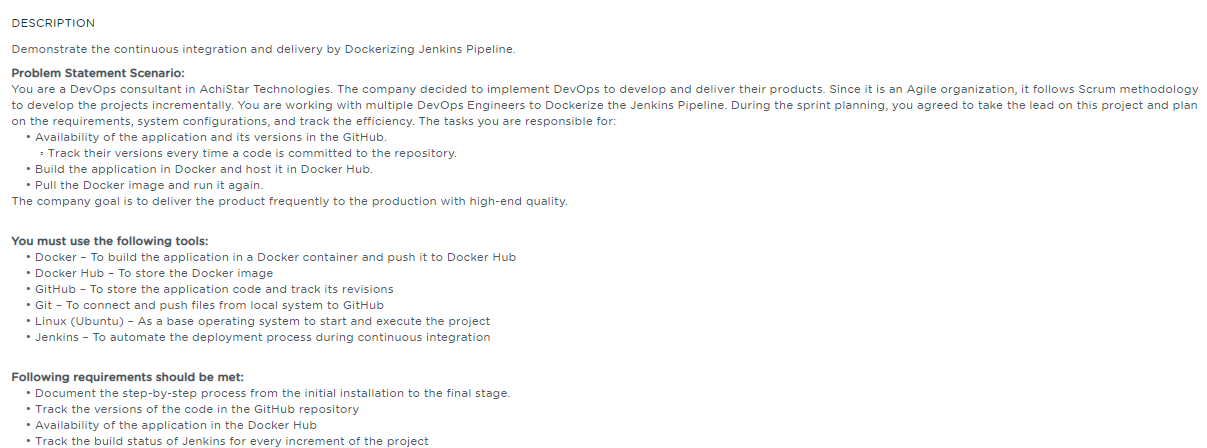
**Dockerizing Jenkins Pipeline - Assessment**



**Table of Contents**

[**1.** **Introduction** 3](#_Toc43312812)

[**2.** **Project Abstract** 3](#_Toc43312813)

[**3.** **Install Git** 3](#_Toc43312814)

[**4.** **GitHub account setup and Project creation** 3](#_Toc43312815)

[**4.1.** **GitHub account setup** 3](#_Toc43312816)

[**4.2.** **Login from Git local to remote GitHub** 4](#_Toc43312817)

[**4.3.** **Initialize Git and add the project into the local Git repo** 4](#_Toc43312818)

[**4.4.** **Create a Project repository in GitHub account** 4](#_Toc43312819)

[**4.5.** **Creation of SSH Key and adding it to GitHub** 6](#_Toc43312820)

[**5.** **Install and configure Jenkins** 7](#_Toc43312821)

[**5.1.** **Run Jenkins in browser** 7](#_Toc43312822)

[**5.2.** **Install CloudBees Docker Build and Publish Plugin** 8](#_Toc43312823)

[**6.** **Install and setup Docker** 10](#_Toc43312824)

[**6.1.** **Verify Docker** 10](#_Toc43312825)

[**7.** **DockerHub Readiness** 11](#_Toc43312826)

[**7.1.** **Setup Docker account** 11](#_Toc43312827)

[**7.2.** **Login to Docker account** 11](#_Toc43312828)

[**8.** **CI/CD configuration in Jenkins** 12](#_Toc43312829)

[**8.1.** **Create a new build job** 12](#_Toc43312830)

[**8.2.** **Docker Build and Publish Plugin** 18](#_Toc43312831)

[**9.** **Track version change history in GitHub** 18](#_Toc43312832)

[**10.** **Availability of the application in the Docker Hub** 19](#_Toc43312833)

[**11.** **Track the build status of Jenkins for every increment of the project** 20](#_Toc43312834)

[**12.** **Pull the Docker image and run it again** 20](#_Toc43312835)

# **Introduction**

This assessment is intended to demonstrate the Continuous Integration and Continuous Delivery of the sample project by Dockerizing the Jenkins pipeline

# **Project Abstract**

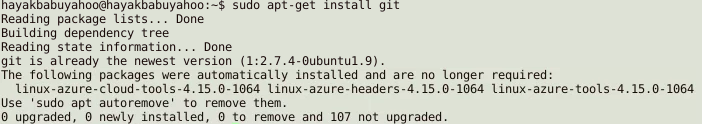
We have considered the Node JS application to demonstrate the Continuous Integration and Continuous Deployment along with the Dockerization using the various tools

# **Install Git**

If Git is already installed in your machine, you can check the version of git by executing the below command in the terminal.

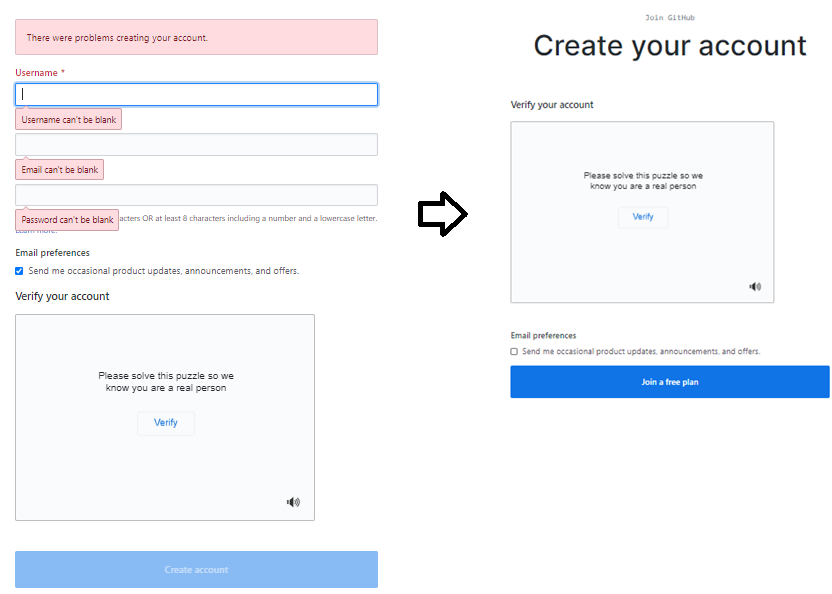


If git is not installed, then you can follow the below steps to install git



# **GitHub account setup and Project creation**

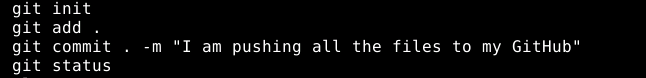
## **GitHub account setup**

****

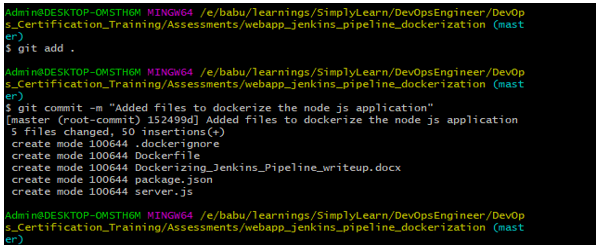
## **Login from Git local to remote GitHub**



## **Initialize Git and add the project into the local Git repo**



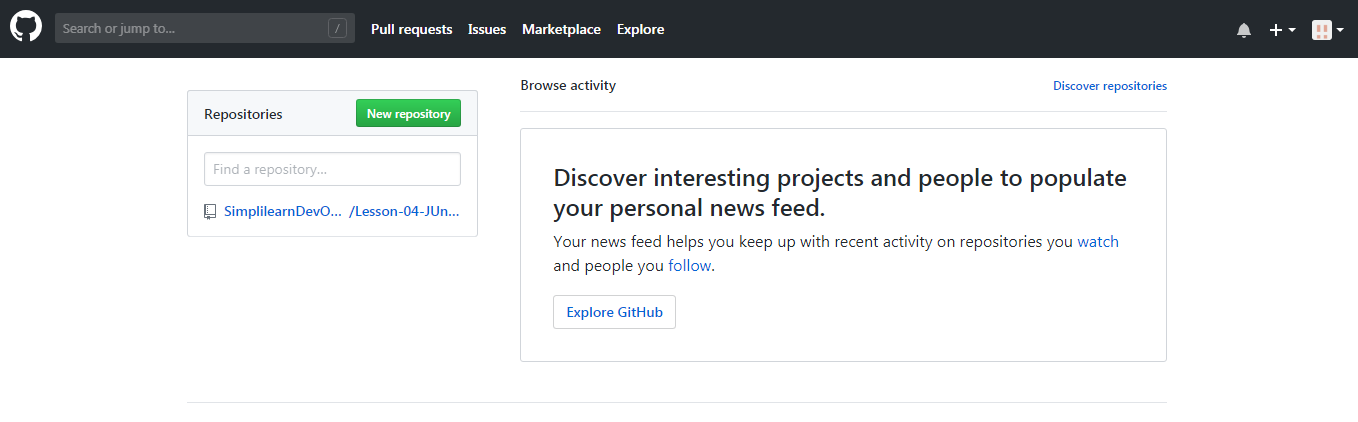
Please follow the below process for step-by-step confirmation of each command execution.

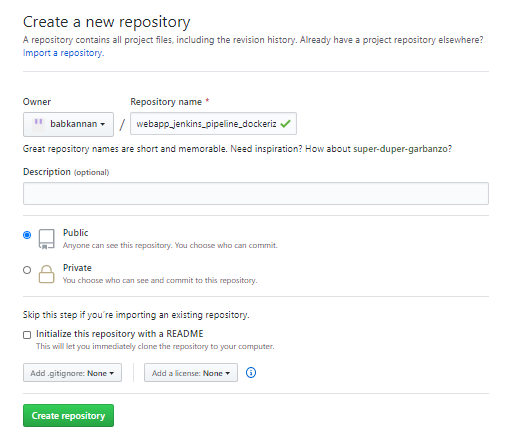


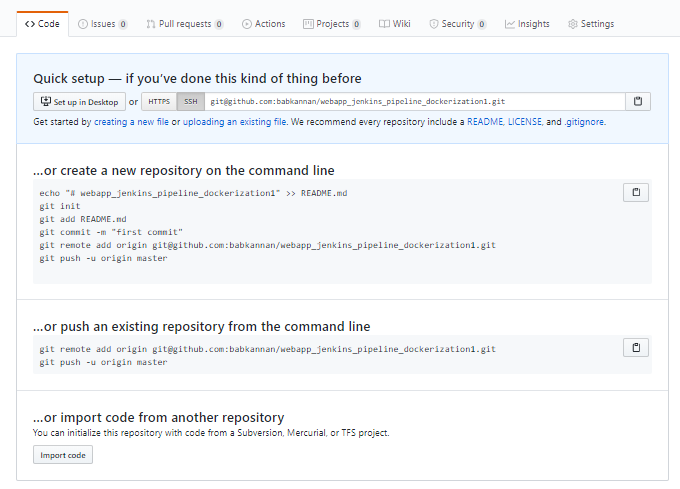
The above commands stage all the project files and pushed into the local Git repository.

## **Create a Project repository in GitHub account**

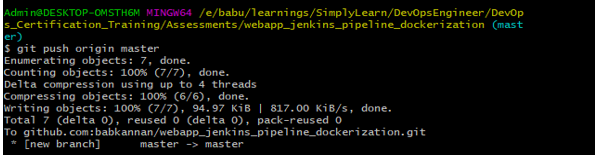
Go to the homepage of GitHub.com and click on **New Repository** as shown below.



Enter the name as “**webapp\_jenkins\_pipeline\_dockerization**” and click on **Create repository.**

You will be redirected to a quick guide page and you will be navigated automatically inside the directory you have created. 

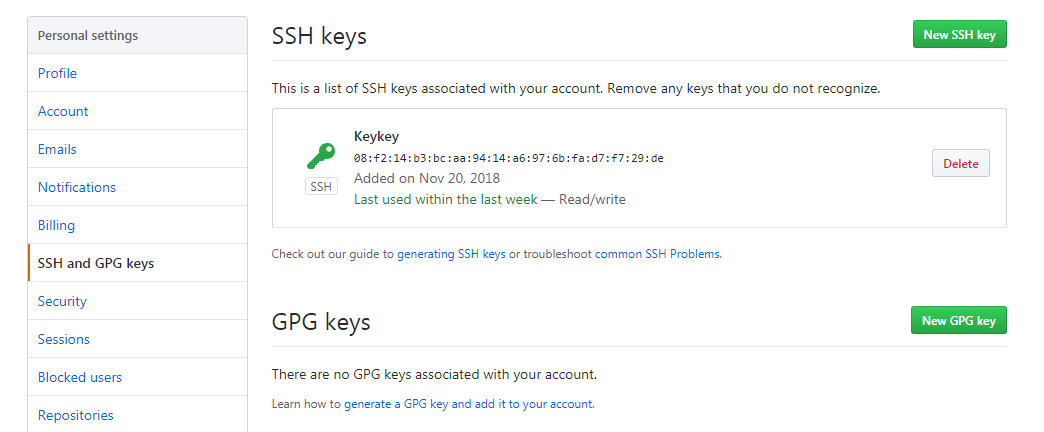
The screenshot below shows that the Node js application is successfully pushed into the Git repository



If you’re unable to push the files to your Github.com account, then follow the below steps:

## **Creation of SSH Key and adding it to GitHub**

Copy the entire key from the clipboard. Choose **Your avatar** > settings **> SSH & GPG Keys** and click on **New SSH key** and paste the key and save it.

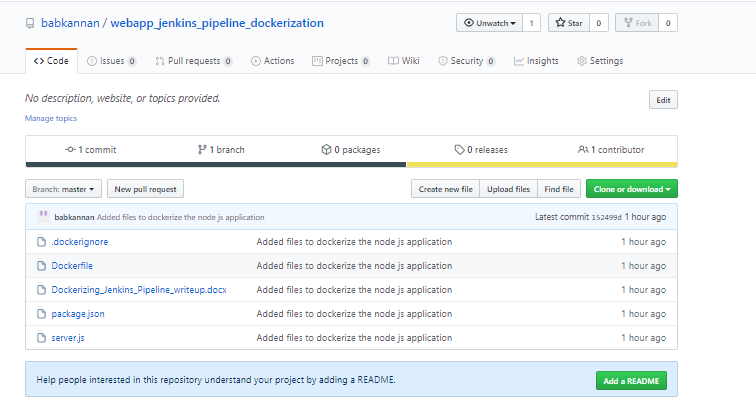


In the terminal, execute **ssh-add** to save the key and link it with local git.

Copy the git remote add origin <URL\_of\_Your\_GitHub\_Repository> and execute it in the terminal.

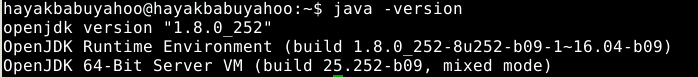
|  |
| --- |
| git remote add origin git@github.com:babkannan/webapp\_jenkins\_pipeline\_dockerization.git  git push -u origin master |

Reload your GitHub.com account to confirm the output shown below.



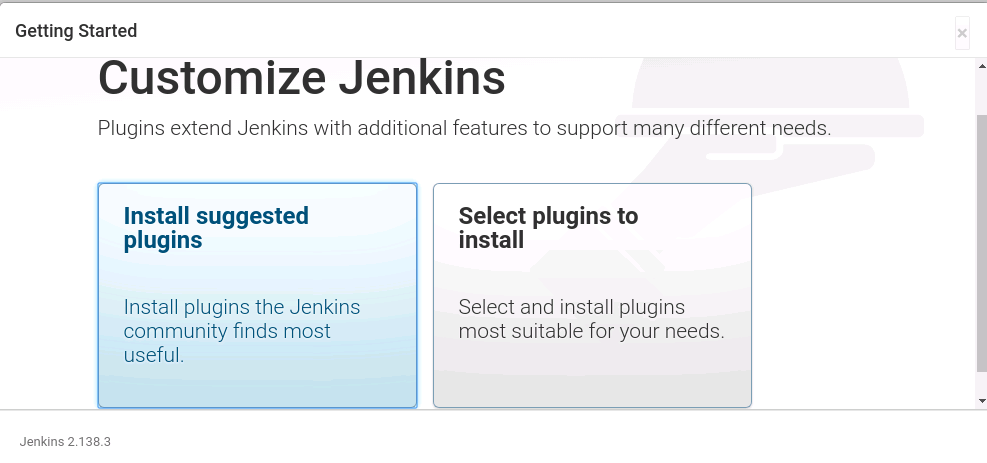
# **Install and configure Jenkins**

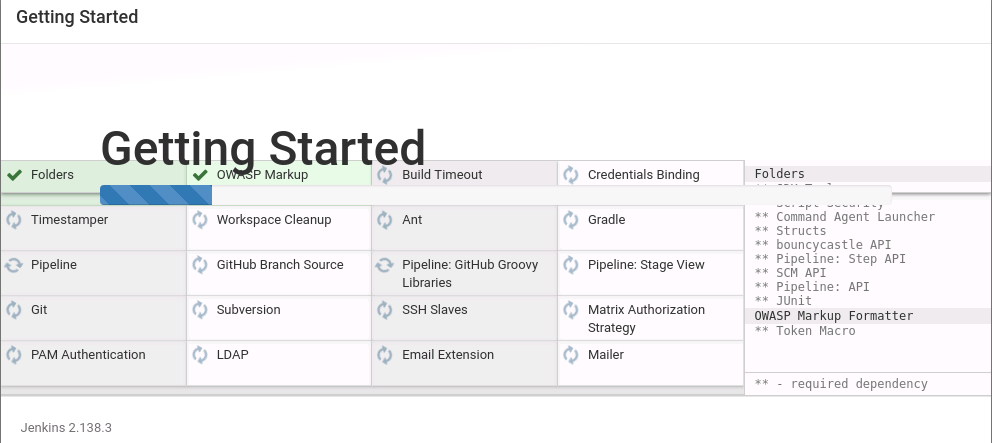
If you could see the result like as shown below, then Java environment is already installed in the Ubuntu system



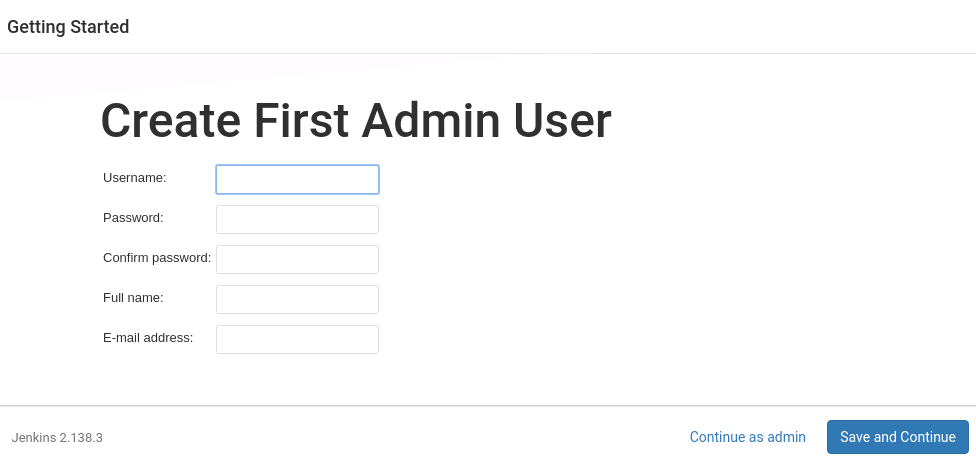
## **Run Jenkins in browser**

Once you are logged in, you will be redirected to the page below:

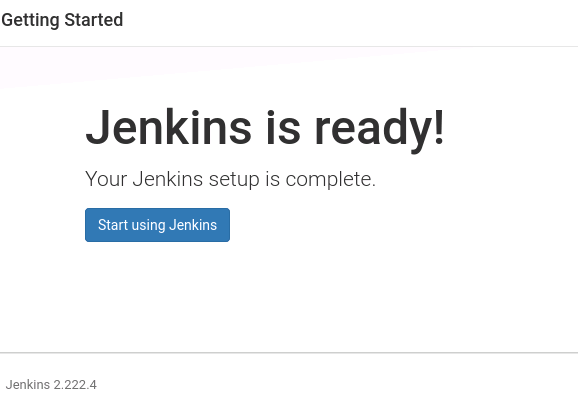
 Select **Install suggested plugins**. You will be redirected to the page below:



After installing recommended plugins, you can create “First Admin User” or continue as Admin by filling the required details as shown in the screenshot.

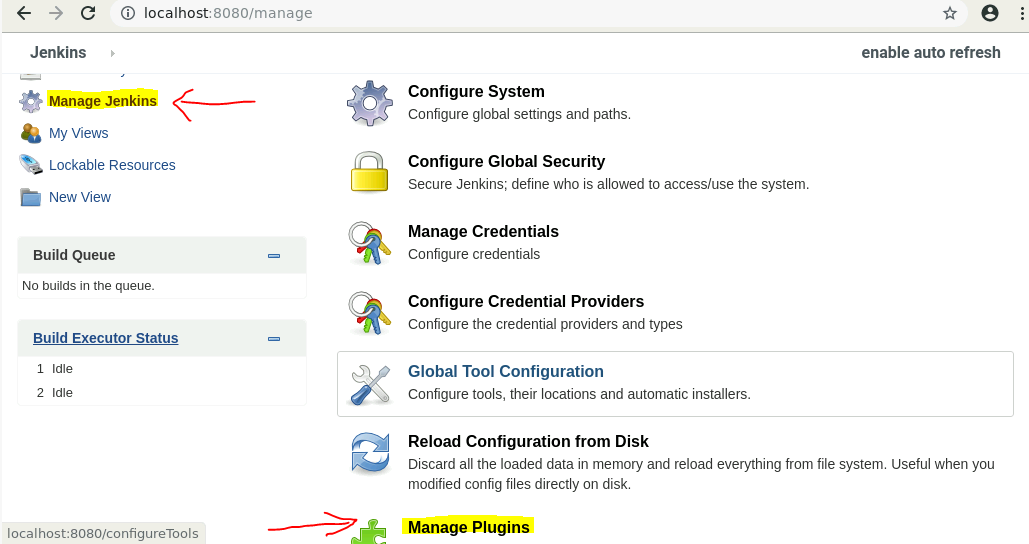


Once that is done, the below page is displayed which informs the user like “Jenkins is ready! and we shall start using Jenkins.

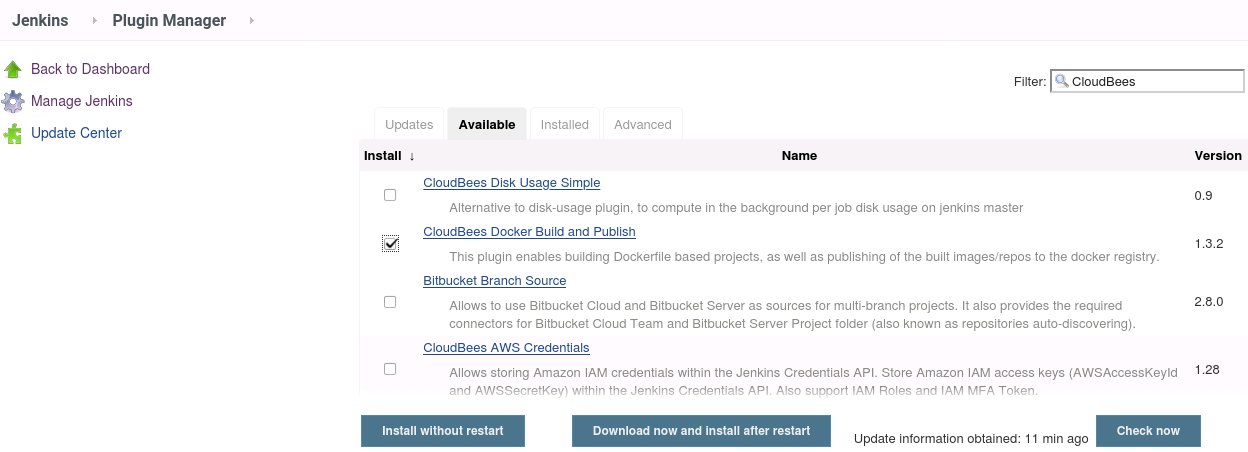


## **Install CloudBees Docker Build and Publish Plugin**

* Click **Manage Jenkins | Manage Plugin** option as shown below in the screeeshot

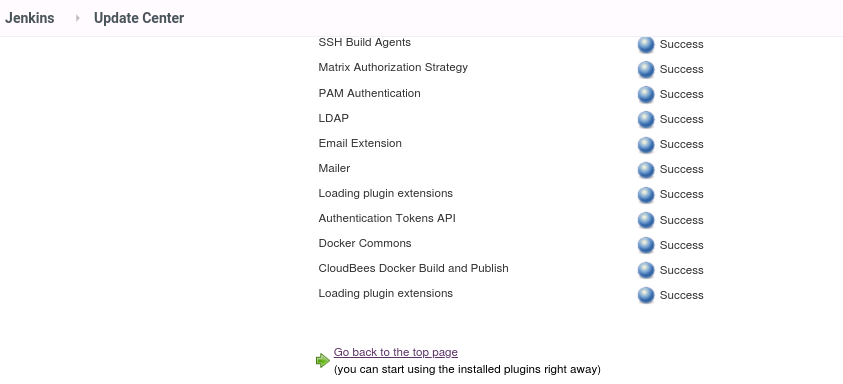


* Click the “Available” tab and enter the text like “CloudBees” in the Search filter as shown below in the screenshot which displays the “CloudBees Docker Build and Publish” plugin



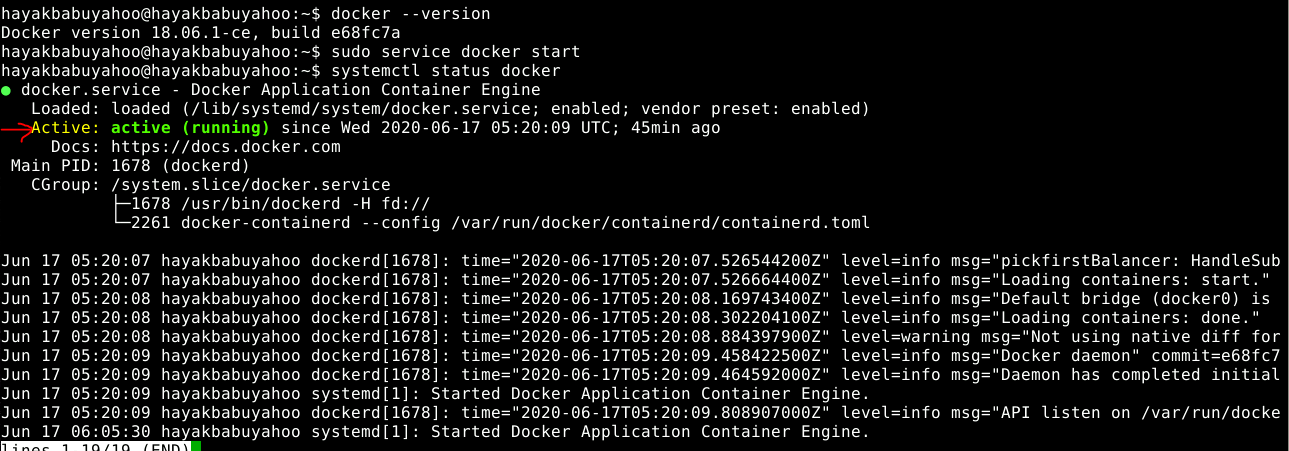
* Select the CloudBees **Docker Build and Publish** and install the plugin using the option “Install without restart”

After successful installation of plugin, the below page appears which informs the user to start using the plugins right away



# **Install and setup Docker**

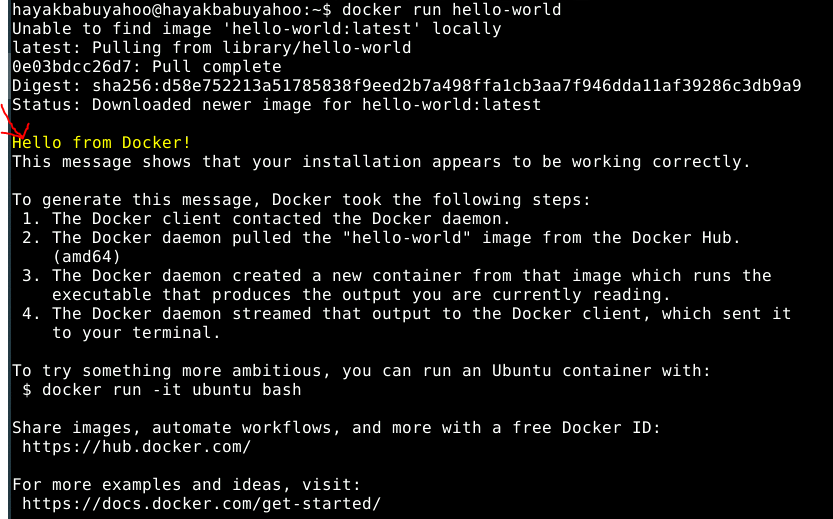
For commands needs to be used, refer the write\_up documents. The screenshot below is used to verify the docker running state



## **Verify Docker**

|  |
| --- |
| sudo docker run hello-world |

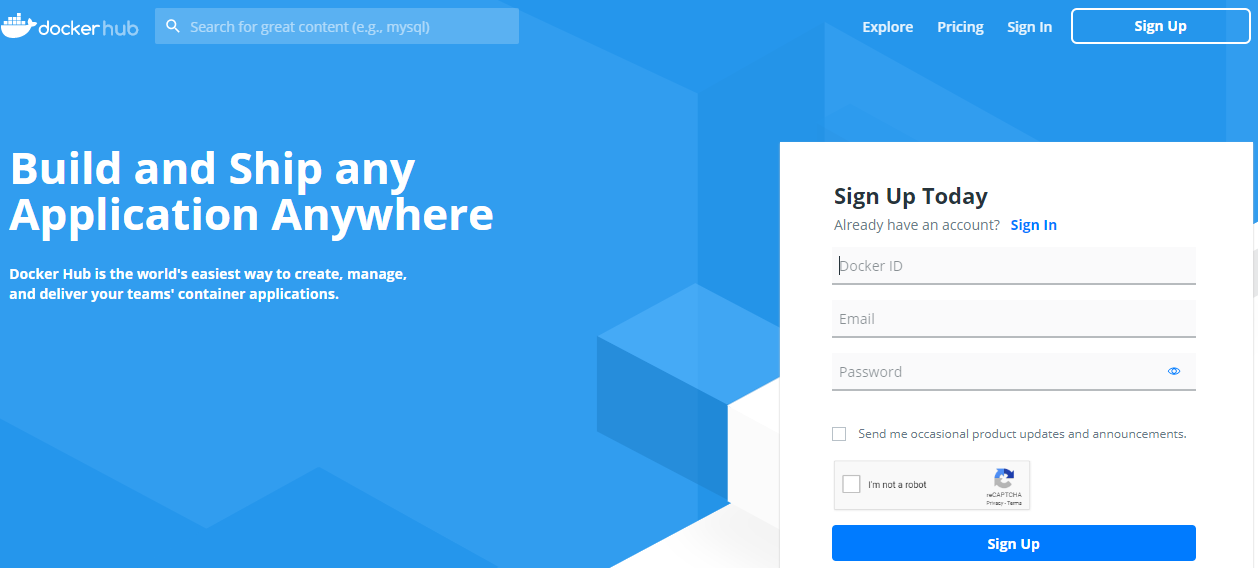
Just pull the hello-world image from the docker and check the docker is running properly in the system



# **DockerHub Readiness**

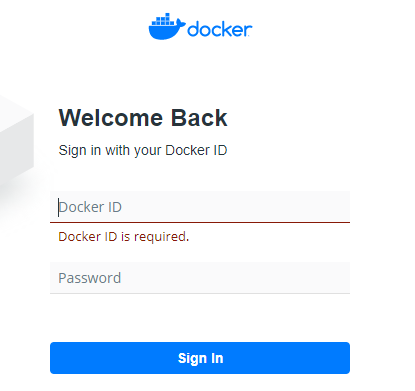
## **Setup Docker account**

Go to [**https://hub.docker.com/**](https://hub.docker.com/) and sign-up to create a docker account



## **Login to Docker account**

Once you register and verify your Docker ID email address, you can log in to [Docker](https://hub.docker.com/) Hub



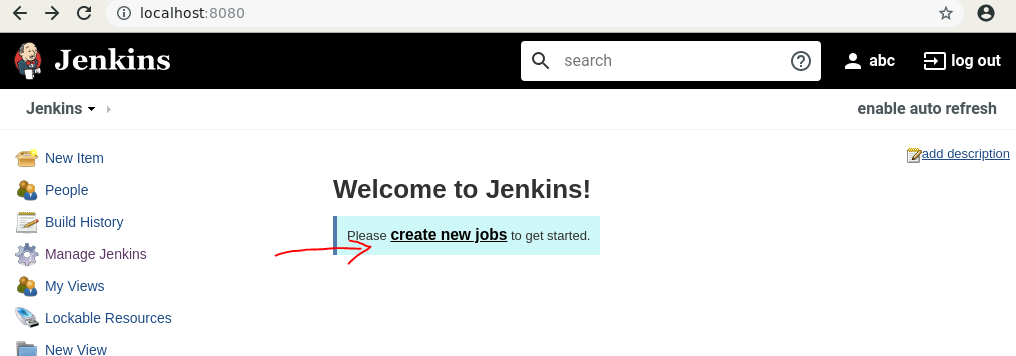
Refer: For more details, refer <https://docs.docker.com/docker-id/>

# **CI/CD configuration in Jenkins**

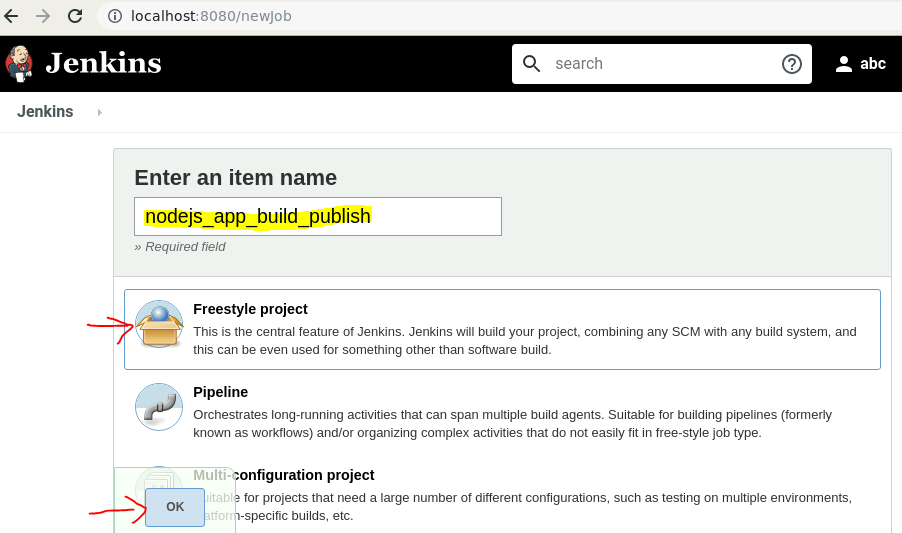
Continuous Integration and Continuous Deployment Configuration in Jenkins

## **Create a new build job**

**Step 1:** Click “create new jobs” to create a new build project which is responsible for building and deploying our project to the docker hub

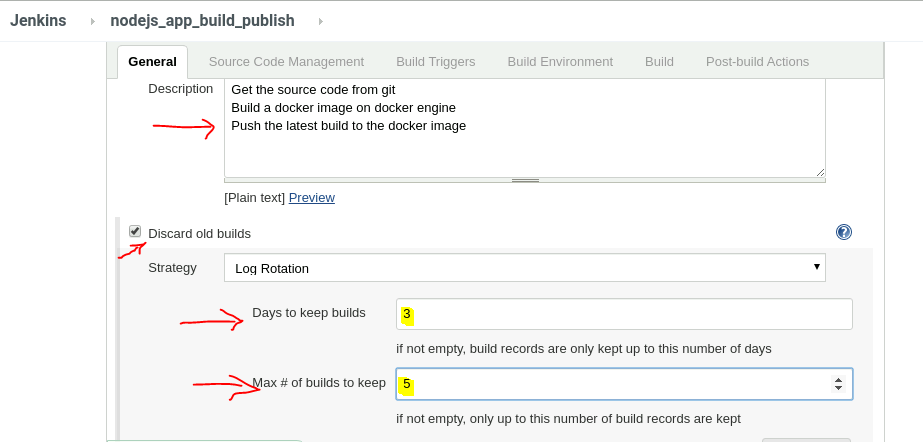


**Step 2**: Select “Freestyle Project” and enter an item name as “nodejs\_app\_build\_publish” and click OK

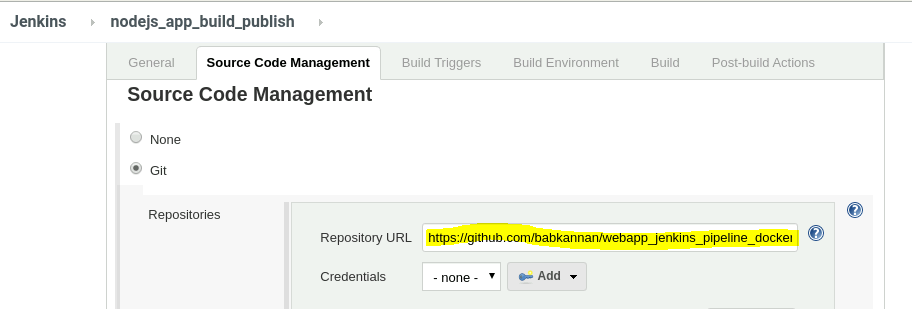


**Step 3**: As the best practice, In the **General** tab of the configuration, fill-in the below fields (as shown in the screenshot below)

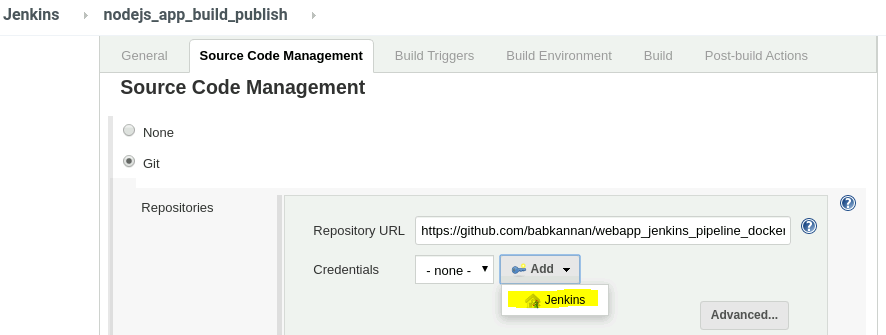
* Description
* Discard old-builds
* Days to keep builds
* Max # builds to keep



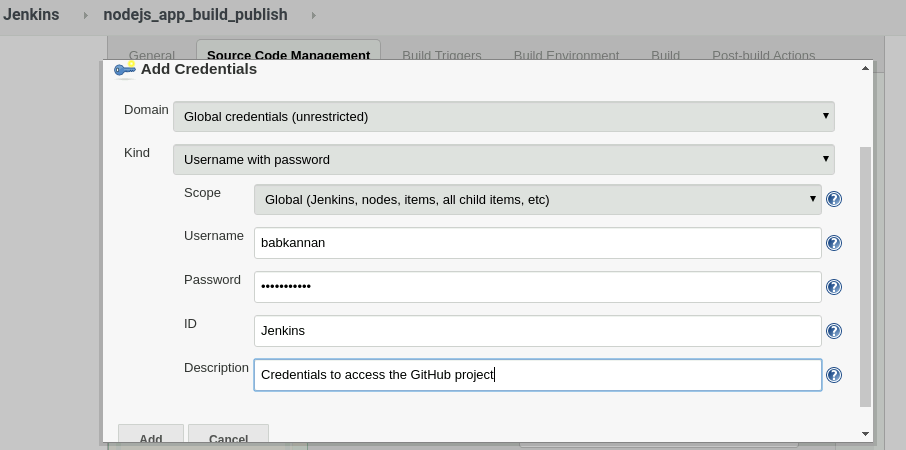
**Step 4:**  In the **Source Code Management** tab of the configuration, fill-in the project repository URL field (as shown in the screenshot below). Copy the Project URL from the GitHub web repository



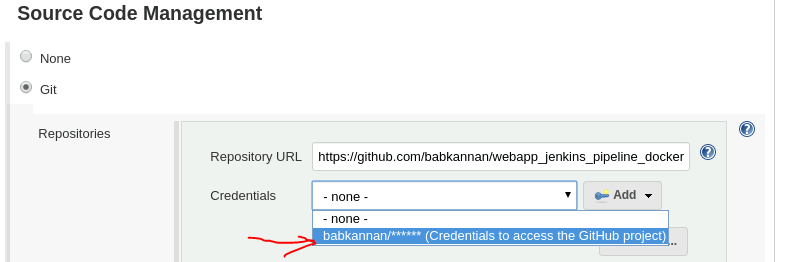
**Step 5:** To access the project in the GitHub securely from the Jenkins, add the project access credentials information. This can be achieved by clicking **Add | Jenkins**



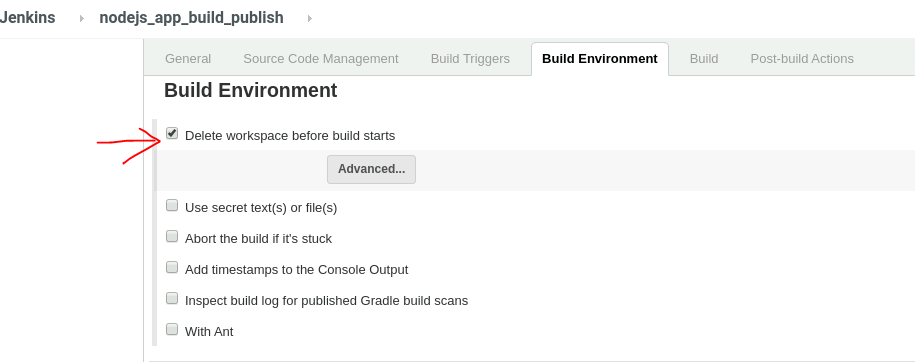
**Step 6:** Fill the GitHub access credentials and click **Add** as shown in the below screenshot



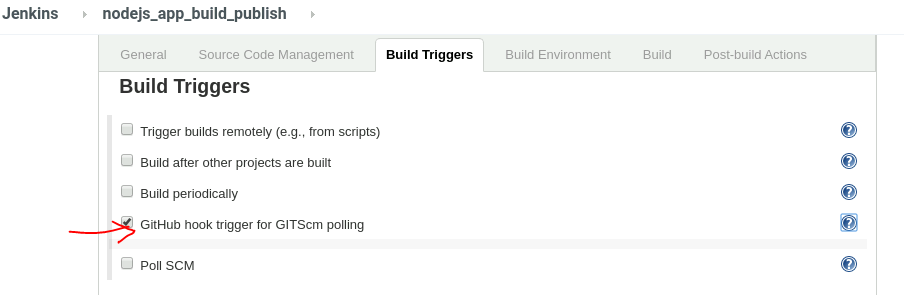
Once it’s successfully added, you can verify the credential information gets added as shown in the below screenshot



**Step 7:** Select the “**Build Environment**” tab of the configuration and select the “Delete workspace before build starts as shown in the below screenshot

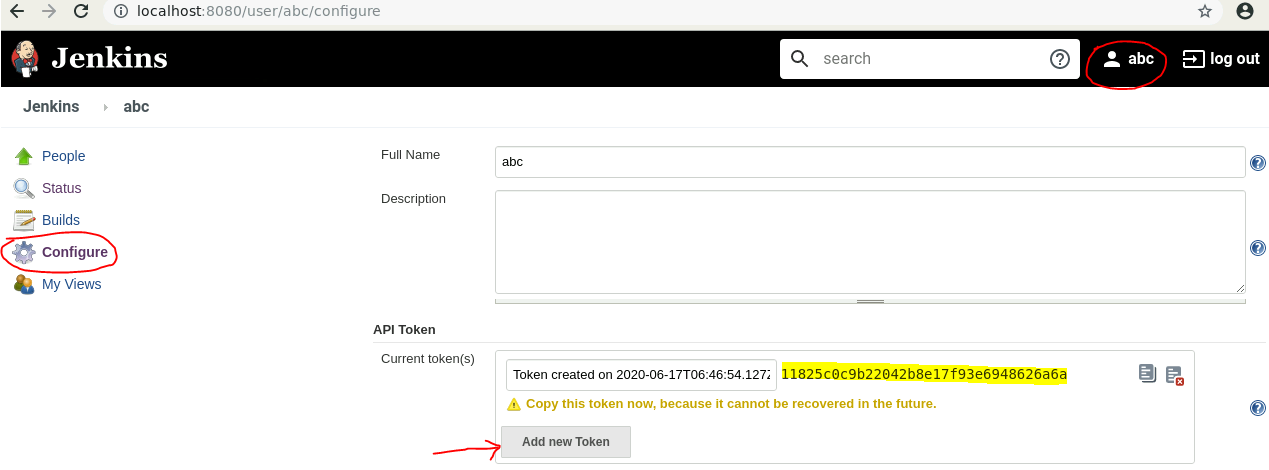


**Step 8:** Under the **Build Triggers** tab of the configuration, select the “**GitHub hook trigger for GitSCM polling**” option as shown below in the screenshot: This option basically enables in triggering the build whenever changes are pushed into the GitHu

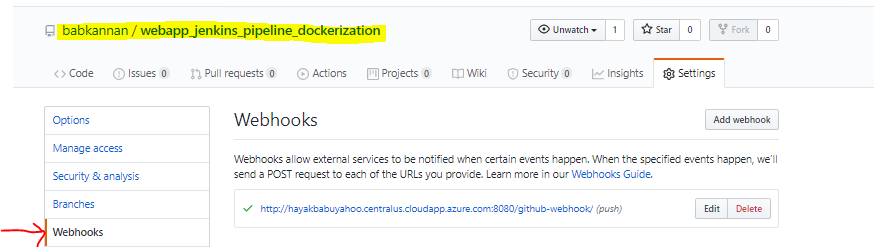


Enabling the GitHub hook itself is not enough, few steps needs to be performed to completely enable this feature.

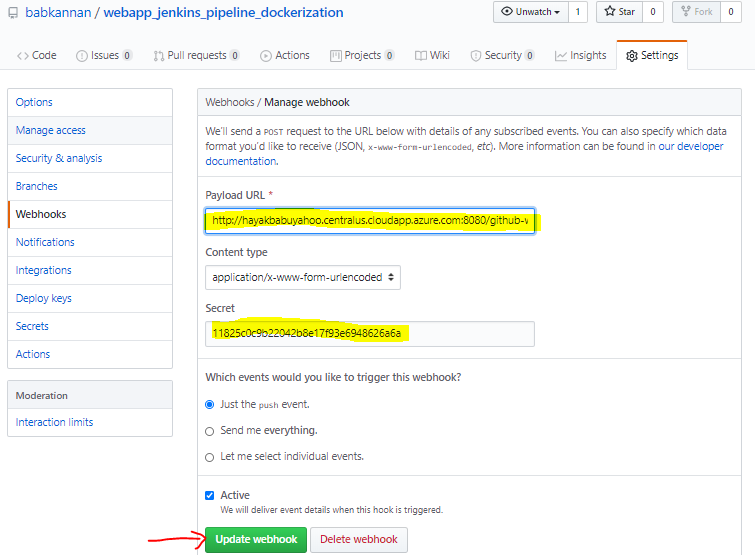
* Go to the user (here, **abc**) and click **Configure**
* Provide the name of the Token and generate the new Token and save the generated token (as highlighted in the screenshot below)
* Click **Save** the changes



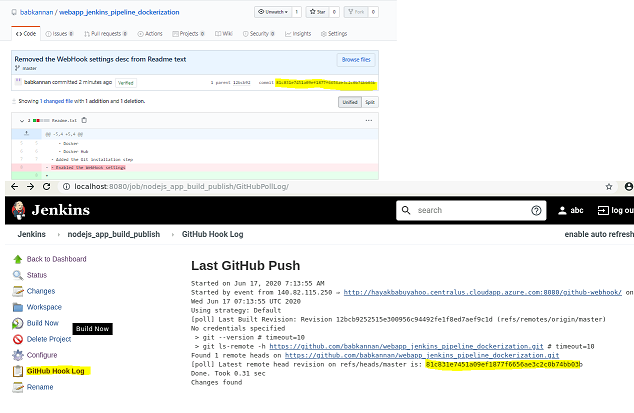
* Next step is to **Enable the WebHook** and add few configurations in the **GitHub** project settings
* Goto Project settings in the GitHub repository
* Click WebHook as shown in the below screenshot



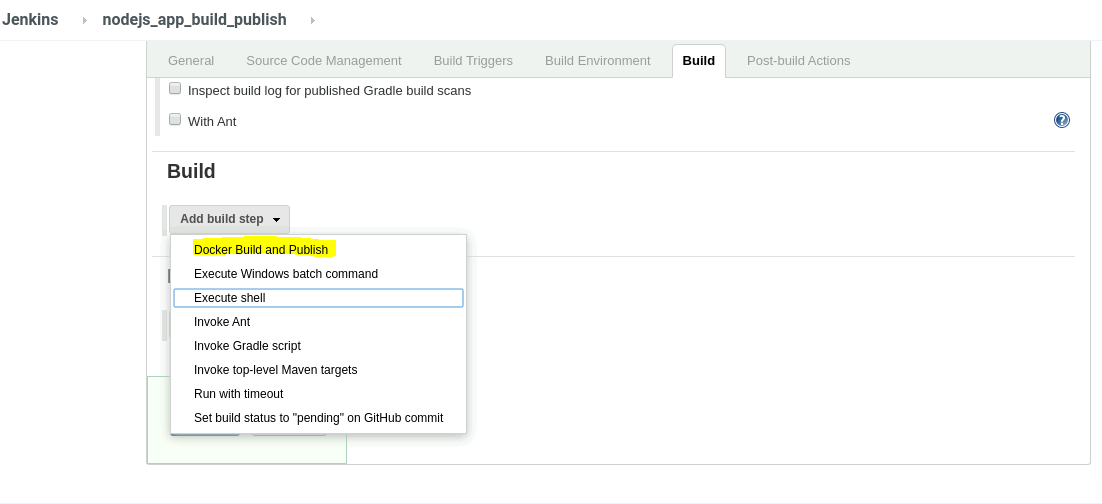
* Click Edit to update the WebHook
* Enter the “Secret” token generated in the previous steps in the “Secret” edit box as highlighted below in the screenshot
* Enter the Jenkins URL as highlighted below in the screenshot
* Click **Update webhook** to save the changes

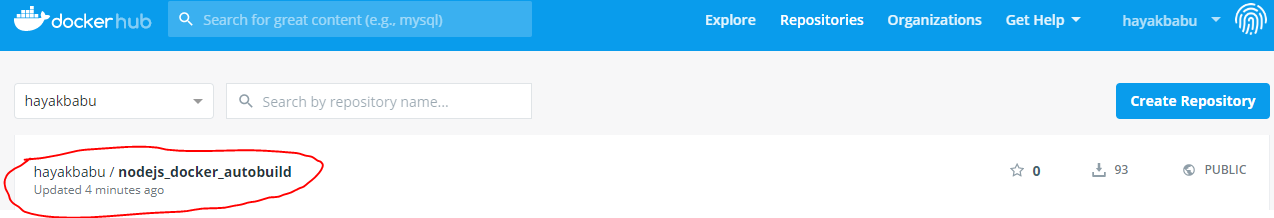


* Just to test the build job trigger execution, made some changes in the project and pushed the changes into the GitHub project repository
* The build job is triggered immediately in the Jenkins (without any wait) as shown in the below screenshot



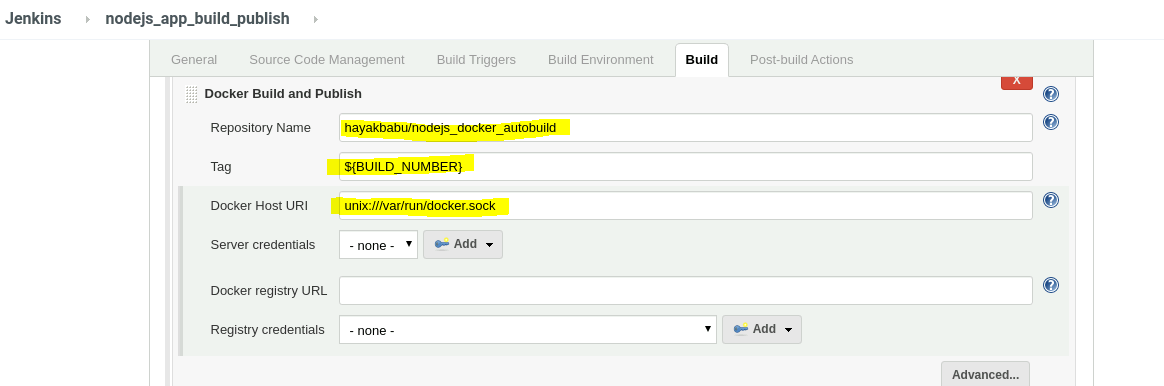
## **Docker Build and Publish Plugin**





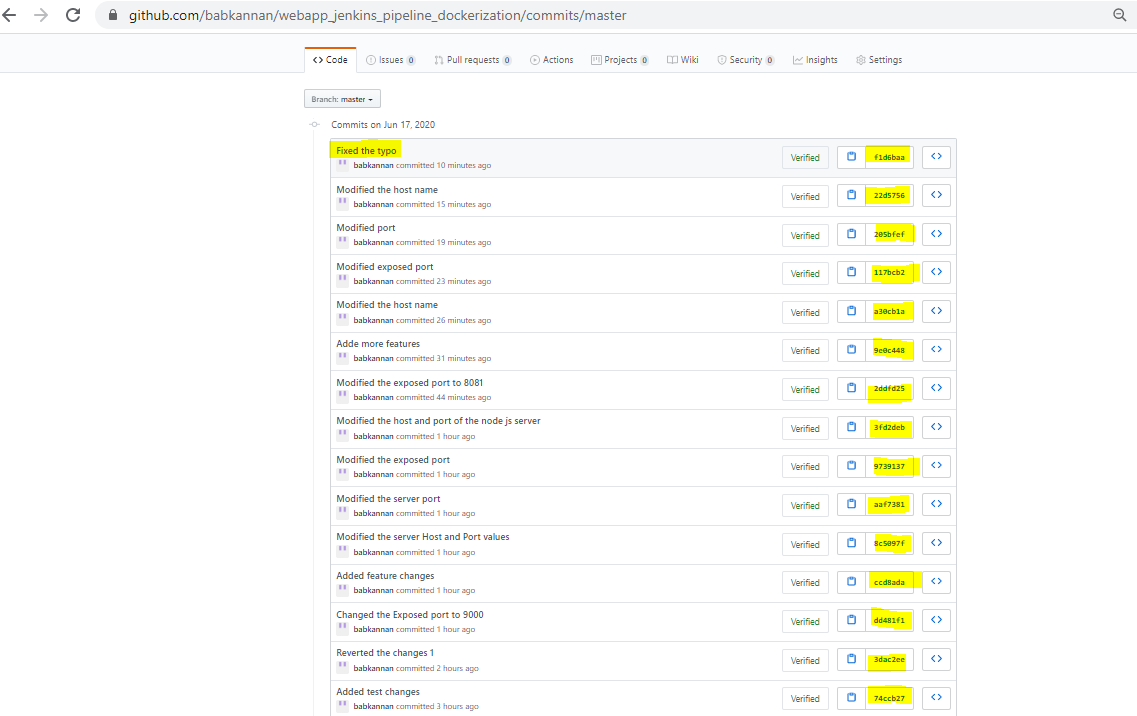
Also, adding the build tab, Fill-in the Build Number which Jenkins internally attach the Build number to the build.

Docker host URI as mentioned in the screenshot below.



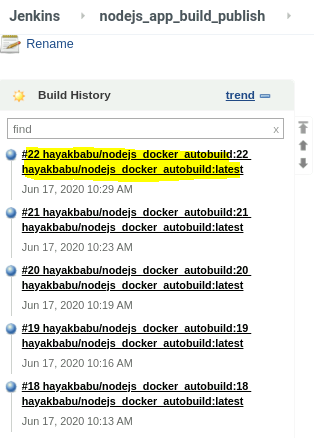
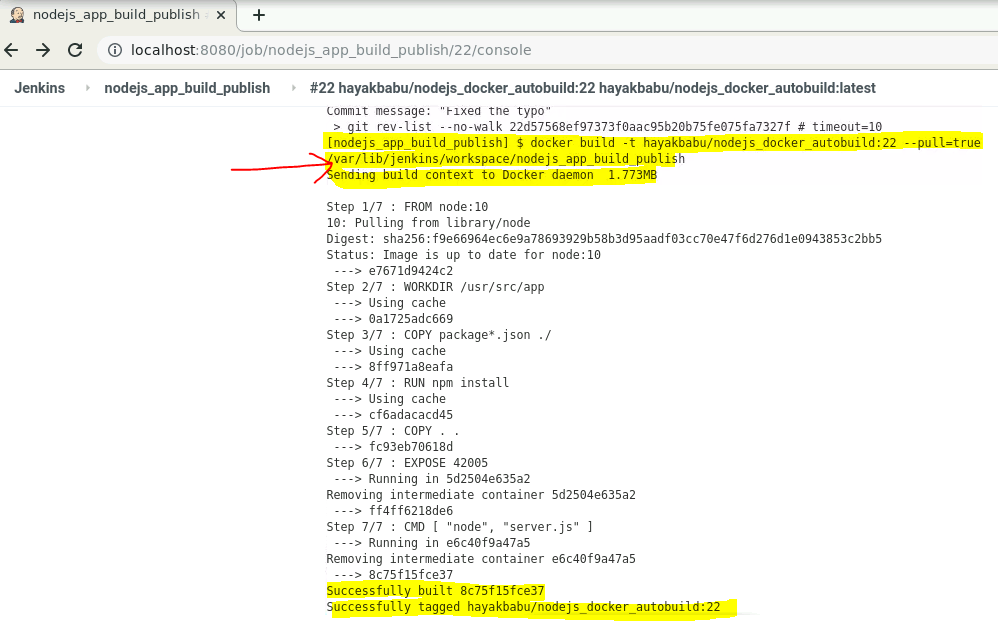
# **Track version change history in GitHub**

Below screenshot shows the source code version history in the GitHub



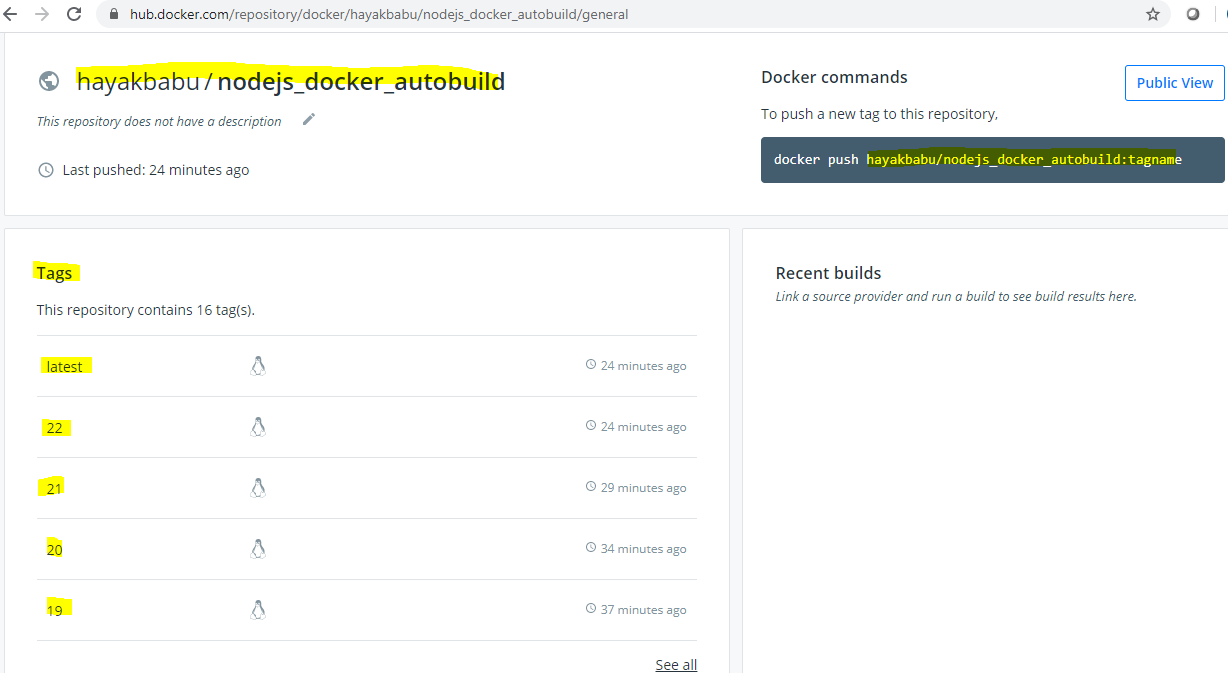
Whenever the changes are pushed into the GitHub via Git, the build will be triggered through the WebHook (configured in the Section 9).

The highlighted ones below where the application build context sends to the docker daemon to build the image. The below screenshot is based on the build #22

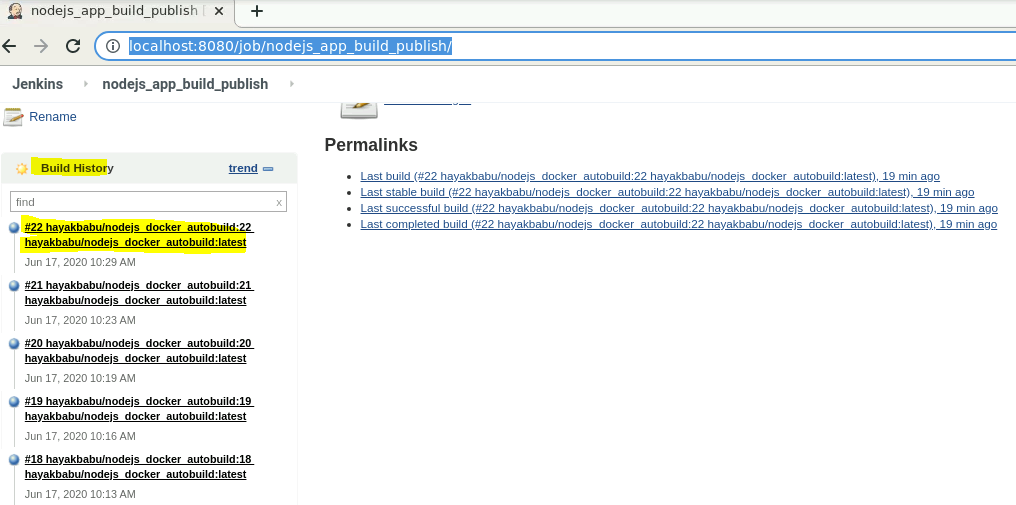
# **Availability of the application in the Docker Hub**

Below screenshot shows the availability of the Node JS application in the public Docker Hub



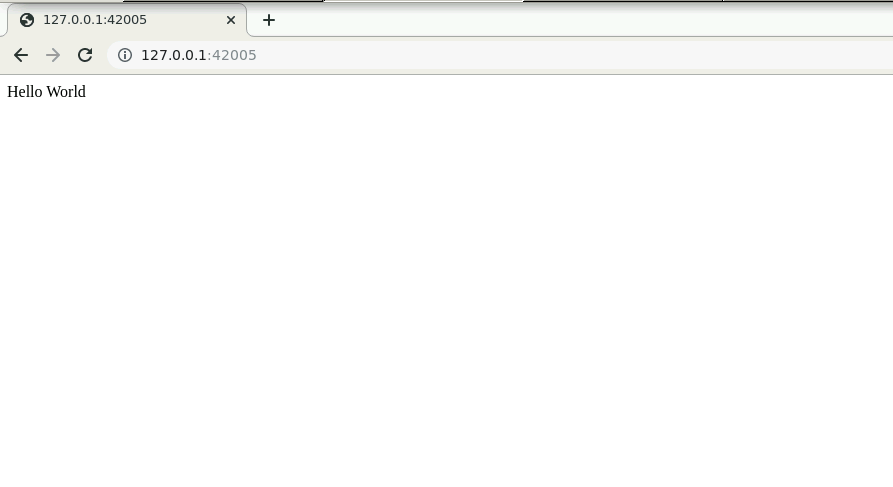
# **Track the build status of Jenkins for every increment of the project**

The below screenshot shows the build status of each build corresponds to the changes made in the Project and pushed into the GitHub.



# **Pull the Docker image and run it again**

Test the application in the browser with the exposed port and the result is displayed in the browser as shown in the below screenshot



Refer the below screenshot extracted from the Docker Hub

