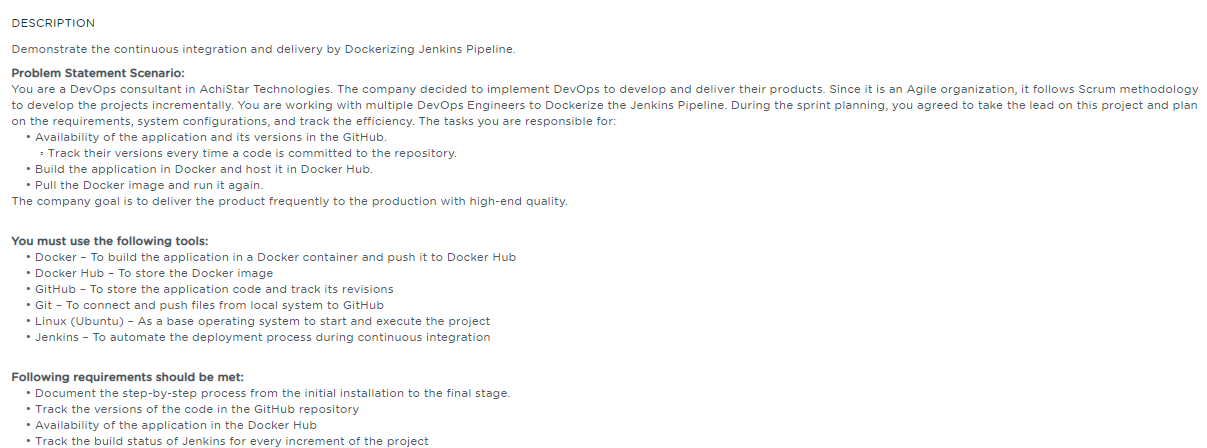
**Dockerizing Jenkins Pipeline - Assessment**



**Table of Contents**

[**1.** **Introduction** 3](#_Toc43305041)

[**2.** **Project Abstract** 3](#_Toc43305042)

[**3.** **Install Git** 3](#_Toc43305043)

[**4.** **GitHub account setup and Project creation** 3](#_Toc43305044)

[**4.1.** **GitHub account setup** 3](#_Toc43305045)

[**4.2.** **Login from Git local to remote GitHub** 4](#_Toc43305046)

[**4.3.** **Initialize Git and add the project into the local Git repo** 4](#_Toc43305047)

[**4.4.** **Create a Project repository in GitHub account** 5](#_Toc43305048)

[**4.5.** **Creation of SSH Key and adding it to GitHub** 7](#_Toc43305049)

[**5.** **Install and configure Jenkins** 8](#_Toc43305050)

[**5.1.** **Add a key to use Debian repository** 8](#_Toc43305051)

[**5.2.** **Make changes to /etc/opt/sources.list** 9](#_Toc43305052)

[**5.3.** **Update local package index** 9](#_Toc43305053)

[**5.5.** **Run Jenkins in browser** 9](#_Toc43305054)

[**5.6.** **Install CloudBees Docker Build and Publish Plugin** 11](#_Toc43305055)

[**6.** **Install and setup Docker** 12](#_Toc43305056)

[**6.1.** **Install Docker** 13](#_Toc43305057)

[**6.2.** **Start Docker** 13](#_Toc43305058)

[**6.3.** **Verify Docker** 14](#_Toc43305059)

[**7.** **DockerHub Readiness** 14](#_Toc43305060)

[**7.1.** **Setup Docker account** 14](#_Toc43305061)

[**7.2.** **Login to Docker account** 15](#_Toc43305062)

[**8.** **CI/CD configuration in Jenkins** 16](#_Toc43305063)

[**8.1.** **Create a new build job** 16](#_Toc43305064)

[**8.2.** **Docker Build and Publish Plugin** 22](#_Toc43305065)

[**9.** **Track version change history in GitHub** 23](#_Toc43305066)

[**10.** **Availability of the application in the Docker Hub** 24](#_Toc43305067)

[**11.** **Track the build status of Jenkins for every increment of the project** 24](#_Toc43305068)

[**12.** **Pull the Docker image and run it again** 25](#_Toc43305069)

[**13.** **References** 26](#_Toc43305070)

# **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

The scope of this assessment is assumed to build and publish the docker image using the “CloudBees Docker Build and Publish” Plugin

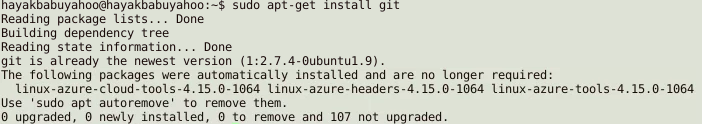
We can also use the Jenkins Pipeline script to implement the same feature for the better readability of Cloning the Project repository, Build the project, Test the Project and Deploy application. However, I didn’t cover under this assessment.

# **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

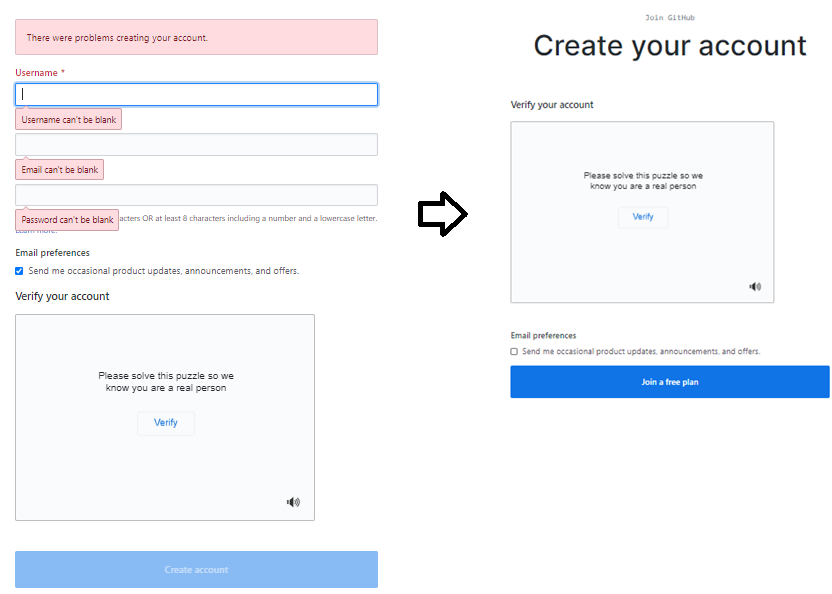


# **GitHub account setup and Project creation**

## **GitHub account setup**

**About GitHub**: It is a web-based hosting service for version control using Git. It offers plans for public and private repositories. You can add multiple projects by creating multiple public repositories. In this section, you will only demonstrate on the public repository and its usage.

Navigate to **https://github.com/** and click on Sign up for GitHub. Enter the details and click on **Create account**.

****

Click **Join a free plan** to create a free plan. You will receive an email to confirm your account. It is important to confirm your account before you use GitHub. Once confirmed, your GitHub account is set up successfully.

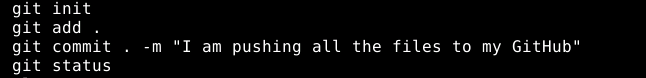
## **Login from Git local to remote GitHub**

Open the terminal in your lab and execute the below commands by replacing <your\_Email\_Id with your registered email address in GitHub and <Your Username> with your GitHub username. This basically configures the Git with your user account

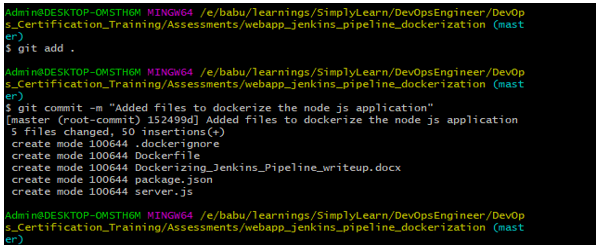


## **Initialize Git and add the project into the local Git repo**

Since the project (here, Node JS application) files are to be pushed, first initialize a .git folder inside the directory by executing the git init command followed by other commands as shown below:



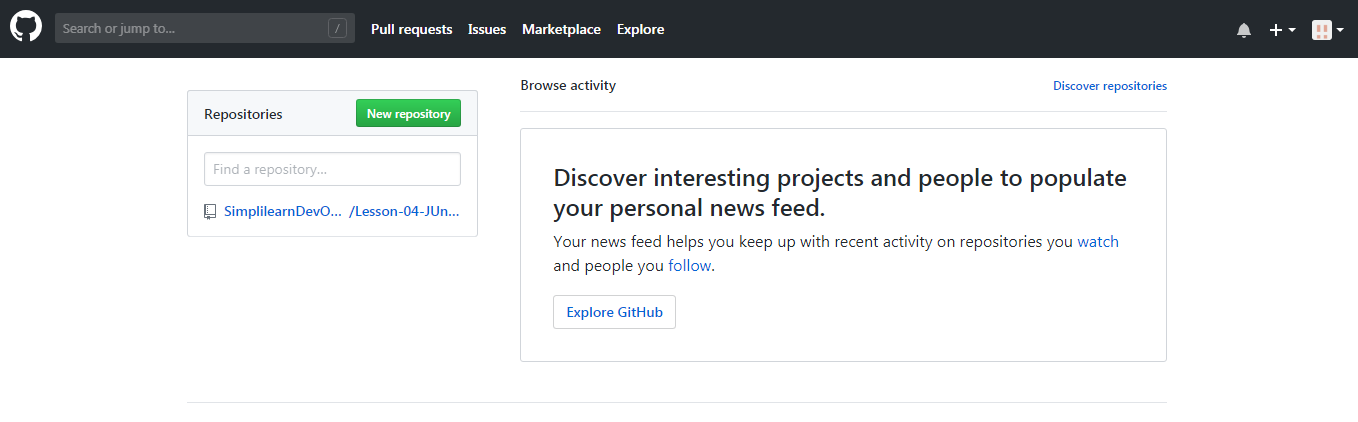
Please follow the below process for step-by-step confirmation of each command execution.



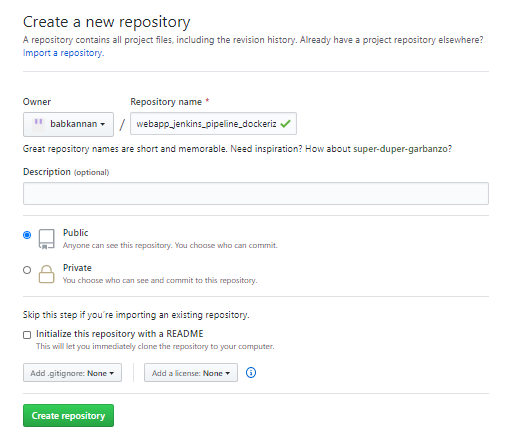
The above commands stage all the project files and pushed into the local Git repository.

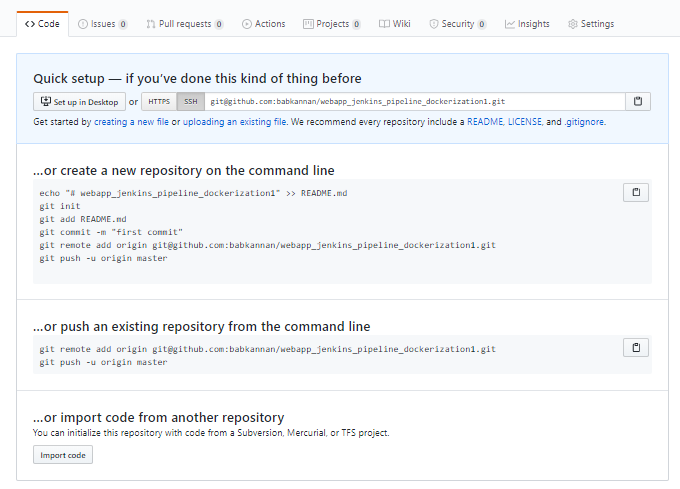
## **Create a Project repository in GitHub account**

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.**



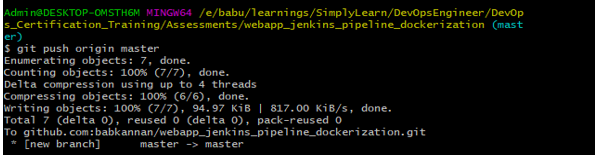
You will be redirected to a quick guide page and you will be navigated automatically inside the directory you have created. 

Since a repository is already created, “…**or create a new repository on the command line**” should be skipped. Click on **SSH** to change the instructions from **HTTPS** to **SSH**.

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 origin master |

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**

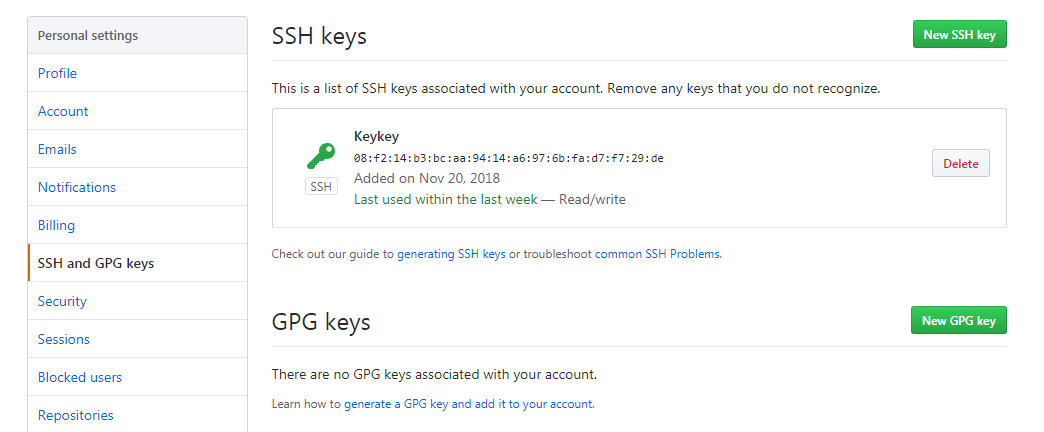
Switch the current directory to ssh by executing below command.

cd ~/.ssh

Generate an RSA key for the registered email Id. (An example is available below)

|  |
| --- |
| ssh key-gen -t rsa -C “<your email\_id>”  gedit id\_rsa.pub |

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.

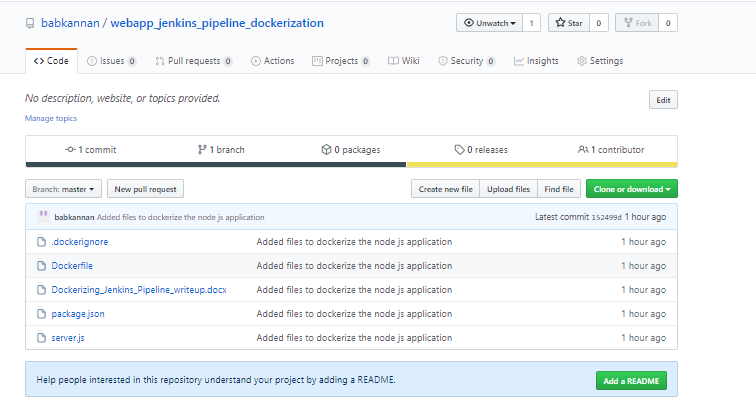


In the terminal, execute **ssh-add** to save the key and link it with local git.

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**

Go to [**https://pkg.jenkins.io/debian-stable/**](https://pkg.jenkins.io/debian-stable/) to understand how to install Jenkins on a Ubuntu/Debian machine.

## **Add a key to use Debian repository**

To use the Debian repository of Jenkins to automate installation and upgrade, first add the key to your system using the following command:

|  |
| --- |
| wget -q -O – <https://pkg.jenkins.io/debian-stable/jenkins.io.key> | sudo apt-key add – |

## **Make changes to /etc/opt/sources.list**

Open the file **/etc/apt/sources.list** using sudo command

|  |
| --- |
| sudo vi /etc/apt/sources.list |

Now add the following entry in the /etc/apt/sources.list command

|  |
| --- |
| deb [https://pkg.jenkins.io/debian-stable binary/](https://pkg.jenkins.io/debian-stable%20binary/) |

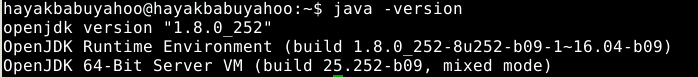
## **Update local package index**

|  |
| --- |
| sudo apt-get update |

Make sure you have jdk installed just by typing the below command

|  |
| --- |
| java -version |

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



If not, install it using the following command:

|  |
| --- |
| sudo apt-get install openjdk-8-jdk |

Finally, install Jenkins by executing the below command

|  |
| --- |
| sudo apt-get install jenkins |

## **Run Jenkins in browser**

Once Jenkins is installed, open a browser on your local machine and enter the

URL: x.x.x.x:8080

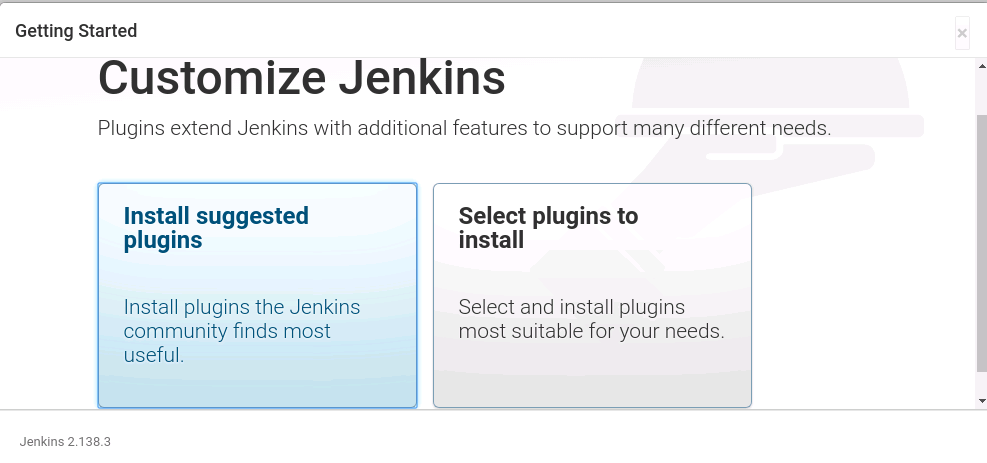
replacing x.x.x.x with the external IP address of your virtual machine or run(or localhost)

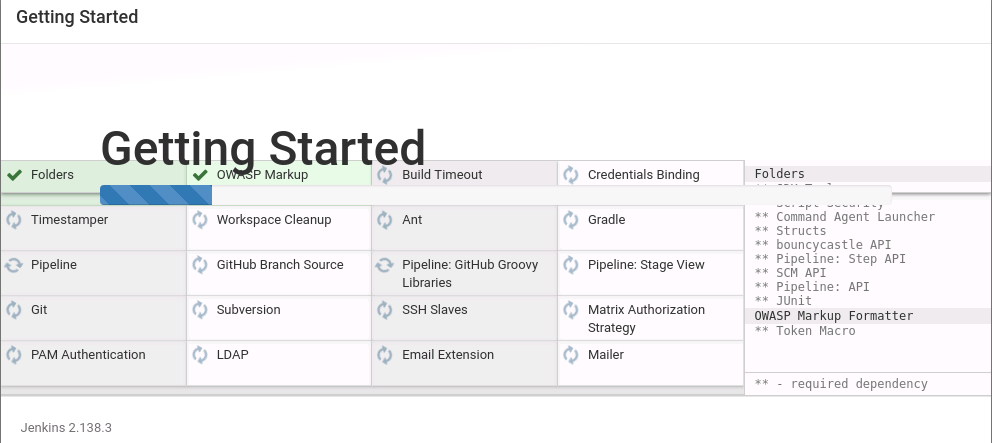
You will need to enter the admin password.

To find the password, type the following command:

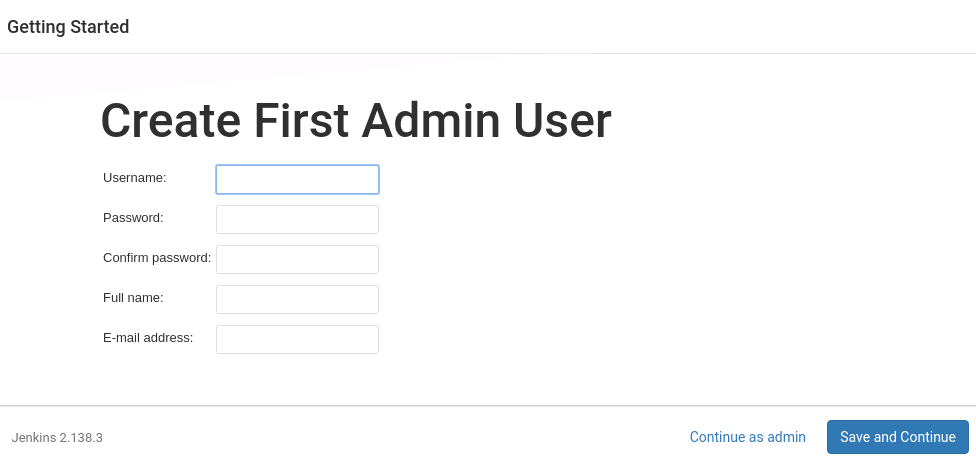
|  |
| --- |
| sudo cat /var/lib/jenkins/secrets/initialAdminPassword |

Use the string of numbers as password. Once you are logged in, you will be redirected to the page below:

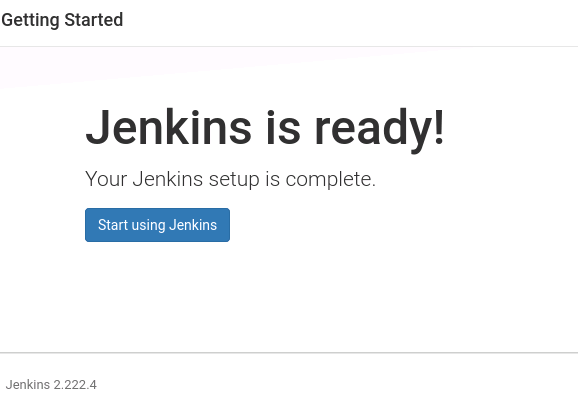
 Select **Install suggested plugins**. You will be redirected to the page below:



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.



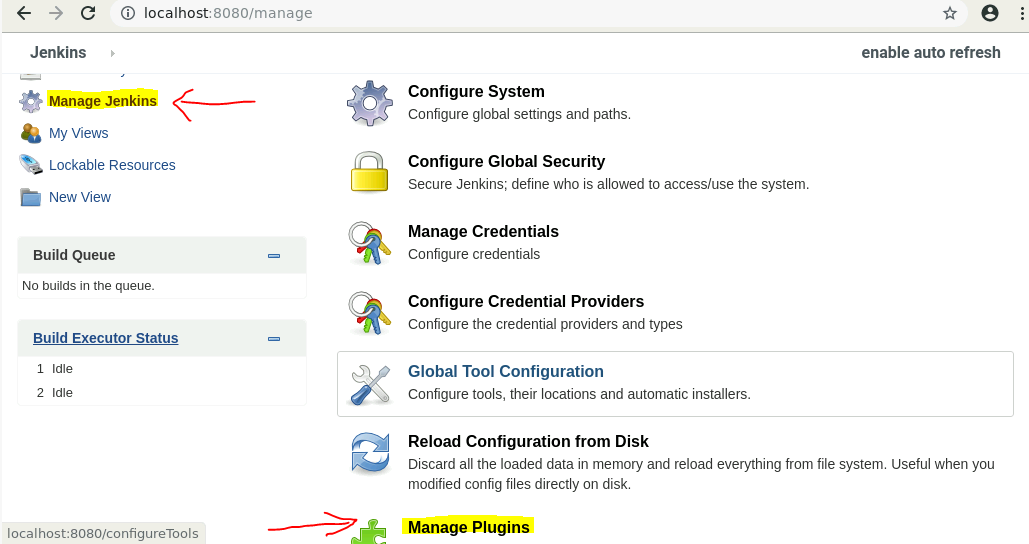
## **Install CloudBees Docker Build and Publish Plugin**

One of the **Project requirement is to build the application in a docker container and publish it to the Docker Hub**. To fulfill this requirement, we can use one of the option like “CloudBees Docker Build and Publish” Plugin.

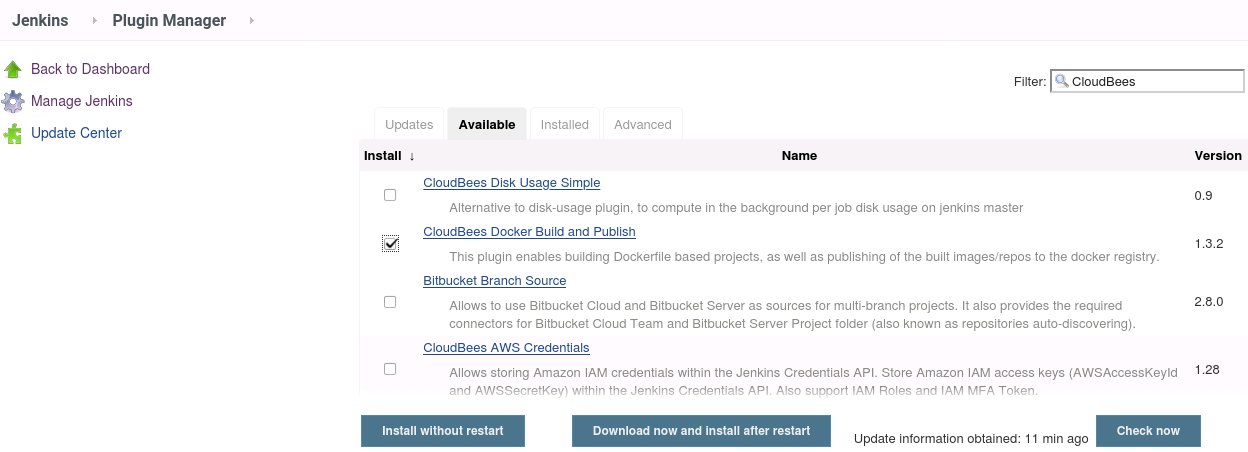
This plugin enables building the Dockerfile based projects, as well as publishing the built images to the Docker Hub.

Steps to install the **CloudBees Docker Build and Publish**

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

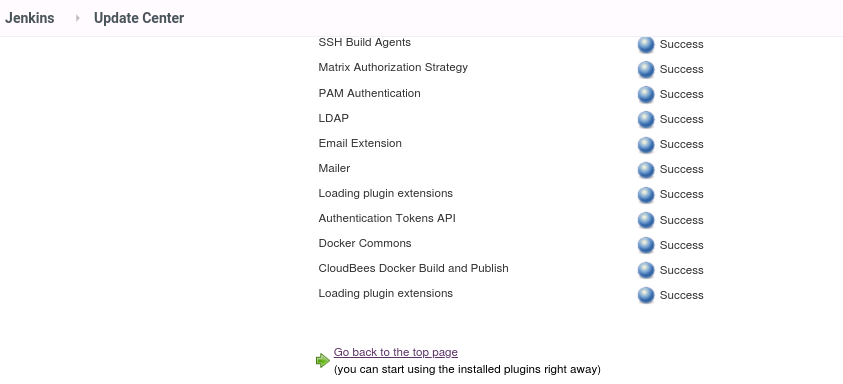


* Click the “Available” tab and enter the text like “CloudBees” in the Search filter as shown below in the screenshot which displays the “CloudBees Docker Build and Publish” plugin



* 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



**Note:**

Refer the Plugin usage under the Project Continuous Integration process steps (Refer: Step)

# **Install and setup Docker**

If you use Ubuntu Trusty, Wily, or Xenial, install the linux-image-extra kernel package:

|  |
| --- |
| sudo apt-get update –y && sudo apt-get install –y linux-image-extra-$(uname –r) |

## **Install Docker**

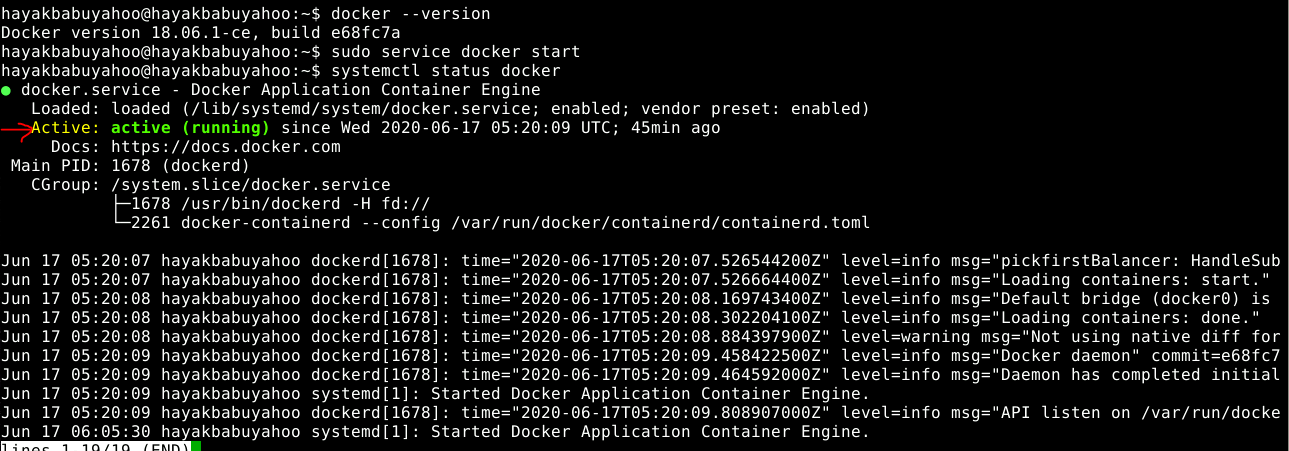
|  |
| --- |
| sudo apt-get install docker-engine -y |

## **Start Docker**

|  |
| --- |
| sudo service docker start |

Verify the docker running status by executing the following command and if the docker is running, then the status will be shown as below:

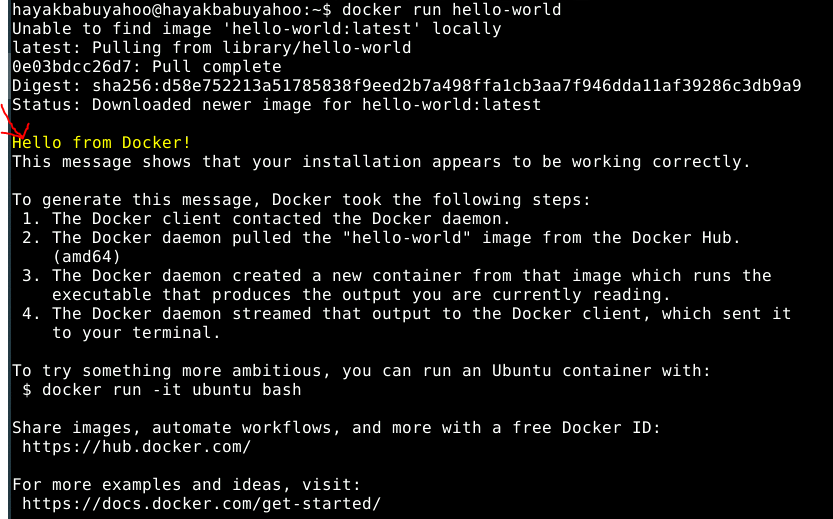
|  |
| --- |
| systemctl status docker |



## **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



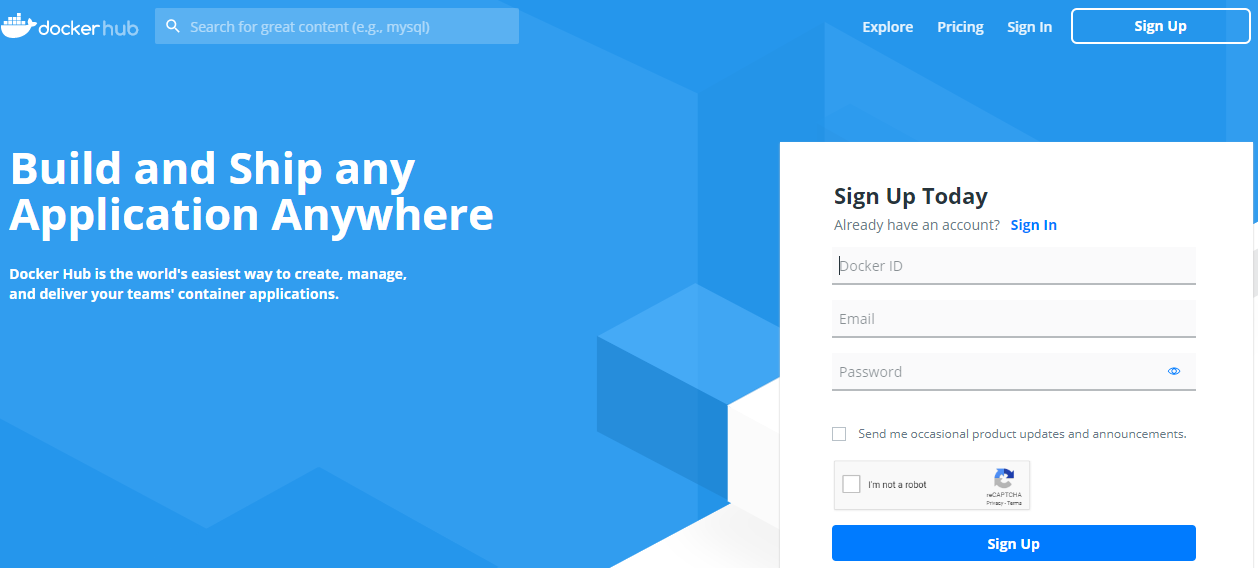
# **DockerHub Readiness**

[Docker Hub](https://hub.docker.com/) is a service provided by Docker for finding and sharing container images with the team. It provides the following significant features:

* [**Repositories**](https://docs.docker.com/docker-hub/repos/): Push and pull container images.
* [**Builds**](https://docs.docker.com/docker-hub/builds/): Automatically build container images from GitHub and BitBucket and push them to Docker Hub.

## **Setup Docker account**

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



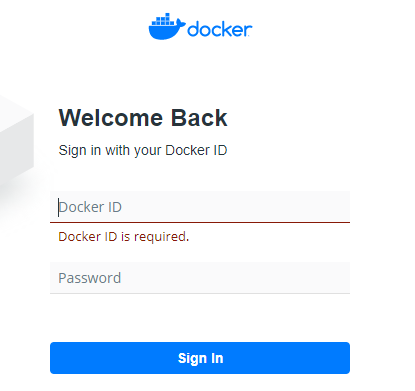
Follow the below steps to create a new docker account

* Enter a username that is also your Docker ID.
  + Your Docker ID must be between 4 and 30 characters long, and can only contain numbers and lowercase letters.
* Enter a unique, valid email address.
* Enter a password between 6 and 128 characters long.
* Click Sign up.
  + Docker sends a verification email to the address you provided.
* Click the link in the email to verify your address.

**Note**: You cannot log in with your Docker ID until you verify your email address.

## **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



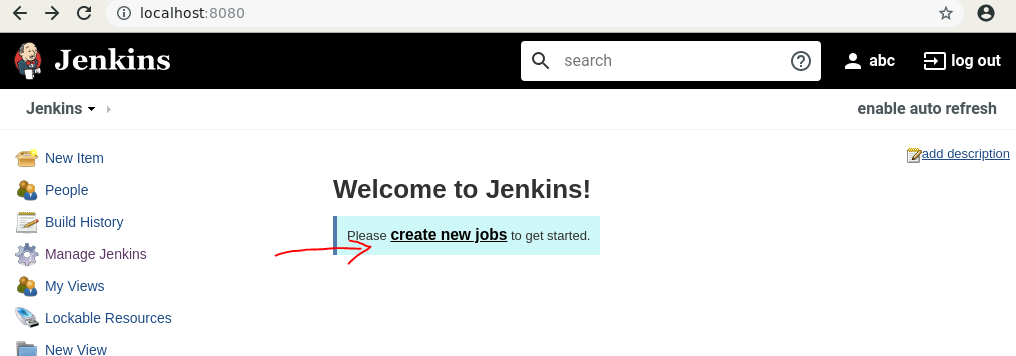
Refer: For more details, refer <https://docs.docker.com/docker-id/>

# **CI/CD configuration in Jenkins**

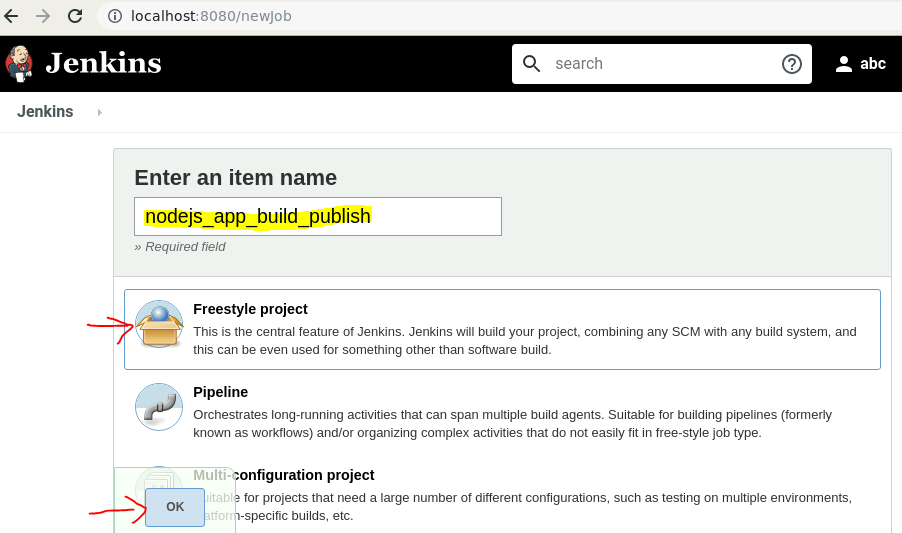
Continuous Integration and Continuous Deployment Configuration in Jenkins

## **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

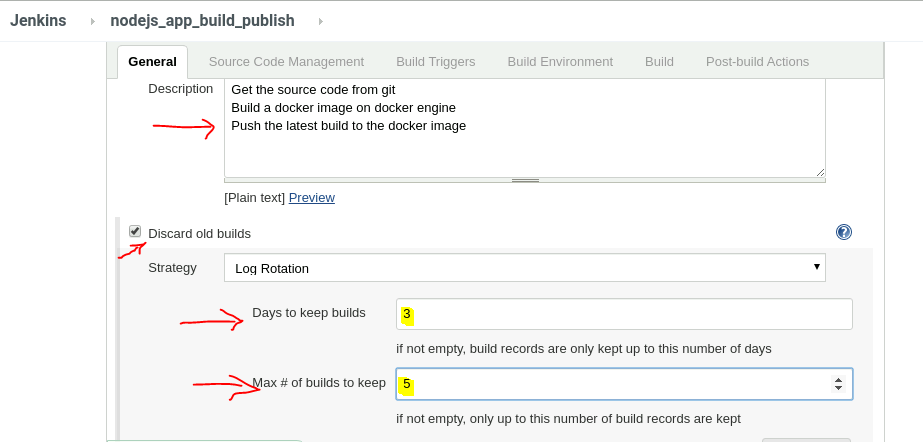


**Step 2**: Select “Freestyle Project” and enter an item name as “nodejs\_app\_build\_publish” and click OK

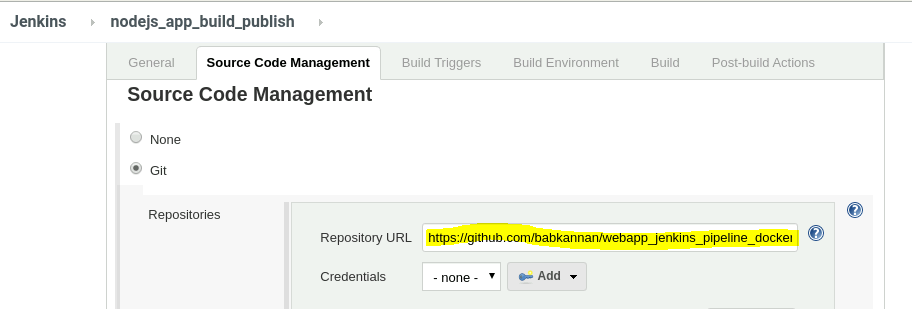


**Step 3**: As the best practice, In the **General** tab of the configuration, fill-in the below fields (as shown in the screenshot below)

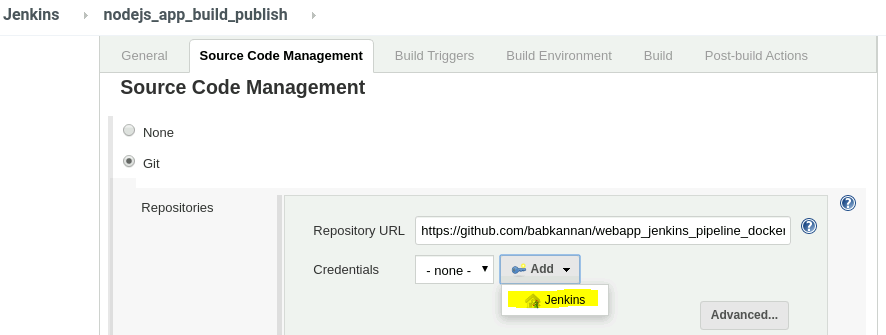
* Description
* Discard old-builds
* Days to keep builds
* Max # builds to keep



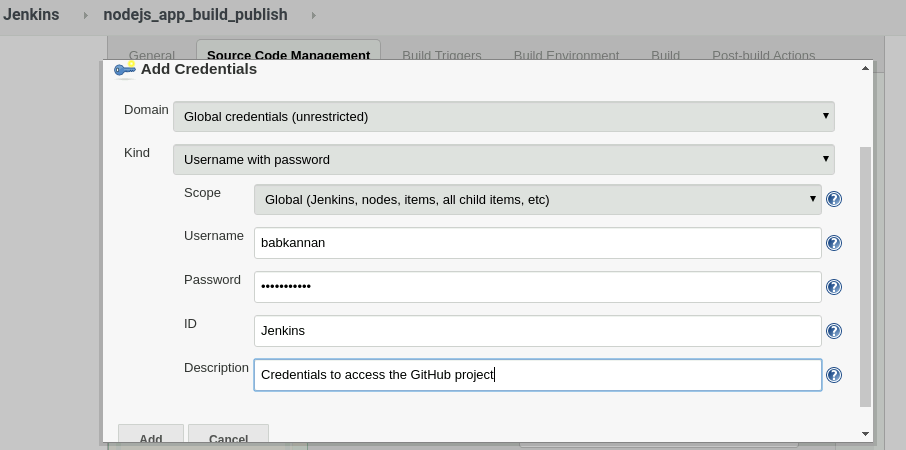
**Step 4:**  In the **Source Code Management** tab of the configuration, fill-in the project repository URL field (as shown in the screenshot below). Copy the Project URL from the GitHub web repository



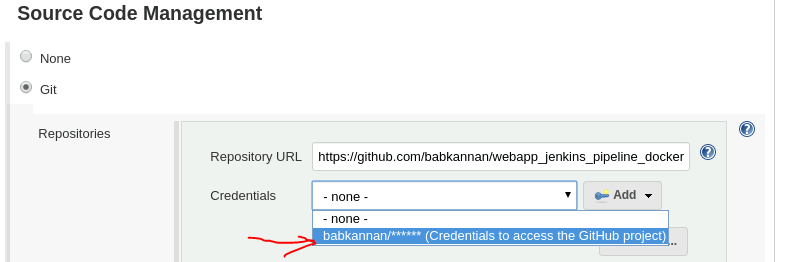
**Step 5:** To access the project in the GitHub securely from the Jenkins, add the project access credentials information. This can be achieved by clicking **Add | Jenkins**



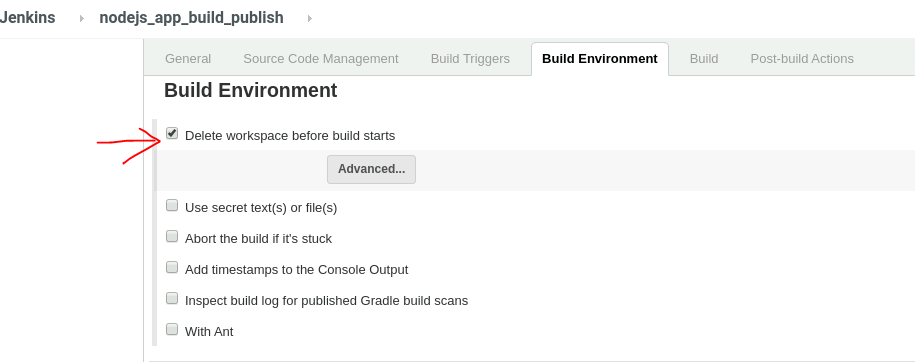
**Step 6:** Fill the GitHub access credentials and click **Add** as shown in the below screenshot



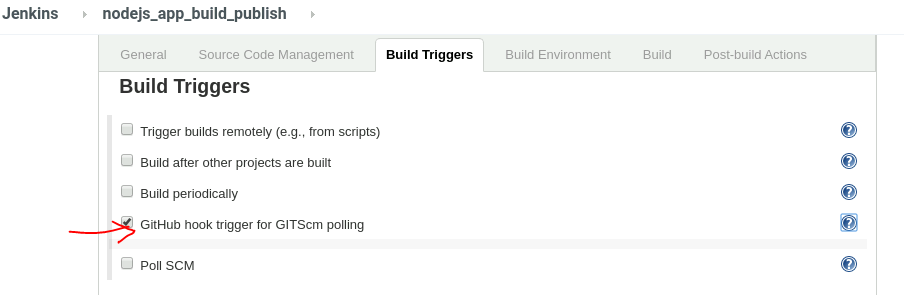
Once it’s successfully added, you can verify the credential information gets added as shown in the below screenshot



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

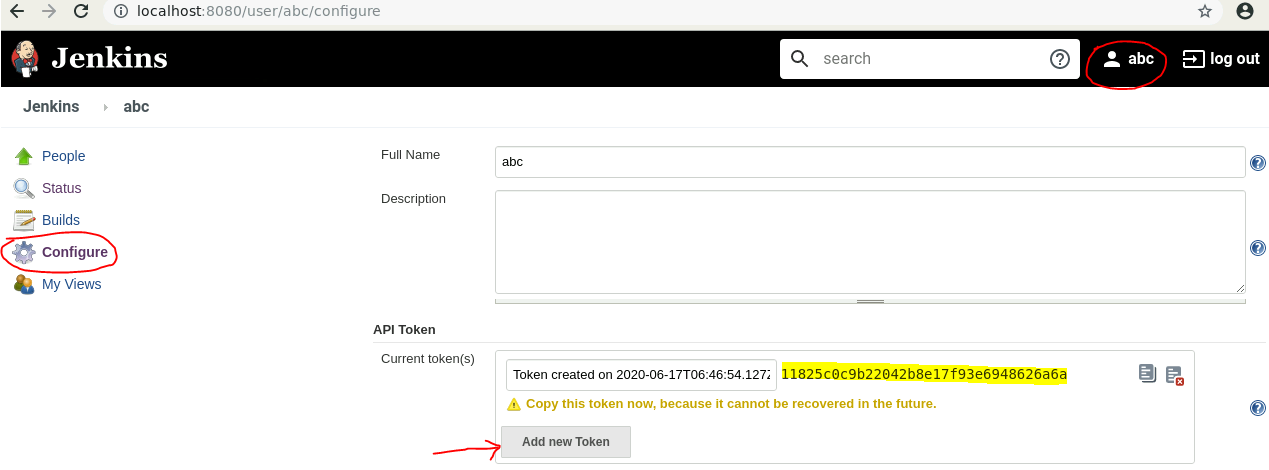


**Step 8:** Under the **Build Triggers** tab of the configuration, select the “**GitHub hook trigger for GitSCM polling**” option as shown below in the screenshot: This option basically enables in triggering the build whenever changes are pushed into the GitHu

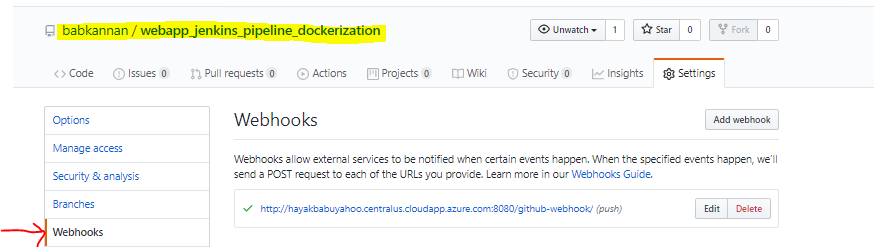


Enabling the GitHub hook itself is not enough, few steps needs to be performed to completely enable this feature.

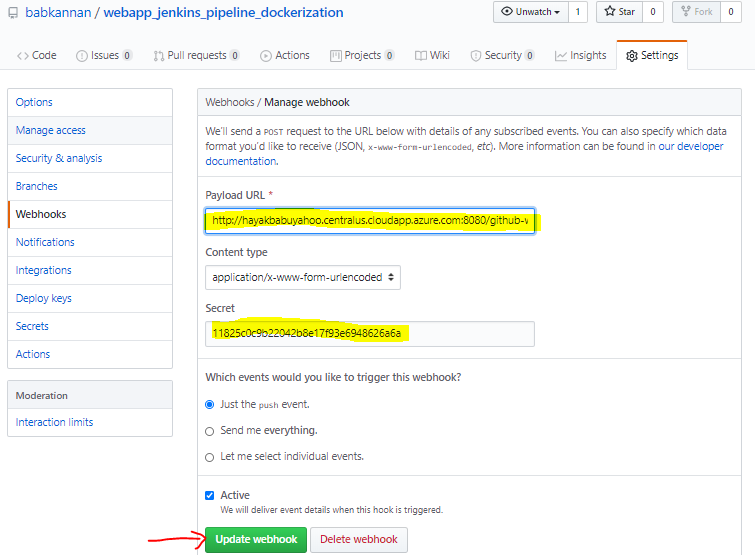
* Go to the user (here, **abc**) and click **Configure**
* Provide the name of the Token and generate the new Token and save the generated token (as highlighted in the screenshot below)
* Click **Save** the changes



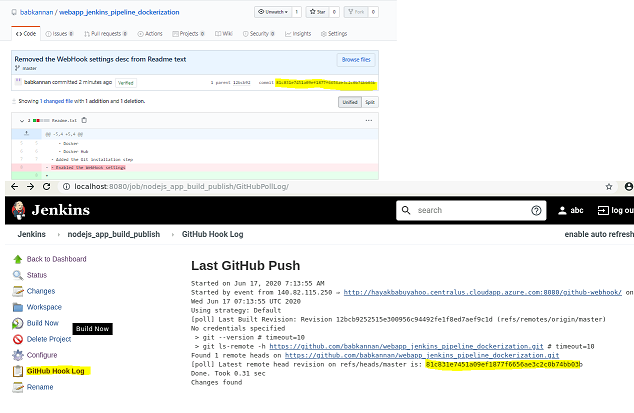
* Next step is to **Enable the WebHook** and add few configurations in the **GitHub** project settings
* Goto Project settings in the GitHub repository
* Click WebHook as shown in the below screenshot



* 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

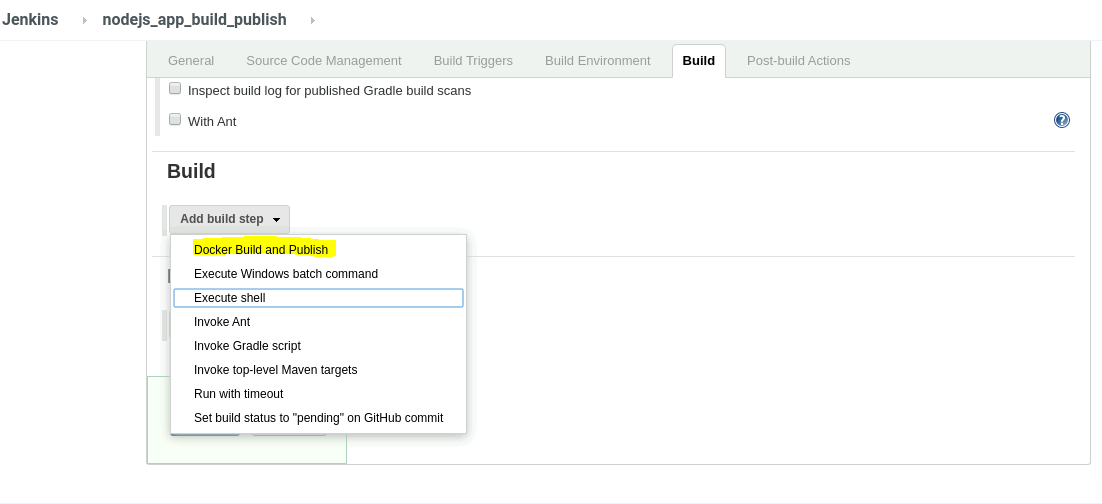


* Just to test the build job trigger execution, made some changes in the project and pushed the changes into the GitHub project repository
* The build job is triggered immediately in the Jenkins (without any wait) as shown in the below screenshot

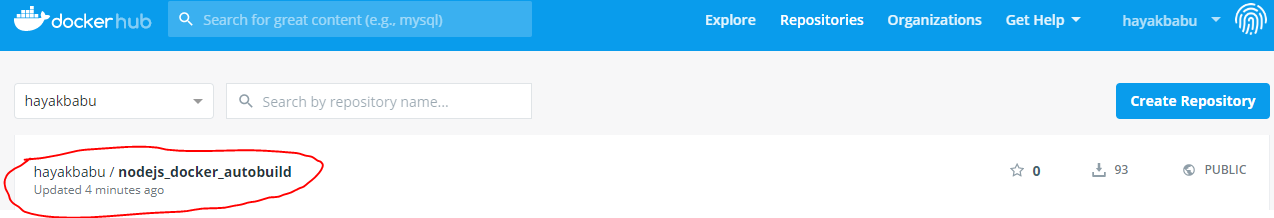


## **Docker Build and Publish Plugin**

**Step 1:** Now it’s time to use the Docker Build and Publish Plugin. To make use of this, perform the following steps:

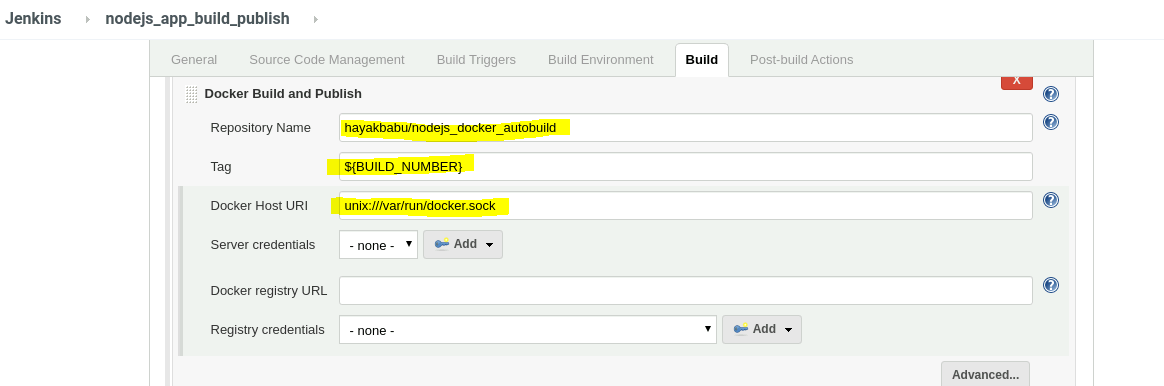
Go to “Build” tab of configuration. Under the **Build** tab of the configuration, select “Add build step’ and **Choose Docker Build and Publish** as shown below in the screenshot

**Step 2:** Fill-in the docker repository name (Refer the Docker Hub account for the Repository name as shown below in the screenshot) in which we need to push the images



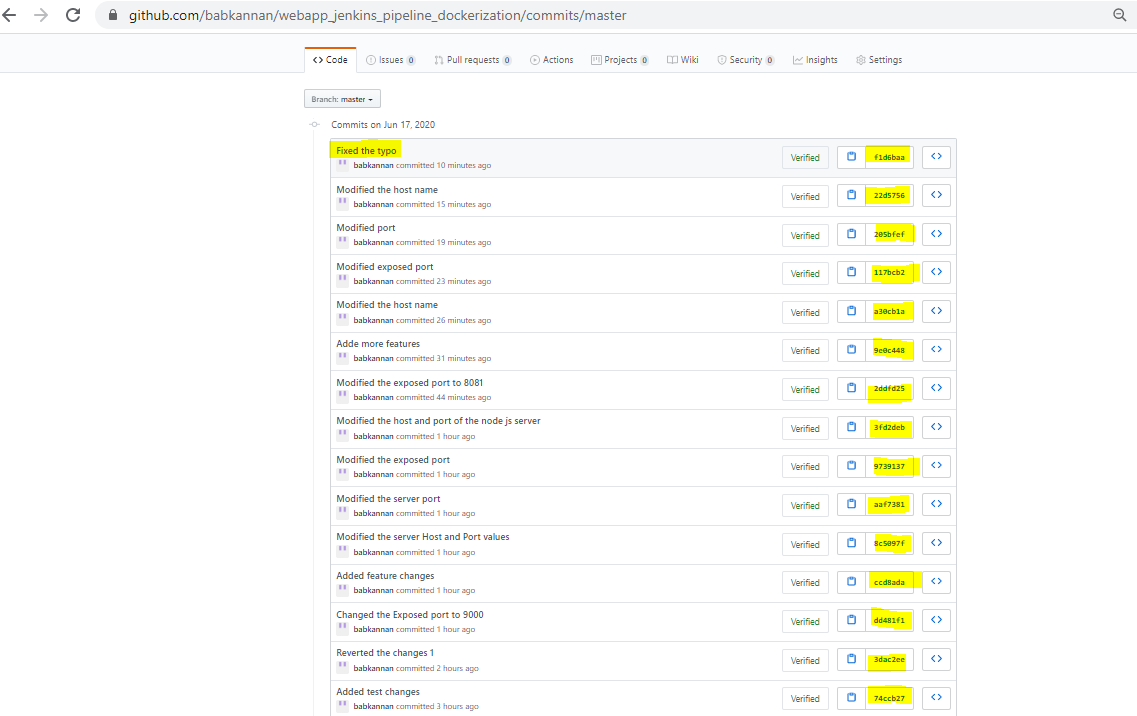
Also, adding the build tab, Fill-in the Build Number which Jenkins internally attach the Build number to the build.

Docker host URI as mentioned in the screenshot below.



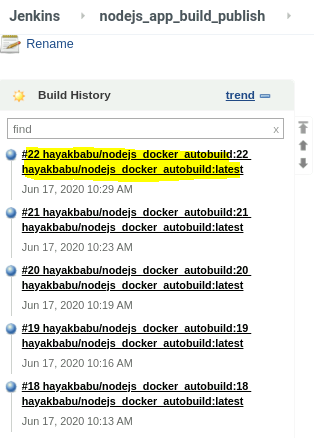
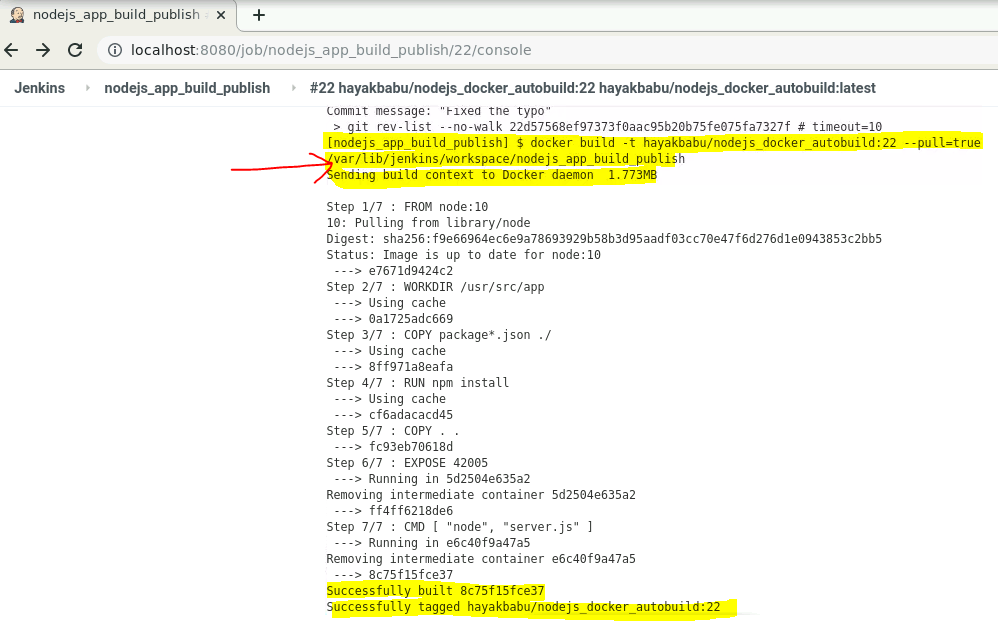
# **Track version change history in GitHub**

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**

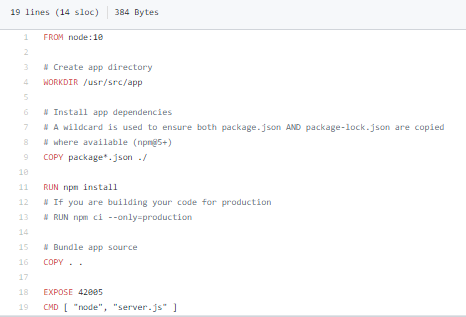
The sequence when the image gets uploaded to the docker hub

1. Changes pushed into the GitHub via Git
2. Build gets triggered through Git Webhook

The below steps is being performed through the CloudBees Docker Build and Publish Plugin

1. After the build is successful, build context (Dockerfile) is sent to the docker daemon to build the image (based on the changes, image layers will be built accordingly)

The below **Dockerfile is being used** to build the image for this project

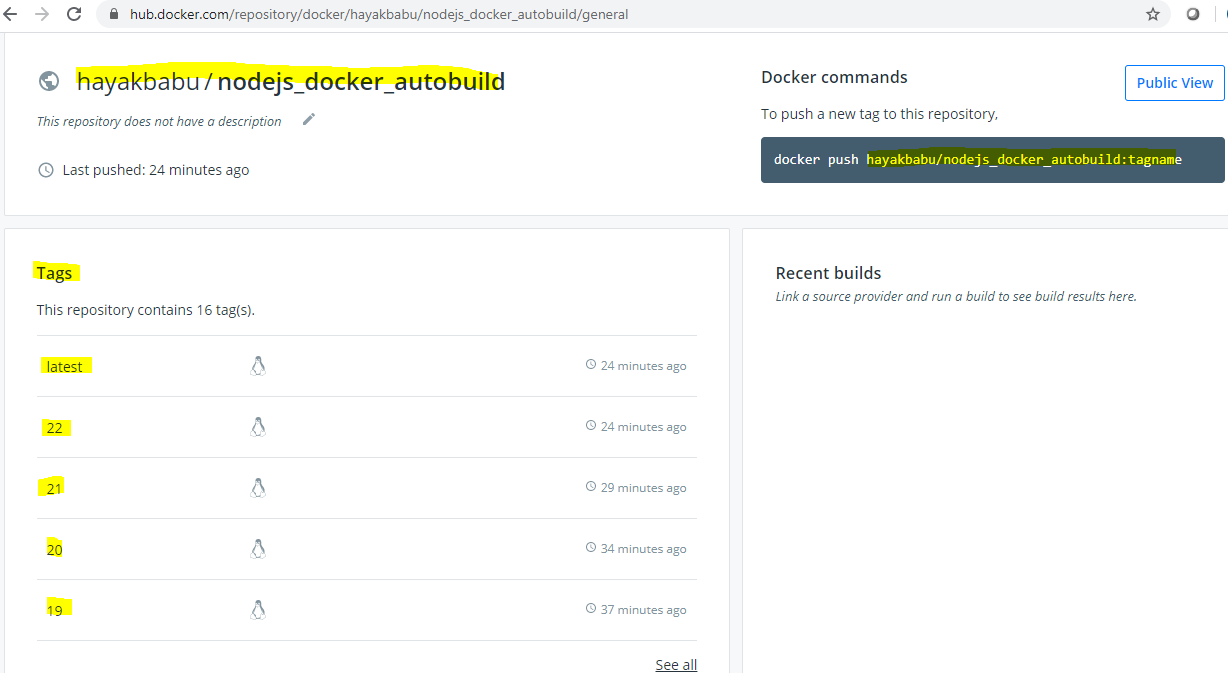


1. Docker image is built with the build tag (Build Number generated from the Jenkins)

Docker image is uploaded to the Docker Hub

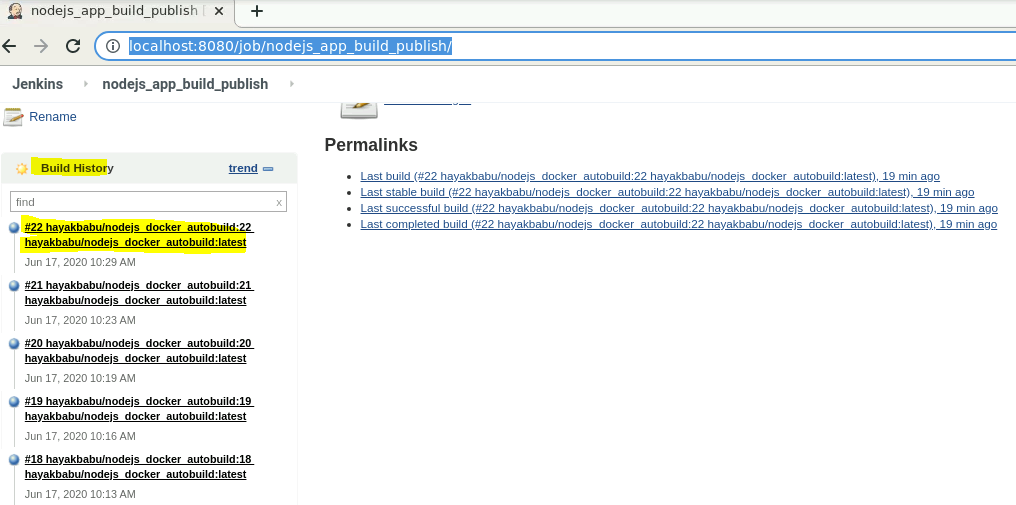
Below screenshot shows the availability of the Node JS application in the public Docker Hub

Reference: <https://hub.docker.com/repository/docker/hayakbabu/nodejs_docker_autobuild/general>



# **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.



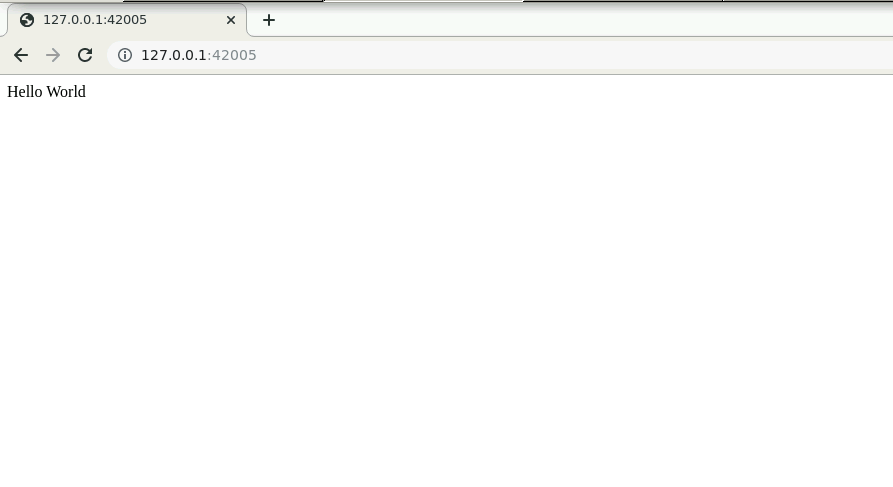
# **Pull the Docker image and run it again**

Below command is being used for pulling the image and perform the test.

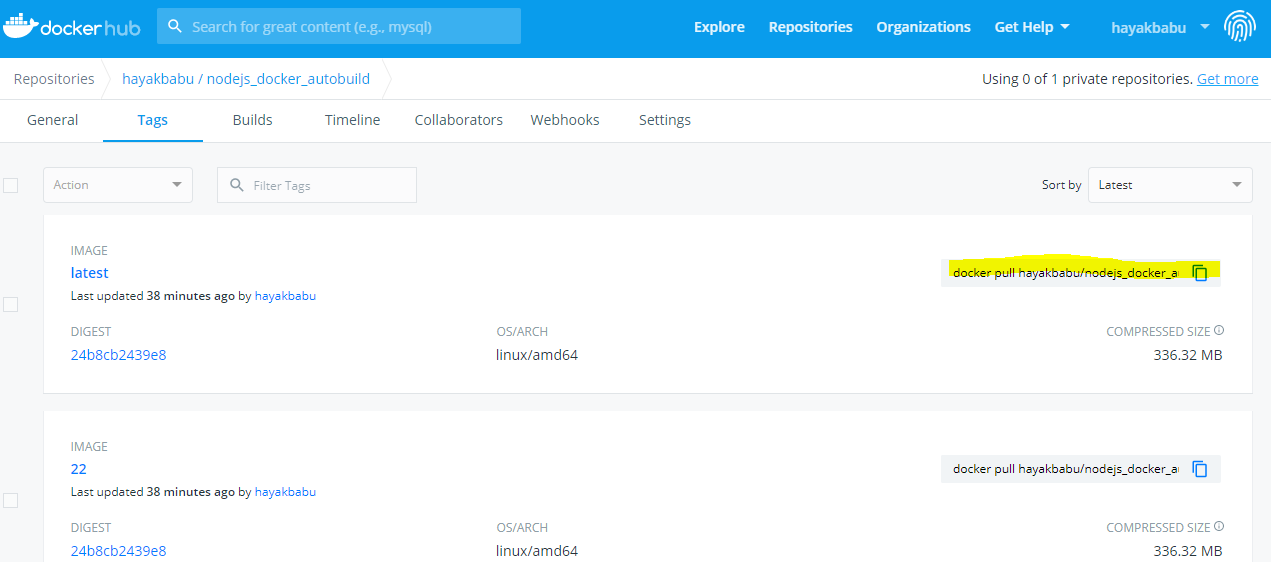
We can run the docker container in the interactive mode. In this mode a log of the running application will be displayed.

|  |
| --- |
| docker pull hayakbabu/nodejs\_docker\_autobuild:latest  docker run –d –p 42005:42005 hayakbabu/nodejs\_docker\_autobuild:latest |

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



# **References**

* Test project used to build the application is available in the GitHub

Reference: <https://github.com/babkannan/webapp_jenkins_pipeline_dockerization/>

* Jenkins configuration details, please review the below link of the Virtual machine

<https://hayakbabuyahoo.centralus.cloudapp.azure.com:42001/guacamole/#/client/REVGQVVMVABjAGRlZmF1bHQ=?username=guacadmin&password=guacadmin>

* The application images are published in the Docker Hub

<https://hub.docker.com/repository/docker/hayakbabu/nodejs_docker_autobuild/>