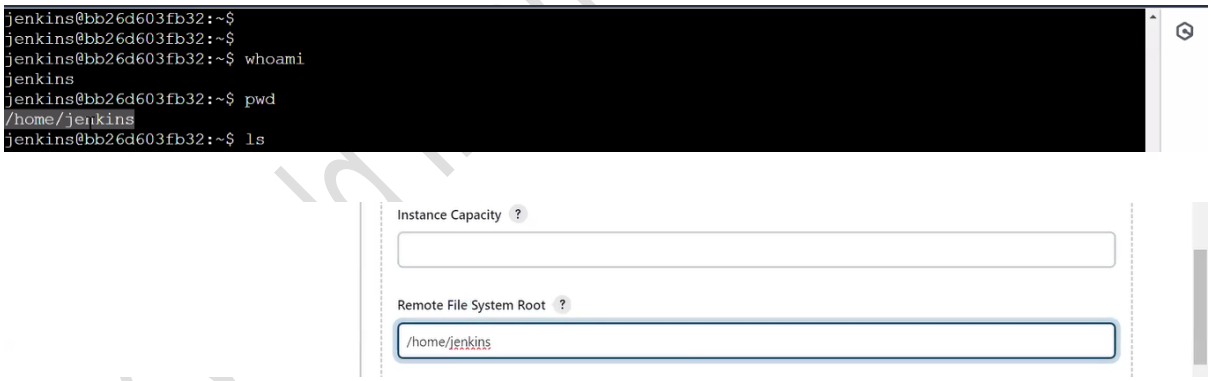
- We can check whether the ssh is enabled or not in this image.



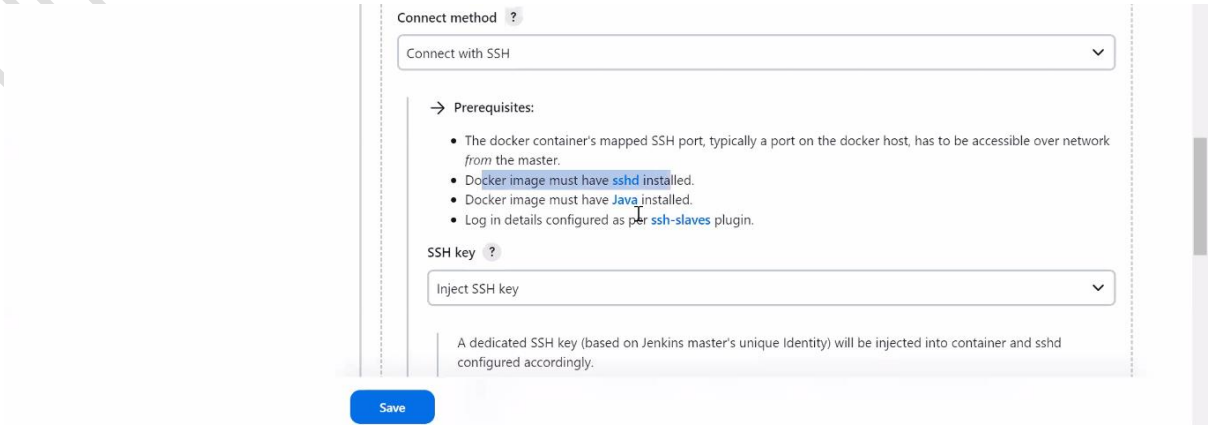
- Jenkins should know which image is to be used so we have to give the image details in the agent template.



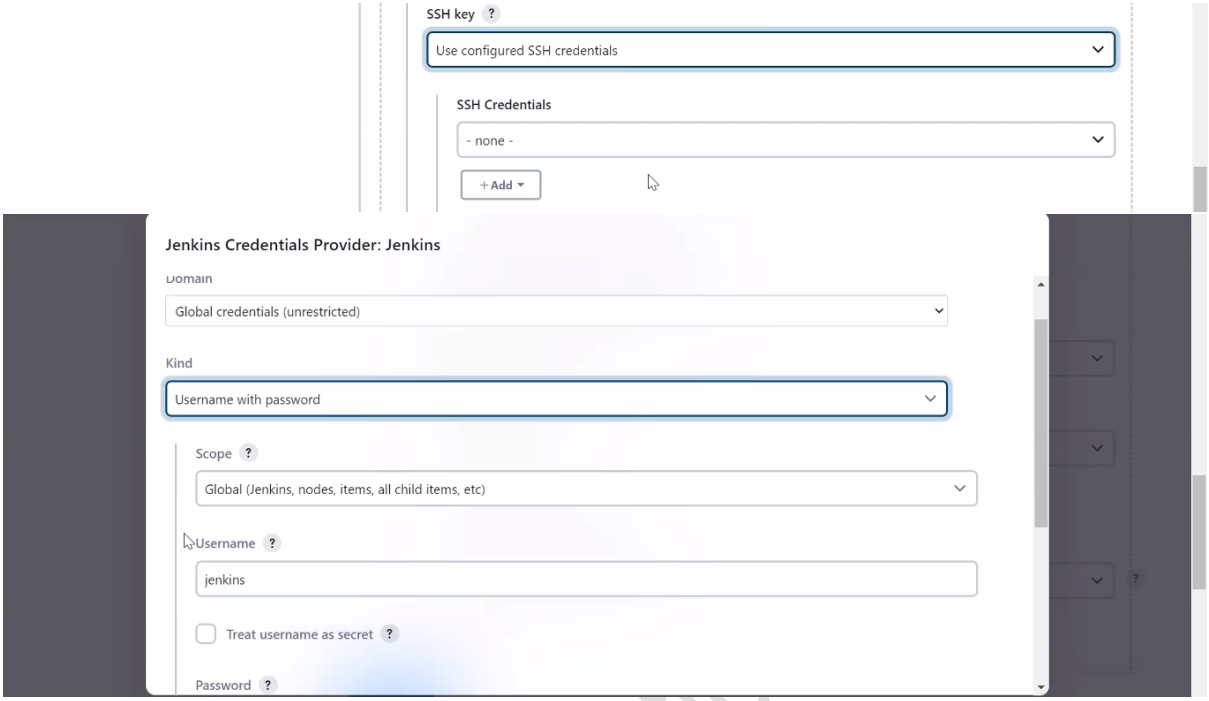
- We have to give the workspace details to the master Jenkins.



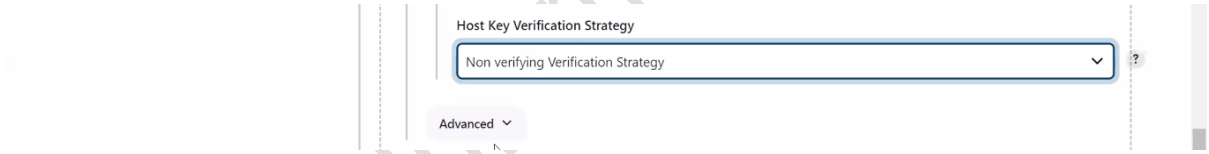
- Now we have to select the connect method as the ssh and for this the docker image must have ssh and java installed in it then only this connect method will work.



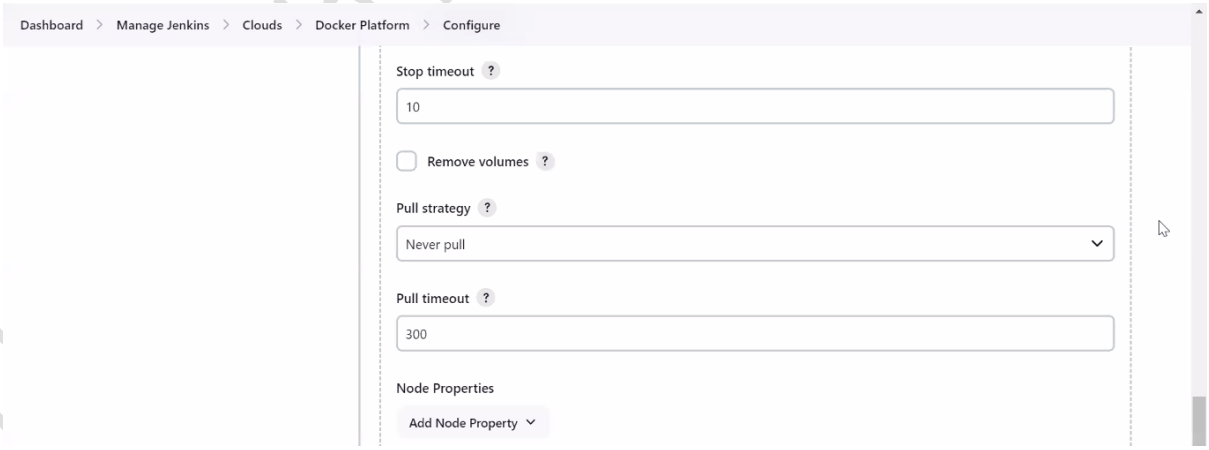
- Add the ssh credentials here.



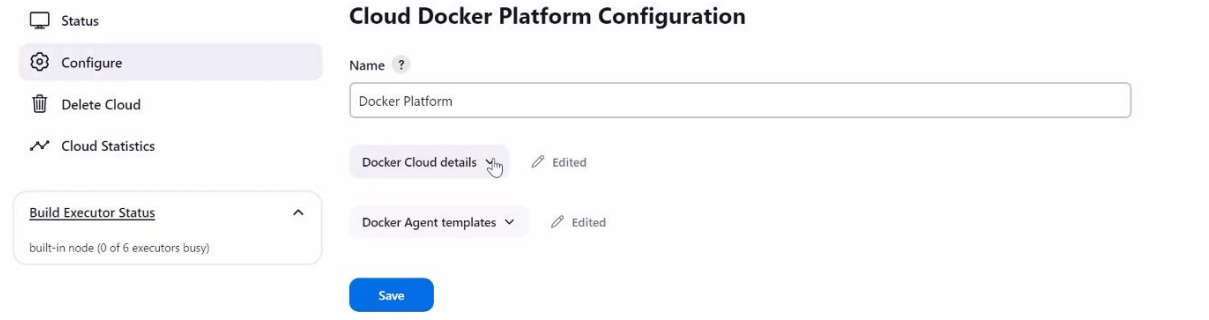
- Select the non-verifying strategy for the host key verification.



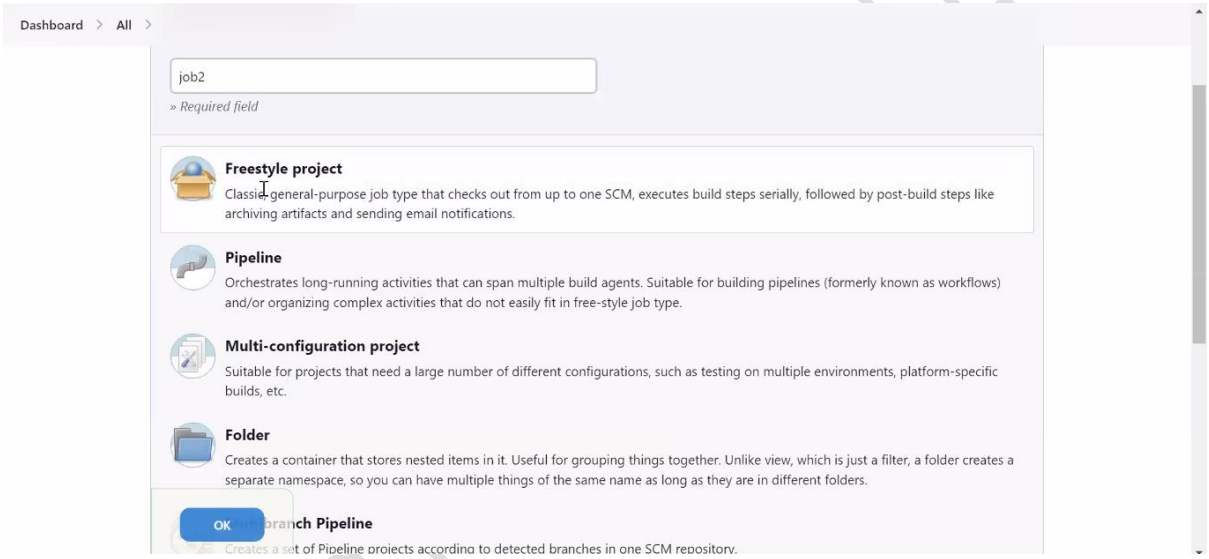
- Set the pull strategy for the image to never pull.



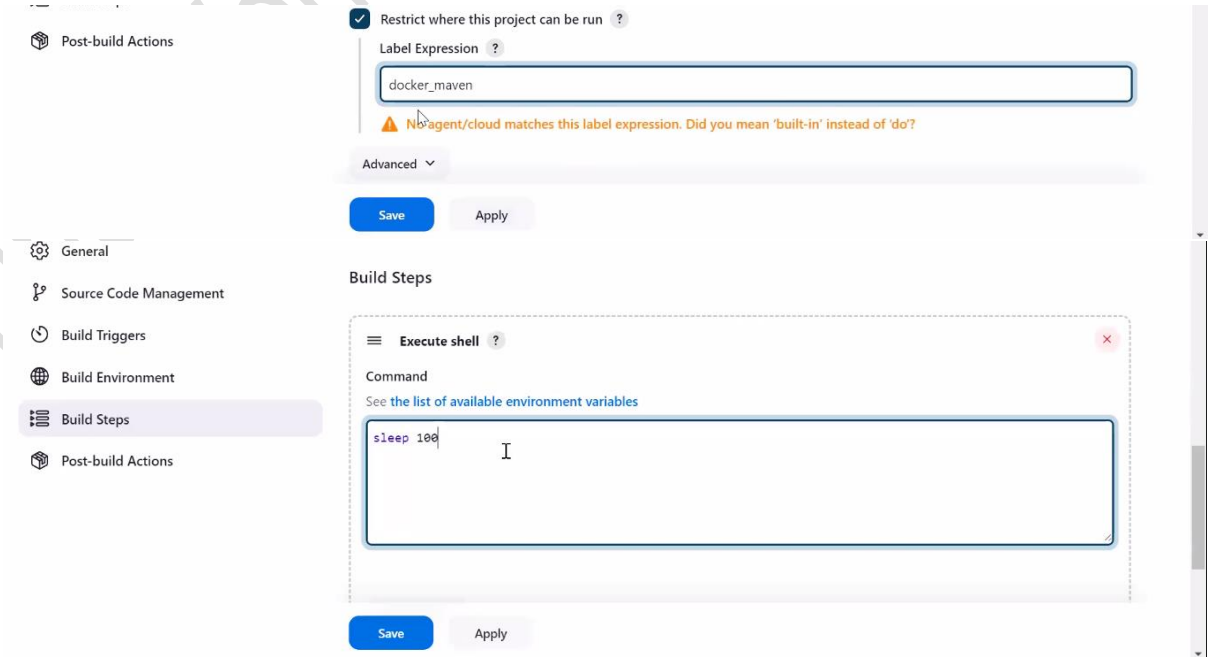
- Now we are done with the agent template part also.



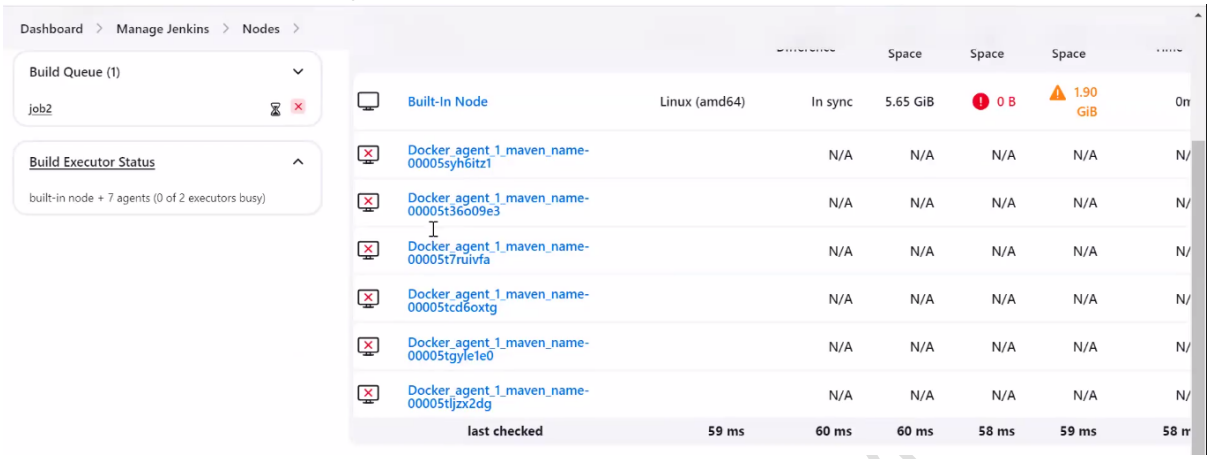
- Will create a job to test the dynamic provisioning of the slave nodes.



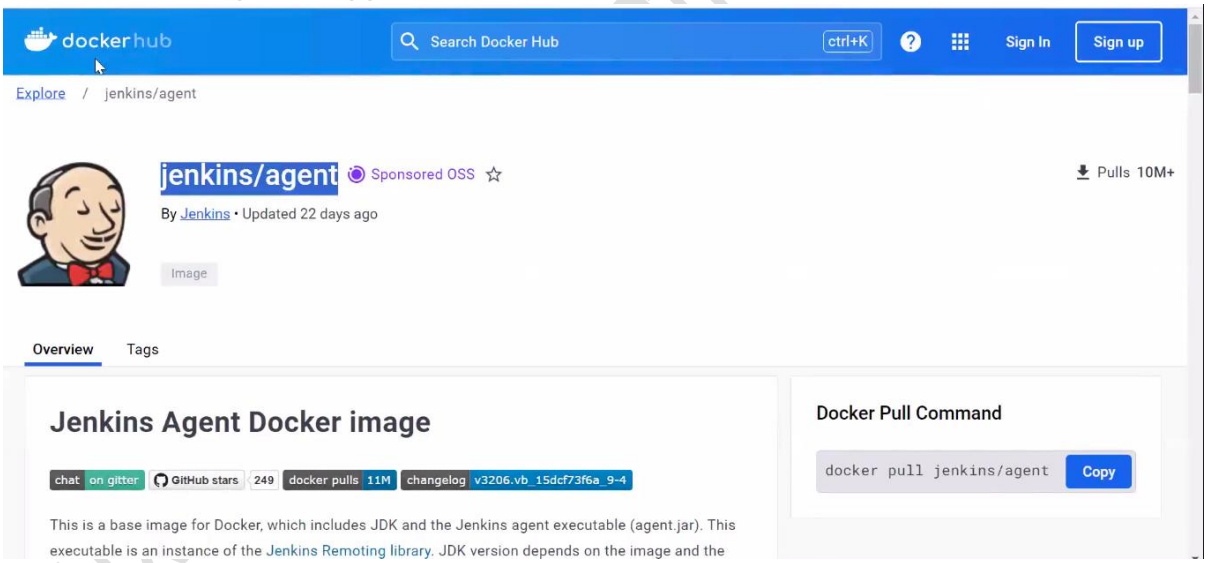
- Restrict this job to run on the platform or the cloud we have created.



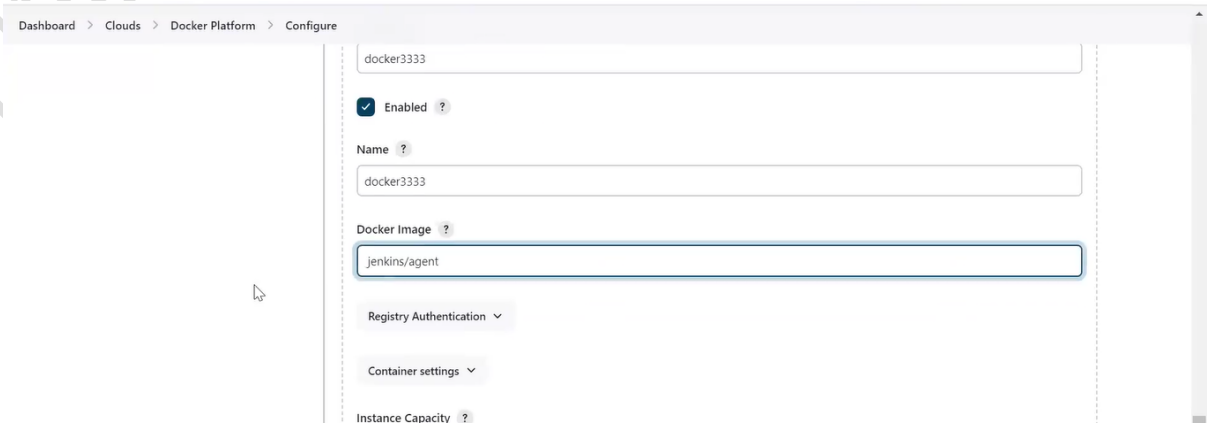
- Now as we run the job we can see that it will start launching some slaves to run the job.



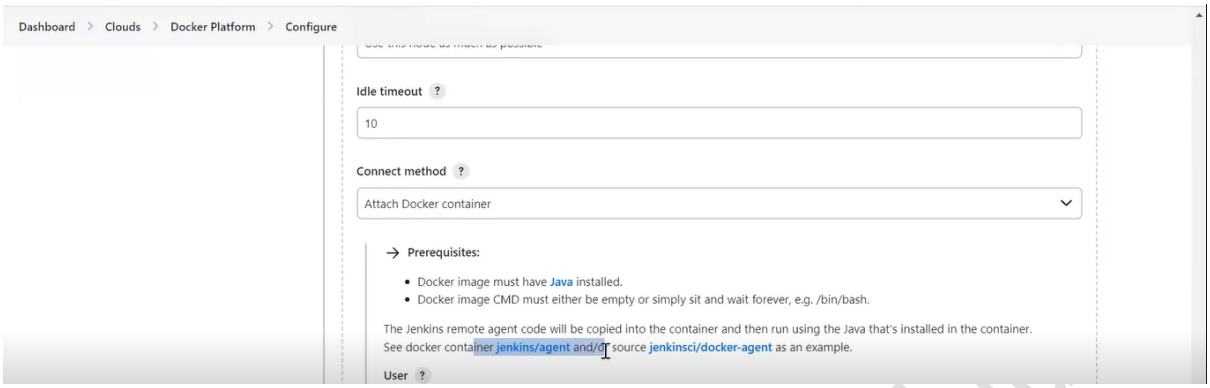
- It is failing to launch the slave because the image we are using have an older version of the java so the version conflict is arising here.
- So to overcome this problem we have to use another image.
- This image is suggested to be used by the Jenkins itself.



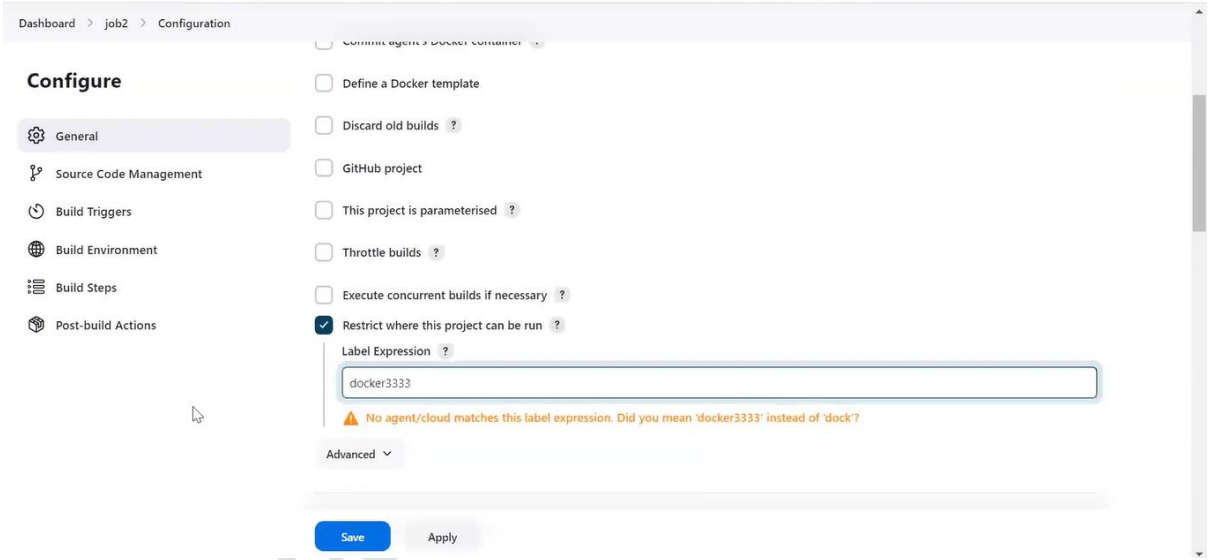
- Add this image in the docker template.



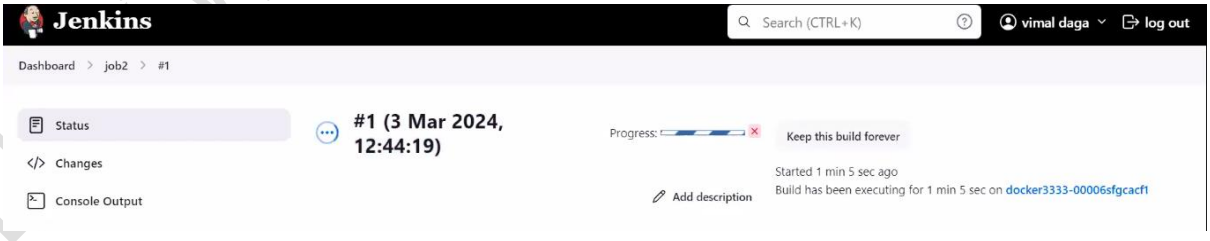
- Select the connect method.

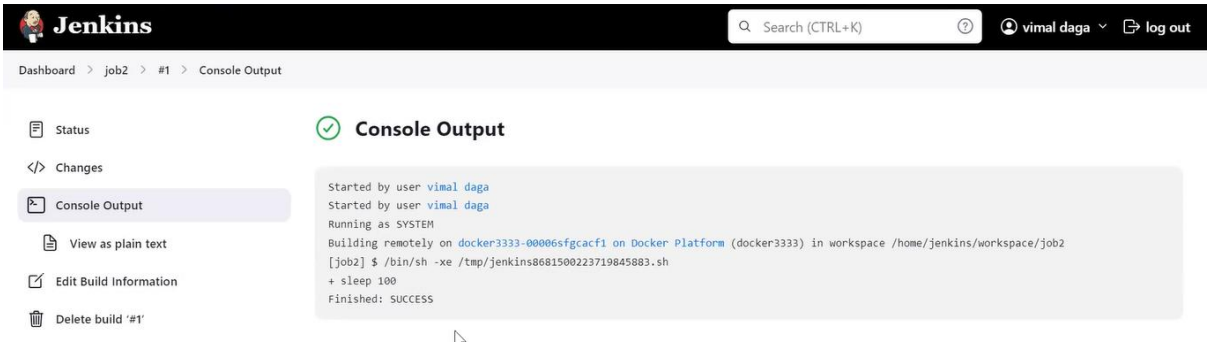


- Configure the job also and restrict it to run on the new agent template.



- 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.