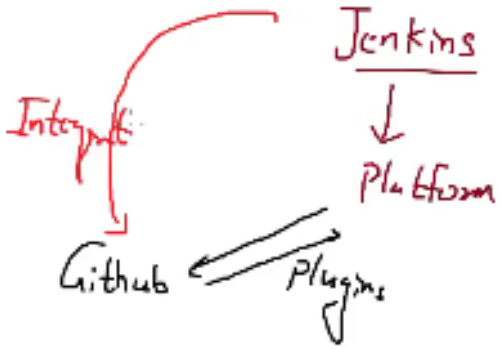
S

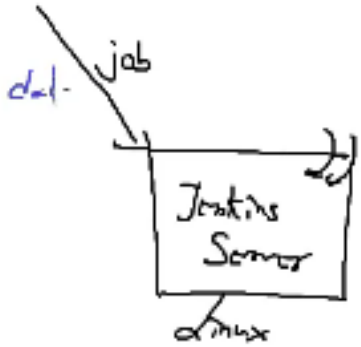
Mastering Jenkins Session No. 03

Summary 18-02-2024

- Jenkins is a tool we use for automation and Jenkins is integrated with any tool eg. Kubernetes, GitHub, Docker, and many more
- To integrate Jenkins with any tool for that we use plugins

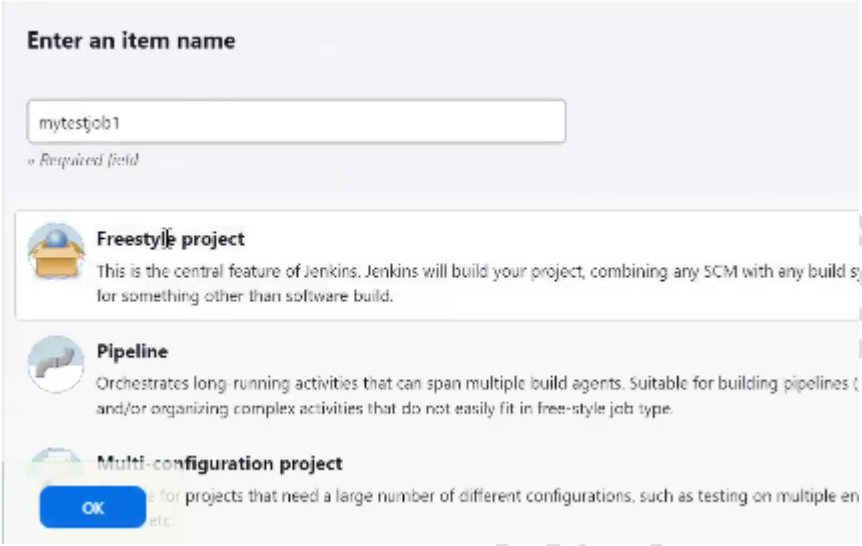


- We install Jenkins in the Linux system means Jenkins runs on the Linux system

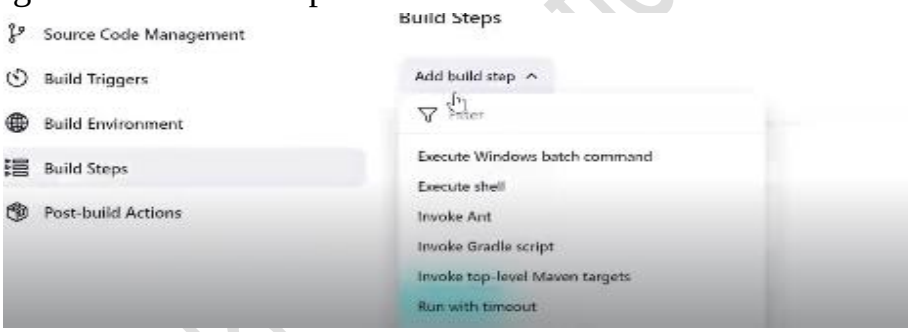


- For example you run the job in Jenkins but this job runs in the base system
- You run the docker command or git commands and your base OS system doesn't have these commands then the Jenkins job fail
- For that first we need to install all the required things then we run the job
- This is not the responsibility of Jenkins to provide these required commands
- If your base OS system has 2GB RAM and your Jenkins job needs 3GB RAM to run the case Jenkins job fail

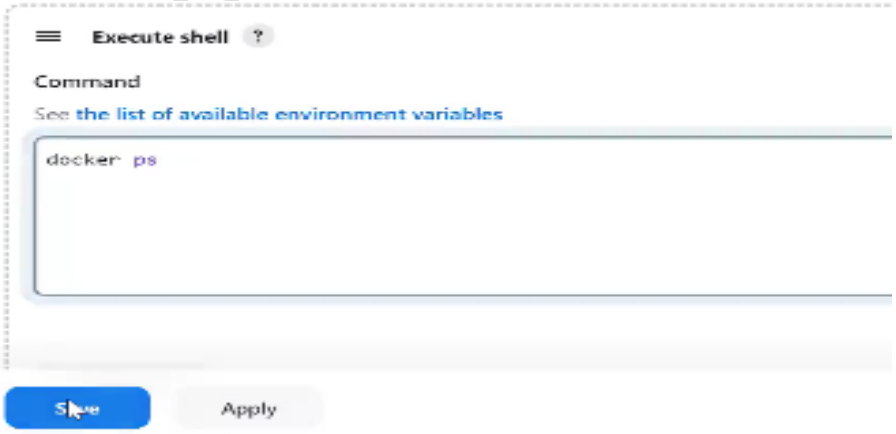
- For example we create a one job
- To create a job click on a new item then give the name of the job and select the freestyle project



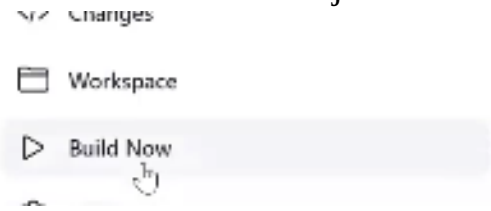
- After that go into the build step and select execute shell



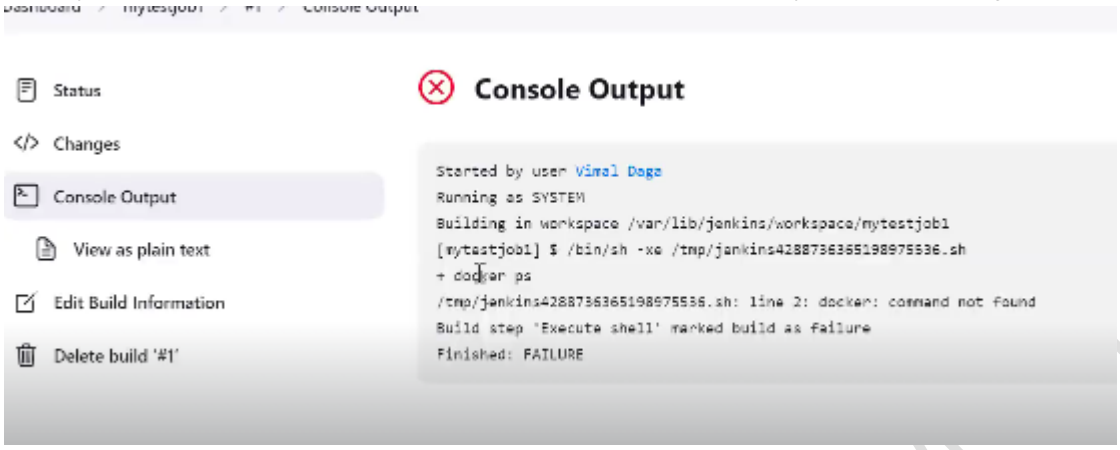
- Then type the docker ps command and save



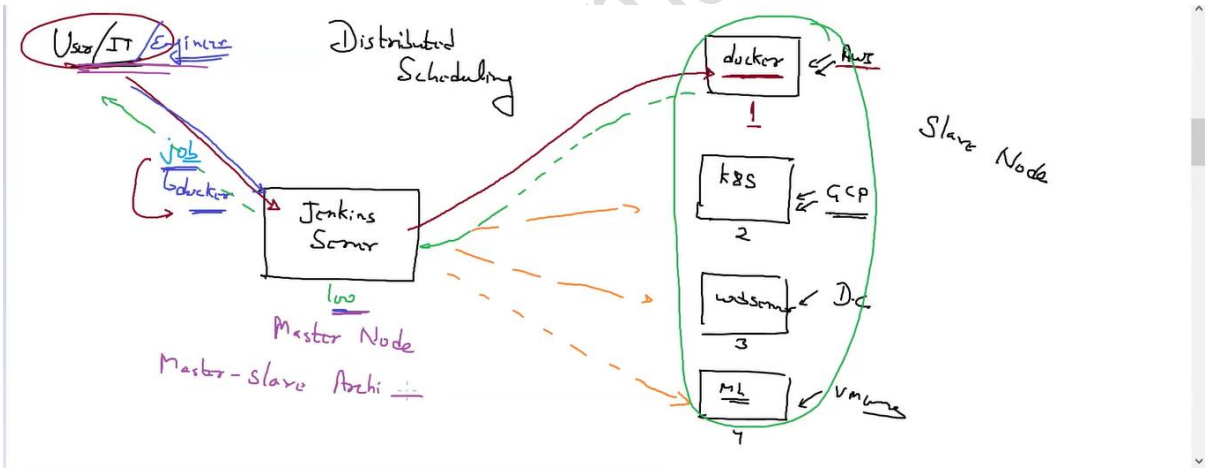
- After that click on Build Now to run the job



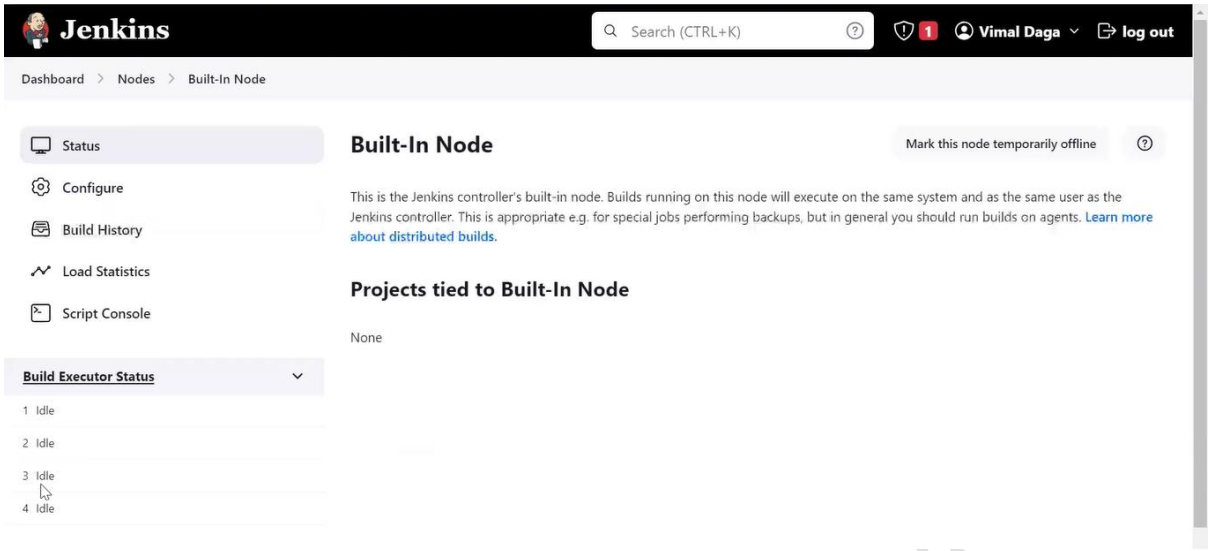
- As we know this job is run in the base OS system but right now your base OS system doesn't have docker commands that's why the Jenkins job fail



- As we see this error shows docker command is not available.
- This type of architecture is known as single-node architecture.
- We are going to set up an environment of different computers in which one computer will have the Jenkins server in it and this server will schedule the jobs across multiple nodes in a distributed computing system. This is known as distributed scheduling.

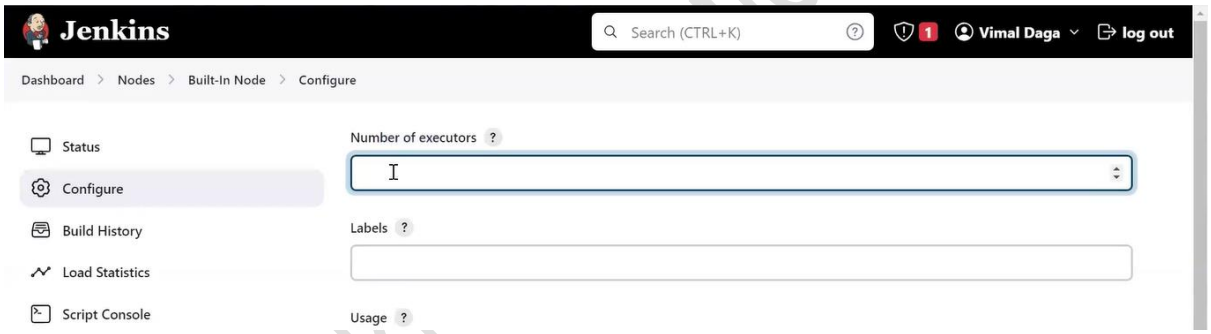


- Here every slave node has its respective tools and resources in it. Users have to contact only the Jenkins server for any job.
- The Jenkins node will automatically distribute the job among the slave nodes.
- To set up the master-slave architecture we need three instances with Jenkins installed in them. One will be the master and the other two will be the slave.
- The number of jobs we can run in parallel in Jenkins depends upon the number of the build executors.

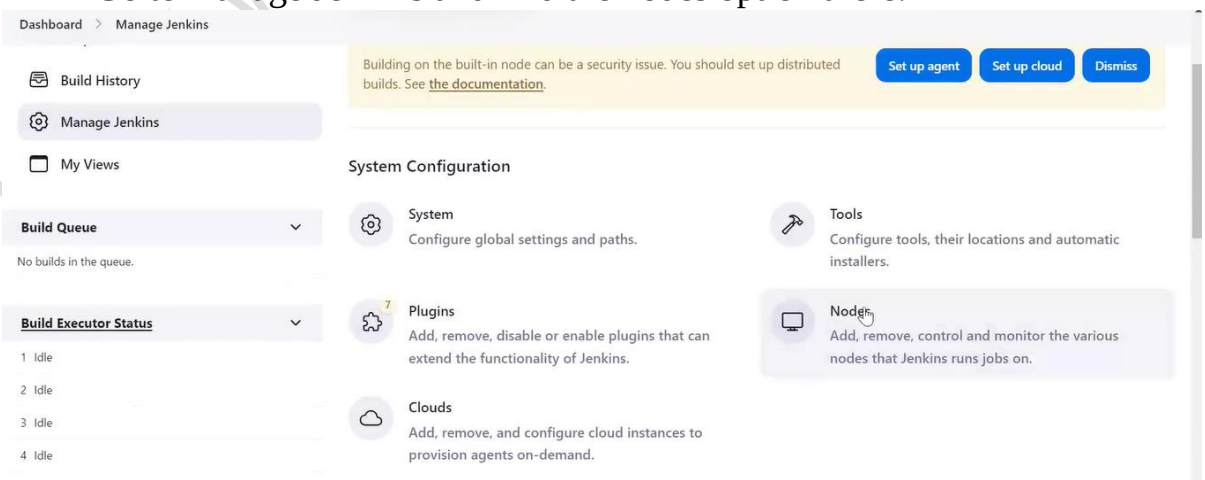


- Here we have 4 build executors which means we can run 4 jobs in parallel.

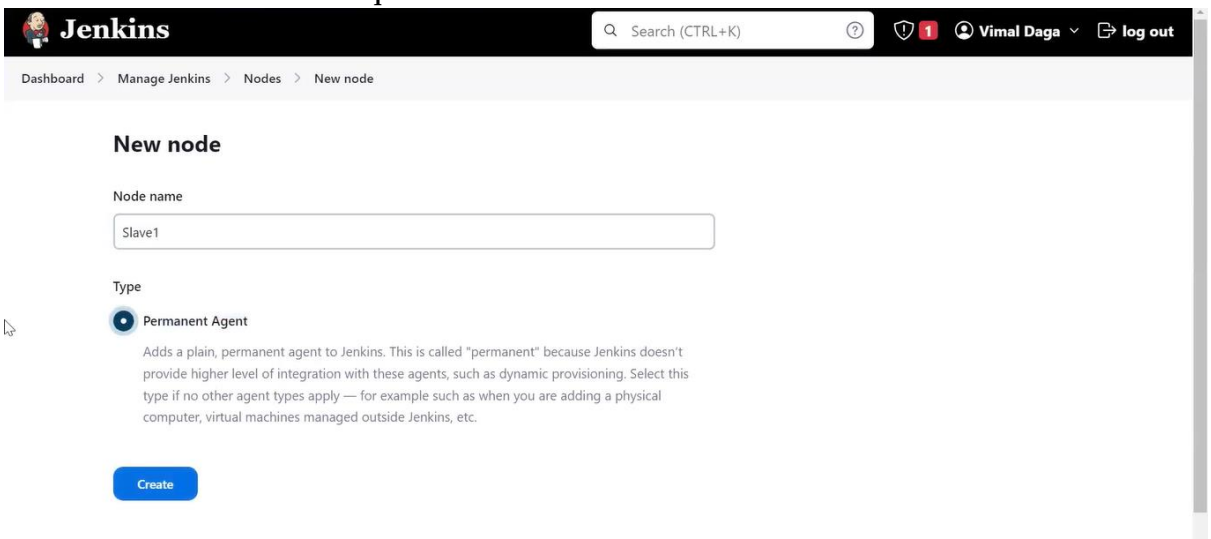
- We can also change the number of the build executors in the configure Jenkins option.



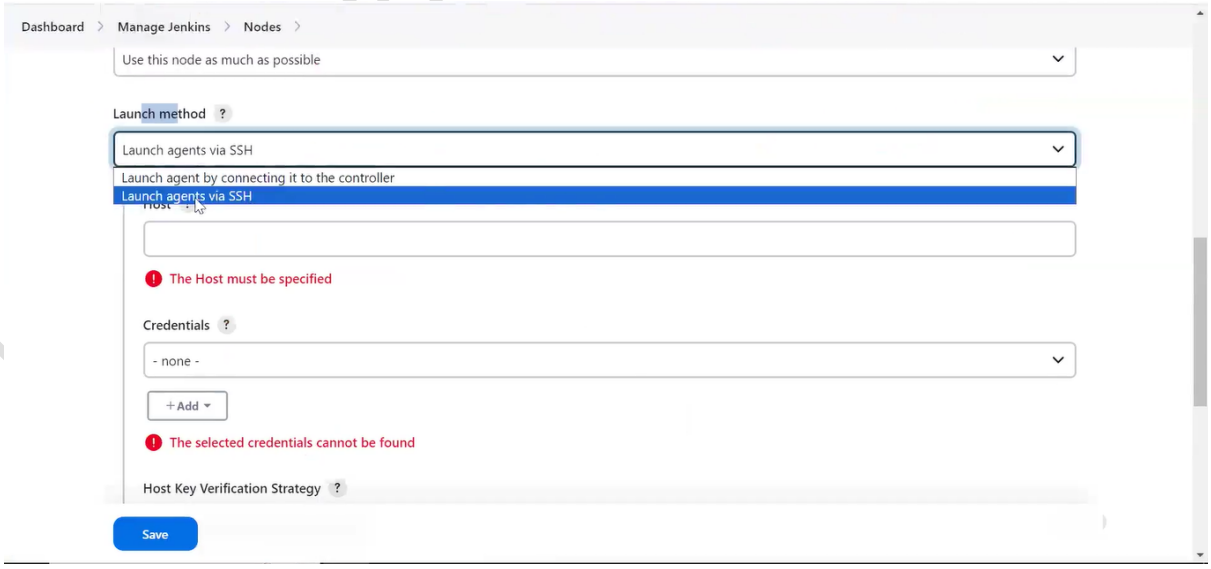
- To add the slave nodes in the master Jenkins nodes
  - Go to manage Jenkins and find the nodes option there.



- Click on the new node and give the name of the node and other configurations.
- There are two ways of adding the nodes, one is static and the other is dynamic.
- In the static node, the nodes always keep on working even when it is not required while dynamic nodes are so smart that they work only when there is a requirement.

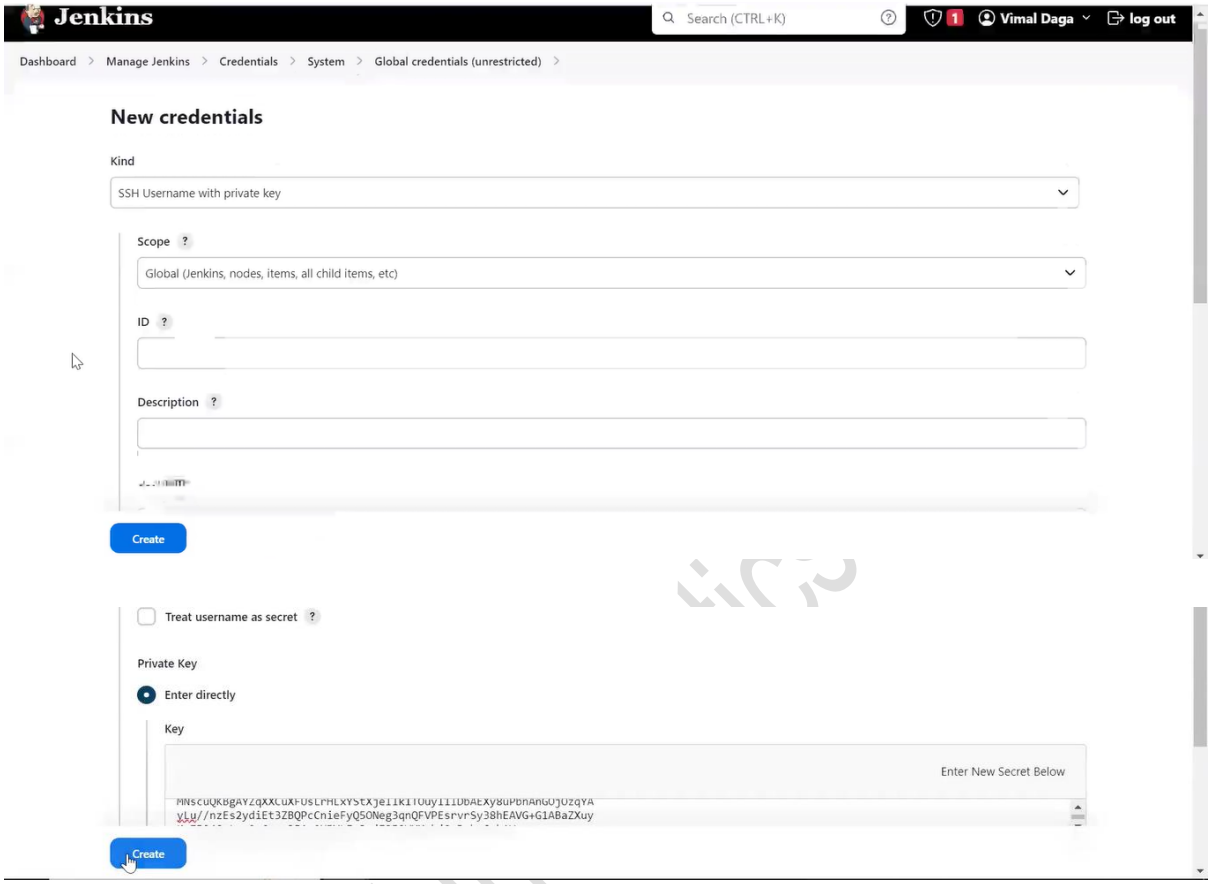


- We have to select the launch method also through which the master and slave will contact each other.
- Also, assign the IP address of the slave node and the credentials to the master node.
- Here the credentials is the private key.

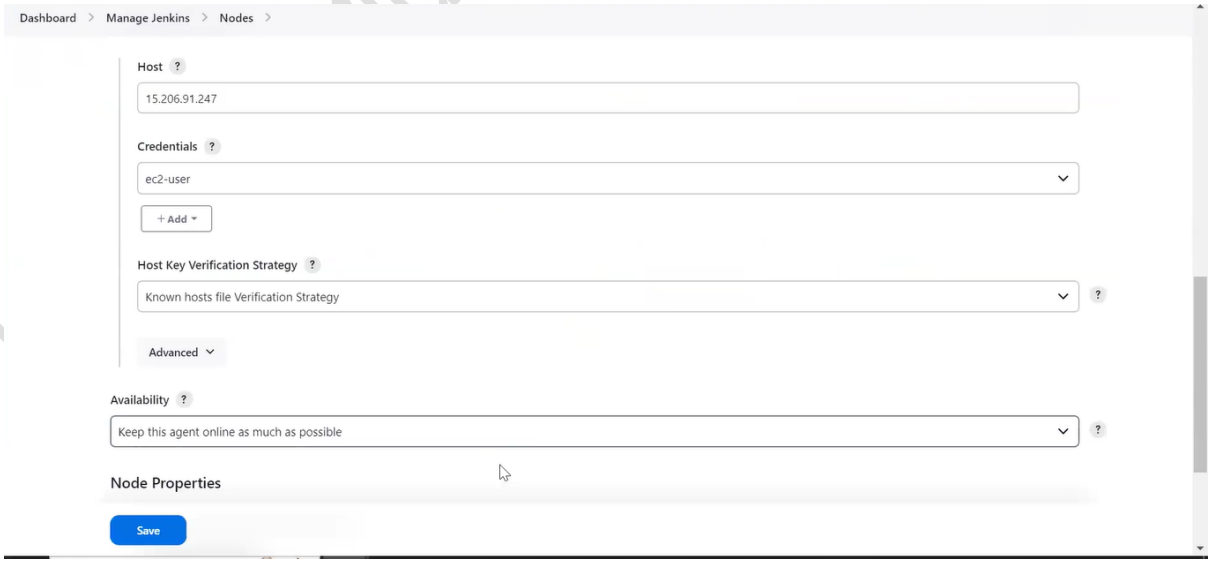


- The requirements to login to the slave are SSH enabled, IP address, username, and the password

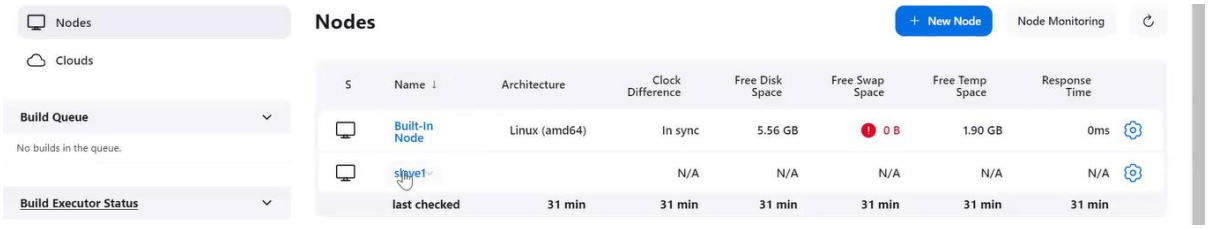
- Sometimes we have to manually add the credentials in the master node, for that go to the manage Jenkins option and select the credentials option from there.



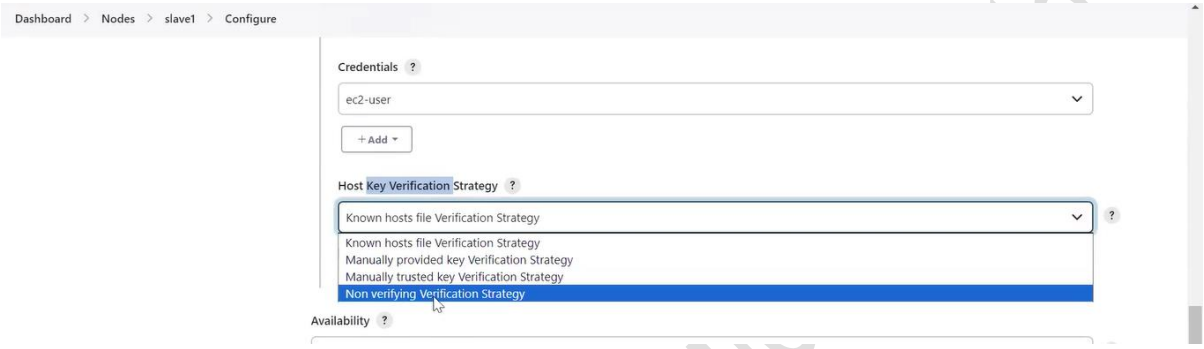
- Now we can add the node easily



- The node has been added



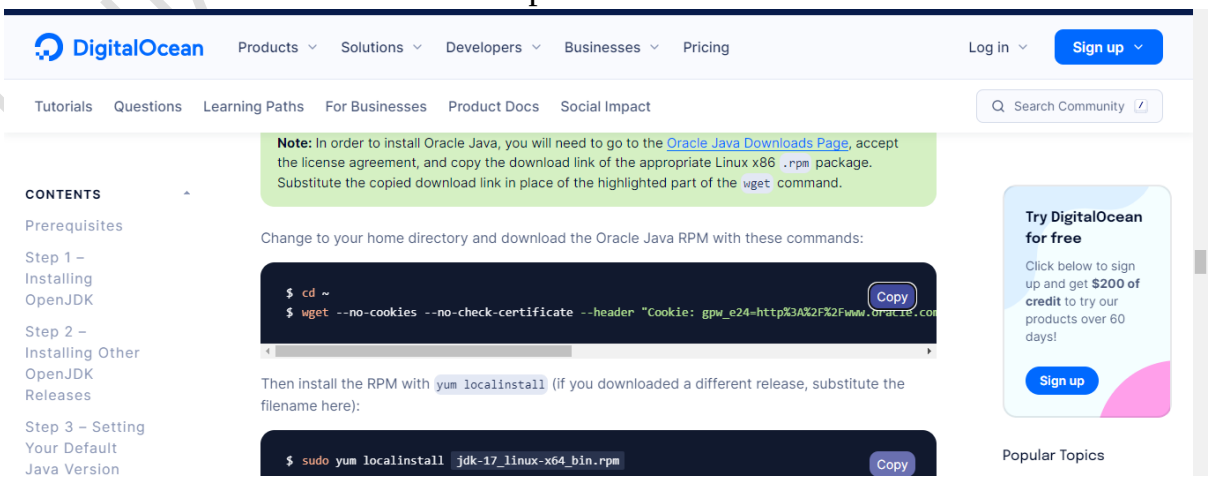
- While adding the credentials you should also allow the authentication by selecting the following option. Otherwise, the slave will fail to launch.



- We can arise with another error while launching the slave and that error can be due to the missing java agent.
- For that, we need to install the Java program in the slave.

```
[ec2-user@ip-172-31-32-53 ~]$ sudo  yum  install jdk
Last metadata expiration check: 0:39:54 ago on Sun Feb 18 10:59:57 2024.
No match for argument: jdk
Error: Unable to find a match: jdk
[ec2-user@ip-172-31-32-53 ~]$ sudo  yum  install openjdk
Last metadata expiration check: 0:40:00 ago on Sun Feb 18 10:59:57 2024.
No match for argument: openjdk
Error: Unable to find a match: openjdk
[ec2-user@ip-172-31-32-53 ~]$ sudo  yum  install  java
```

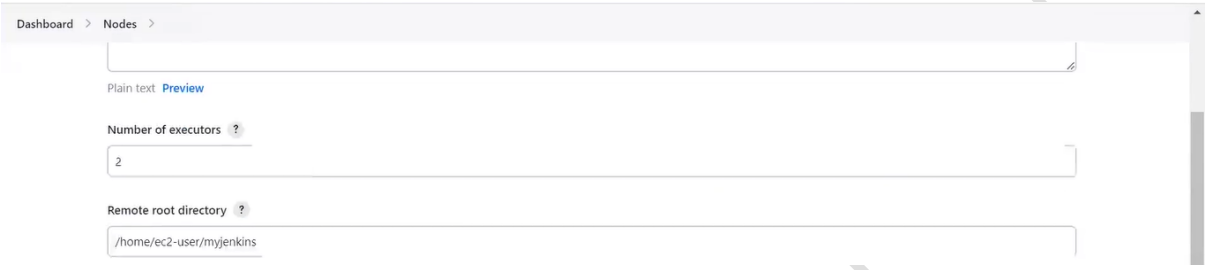
- Now if we want to launch the slave agent again then we will have one more error and that error is because of the Java version incompatibility.
- So we have to install Java from another way.
- Go to Google and search for java rpm download and copy the link from there to download the rpm.



- cd ~

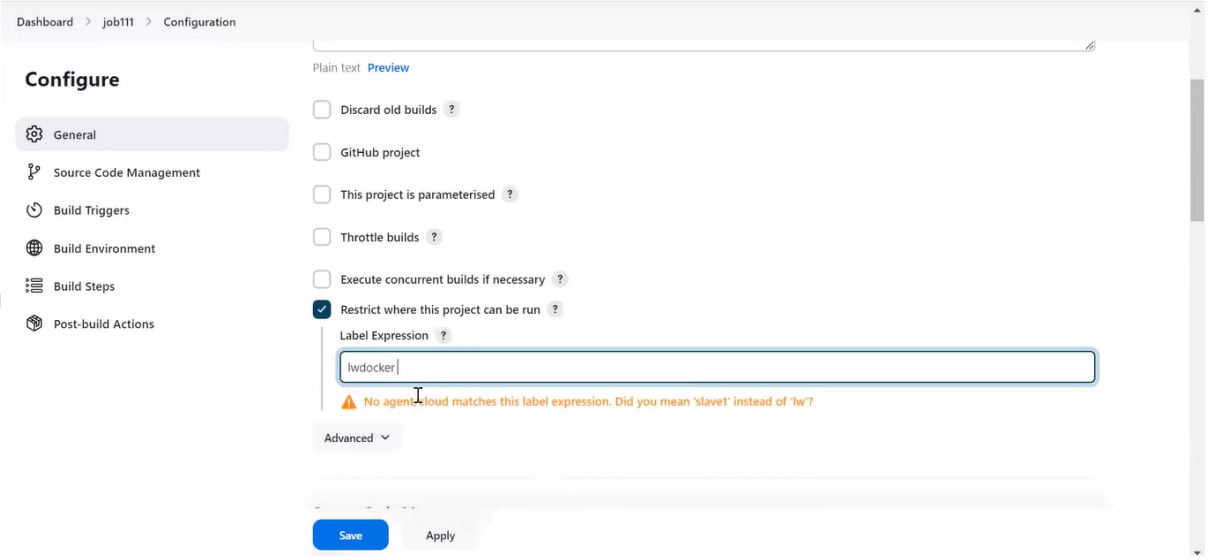
wget --no-cookies --no-check-certificate --header "Cookie: gpw_e24=http%3A%2F%2Fwww.oracle.com%2F; oraclelicense=accept-secure backup-cookie" https://download.oracle.com/java/17/latest/jdk-17_linux-x64_bin.rpm

- we also need to define the working directory for the slave, which can be done will creating or adding the slave.



- labels to the slave are given to easily differentiate it or to find out what exactly this slave does.
- The master-slave architecture is now set up successfully.

- We can decide on which slave the particular job should run.
  - For that, while creating the job we have to restrict it, here the use of labels comes into play.
  - Here we have chosen the lwdocker slave to run the docker jobs, making sure that the docker is installed in the slave node otherwise the job will fail to run.



- Now we can run any docker job on the slave from the master slave.

- Let’s set up an interesting project using the following architecture.

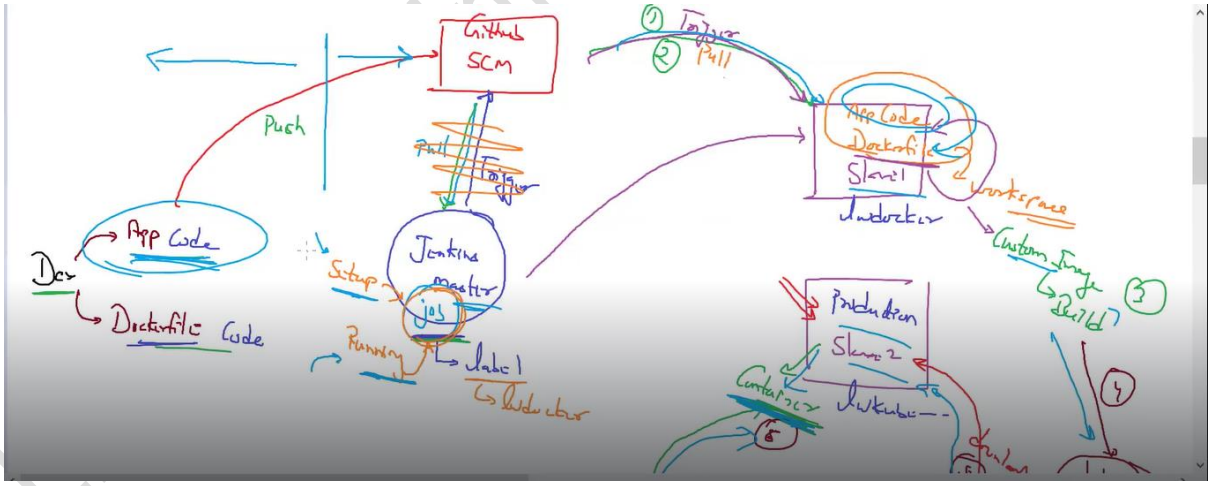
- Nowadays most of the work is done in containers, for example, to use or launch any website we generally put our code in the container and create our custom image.

- We can create the custom image using the Dockerfile.

- Dockerfile contains all the things that we need in our custom image, for example, the OS version, web server setup, custom codes, etc.

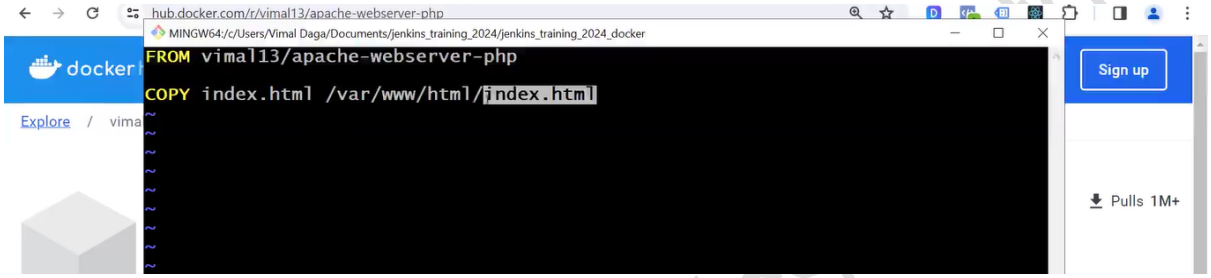
- Here is the entire pipeline for the project we are going to build

  - The developer will upload the app code or the Docker file to the Git SCM.
  - As soon as the code is uploaded the slave will pull or download the Dockerfile from the gitub.
  - After downloading, the slave will build an image from the Dockerfile.
  - Once the image is built, it will be pushed to the docker hub.

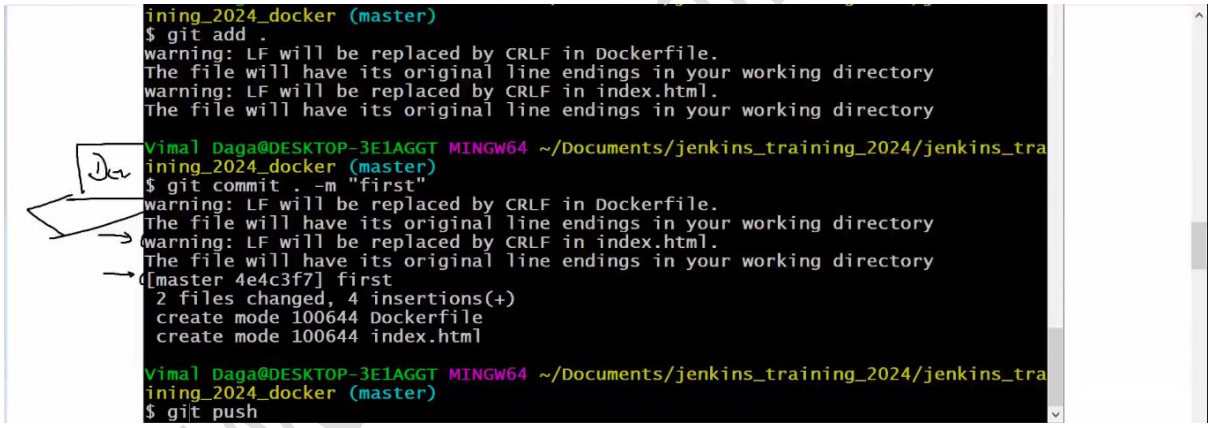


  - We will use two nodes with labels **build** and **production**.
  - Make sure to install the docker in both the slave nodes.
  - Now let's create the app code and the Dockerfile and push it to Github.

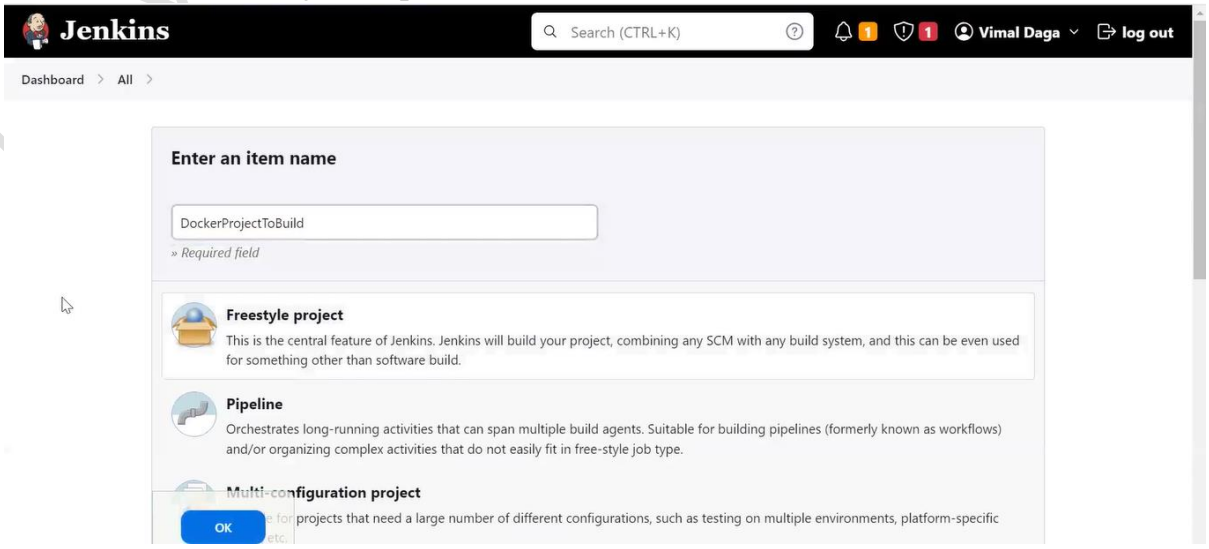
- Dockerfile

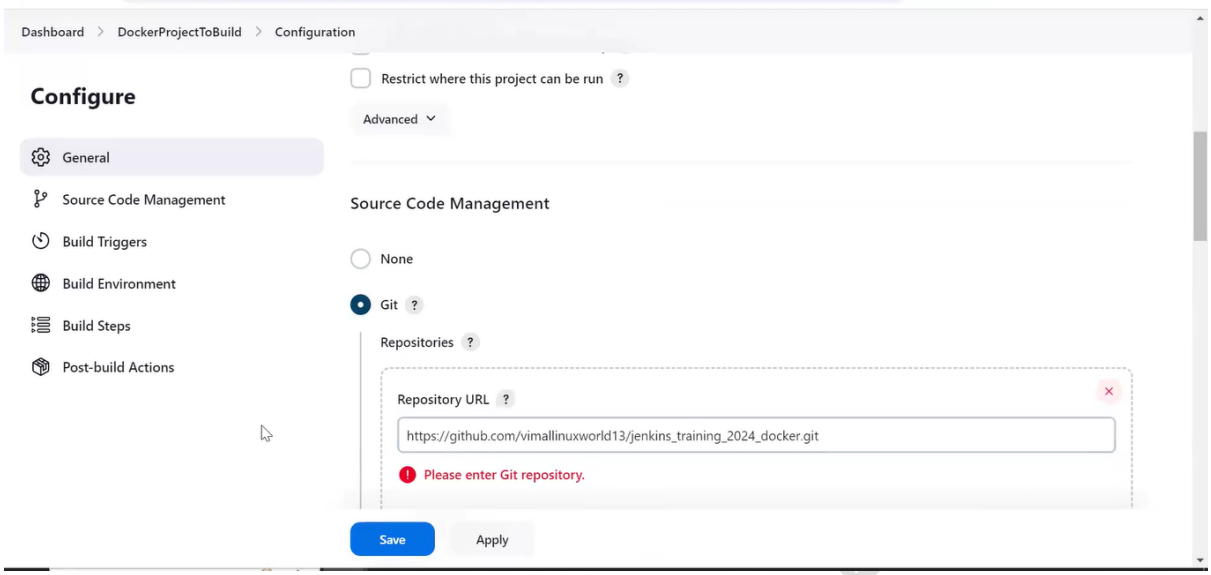


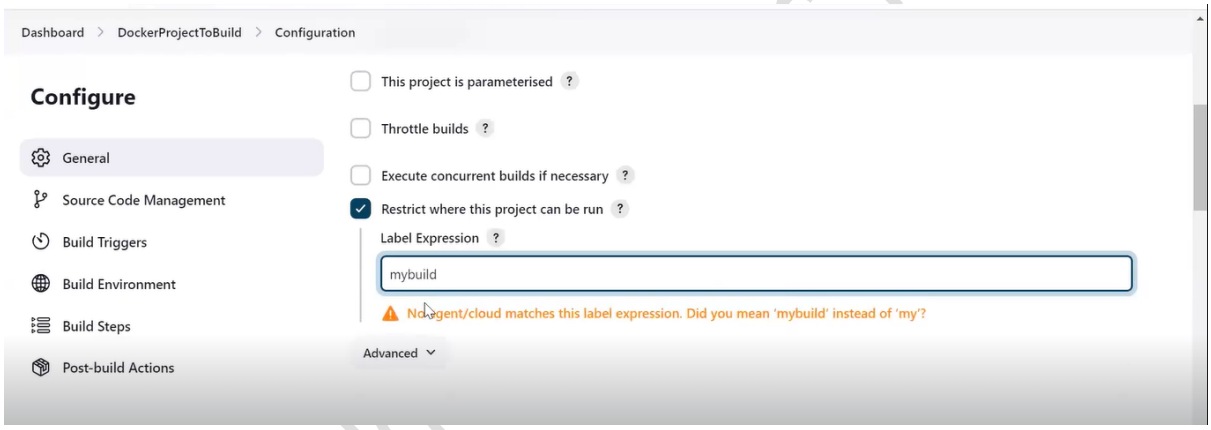
- Put both the files to the GitHub



- Now create a job to pull the files from the GitHub

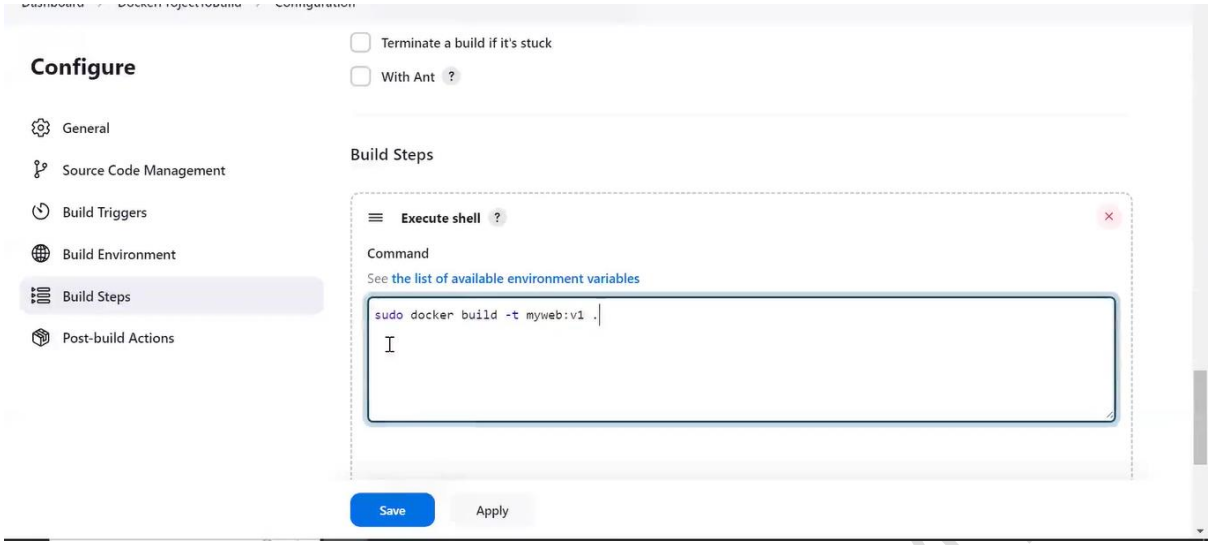






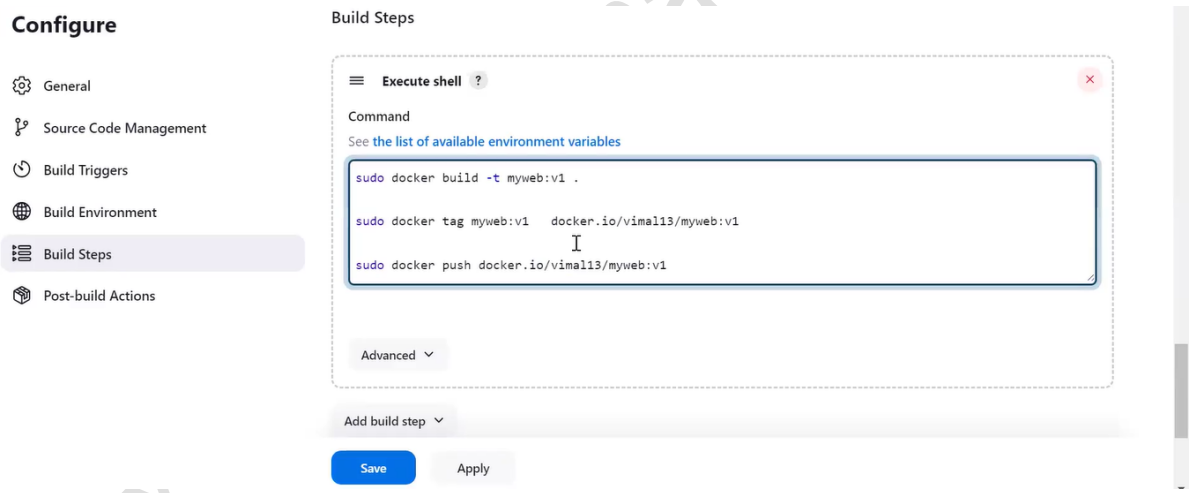
- This job may fail because the slave doesn't have the git installed in it, so first install the git in the salve using the command ***sudo yum install git -y.***
- We can see that the files have been downloaded in the slave

- Now next step is to build the files so for that we have to configure the job

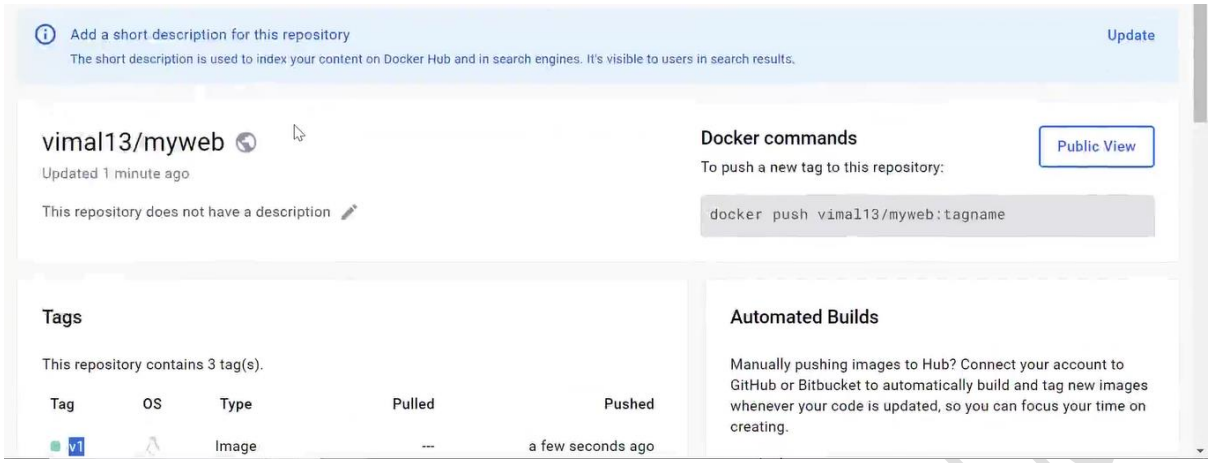


- Once we run the job we can see that an image is created in the slave.

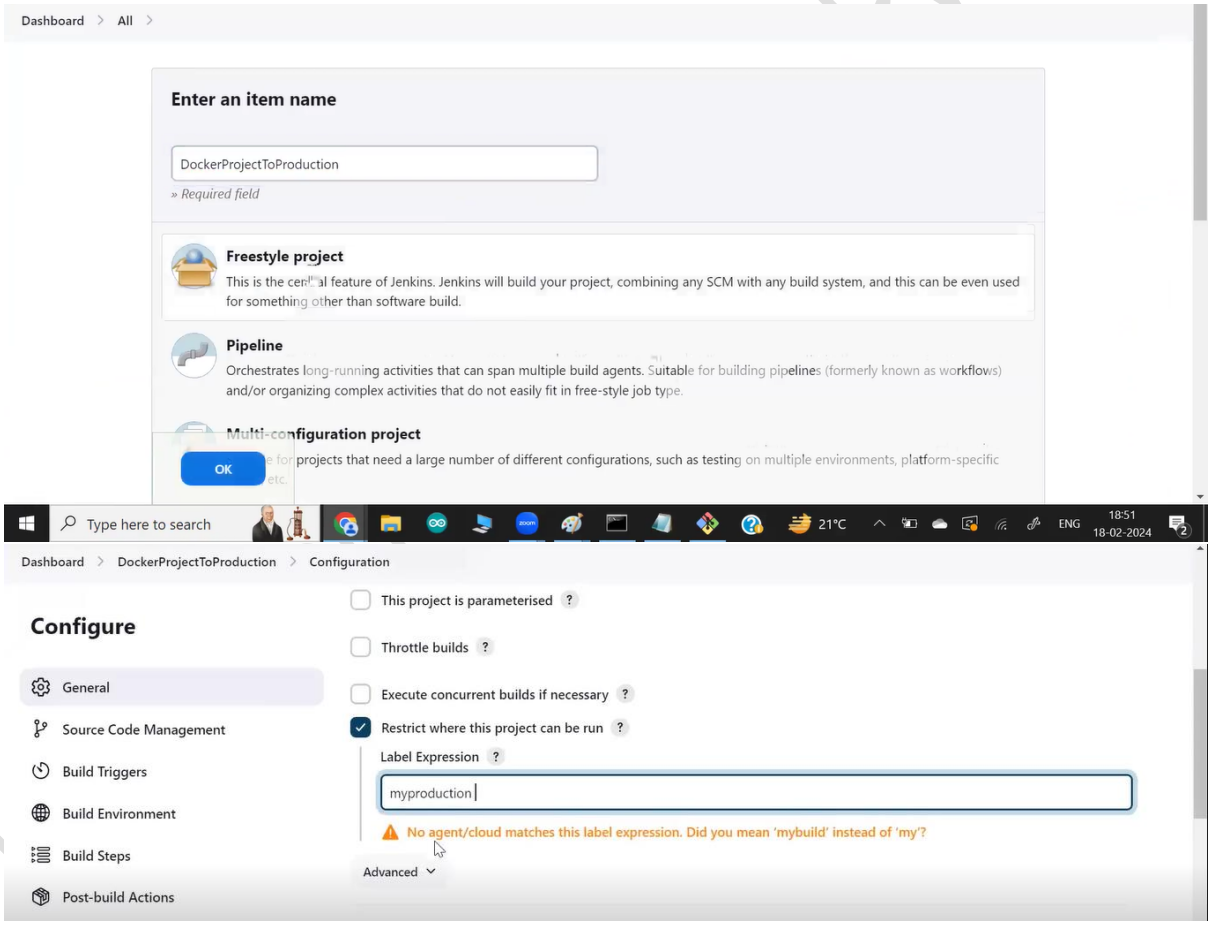
- The next step is to push the build image to the docker hub



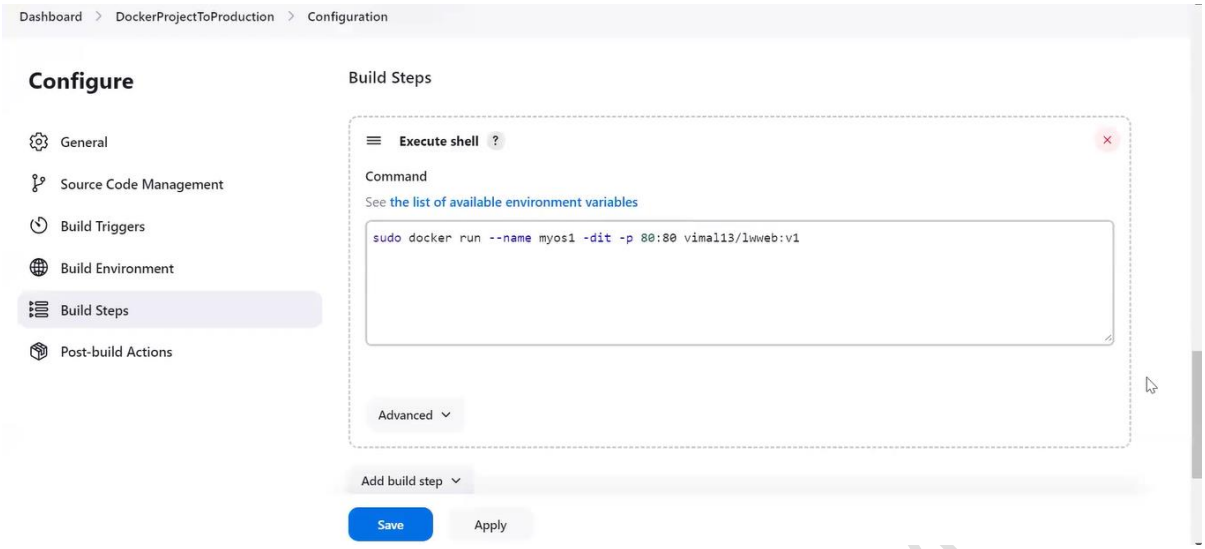
- For pushing the image we need to log in to the docker hub in the slave first and we also need to change the tag or name of the image.
- The image is successfully pushed to the docker hub.



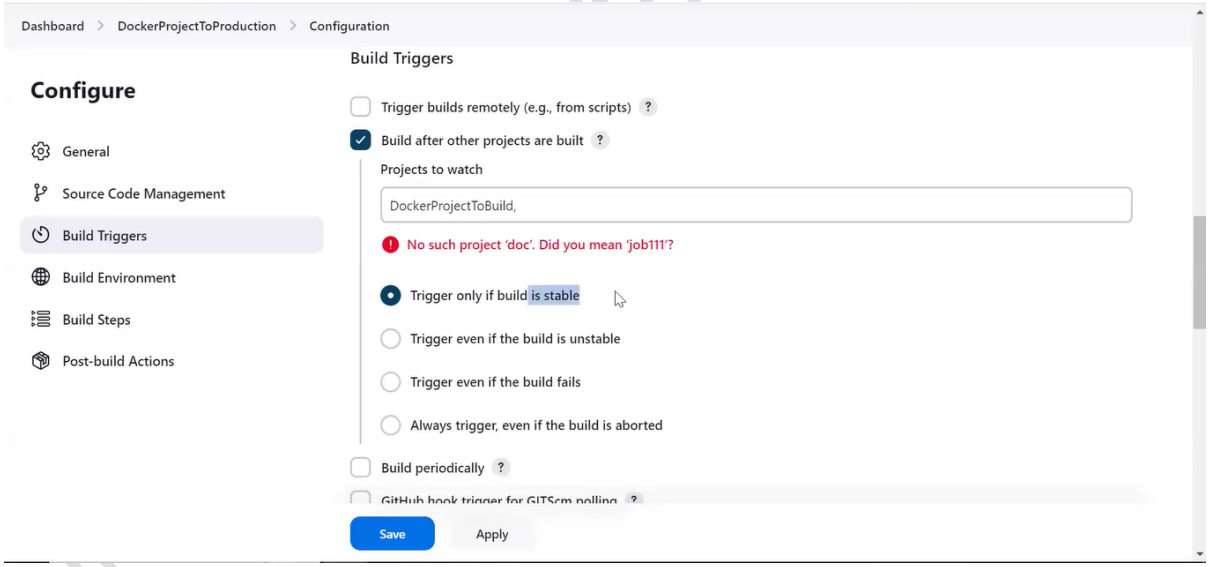
- We will create a new job now for the production work.



- This job will run on the slave2 labeled as the production.
- We need to run some commands to launch the image in the production



- ➢ Both jobs are independent but somehow they are related to each other. One slave is pushing the image and another is launching the image.
- ➢ We have to automate the process of launching the image in a way that as soon as the image is pushed to the docker hub, it should be launched by the slave2.
- ➢ For that go to the second job and configure it



- ➢ The job1 is the upstream project and the job2 is the downstream project.

- Here we are done with the end-to-end automation, we just have to push the code to GitHub and the website will be launched automatically.