A Step-by-Step Guide to Jenkins CI/CD with GitHub, DockerHub and Kubernetes

By

Sauvik Maiti

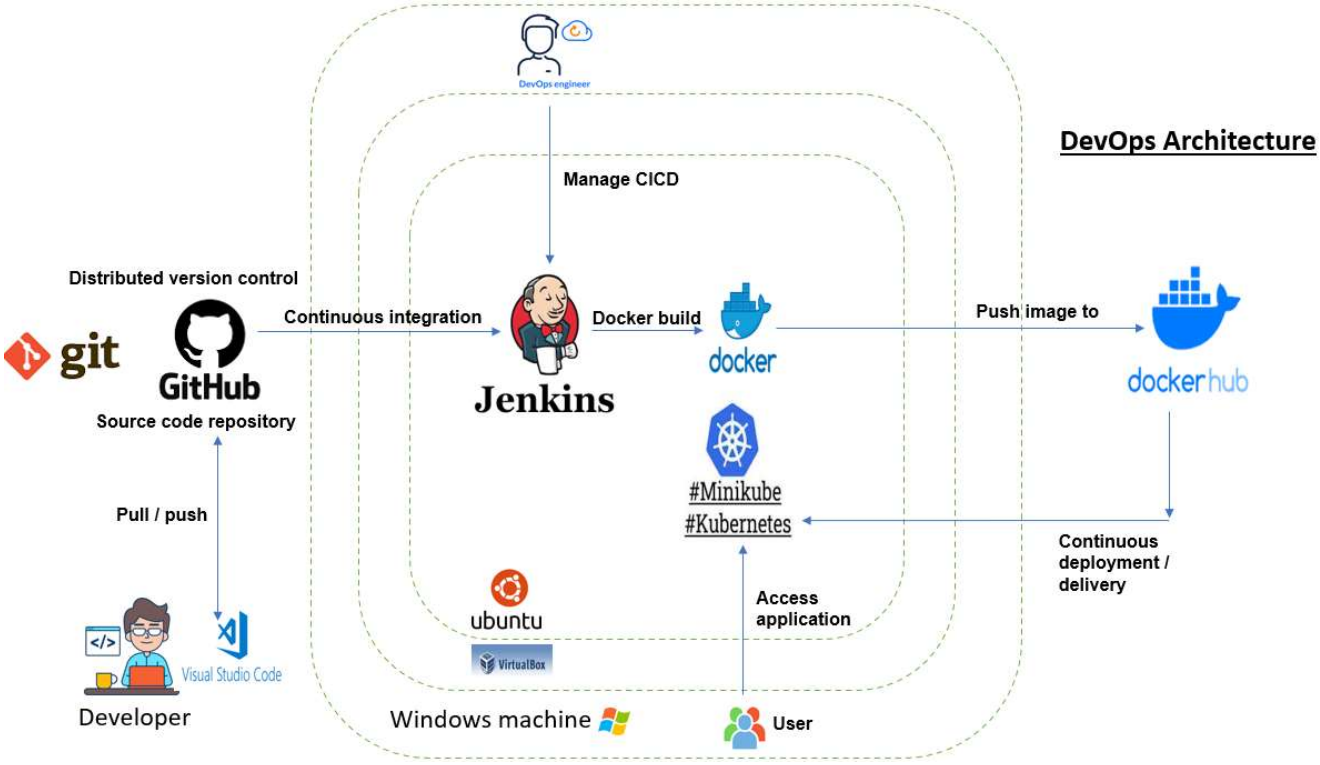
Table of Contents

Overall Architecture 5
Install Git on local machine 6
Install & configure Visual Studio Code on local machine 7
Clone a repository locally 8
Configure your 'user.name' and 'user.email' in git 11
Install VirtualBox on local machine 12
Download Oracle VirtualBox from the below URL 12
Install Oracle VM VirtualBox pre-requisities 12
Install Oracle VirtualBox 13
Download Guest OS for Oracle VM 14
Create VM inside Oracle VirtualBox 15
VirtualBox mouse integration 26
Enable copy/paste from host to guest OS in VirtualBox 27
Enable Drag and Drop (between host and guest OS) 28
How to SSH Into a VirtualBox Ubuntu Server 30
Installing SSH on the Virtual Machine 30
Enable port-forwarding 30
Restart ssh server 33
Start SSH Session (from Host Windows to Guest Ubuntu using Gitbash) 33
Shutdown Ubuntu server 33
Restart Ubuntu server 34
Installing update on Ubuntu server 34
Installing Jenkins on Ubuntu server 34
Install Java as Jenkins pre-requisite 34
Install Jenkins 36
Enable Jenkins 37
Start Jenkins 37
Check status of Jenkins 37
Set port forwarding for Jenkins (guest / Ubuntu to host / Windows) 38
Access Jenkins (on VirtualBox) from Windows machine 39
Unlock Jenkins / first time configuration 39
Customize Jenkins 40
Initial Jenkins configuration 40
Create First Admin User 41
Instance Configuration 42
Jenkins plugins Installation 43
Installing Docker from the Official Repository 46
Update the Package Repository 46

Install Prerequisite Packages 46
Add GPG Key 47
Add Docker Repository 47
Specify Docker Installation Source 47
Install Docker 48
Set docker to start automatically 48
Start docker 49
Check Docker Status 49
Set required permission for your Ubuntu user id to use Docker 49
How to restart docker 49
Create / manage Docker Hub account 50
Sign Up for Docker Hub account 50
Sign In to Docker Hub account 50
Create repository in Docker Hub 51
Create / manage Git Hub account 52
Signup for GitHub account 52
Login to GitHub account 55
Generate Personal Access Token in GitHub 58
Clone a Git repository 62
Install Minikube (for Kubernetes) 65
Install kubectl 65
Add nameserver in Ubuntu server 67
Start Minikube (with Docker Driver) 68
Check minikube status 68
kubectl cluster-info 69
kubectl config view 69
kubectl get nodes 69
kubectl get pods --all-namespaces 70
minikube addons enable metrics-server 70
minikube dashboard 70
Setting up Jenkins CI/CD pipeline 70
Open Jenkins URL 70
Configuring Docker Hub authentication 71
Configure Kubernetes authentication for Jenkins 73
Create CICD pipeline in Jenkins 77
Pipeline syntax 85
Pipeline script to check out the GitHub repository 86
Pipeline script for Docker hub authentication 88
Pipeline script to allow authentication for Kubernetes deployment: 91

Run Jenkins Pipeline 92
Accessing the Kubernetes Dashboard 95
Check deployed application in Kubernetes 96
Create configuration to access application 97
Application URL access from Windows machine 98

Overall Architecture



1. Local machine / host OS = Windows; Oracle VirtualBox is installed on Windows; Ubuntu as guest OS is installed on Oracle VirtualBox.

Local machine (Host Windows) --> Oracle Virtual Box --> Ubuntu

2. Jenkins, docker and minikube (Kubernetes) are installed on Ubuntu guest OS.

3. GitHub and DockerHub are at public cloud over internet.

4. Developer from anywhere (through internet) will push code through Visual Studio Code to GitHub (source code repository) which is at public cloud.

5. Jenkins (in Ubuntu) will fetch the code from GitHub (public cloud) and build the docker image (inside Ubuntu).

6. Once docker image is built, Jenkins (in Ubuntu) will push Docker image from Ubuntu to DockerHub (at public cloud).

7. Kubernetes installed with minikube inside Ubuntu will pull the same image from DockerHub (at public cloud) and deploy it to Kubernetes Cluster (inside Ubuntu). Application will run at Kubernetes.

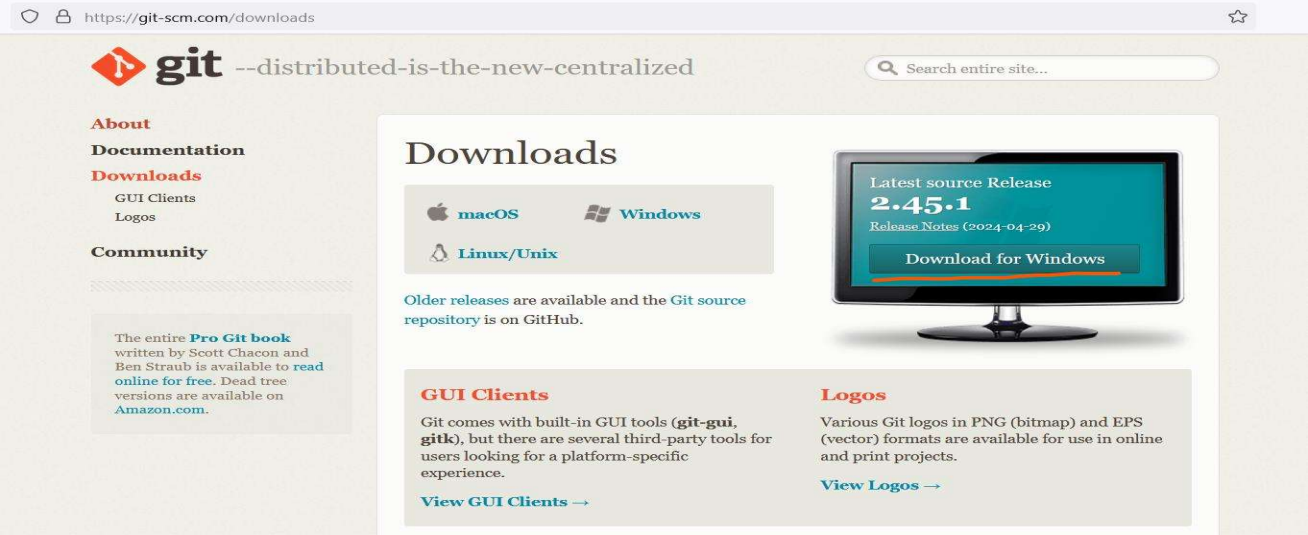
8. Users from local machine (windows) will access application deployed on Kubernetes Cluster (inside Ubuntu).

9. DevOps engineer from local machine (windows) will access Jenkins installed on Ubuntu and manage CICD pipelines.

Install Git on local machine

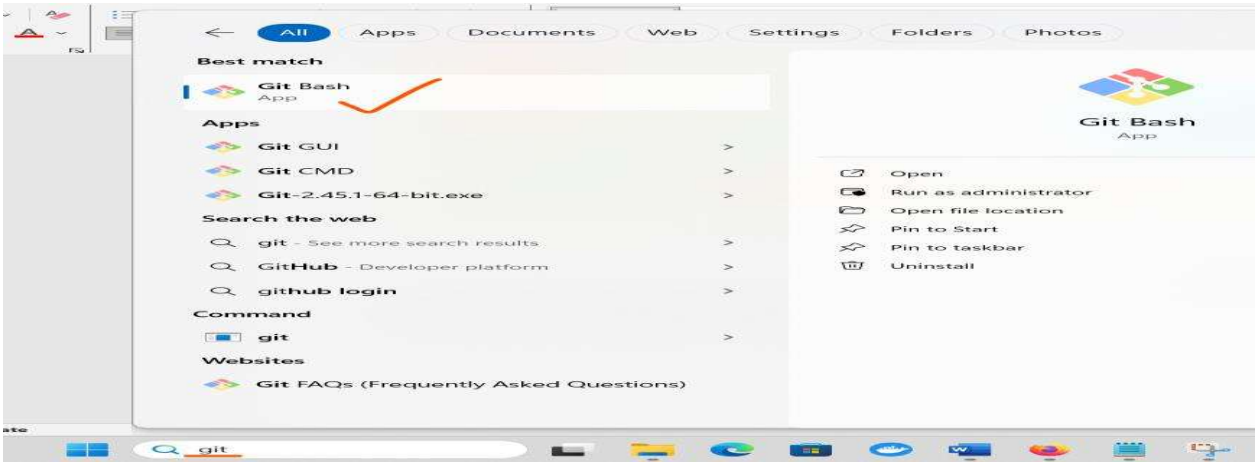
Download Git from the below URL in your local machine.

https://git-scm.com/downloads

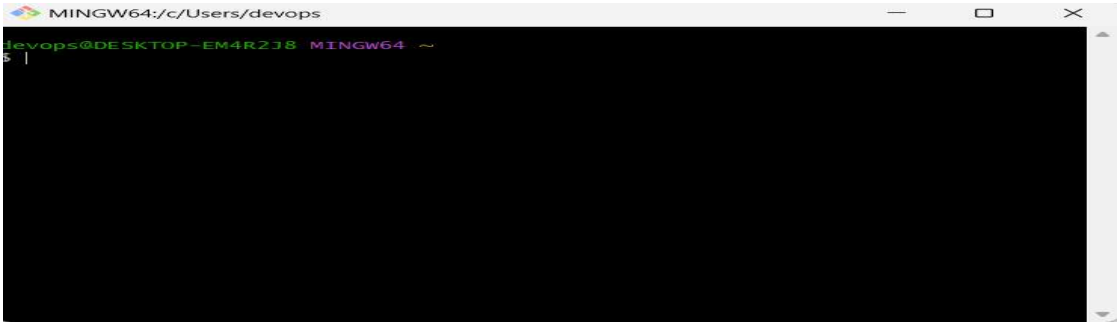


Install it after download. Post installation, we will get Git Bash 

which is an application for Microsoft Windows OS environments that provides Unix based shell utilities and experience for Git command line commands. Git Bash emulates the Git command line experience that Unix environments have, for Windows users.



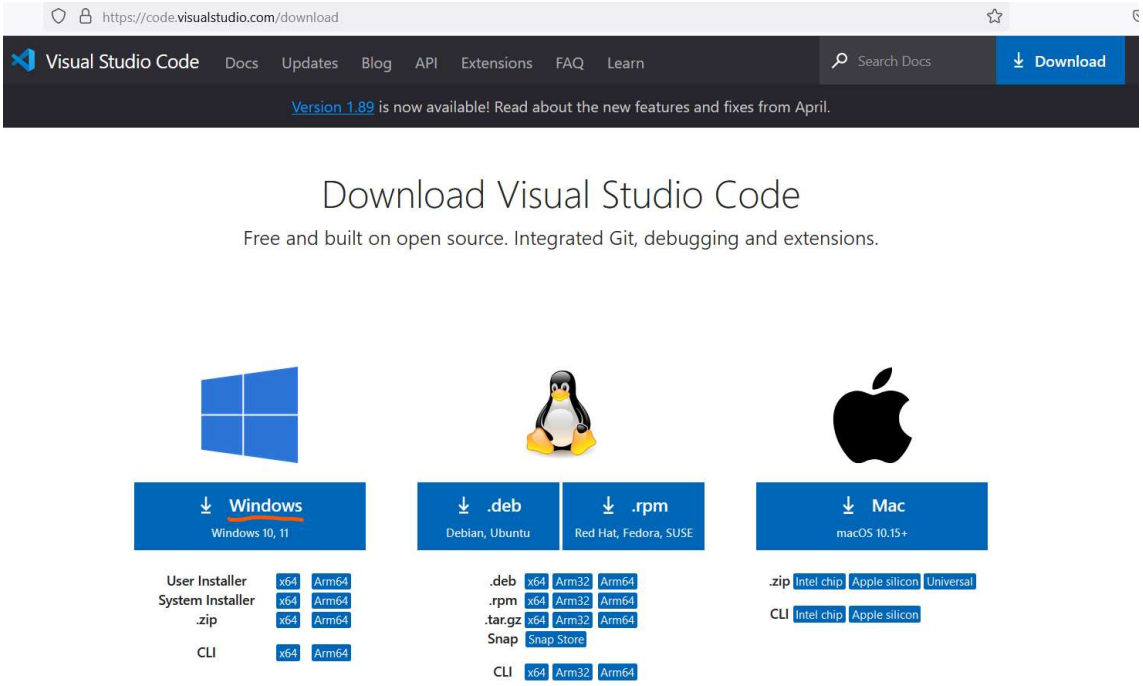
You can open Git bash from Start menu search like the above.

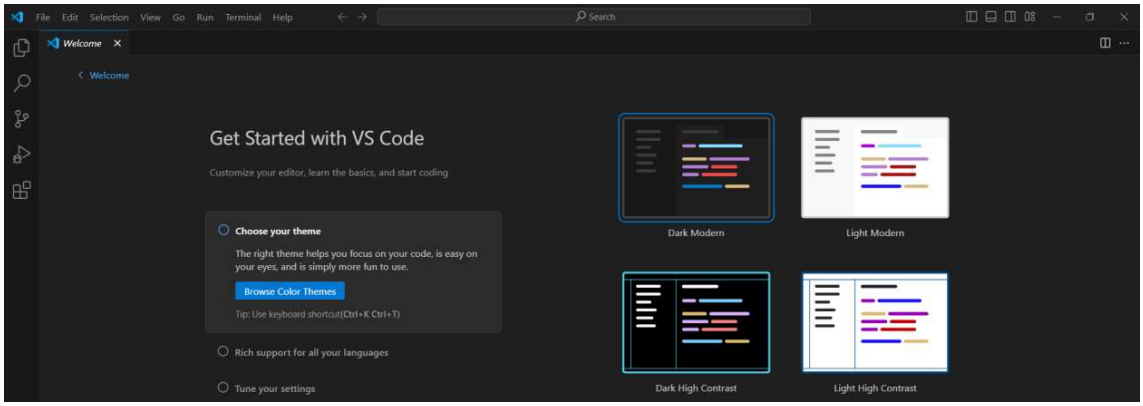


Install & configure Visual Studio Code on local machine

Download and install Visual Studio Code from the below link:

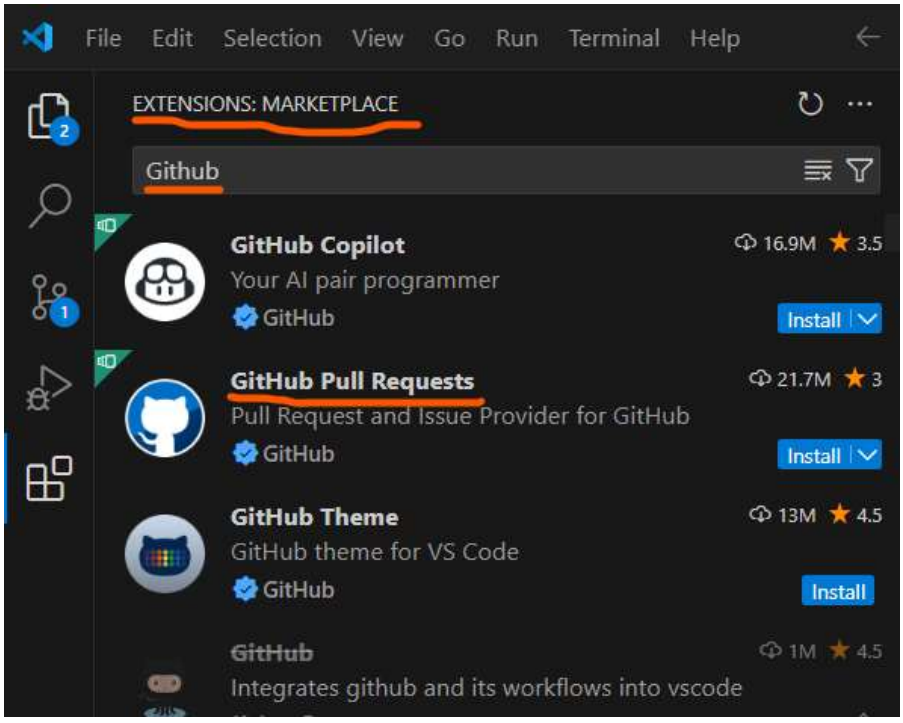
https://code.visualstudio.com/download





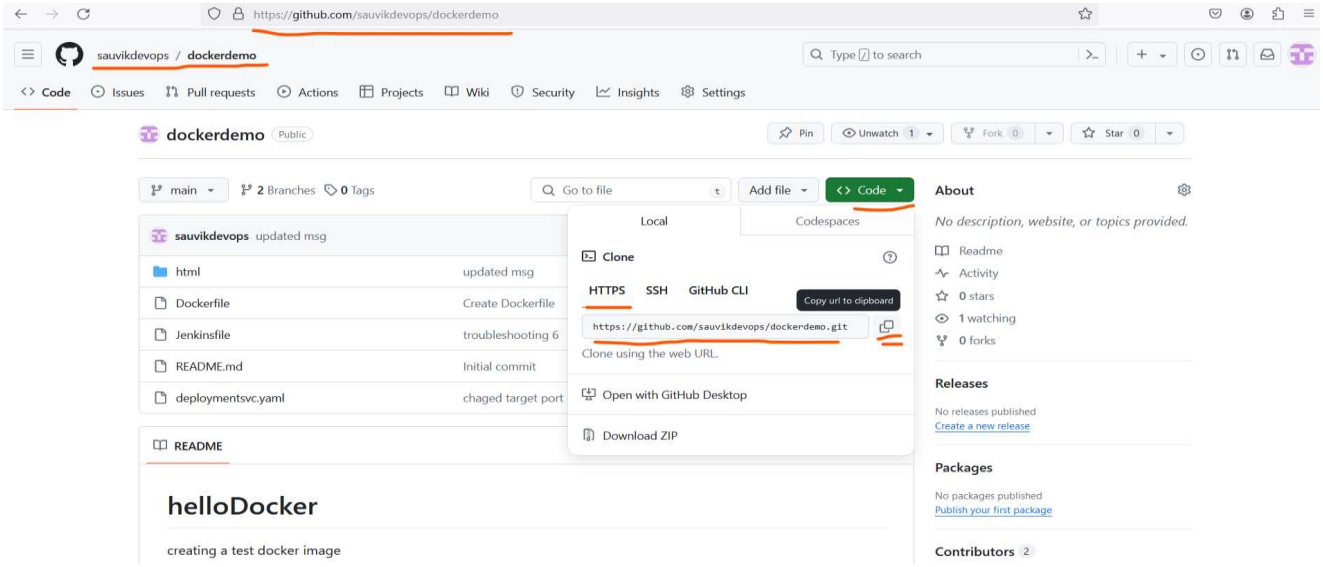
Clone a repository locally

Open VS Code, and click on the gear icon  in the bottom left corner. From Extensions: MARKETPLACE put Github in the search box:

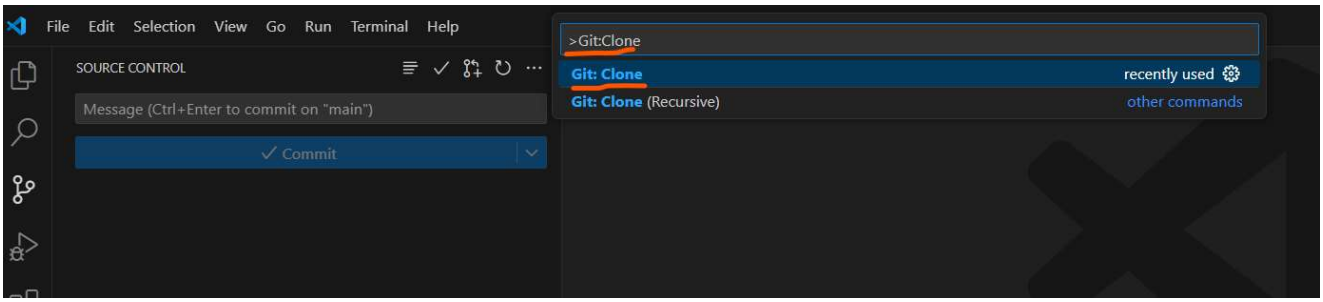


Select GitHub Pull Requests and click on install.

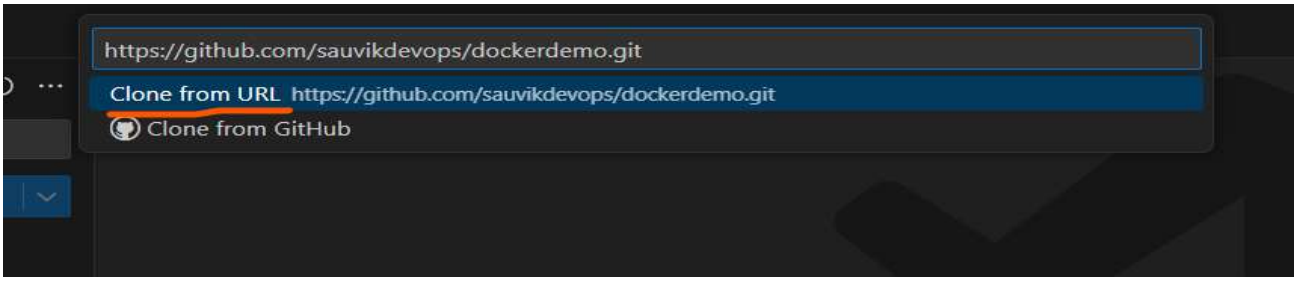
Now, we're going to add a GitHub repository to VS Code. You'll want to go back to your GitHub account (https://github.com) with your default browser and locate the address of the repository you want to add. Once you've navigated to the repository in question, click the Code dropdown, and copy the URL under HTTPS:



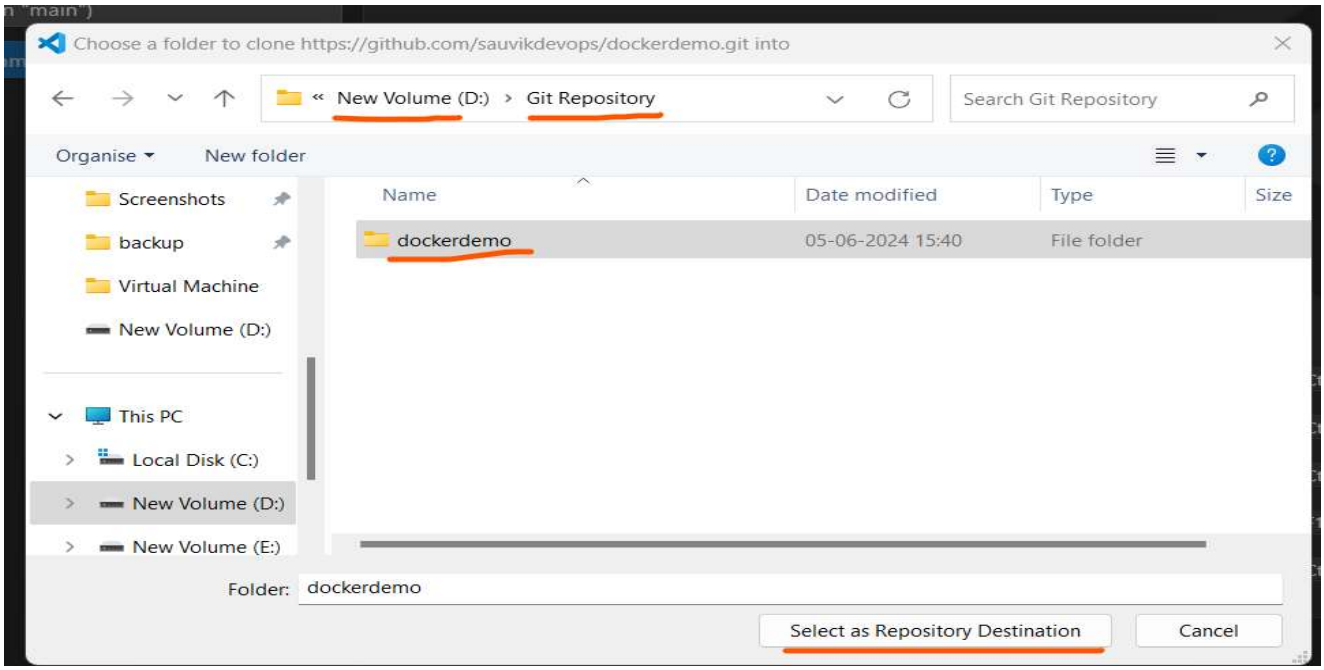
Now, come back to Visual Studio Code and press Ctrl + Shift + P to open command palette. Put Git:clone in search box:



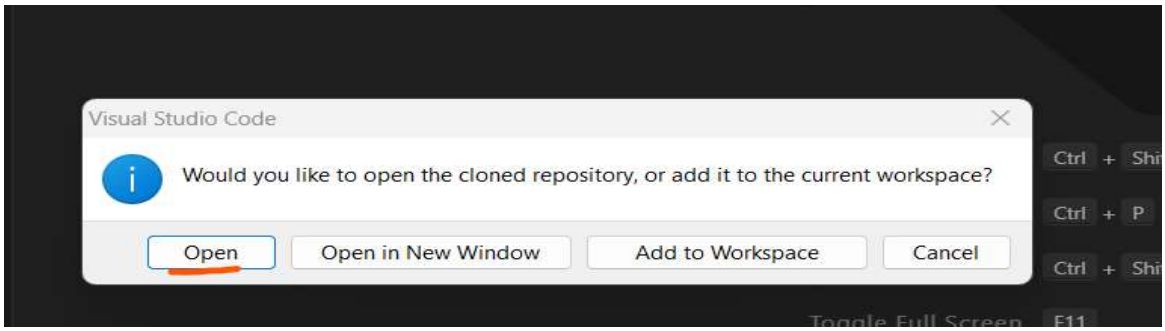
Put the copied URL of your Github repo and click on "Copy from URL …"



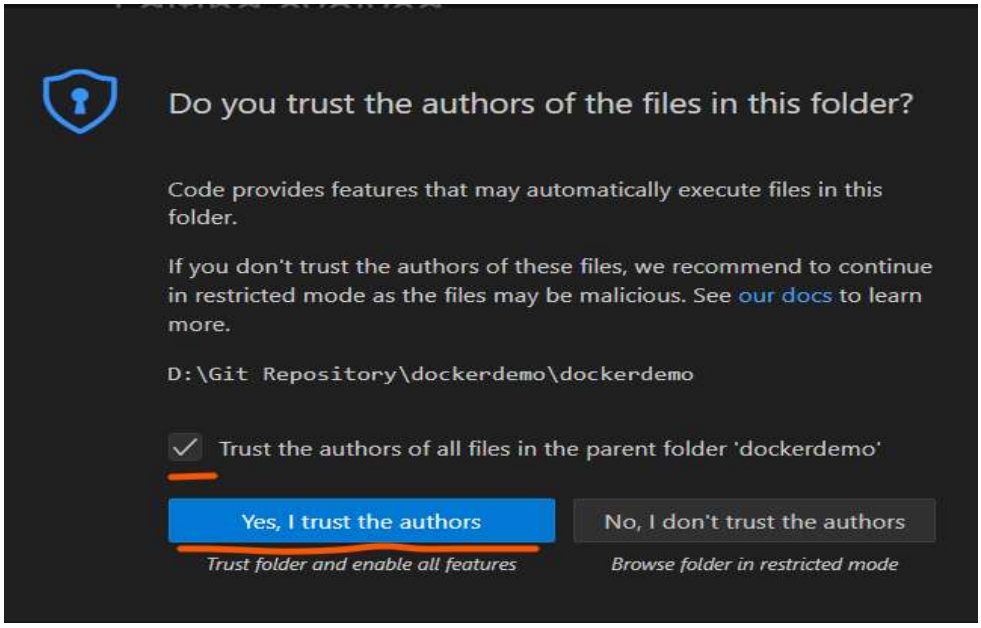
It will now as for Repository Destination i.e. the folder in your local computer where the Github repo would be cloned. For example I have created a folder called “deckerdemo” (usually we keep it as our project name) inside D:\Git Repository folder. You can keep it anywhere you would like.



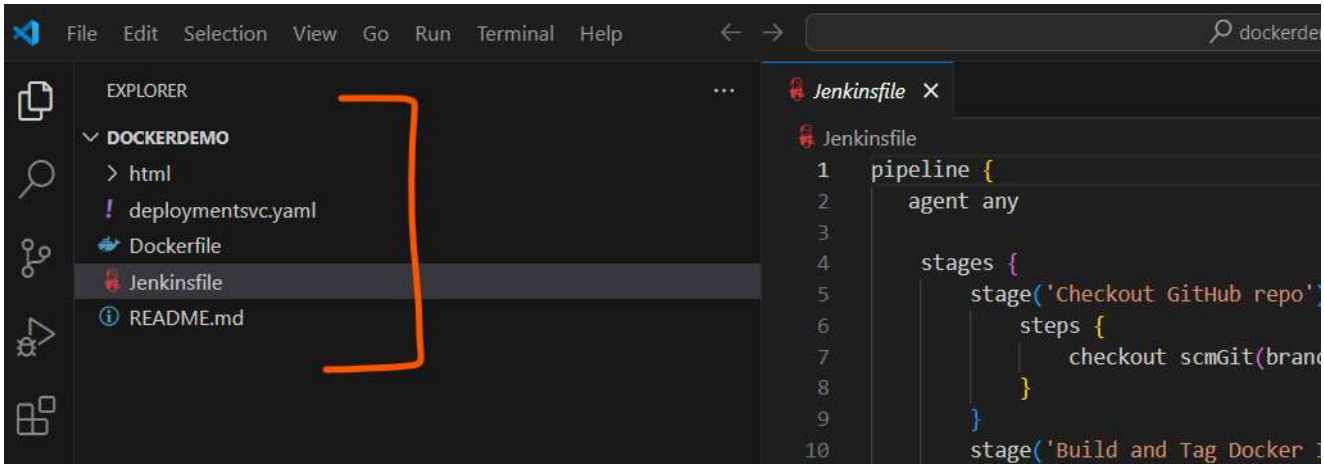
Click on “Open” if it asks. You can also add it into current VsCode workspace as well.



Check mark on – “Trust the authors of …” and click on “Yes, I trust the authors”

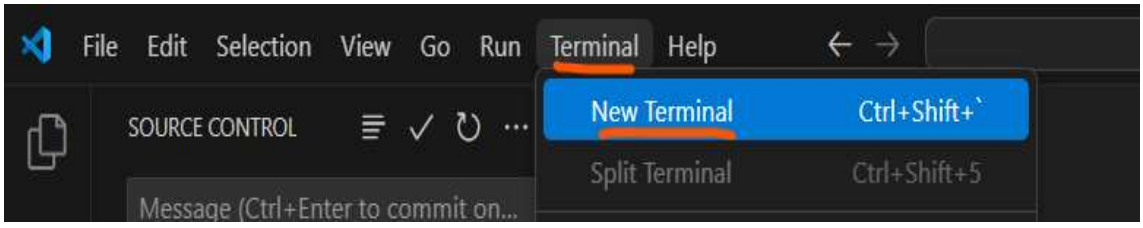


Now, you should be able to see the cloned repo in local.



Configure your ‘user.name’ and ‘user.email’ in git

Go to Terminal menu of Visual Studio Code, click on “New Terminal”



Run the below commands in the terminal.

git config --global user.email “sauvik.devops@gmail.com”

git config --global user.name “sauvik.devops@gmail.com”

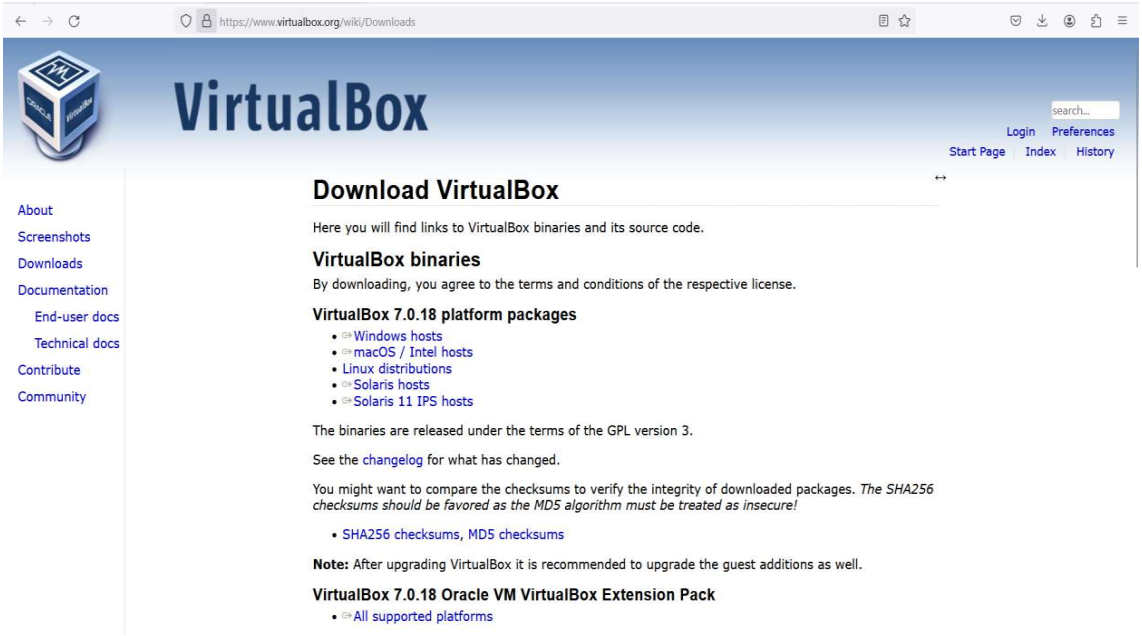
Note: Replace my email with your.

```
PS D:\sauvikdevops\dockerdemo> git config --global user.email sauvik.devops@gmail.com
PS D:\sauvikdevops\dockerdemo> git config --global user.name sauvik.devops@gmail.com
PS D:\sauvikdevops\dockerdemo>
```

Install VirtualBox on local machine

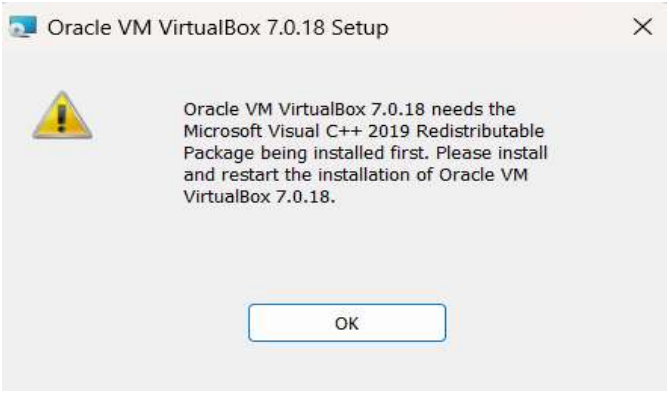
Download Oracle VirtualBox from the below URL

https://www.virtualbox.org/wiki/Downloads



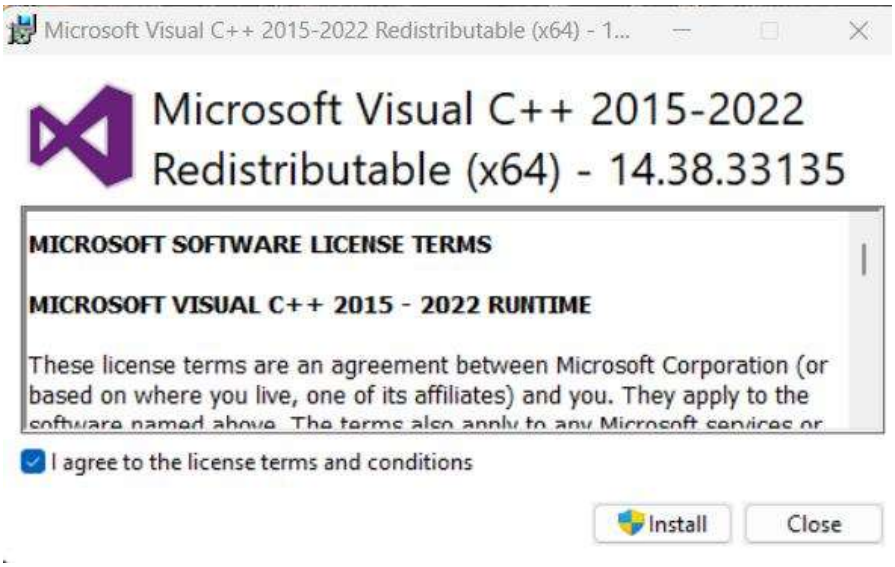
Install Oracle VM VirtualBox pre-requisities

Once you try to install Oracle VirtualBox, it will ask for below requirement:

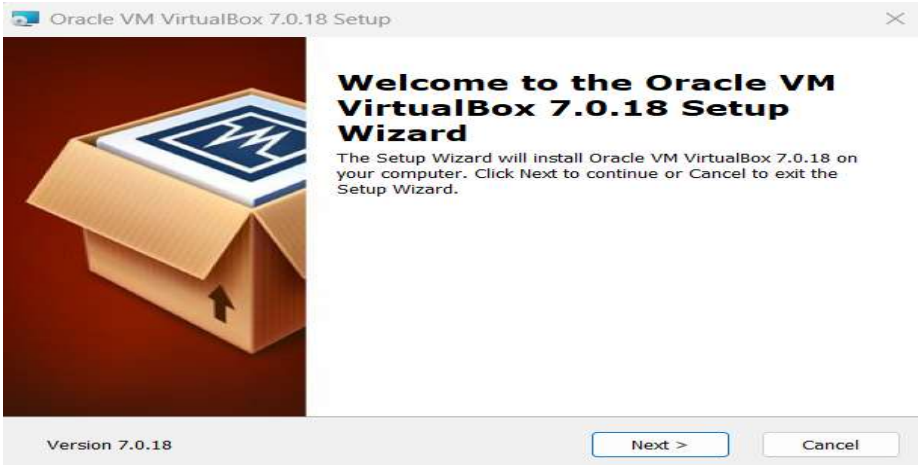


Download and Install Oracle VirtualBox from this link:

https://learn.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist?view=msvc-170



Install Oracle VirtualBox



Click on Next to proceed installation.

Click on Finish to complete installation.

Download Guest OS for Oracle VM

There are lots of websites from where you can download ISO image of Guest OS to run inside Oracle VM, for example.

https://www.linux.org/pages/download/

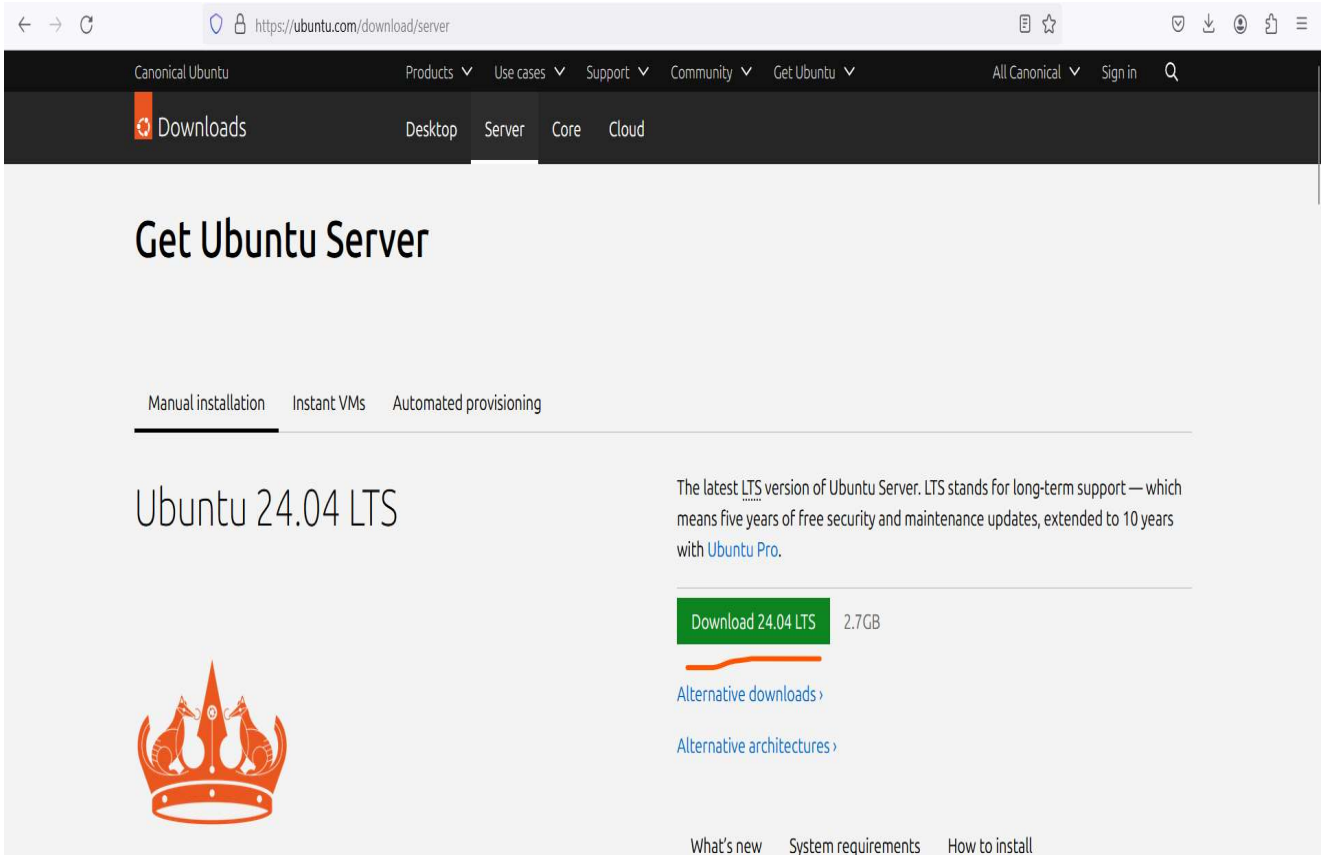
https://www.linuxlookup.com/linux_iso

https://ubuntu.com/download/desktop

https://ubuntu.com/download/server

https://yum.oracle.com/oracle-linux-isos.html

For our demo, we are downloading Linux OS Ubuntu Server 24.04 LTS from - https://ubuntu.com/download/server

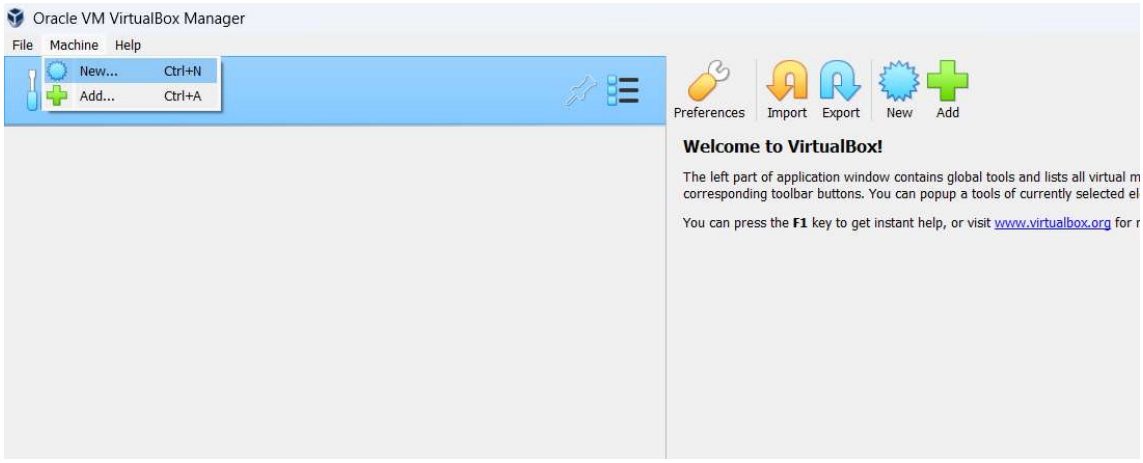


Click on Download to proceed.

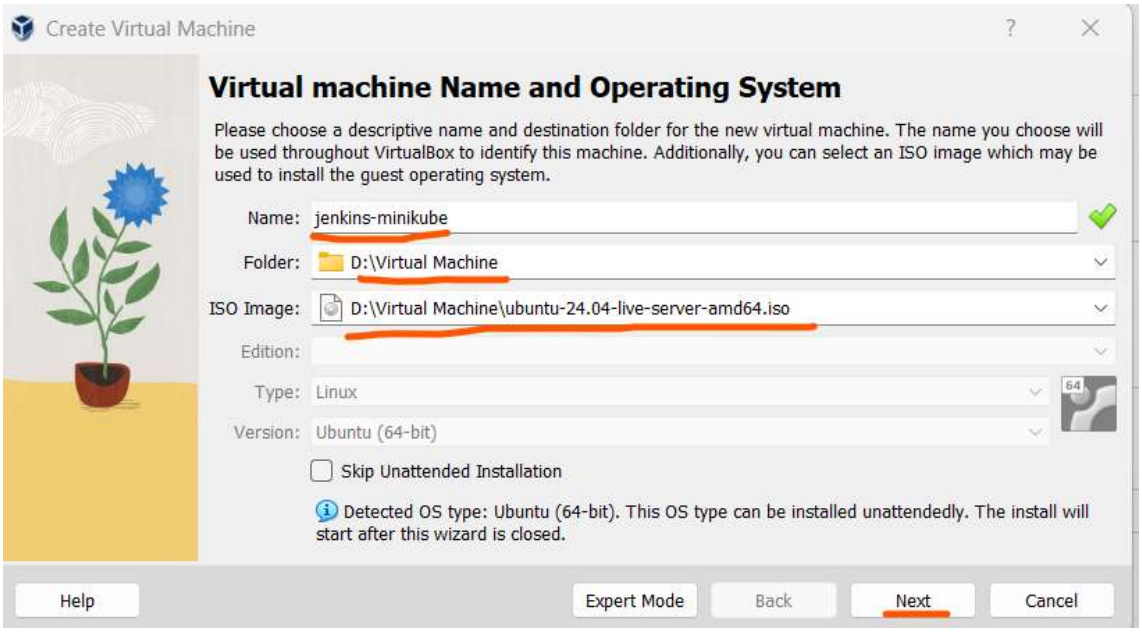
Note: For your lab, you do not need to download this Ubuntu ISO on every machine separately. Just download once and then copy to all the machines.

Create VM inside Oracle VirtualBox

Open Oracle VirtualBox and click on “Machine” menu → New…

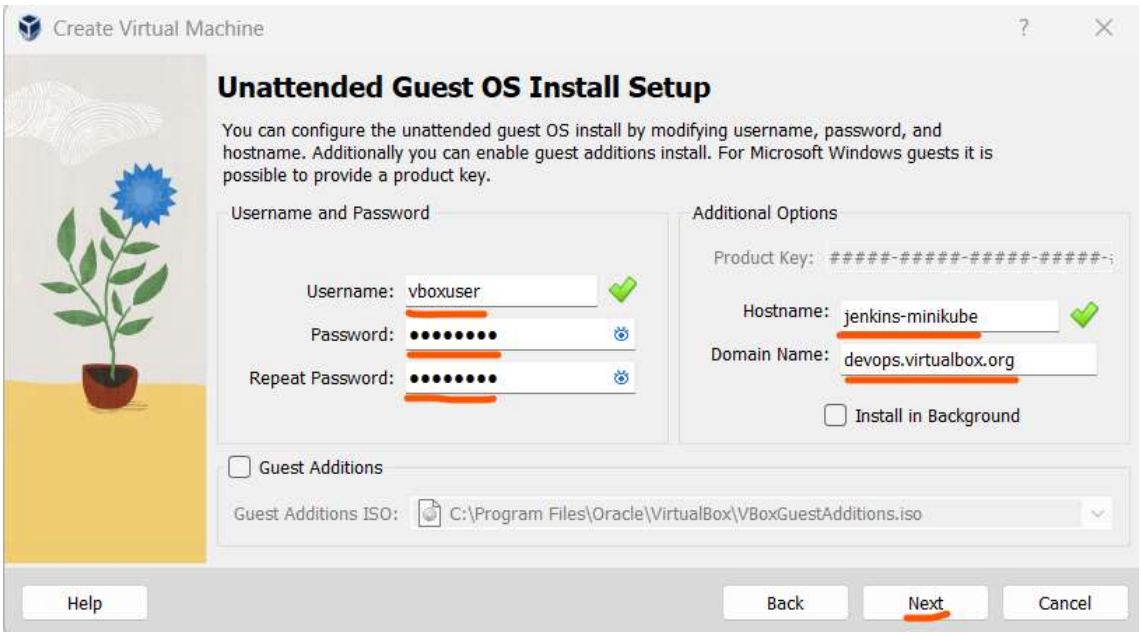


Put machine name of choice. I have given – jenkins-minikube and the folder location in your local machine where the Virtual machine including the disks would be created and location of Ubuntu ISO image which you have downloaded.

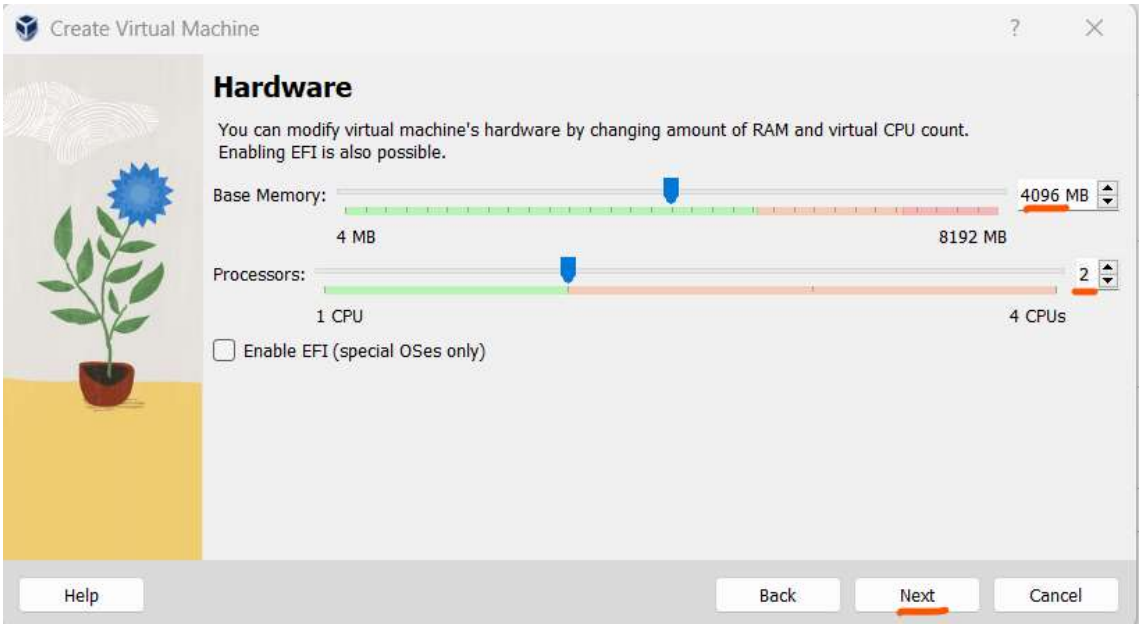


Click Next to proceed.

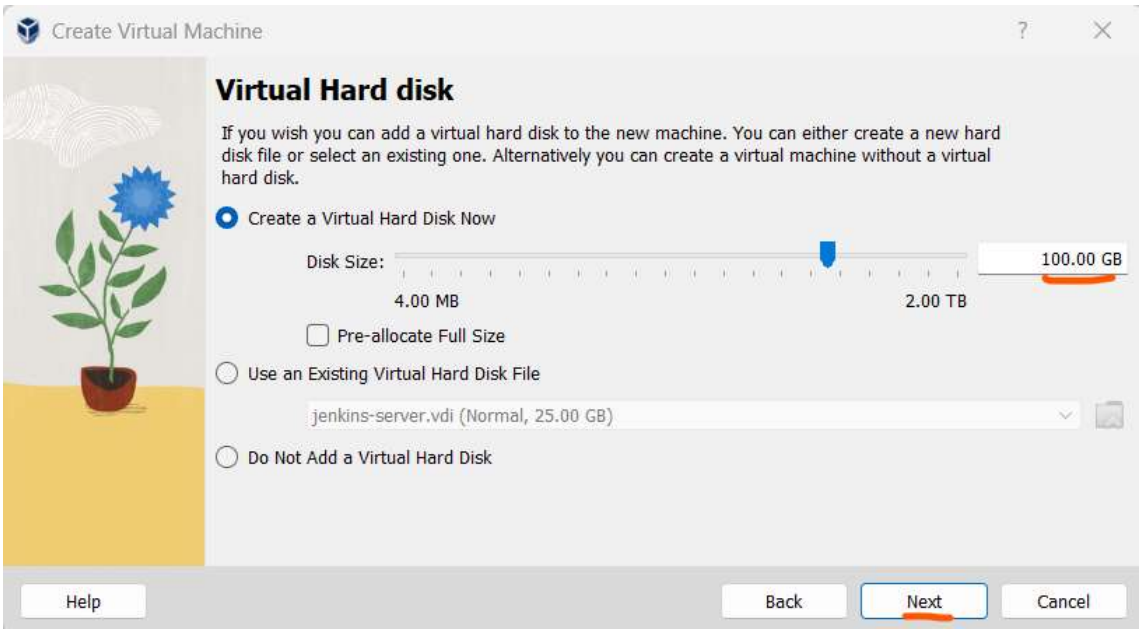
In the next screen, put Ubuntu username of your choice followed by password and confirmation of this new password. Also, put a hostname of this ubuntu guest OS and domain name. For this example, I have given domain name as “devops-virtualbox.org”.



Now set the hardware i.e. memory & CPU for this new Ubuntu guest OS. I would recommend you to set Base Memory = 4096 MB and Processors = 2

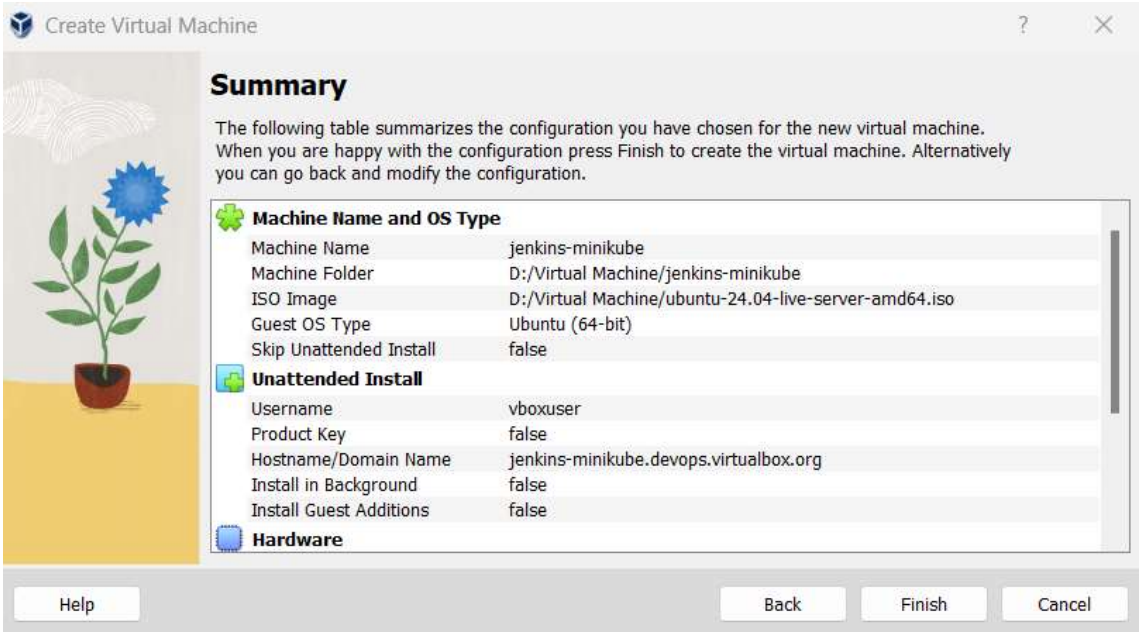


Click Next to proceed.



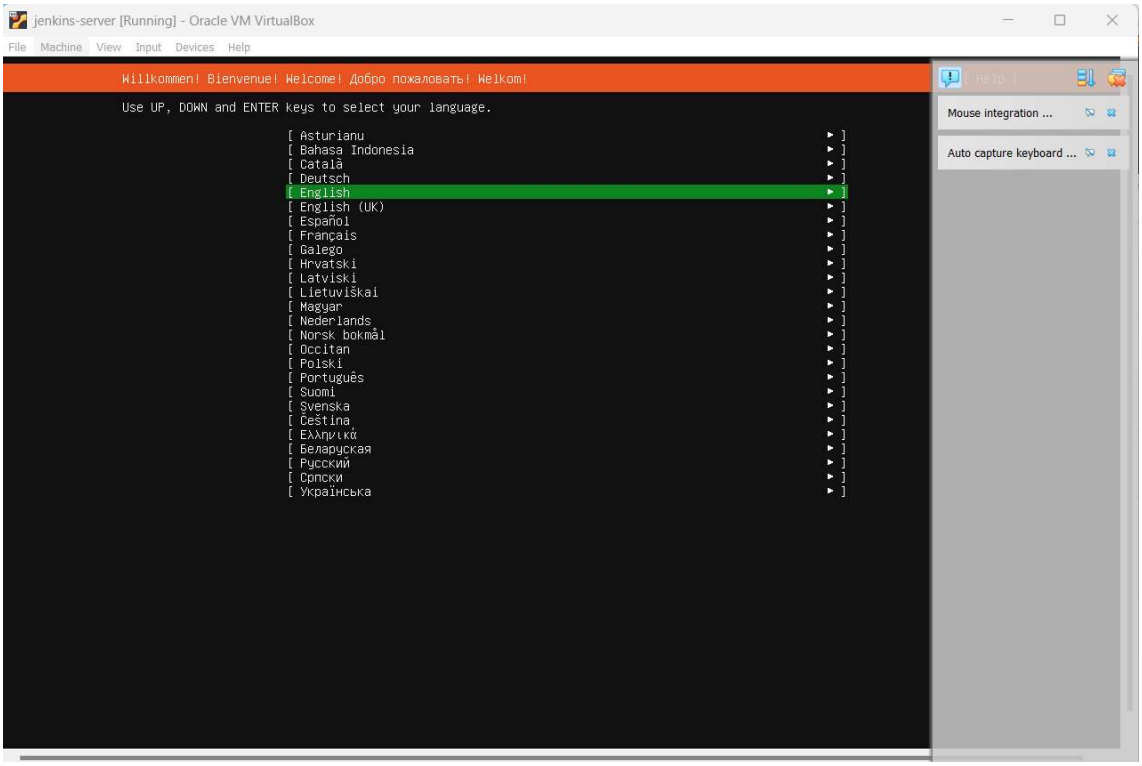
Set the virtual hard disk for this Ubuntu guest OS. I would recommend to have atleast 100 GB.

Click Next to proceed.

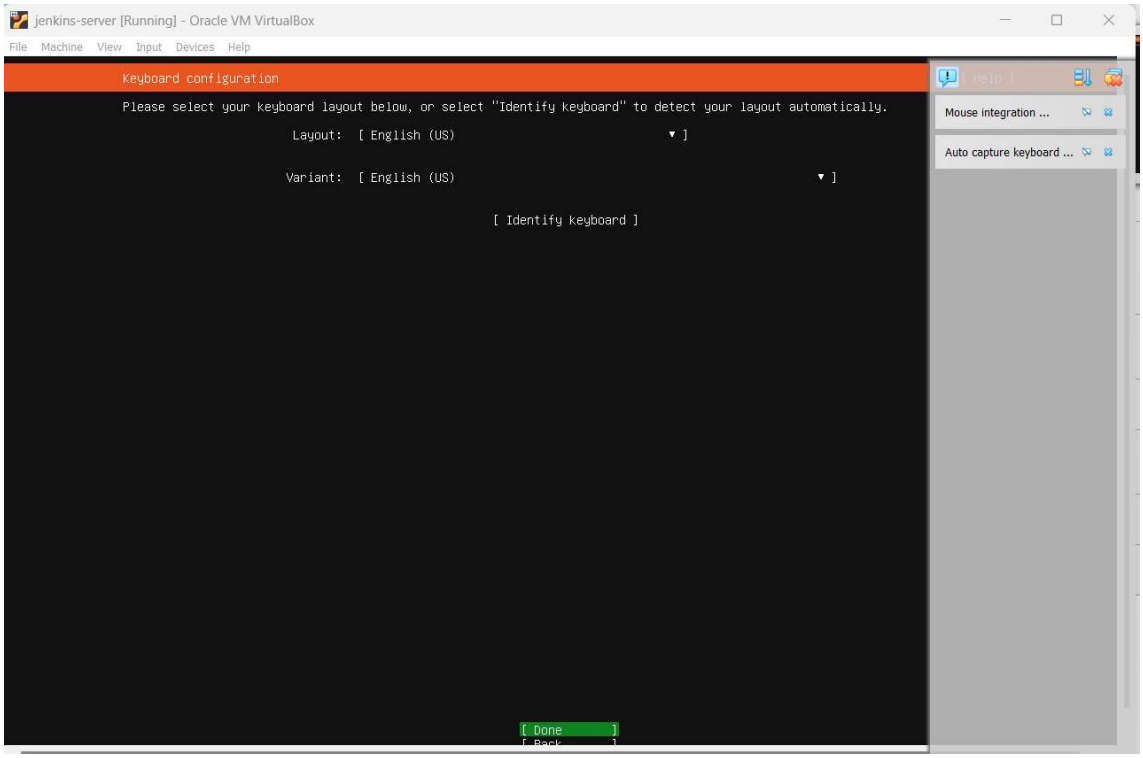


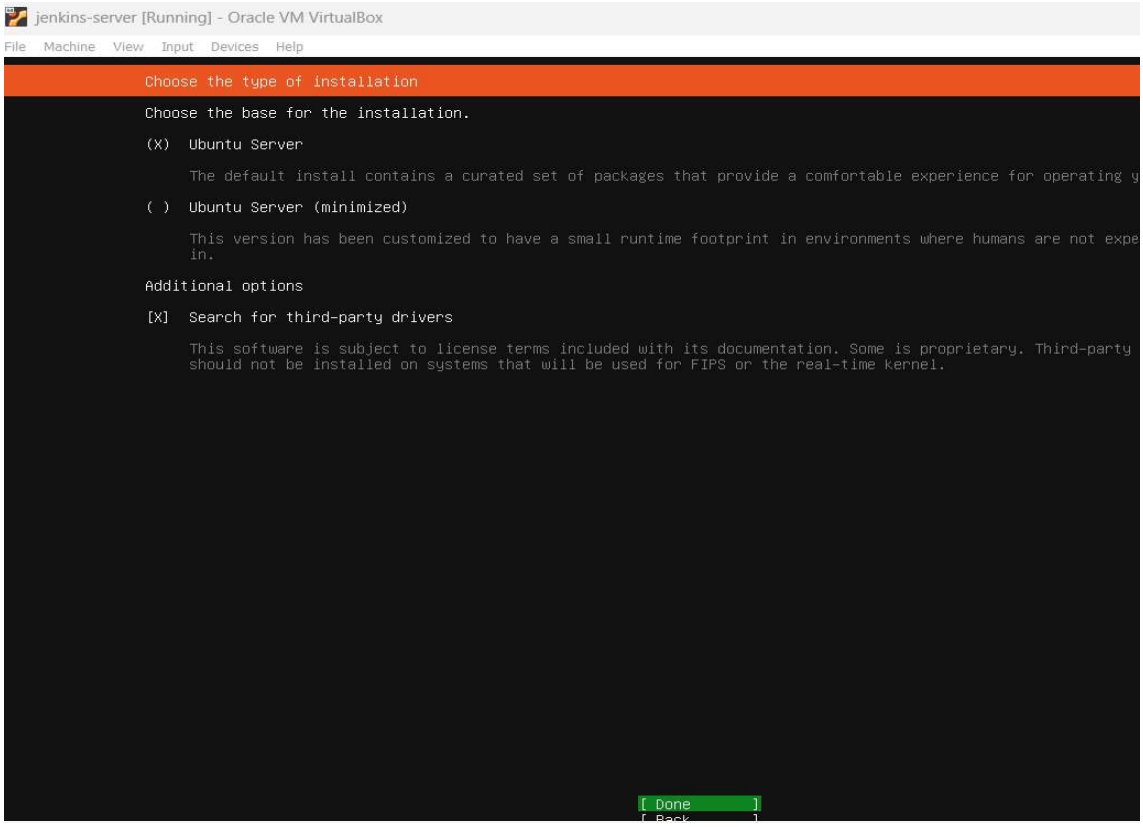
In the Summary page, just verity the settings you have set and click on Finish. It would then ask you for few preferences.

Select (or keep as it is if already selected) English as language and press enter to go to next screen.

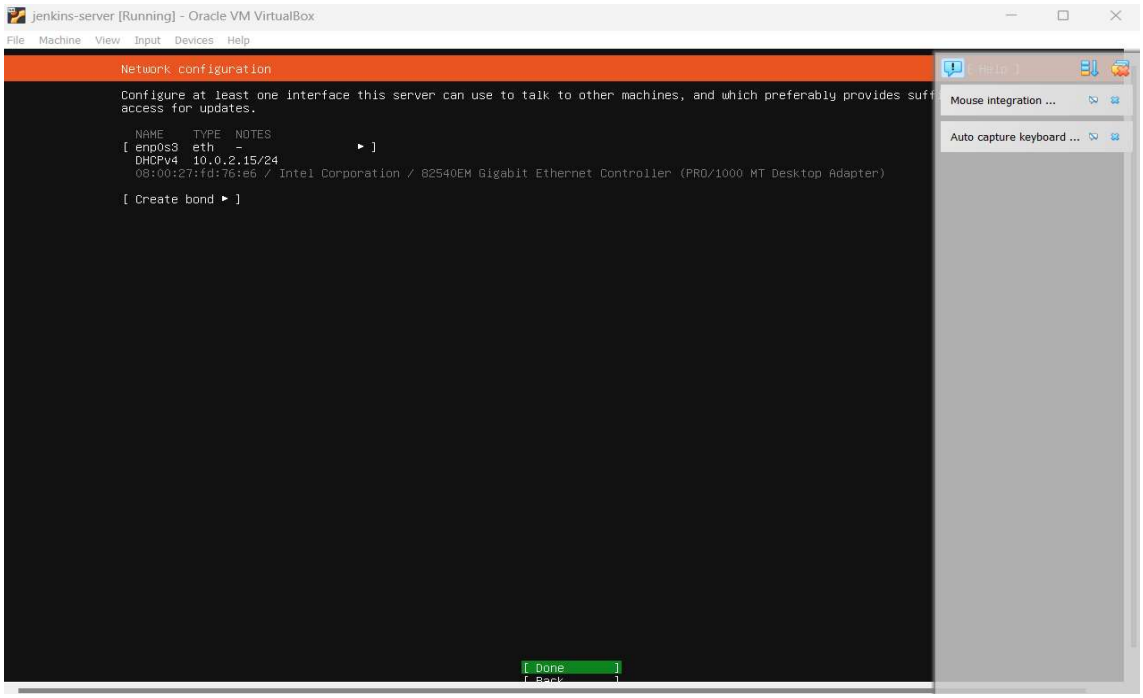


Leave the default keyboard layout and variant as it is and press enter to go to next.

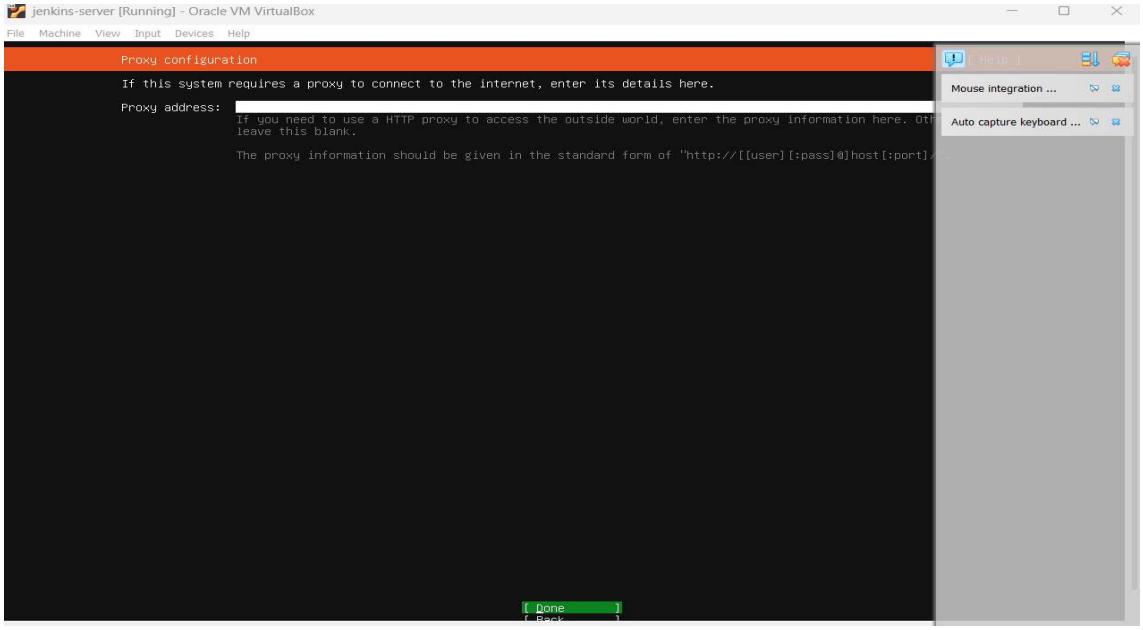




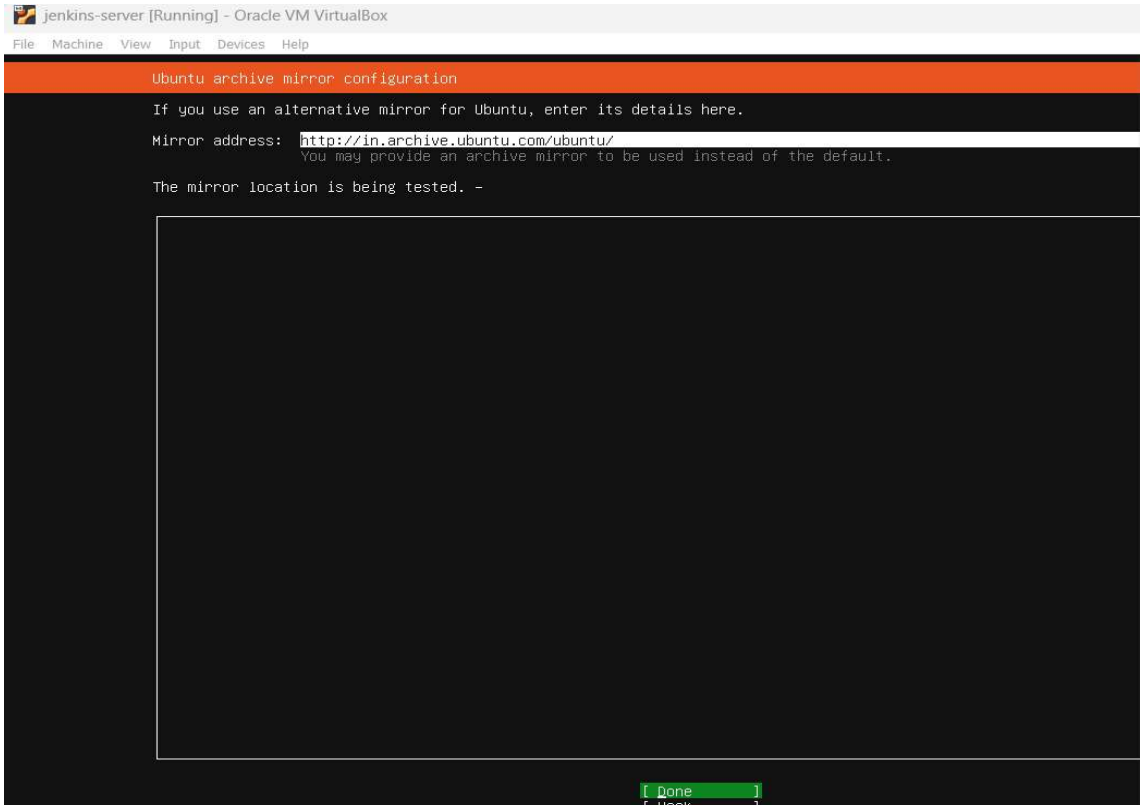
Select (or leave it if already selected) “Ubuntu Server” and press enter on “Done” to proceed.



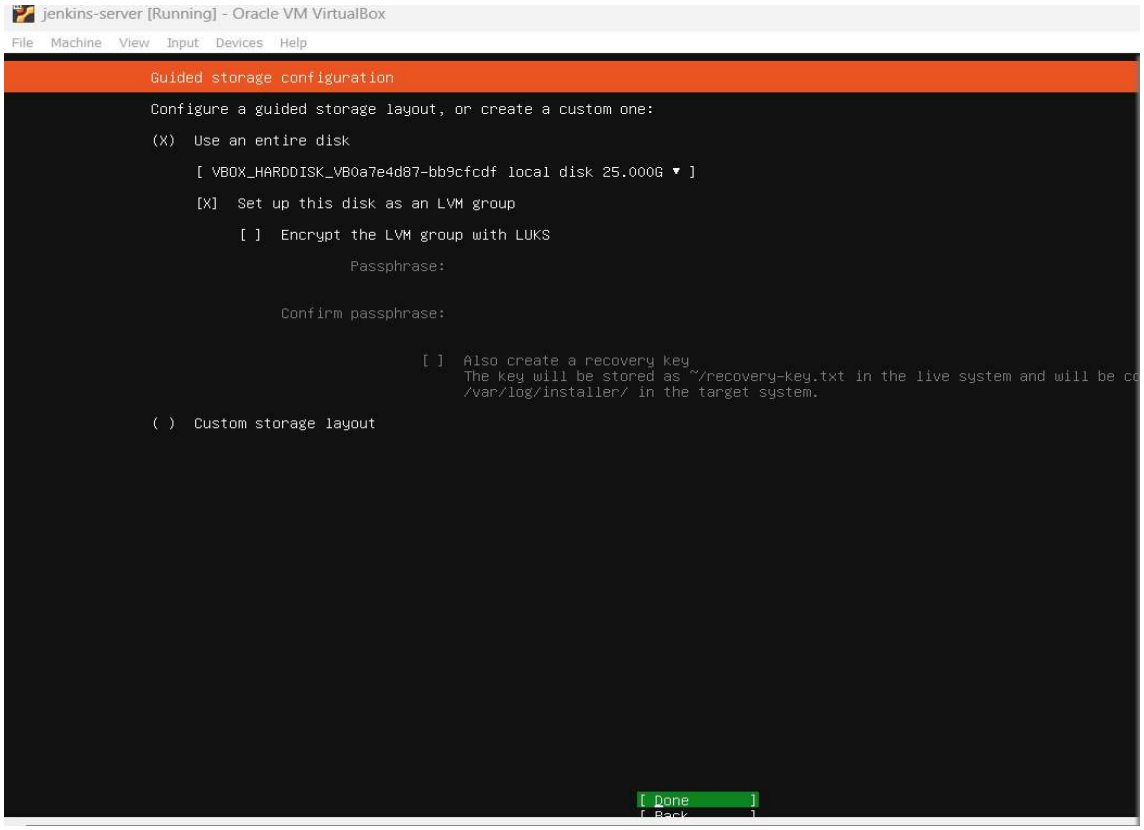
Leave the default network configuration and press enter on “Done” to proceed.



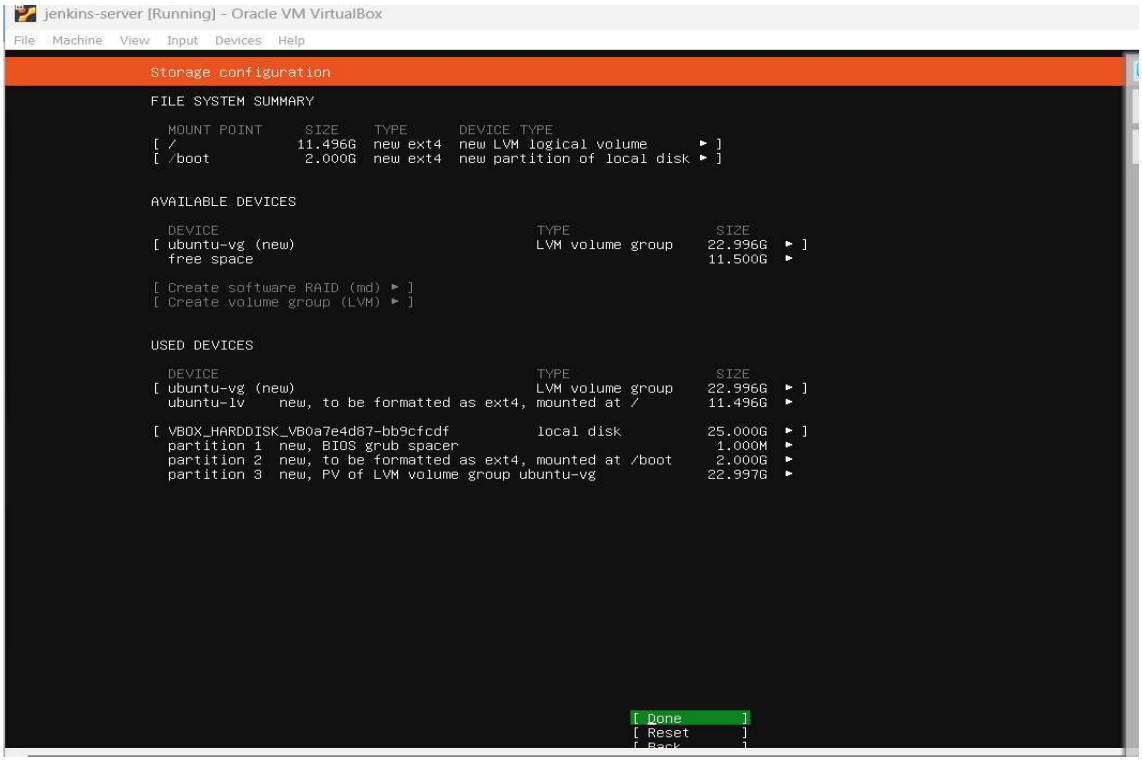
Leave the default Proxy configuration and press enter on “Done” to proceed.



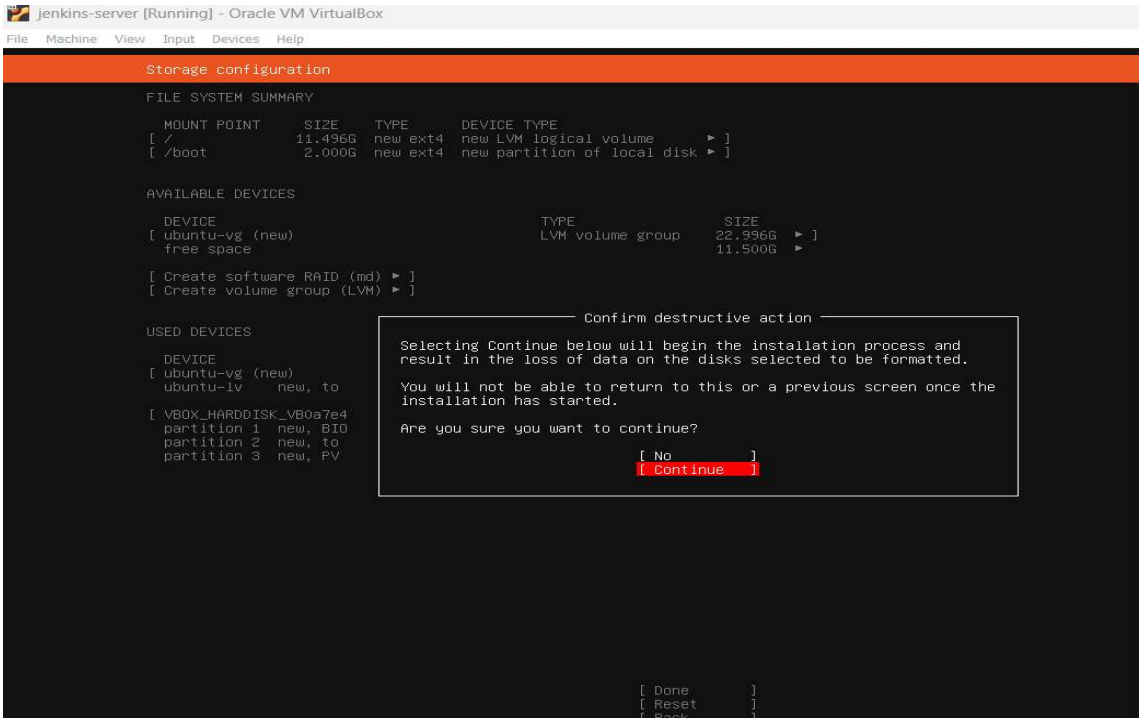
Leave the default Ubuntu archive mirror configuration and press enter on “Done” to proceed.



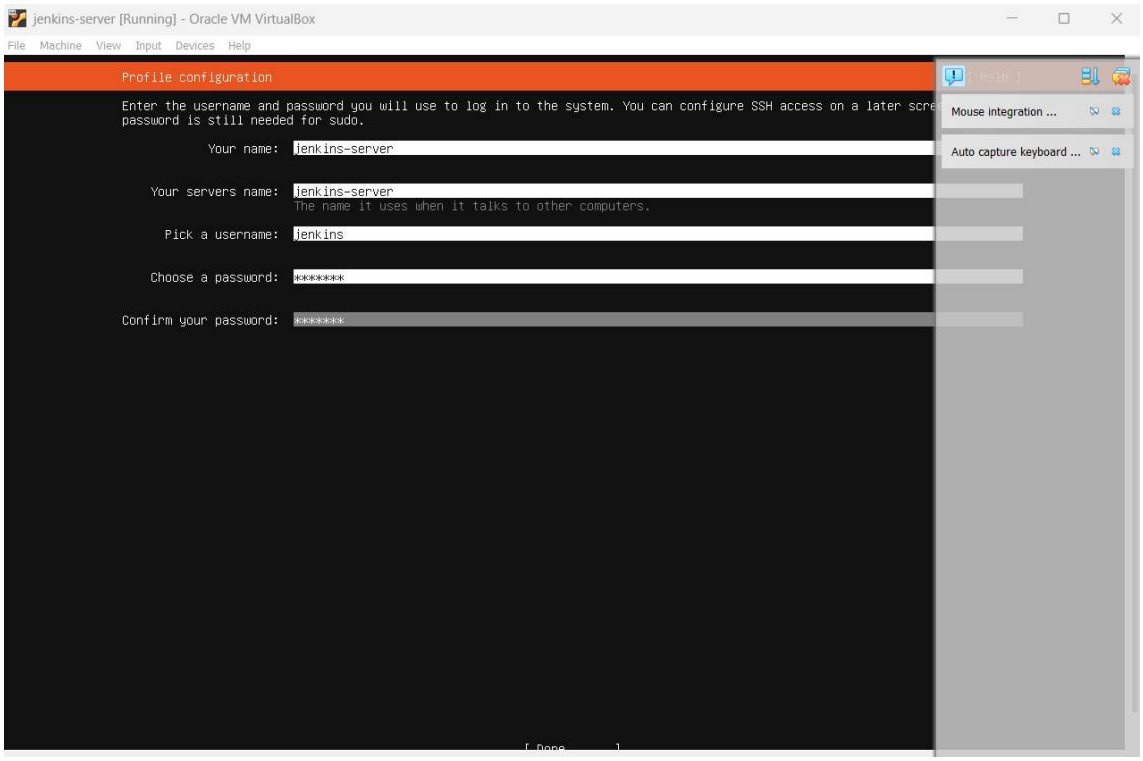
Leave the default storage configuration and press enter on “Done” to proceed.



Leave the default Storage configuration and press enter on “Done” to proceed.

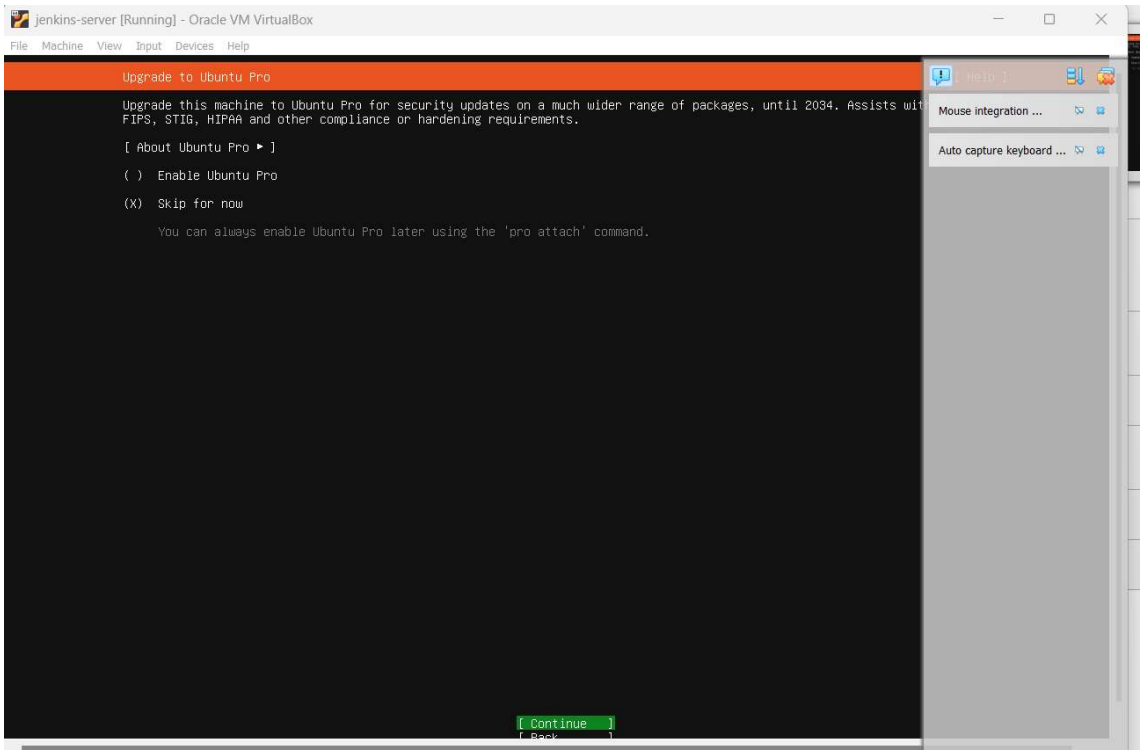


Click on Continue.

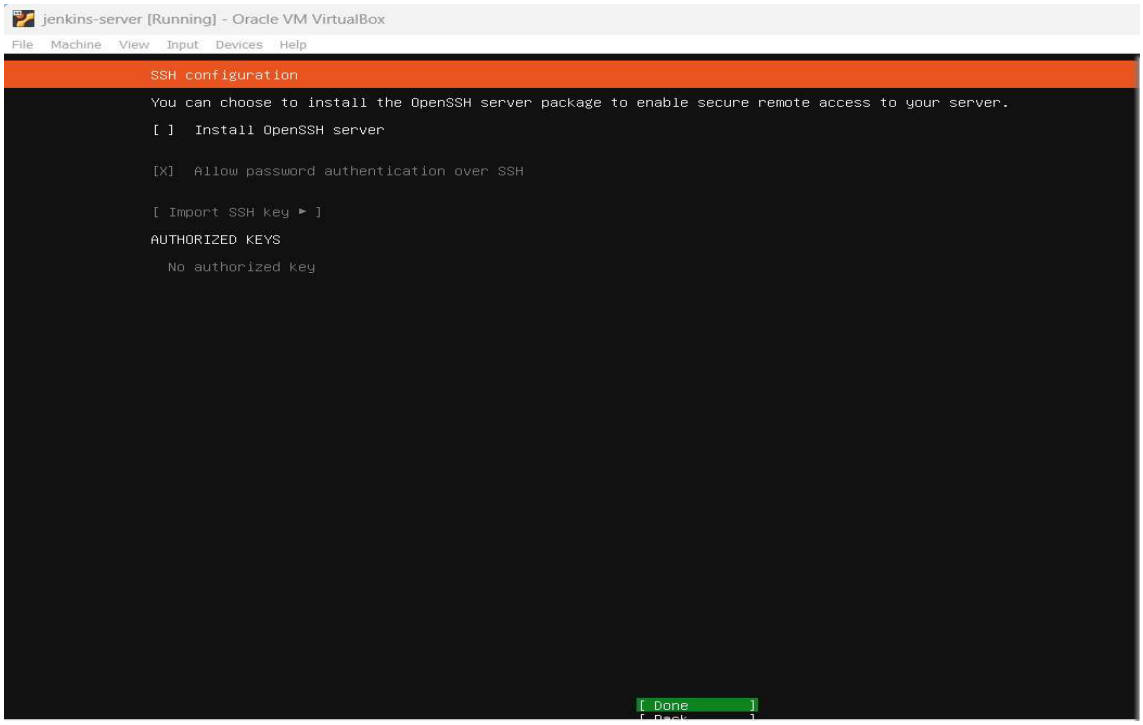


Put your name (any name), your server's name (I have put jenkins-minikube-server), give a username (I have put devops) and password.

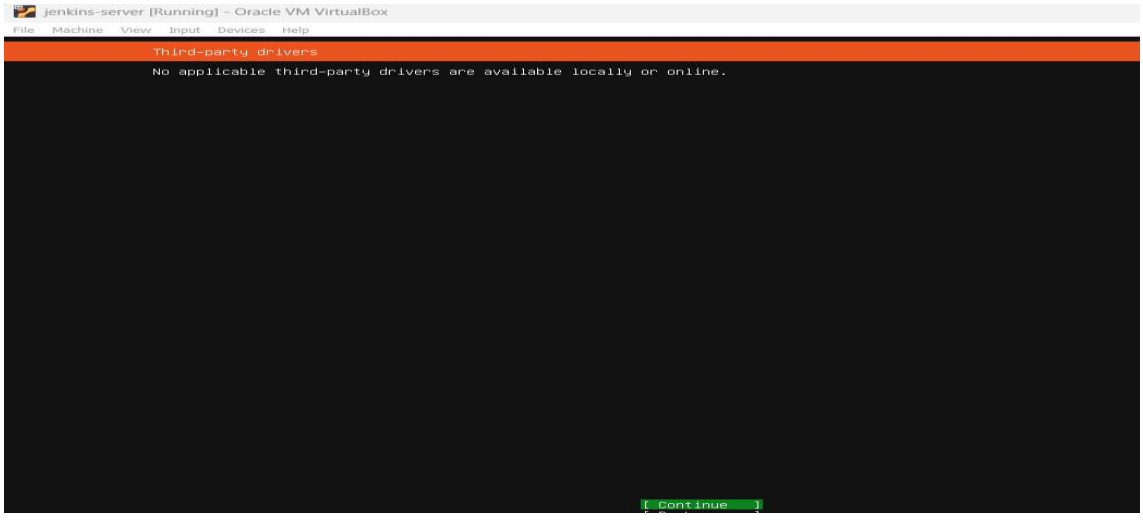
Press enter on "Done" to proceed.



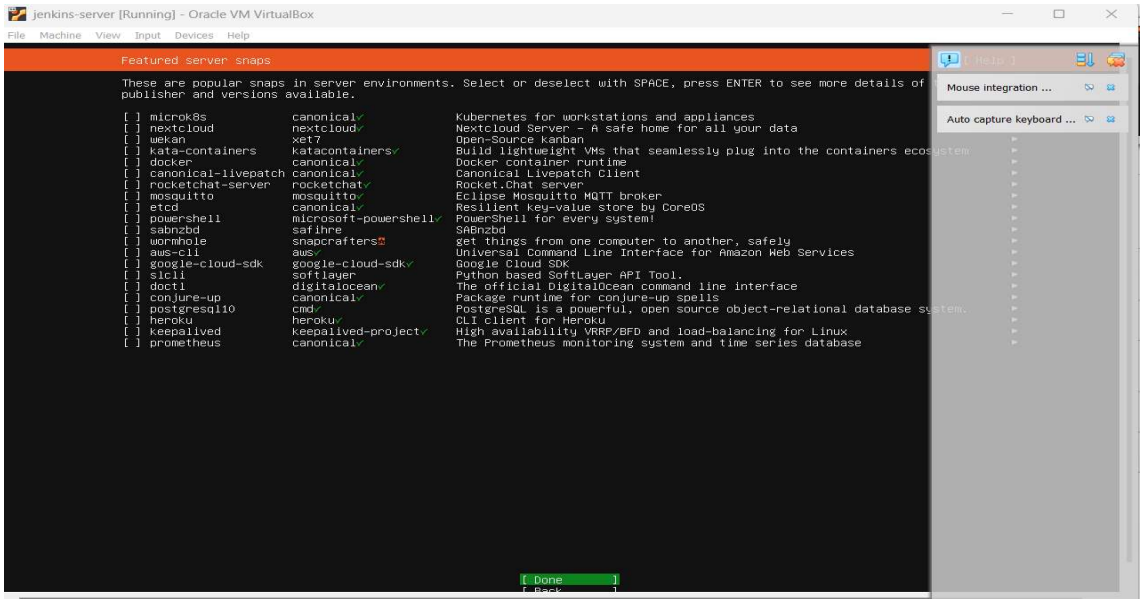
Select skip for now and press enter on “Continue” to proceed.



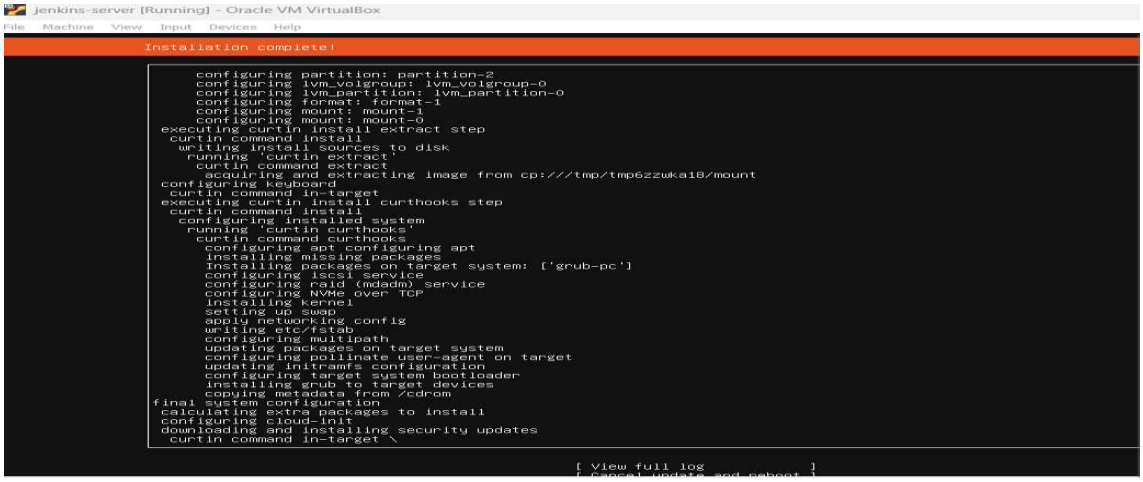
Leave the default option as black and press enter on “Done” to proceed.

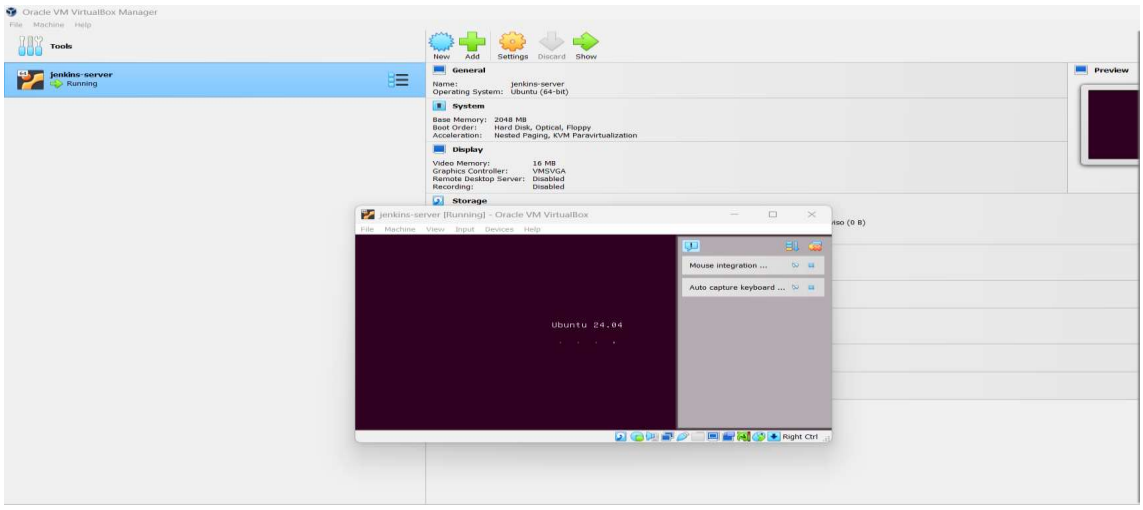


Leave the default option and press enter on “Continue” to proceed.



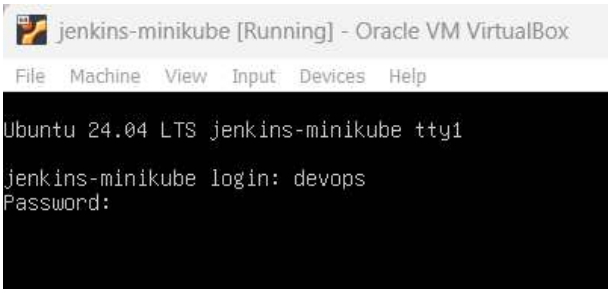
Leave all fields blank and press enter on “Done” to proceed.

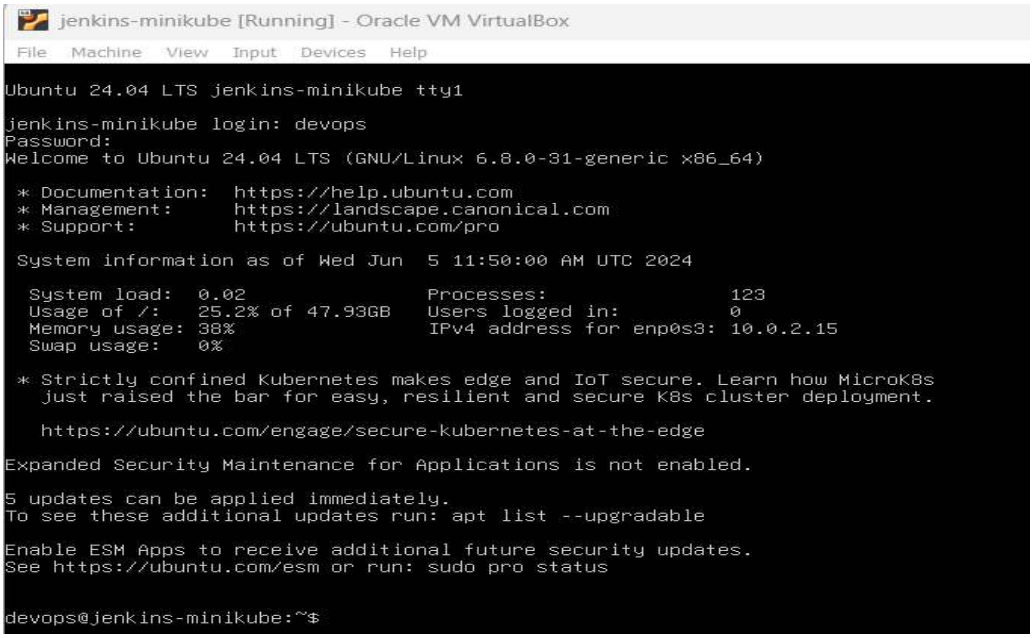




When you see installation complete, reboot the server.

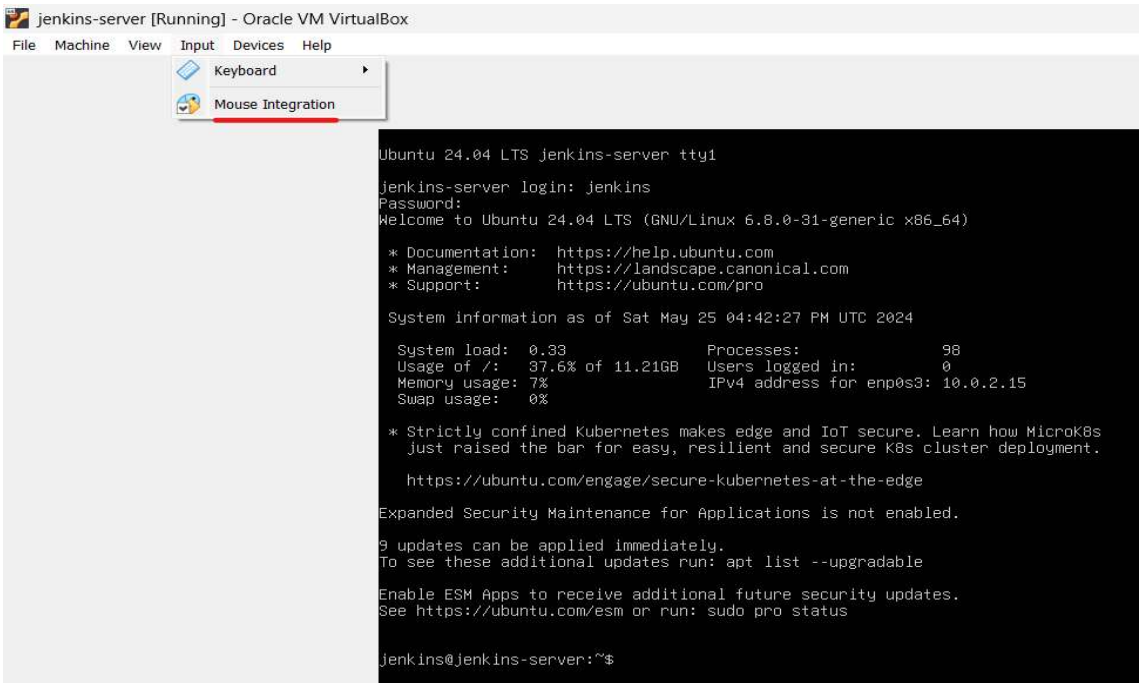
After reboot, try login to this server using the credential you had created after installation.

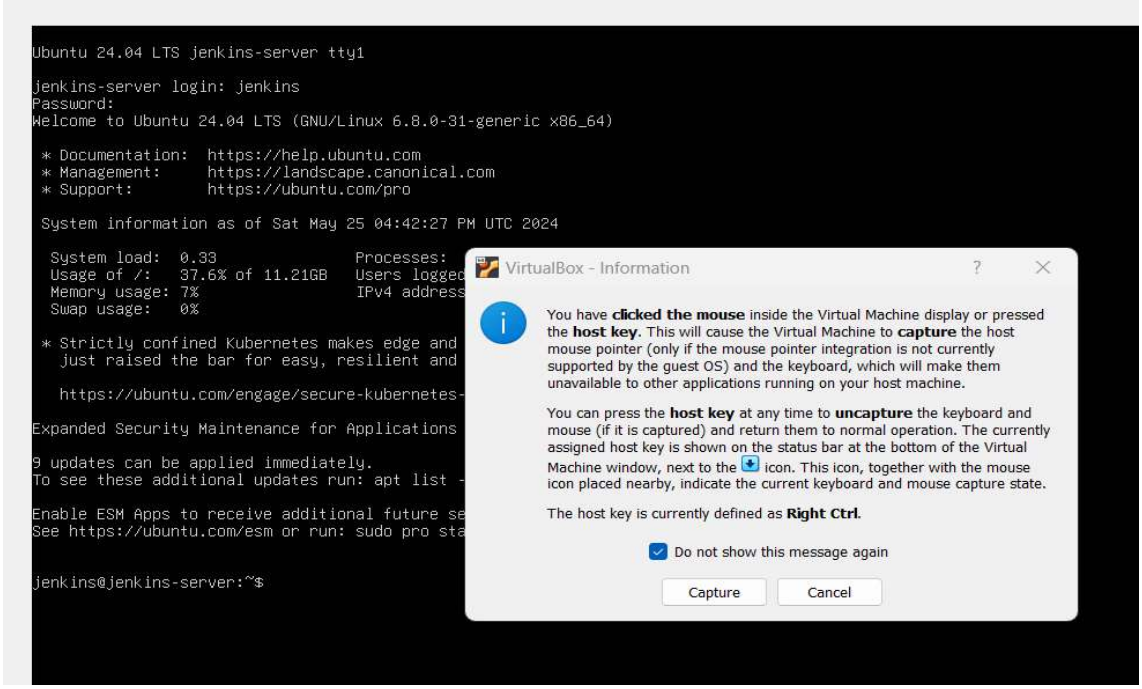




VirtualBox mouse integration

Mouse integration allows the mouse to flow smoothly from guest (Ubuntu) back to the host (Windows) without doing anything special.





To come out the mouse cursor from guest (Ubuntu) back to the host (Windows), press CTRL from right hand side of keyboard.

Enable copy/paste from host to guest OS in VirtualBox

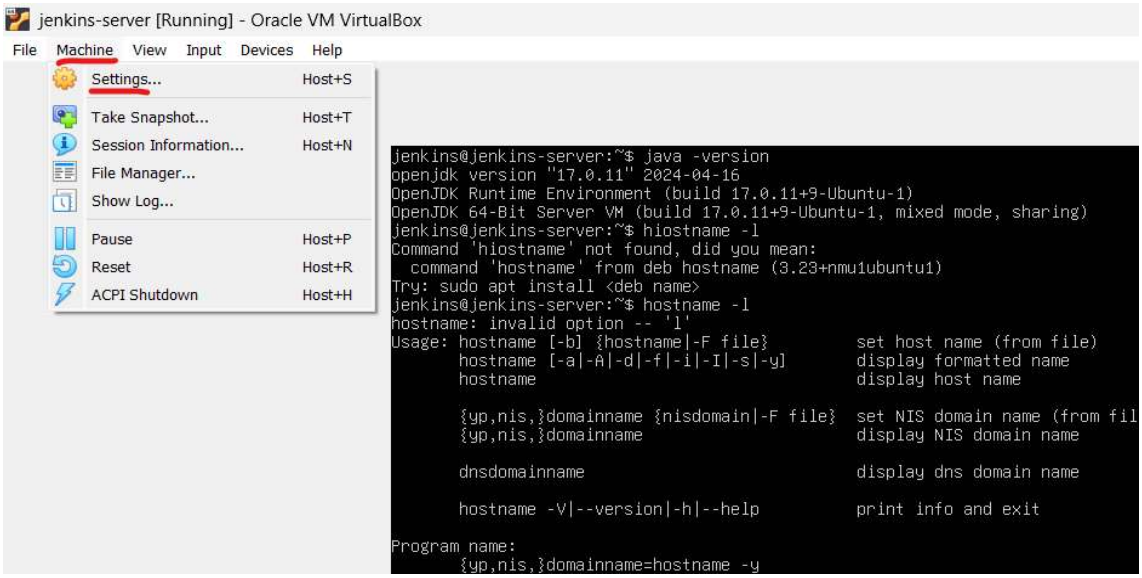
Occasionally, there is a requirement to transfer files or text from our host machine to the Virtual Machine (VM) running inside VirtualBox. This can be accomplished effortlessly by following the steps provided below:

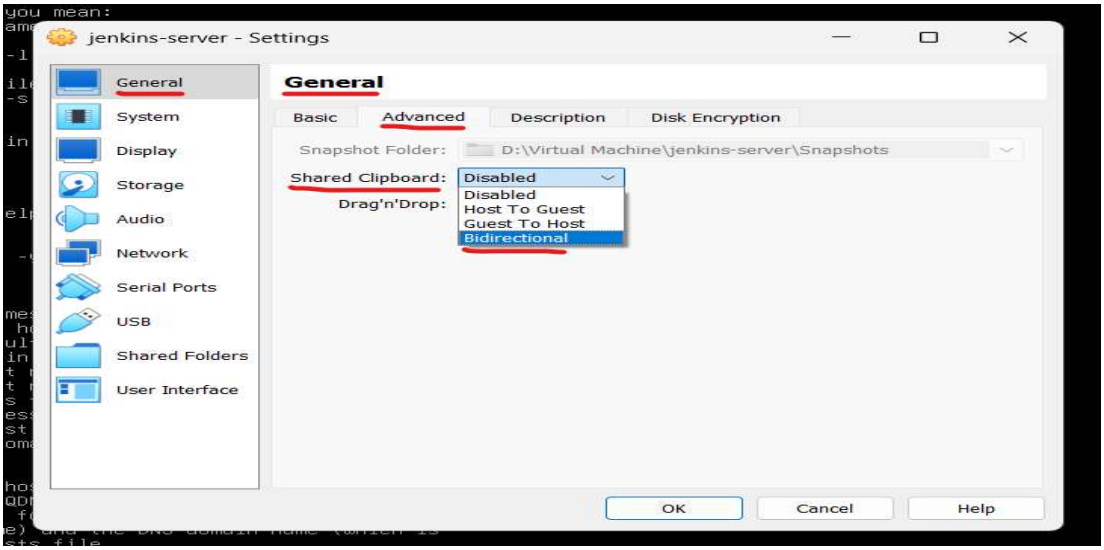
1. Launch VirtualBox.

2. Access the settings of the VM by navigating to the General section and selecting the Advanced tab.

3. Within the Advanced tab, you will find a setting to enable the “shared clipboard.” It offers three options: disabled, host to guest, and bidirectional. Opt for the bidirectional option.

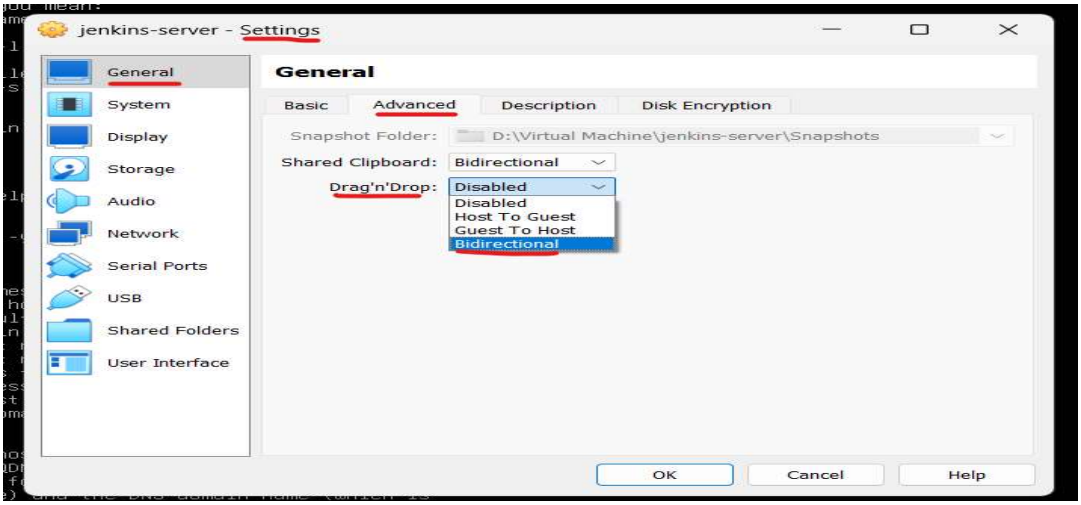
Attached below is a screenshot depicting the aforementioned setting.





Enable Drag and Drop (between host and guest OS)

Oracle VM VirtualBox enables you to drag and drop content from the host to the guest, and vice versa. For this to work the latest version of the Guest Additions must be installed on the guest.



Note: There might be problem is with the virtualbox-guest-x11 package missing.

From the Ubuntu Server command prompt,

Execute the below commands:

sudo apt-get update

sudo apt-get install virtualbox-guest-x11

If it asks you about keeping a file or installing the new one, select the new one.

sudo VBoxClient --clipboard

This should enable clipboard sharing.

```
jenkins@jenkins-server:~$ sudo apt-get update
[sudo] password for jenkins:
Hit:1 http://security.ubuntu.com/ubuntu noble-security InRelease
Hit:2 http://in.archive.ubuntu.com/ubuntu noble InRelease
Get:3 http://in.archive.ubuntu.com/ubuntu noble-updates InRelease [89.7 kB]
Hit:4 http://in.archive.ubuntu.com/ubuntu noble-backports InRelease
Get:5 http://in.archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [77.1 kB]
Get:6 http://in.archive.ubuntu.com/ubuntu noble-updates/universe amd64 Packages [35.6 kB]
Fetched 202 kB in 3s (65.8 kB/s)
Reading package lists... Done
jenkins@jenkins-server:~$ sudo apt-get install virtualbox-guest-x11
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  cpp cpp-13 cpp-13-x86-64-linux-gnu cpp-x86-64-linux-gnu gcc-13-base libegl-mesa0 libegl1 libepoxy0 libfontenc1 libgbm1 libisl23 libmpc3 libnotify-bin
  libnotify4 libwayland-server0 libxcvt0 libxfont2 virtualbox-guest-utils x11-xkb-utils x11-xserver-utils xcvt xfonts-base xfonts-encodings xfonts-utils
  xserver-common xserver-xorg-core
Suggested packages:
  cpp-doc gcc-13-locales cpp-13-doc notification-daemon nickle cairo-5c xorg-docs-core xfs | xserver xfonts-100dpi | xfonts-75dpi xfonts-scalable
The following NEW packages will be installed:
  cpp cpp-13 cpp-13-x86-64-linux-gnu cpp-x86-64-linux-gnu gcc-13-base libegl-mesa0 libegl1 libepoxy0 libfontenc1 libgbm1 libisl23 libmpc3 libnotify-bin
  libnotify4 libwayland-server0 libxcvt0 libxfont2 virtualbox-guest-utils virtualbox-guest-x11 x11-xkb-utils x11-xserver-utils xcvt xfonts-base
  xfonts-encodings xfonts-utils xserver-common xserver-xorg-core
0 upgraded, 27 newly installed, 0 to remove and 10 not upgraded.
Need to get 21.8 MB of archives.
After this operation, 57.0 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

How to SSH Into a VirtualBox Ubuntu Server

Installing SSH on the Virtual Machine

On the virtual machine, **install SSH** using the command:

sudo apt install openssh-server

Your SSH server will start up automatically. You can **check its status** using the following command:

sudo systemctl status ssh:

Otherwise **start ssh** server using this command:

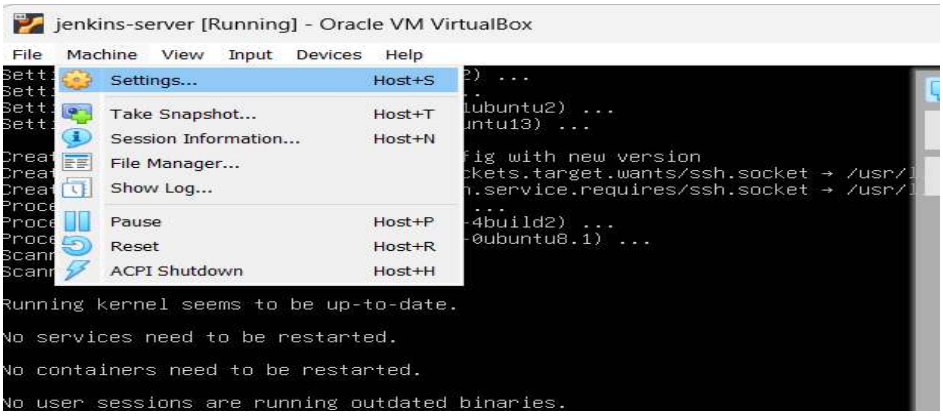
sudo systemctl start ssh

Now run this to get the **status of ssh** server:

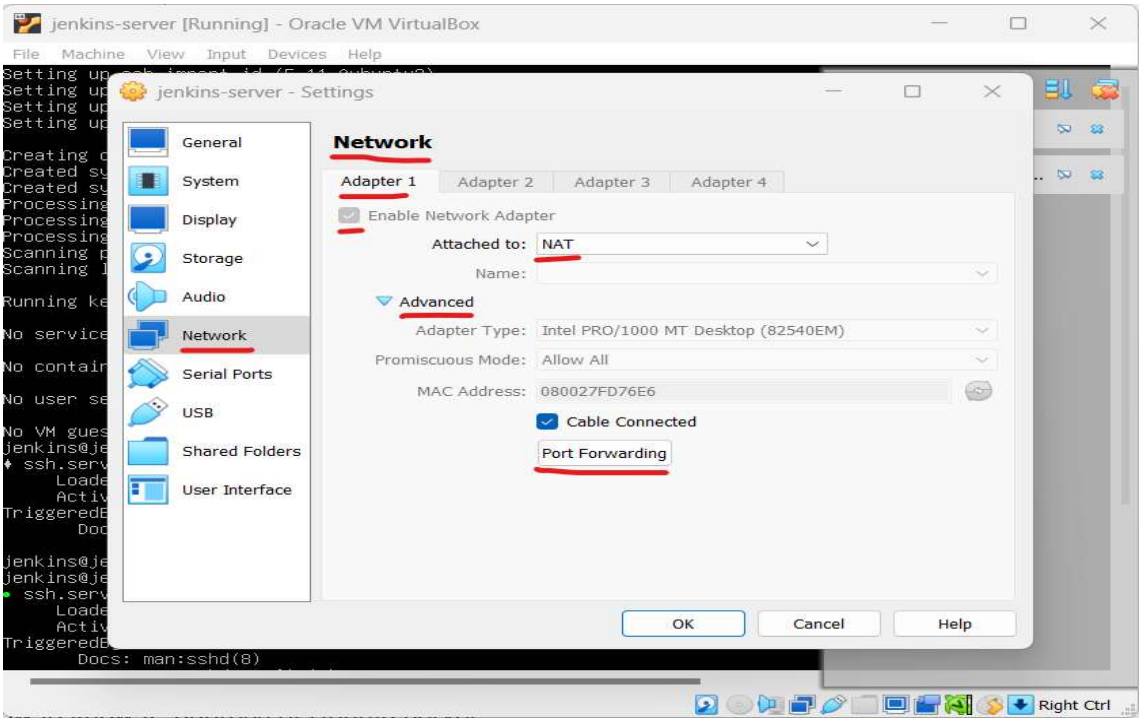
sudo systemctl status ssh

Enable port-forwarding

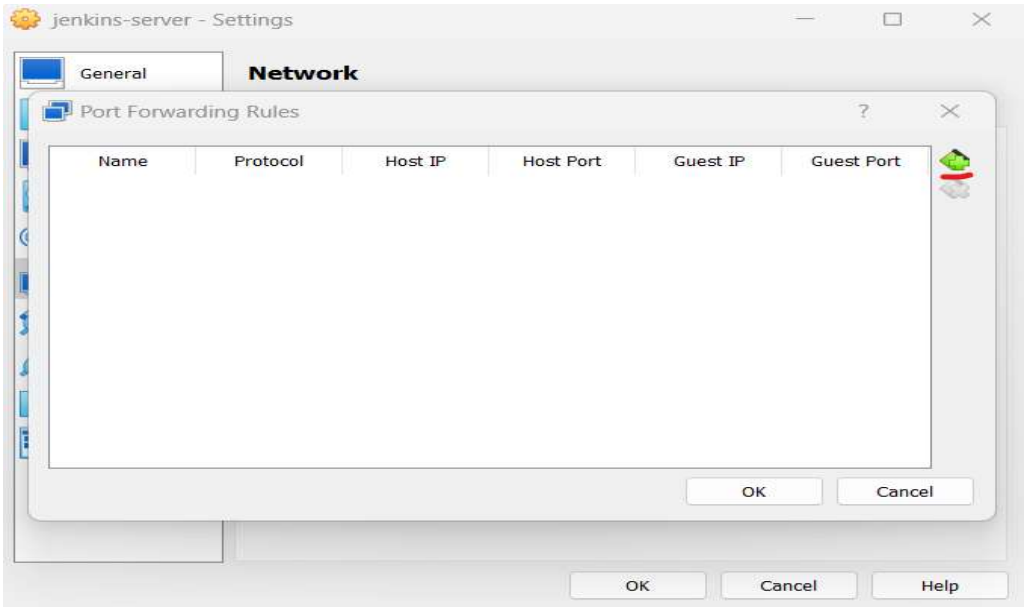
Open settings from guest OS → Machine → Settings…



Go to Network → Adapter 1 → Select NAT → Click on Advanced → Port Forwarding

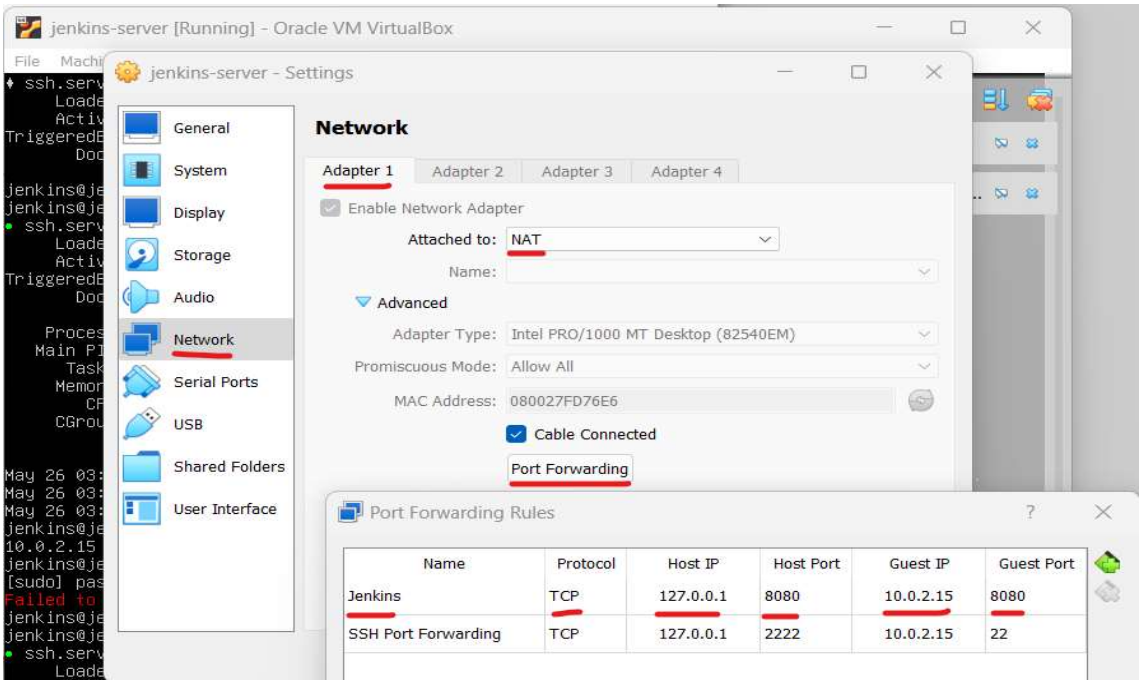


Click on Add sign as shown below:

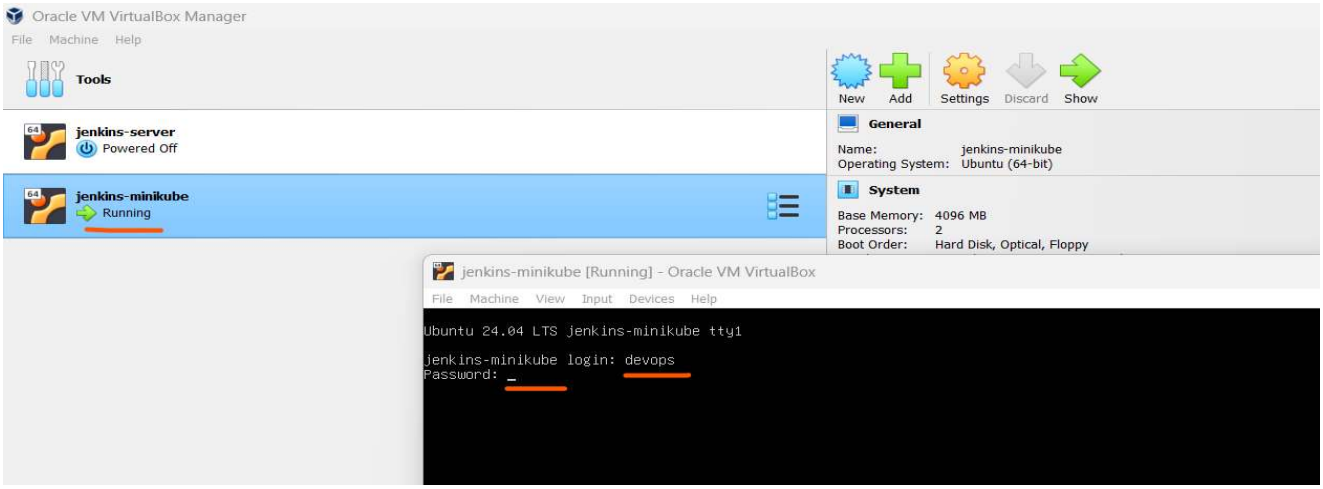


The real problem here is that the "Host IP" parameter in your VirtualBox's port forwarding rule only tells it to listen on 127.0.0.1 (loopback address). This means it will only accept connections made to 127.0.0.1

Make these entries like below:



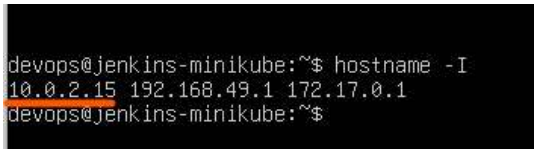
Please note that the Guest IP might be different for you. To know what is the IP address of your Ubuntu Guest OS, please go the Ubuntu terminal from VirtualBox, login to Ubuntu using the credential you had created after installation.



Login and run the below command to get IP address of your Ubuntu guest OS:

Hostname -I

(here ‘i' would be in UPPERCASE)



The IP address that starts with 10.0.X.X is what you should consider to put into NAT port forwarding.

Restart ssh server

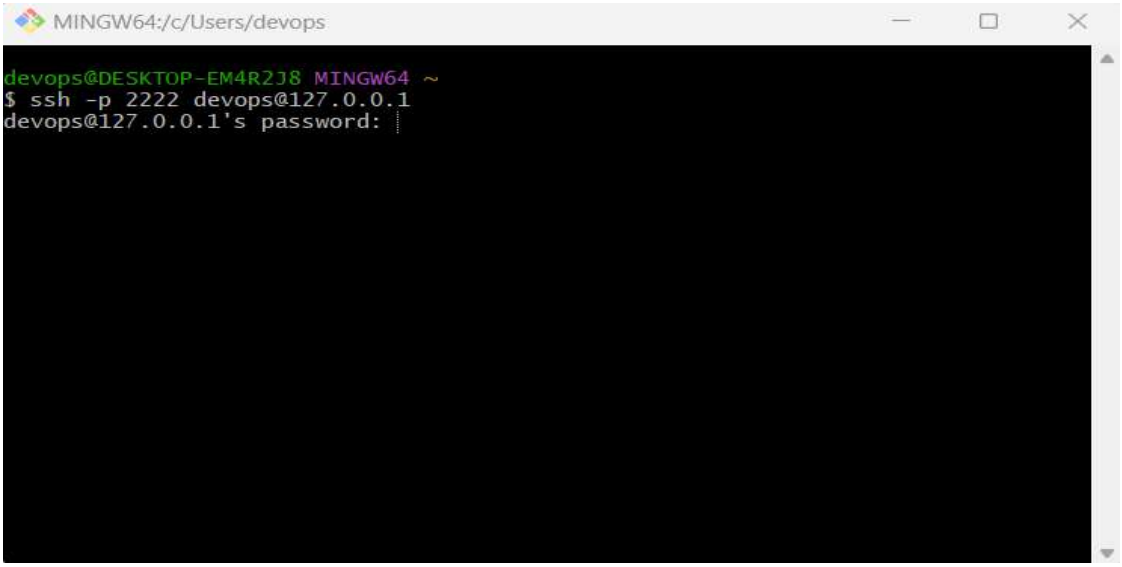
you might need to **restart ssh service** on VirtualBox for the changes to take effect.

sudo systemctl restart ssh

Start SSH Session (from Host Windows to Guest Ubuntu using Gitbash)

From the terminal in your main operating system, run the SSH command in the following format: ssh -p 2222 guest_os_username@127.0.0.1. For example:

ssh -p 2222 devops@127.0.0.1



Please note that 'jdevops', in this case, is the login username for the virtual machine. Finally, enter the password for the guest OS user when prompted to initialize the connection.

Now onwards, we will be using Gitbash to login to Ubuntu guest OS in VirtualBox from our Windows host machine.

Shutdown Ubuntu server

sudo poweroff

Restart Ubuntu server

sudo reboot

Installing update on Ubuntu server

Clear the console using the ”clear” command and update the system using “**sudo apt update**” command.

```
jenkins@jenkins-server:~$ sudo apt update
[sudo] password for jenkins:
Hit:1 http://security.ubuntu.com/ubuntu noble-security InRelease
Hit:2 http://in.archive.ubuntu.com/ubuntu noble InRelease
Get:3 http://in.archive.ubuntu.com/ubuntu noble-updates InRelease [89.7 kB]
Hit:4 http://in.archive.ubuntu.com/ubuntu noble-backports InRelease
Get:5 http://in.archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [77.0 kB]
Get:6 http://in.archive.ubuntu.com/ubuntu noble-updates/universe amd64 Packages [35.6 kB]
Fetched 202 kB in 4s (51.2 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
10 packages can be upgraded. Run 'apt list --upgradable' to see them.
jenkins@jenkins-server:~$
```

Installing Jenkins on Ubuntu server

Jenkins is a open-source automation server that lets you build, test and deploy your code. Now, we will see how to install Jenkins on the newly created Ubuntu server.

Install Java as Jenkins pre-requisite

Jenkins is a Java based application. So, Java is a pre-requisite. Install Java with the following command in Ubuntu server –

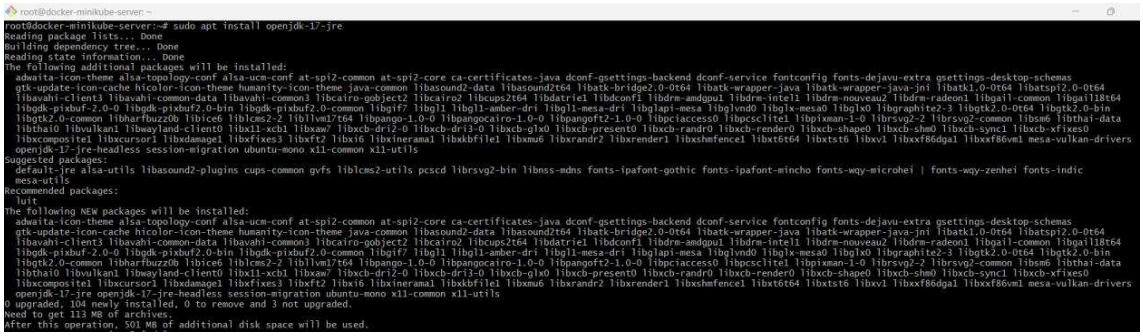
sudo apt update

sudo apt upgrade

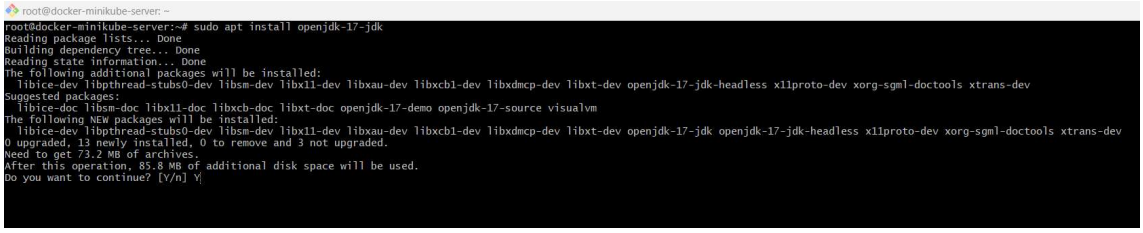
```
root@docker-minikube-server: ~
root@docker-minikube-server:~# apt-cache search openjdk | grep openjdk-17
openjdk-17-dbg - Java runtime based on OpenJDK (debugging symbols)
openjdk-17-jdk - OpenJDK Development Kit (JDK)
openjdk-17-jdk-headless - OpenJDK Development Kit (JDK) (headless)
openjdk-17-jre - OpenJDK Java runtime, using Hotspot JIT
openjdk-17-jre-headless - OpenJDK Java runtime, using Hotspot JIT (headless)
openjdk-17-source - OpenJDK Development Kit (JDK) source files
openjdk-17-demo - Java runtime based on OpenJDK (demos and examples)
openjdk-17-doc - OpenJDK Development Kit (JDK) documentation
openjdk-17-jre-zero - Alternative JVM for OpenJDK, using Zero
uwsgi-plugin-jvm-openjdk-17 - Java plugin for uWSGI (OpenJDK 17)
uwsgi-plugin-jwsgi-openjdk-17 - JWSGI plugin for uWSGI (OpenJDK 17)
uwsgi-plugin-ring-openjdk-17 - Closure/Ring plugin for uWSGI (OpenJDK 17)
uwsgi-plugin-servlet-openjdk-17 - JWSGI plugin for uWSGI (OpenJDK 17)
root@docker-minikube-server:~#
```

Depending on your version of Ubuntu, you may be able to install OpenJDK 17 JDE and JRE using the following command. This is only applicable if these packages are available in your distribution:

sudo apt install openjdk-17-jre



sudo apt install openjdk-17-jdk



Confirm the installation by running the following command.

java -version

```
jenkins@jenkins-server:~$ java -version
openjdk version "17.0.11" 2024-04-16
OpenJDK Runtime Environment (build 17.0.11+9-Ubuntu-1)
OpenJDK 64-Bit Server VM (build 17.0.11+9-Ubuntu-1, mixed mode, sharing)
jenkins@jenkins-server:~$
```

Install Jenkins

Run the below commands to install Jenkins on Ubuntu:

1st command:

sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \

https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

```
root@jenkins-server: ~
root@jenkins-server:~# sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]" \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install jenkins
--2024-05-26 05:12:30--  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
Resolving pkg.jenkins.io (pkg.jenkins.io)... 151.101.130.133, 151.101.66.133, 151.101.2.133, ...
Connecting to pkg.jenkins.io (pkg.jenkins.io)|151.101.130.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3175 (3.1K) [application/pgp-keys]
Saving to: '/usr/share/keyrings/jenkins-keyring.asc'

/usr/share/keyrings/jenkins-keyring.asc             100%[=========================================

2024-05-26 05:12:31 (30.7 MB/s) - '/usr/share/keyrings/jenkins-keyring.asc' saved [3175/3175]

Ign:1 https://pkg.jenkins.io/debian-stable binary/ InRelease
Get:2 https://pkg.jenkins.io/debian-stable binary/ Release [2,044 B]
Get:3 https://pkg.jenkins.io/debian-stable binary/ Release.gpg [833 B]
Hit:4 http://in.archive.ubuntu.com/ubuntu noble InRelease
Hit:5 http://security.ubuntu.com/ubuntu noble-security InRelease
Get:6 http://in.archive.ubuntu.com/ubuntu noble-updates InRelease [89.7 kB]
Hit:7 http://in.archive.ubuntu.com/ubuntu noble-backports InRelease
Get:8 https://pkg.jenkins.io/debian-stable binary/ Packages [26.9 kB]
Fetched 119 kB in 3s (40.3 kB/s)
```

2nd command:

echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]" \

https://pkg.jenkins.io/debian-stable binary/ | sudo tee \

/etc/apt/sources.list.d/jenkins.list > /dev/null

3rd command:

sudo apt-get update

4th command:

sudo apt-get install jenkins

After installation, please run the below commands to configure Jenkins:

Enable Jenkins

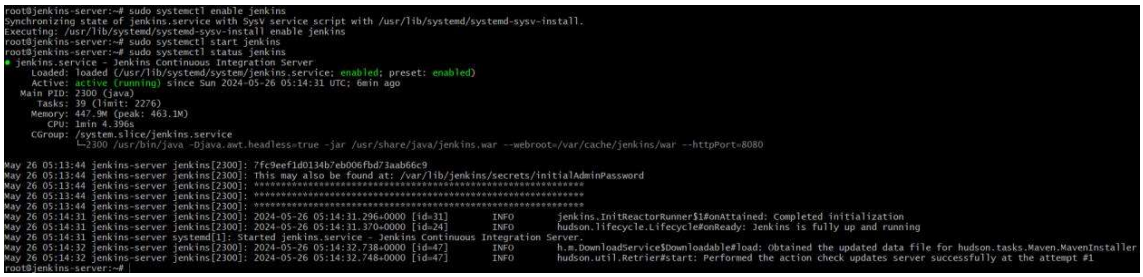
sudo systemctl enable jenkins

Start Jenkins

sudo systemctl start jenkins

Check status of Jenkins

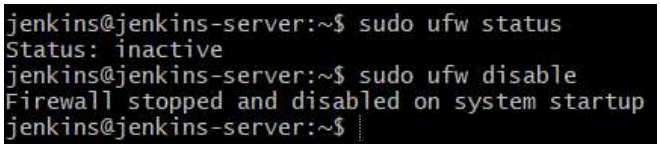
sudo systemctl status jenkins



[Optional] If needed, **disable firewall** on Ubuntu server:

sudo ufw status

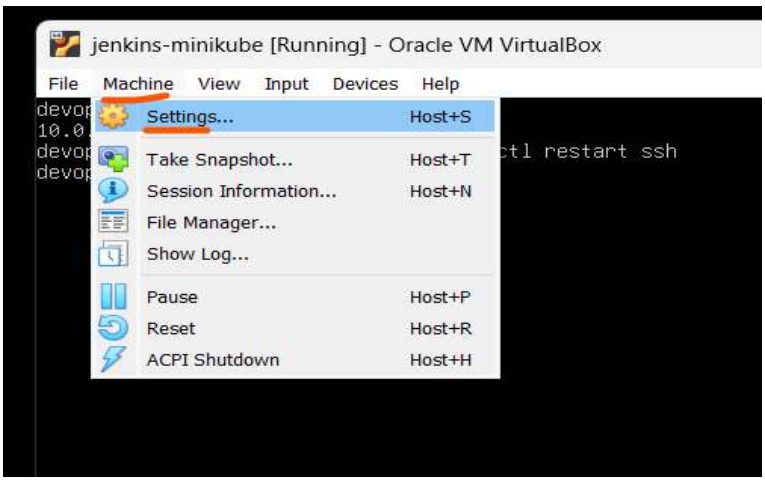
sudo ufw disable



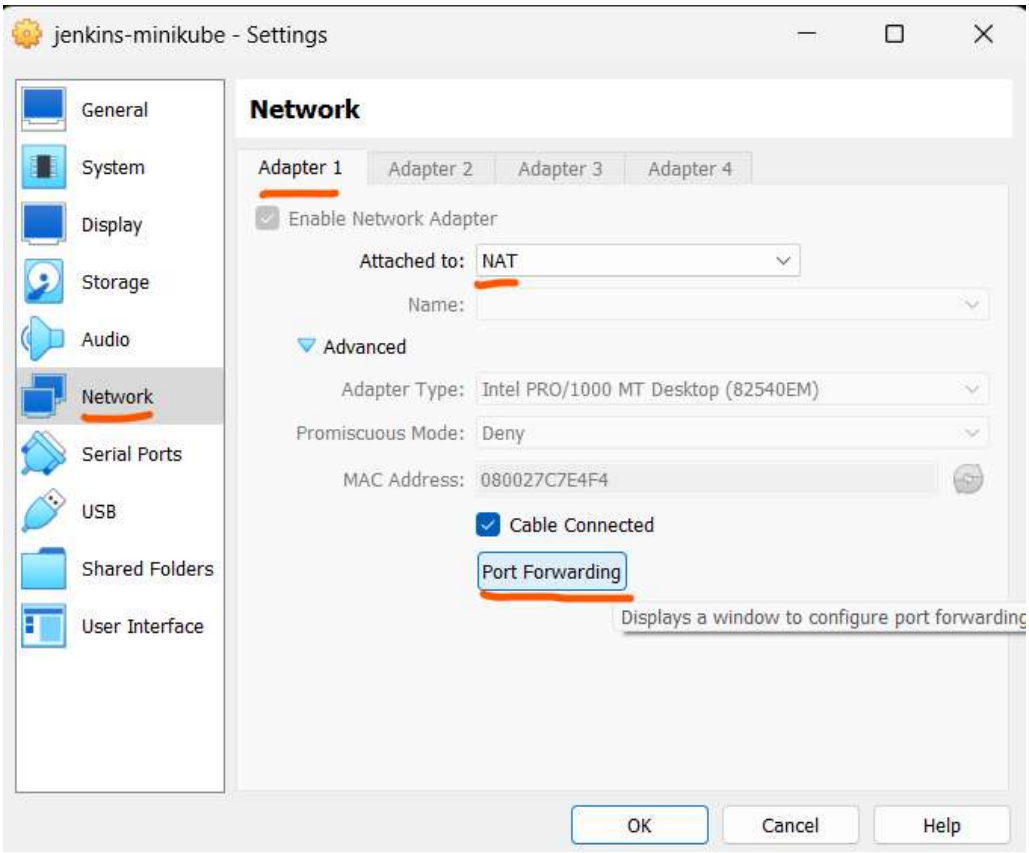
Set port forwarding for Jenkins (guest / Ubuntu to host / Windows)

If you haven't enabled port forwarding yet, please follow the below step. Otherwise ignore.

Go to Ubuntu server VirtualBox console, click on Machine → Settings

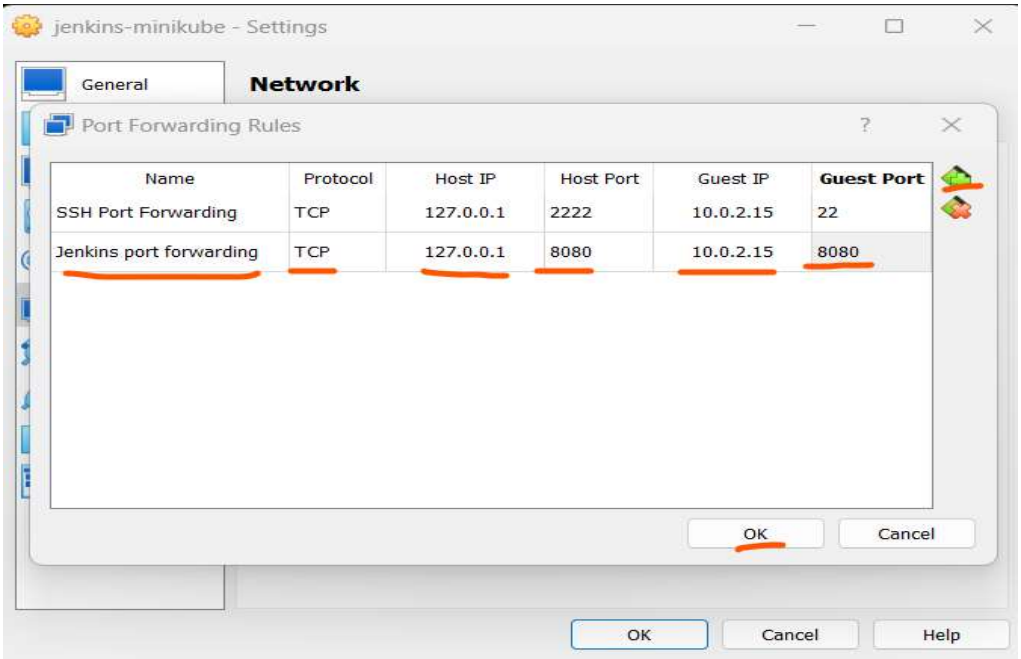


Click on network → Adapter 1 → NAT → click on Port Forwarding



Add port forwarding for Jenkins like this:

Click on Add sign (green colour) and put Jenkins port forwarding as below:



Access Jenkins (on VirtualBox) from Windows machine

If you have already installed Jenkins and configured port-forwarding, please proceed to next step. Otherwise, please follow steps for these.

Now open browser and access Jenkins:

http://<loopback address>:8080/

http://127.0.0.1:8080/

Or

http://localhost:8080

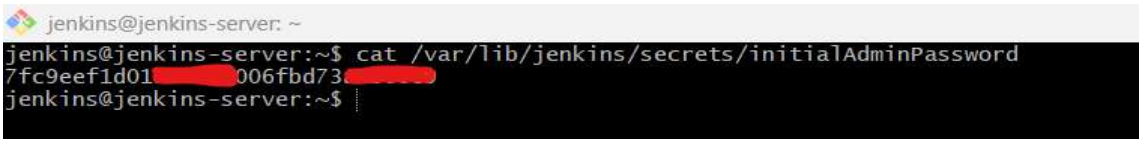
Unlock Jenkins / first time configuration

For the first time when you access Jenkins URL, you would be prompted to unlock Jenkins.

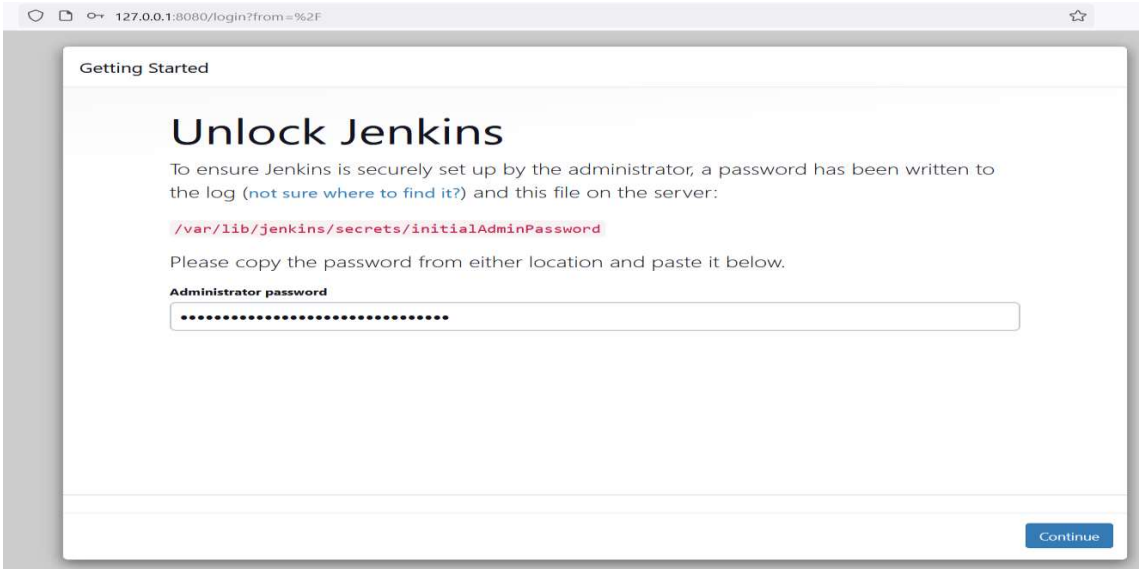
To ensure Jenkins is securely set up by the administrator, a password has been written to the log:

In the Ubuntu server, go to this location and read the file content →

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



Please copy the password from this location and paste it below.



Click on Continue.

Customize Jenkins

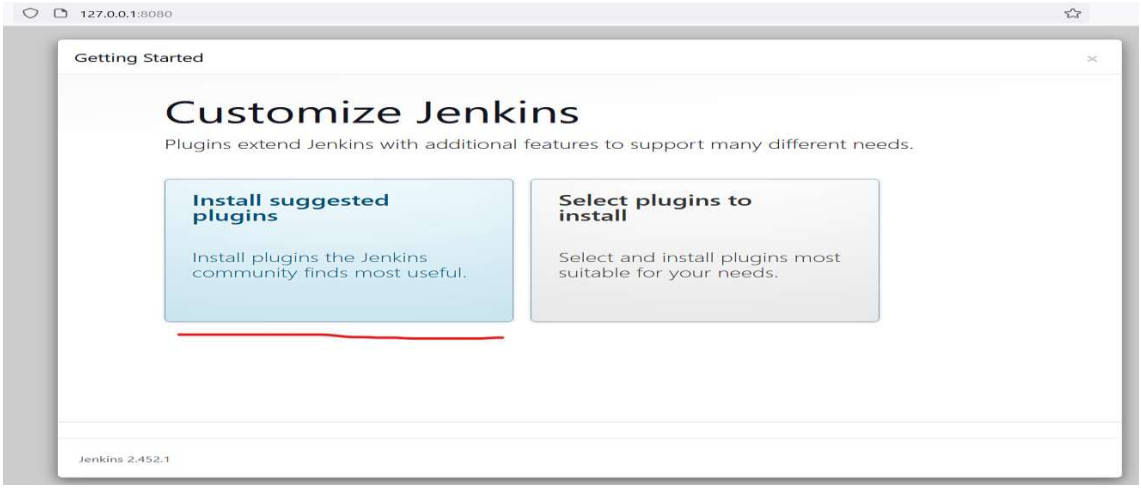
Initial Jenkins configuration

First time installation – default of set plugins.

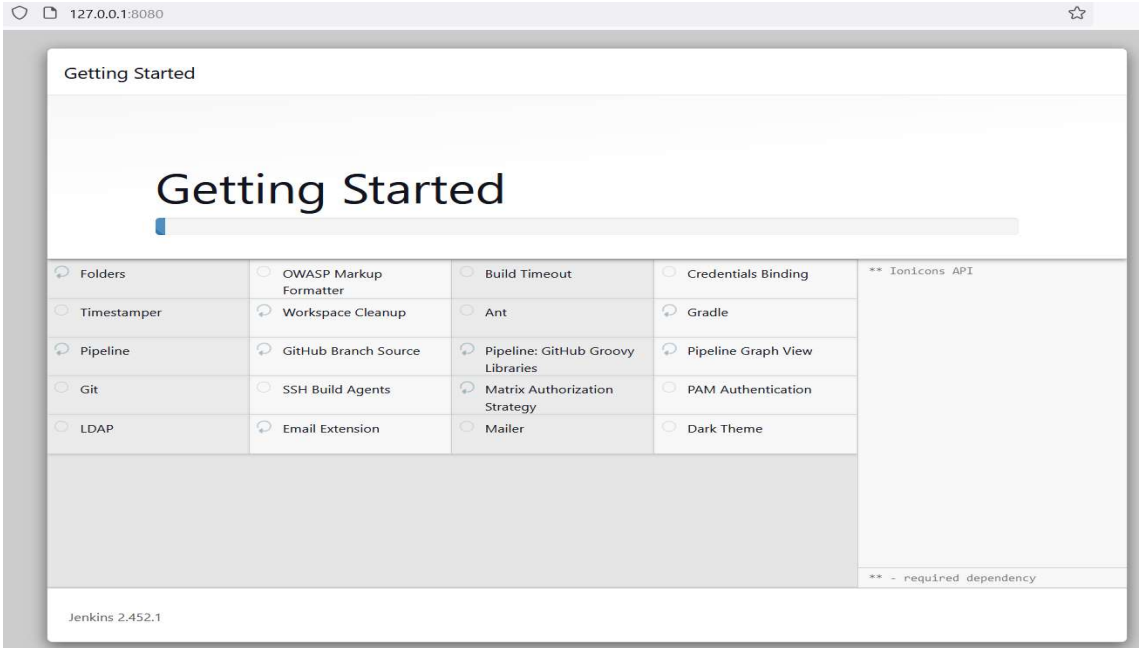
Plugins extend Jenkins with additional features to support many different needs.

For first time, you will get these options.

Please select – “Install suggested plugins”



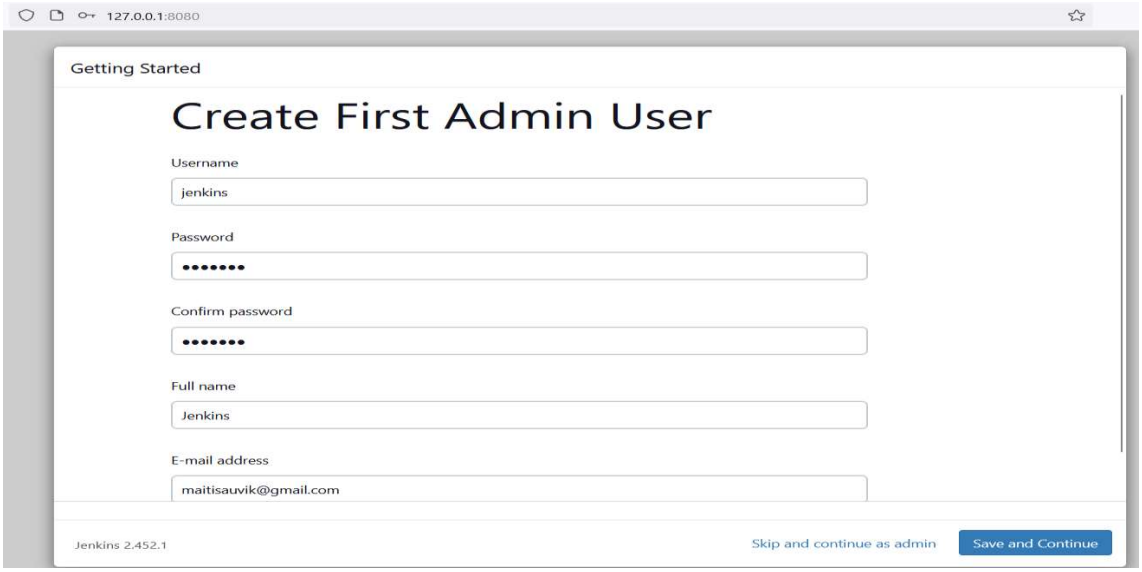
You will able to see the progress of plugin installation.



Create First Admin User

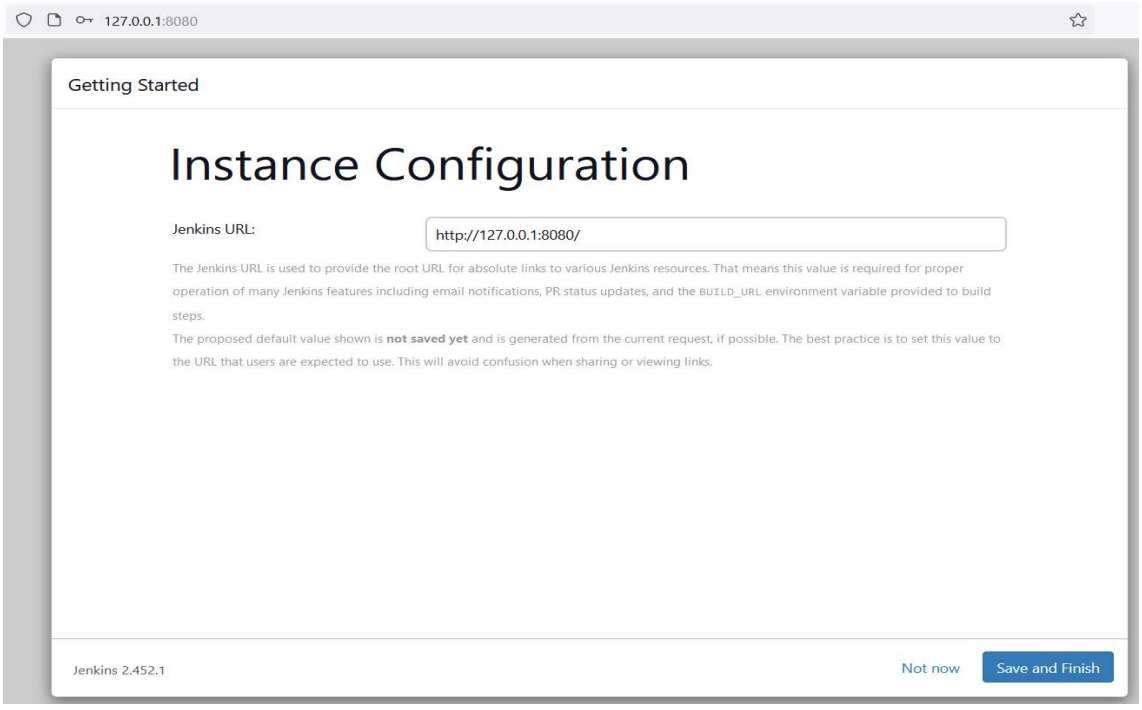
Put your preferred username, password with confirming password, full name of user and e-mail id of user.

Click on Save and continue

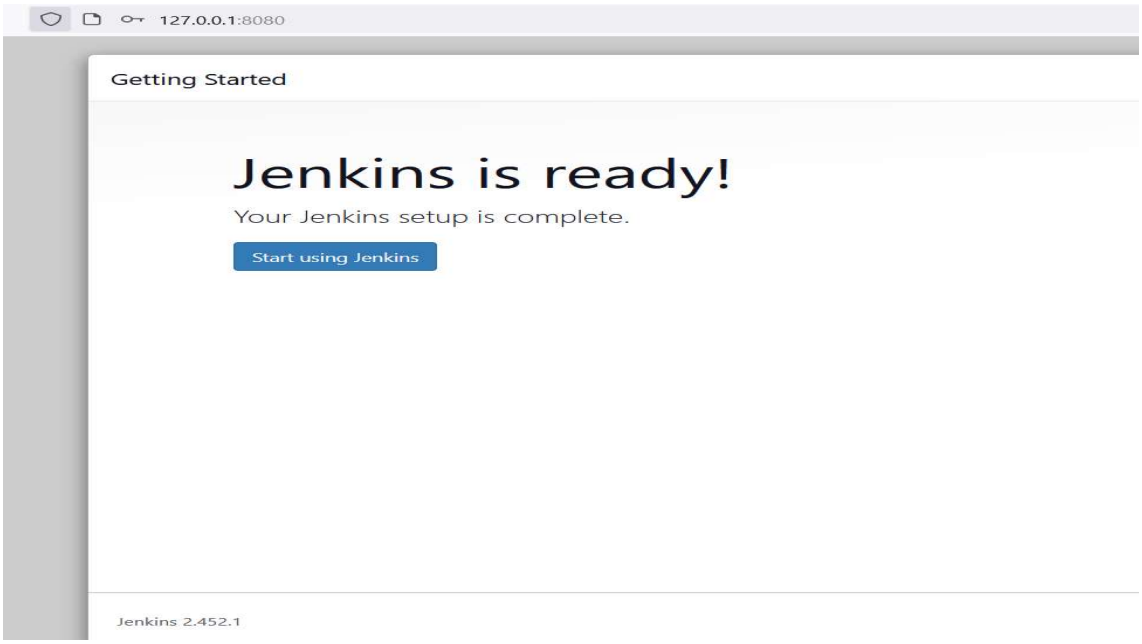


Instance Configuration

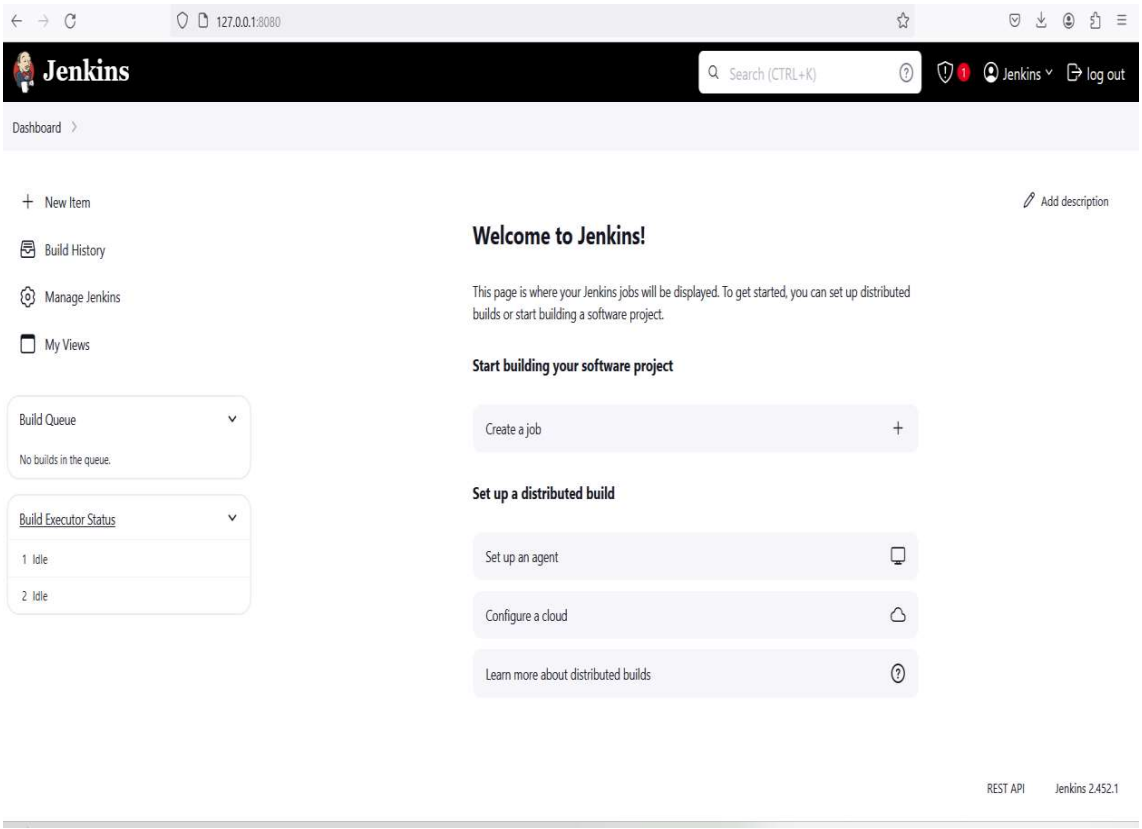
Leave the default Jankins as it is. Click on Save and Finish to continue.



Now you should be able to see that Jenkins is ready for use.



You will get the Jenkins home page like below:



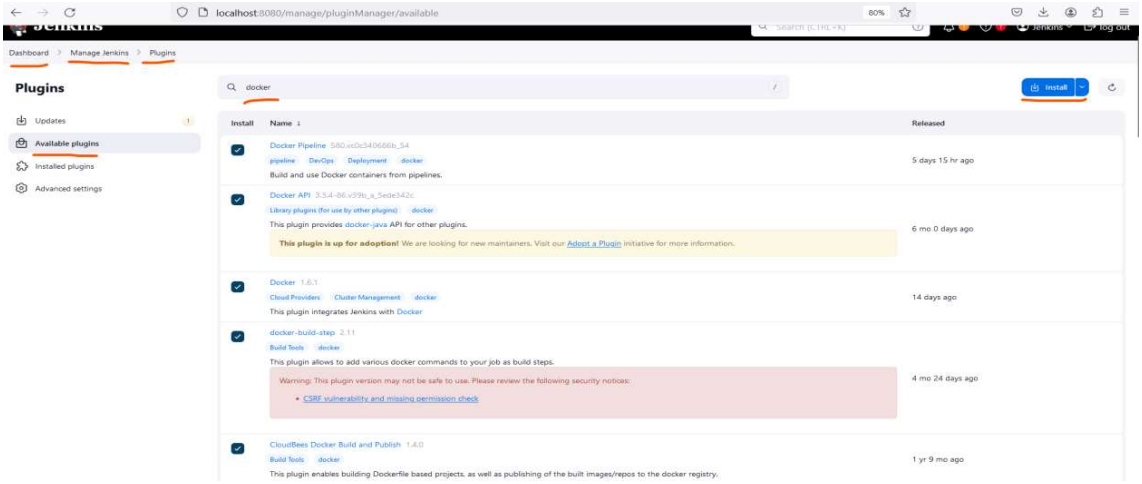
Jenkins plugins Installation

Ensure that the necessary Jenkins plugins are installed. It can be done through Jenkins Web interface: http://localhost:8080 from your windows machine

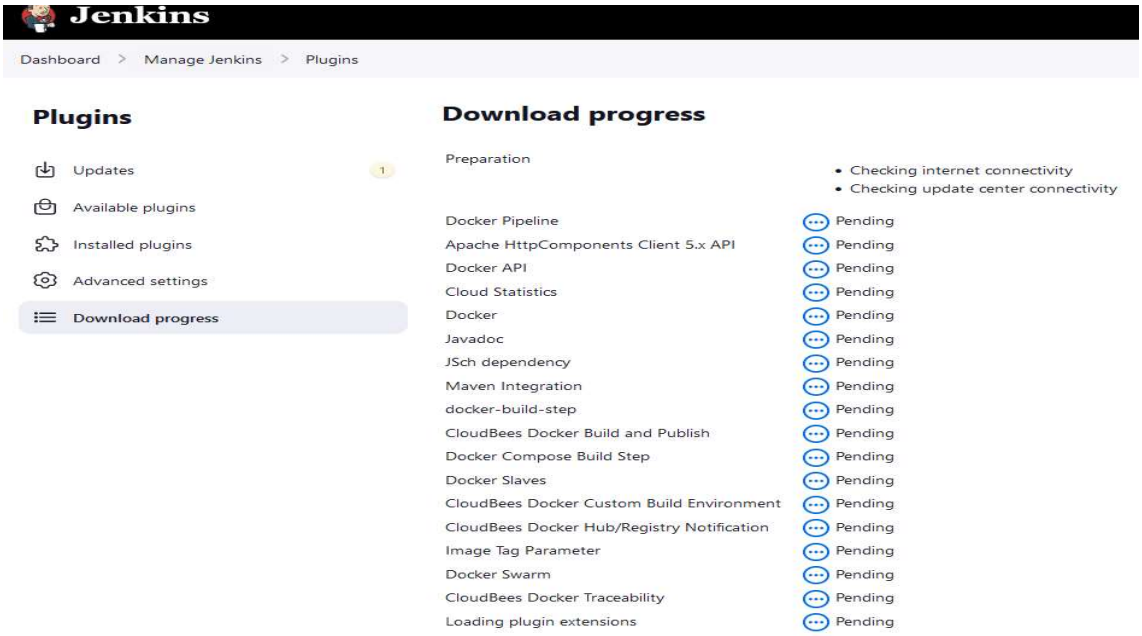
Docker plugin for Jenkins

Dashboard → Manage Jenkins → Plugins → Available plugins

Search for **docker** Select and install all docker related plugins as shown below.



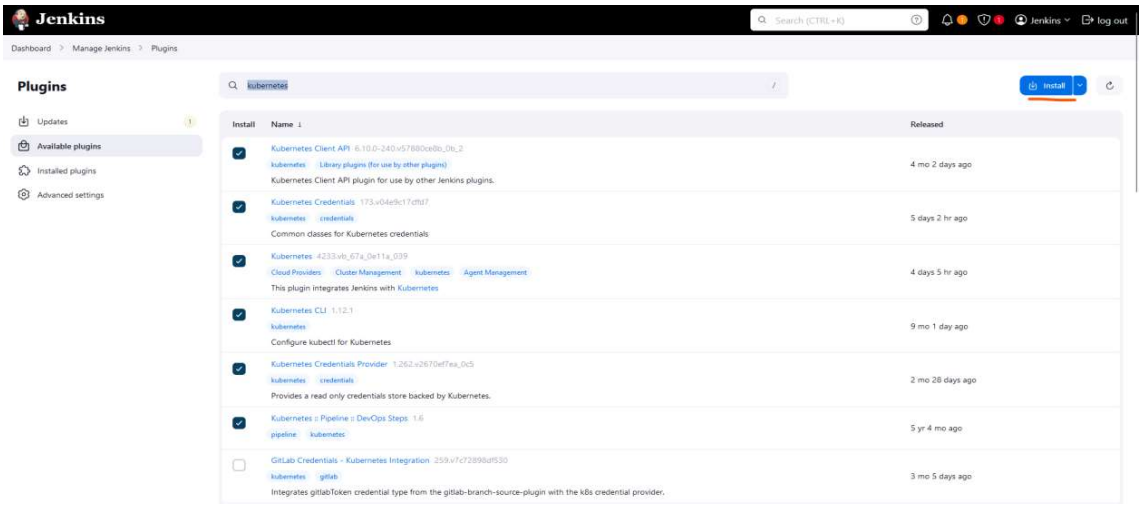
Once you click “Install”, you will see the progress.



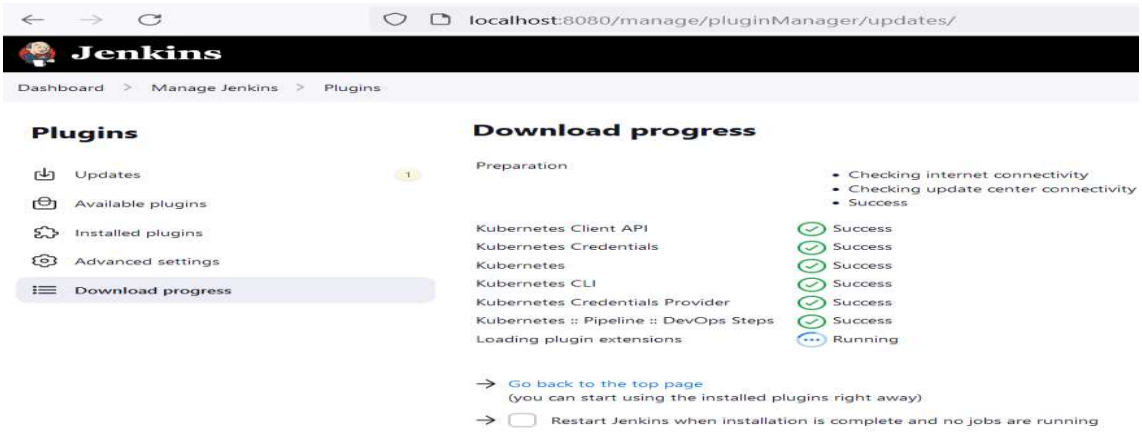
Kubernetes plugin for Jenkins

Jenkins URL → Dashboard → Manage Jenkins → Plugins → Available plugins

Search for **kubernetes**. Select and install all kubernetes related plugins as shown below.



Once you click “Install”, you will see the progress.



Once done, go to Ubuntu server where Jenkins has been installed and run the below command to restart Jenkins service:

sudo service jenkins restart

Open web browser and access Jenkins URL (http://localhost:8080/) once again.

Installing Docker from the Official Repository

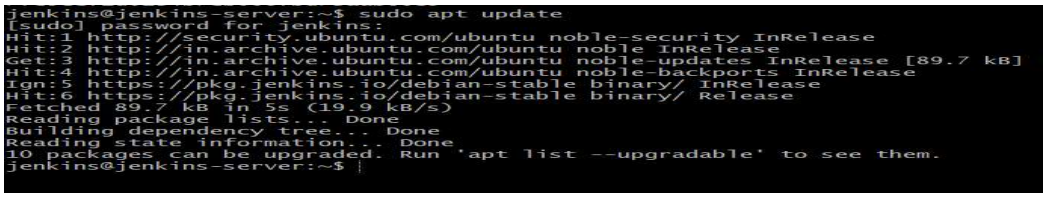
Install Docker from the official Docker repository to ensure you get the latest stable program version. To access the official Docker repository, add the new package source to Ubuntu and then install Docker. Follow the steps below:

Update the Package Repository

Run the following command to update the system's package repository and ensure the latest prerequisite packages are installed:

sudo apt update

When prompted, enter your root password and press Enter to proceed with the update.



Install Prerequisite Packages

The apt package manager requires a few prerequisite packages on the system to use packages over HTTPS. Run the following command to allow Ubuntu to access the Docker repositories over HTTPS:

sudo apt install apt-transport-https ca-certificates curl software-properties-common -y

Installing the prerequisite packages for Docker on Ubuntu.

The command above:

- Allows apt to transfer files and data over https.
- Allows the system to check security certificates.
- Installs curl, a data-transfer utility.

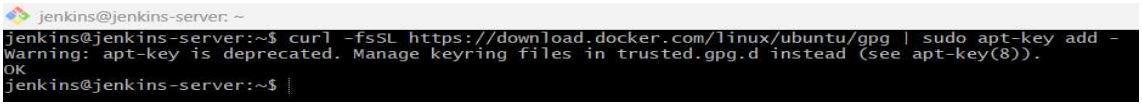
- Adds scripts for software management.

Add GPG Key

A GPG key verifies the authenticity of a software package. Add the Docker repository GPG key to your system by running:

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

Adding the Docker GPG key to verify package authenticity.



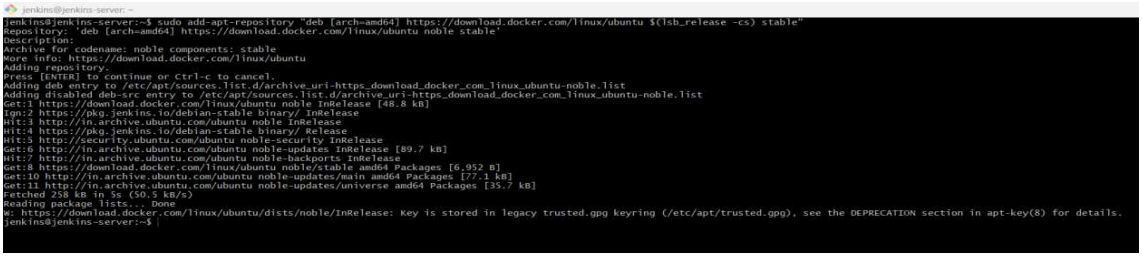
The output should state OK, verifying the authenticity.

Add Docker Repository

Run the following command to add the Docker repository to apt sources:

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

Adding the Docker official repository to the apt package manager.



The command adds the official Docker repository and updates the package database with the latest Docker packages.

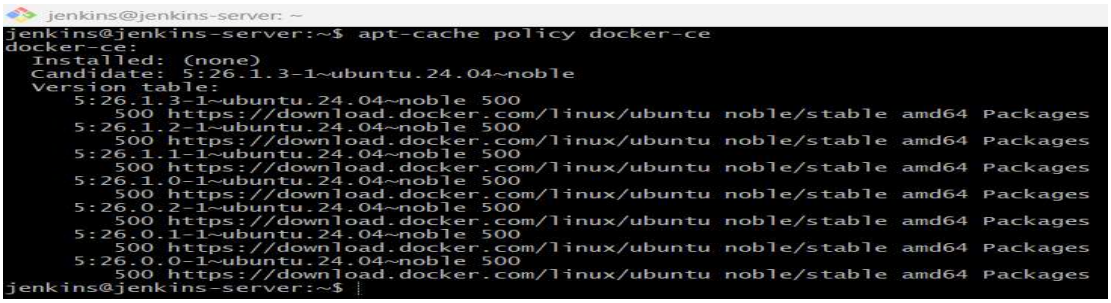
Specify Docker Installation Source

Execute the apt-cache command to ensure the Docker installation source is the Docker repository, not the Ubuntu repository. The apt-cache command queries the package cache of the apt package manager for the Docker packages we have previously added.

Run the following command:

apt-cache policy docker-ce

Specifying the Docker installation source.



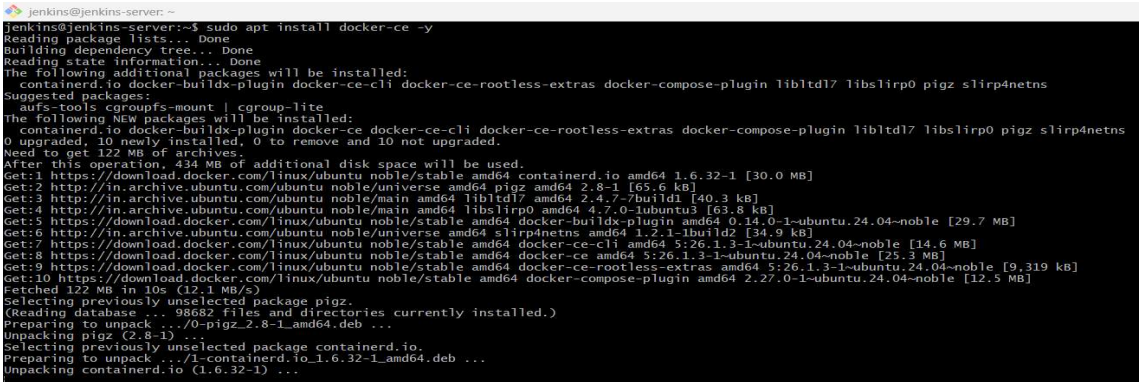
The output states which version is the latest in the added source repository.

Install Docker

Install Docker by running:

sudo apt install docker-ce docker-ce-cli containerd.io -y

Installing Docker on Ubuntu using the official repository.



Wait for the installation process to complete.

Set docker to start automatically

To start docker automatically when the instance starts, you can use the below command:

sudo systemctl enable docker

Start docker

You can use the below command:

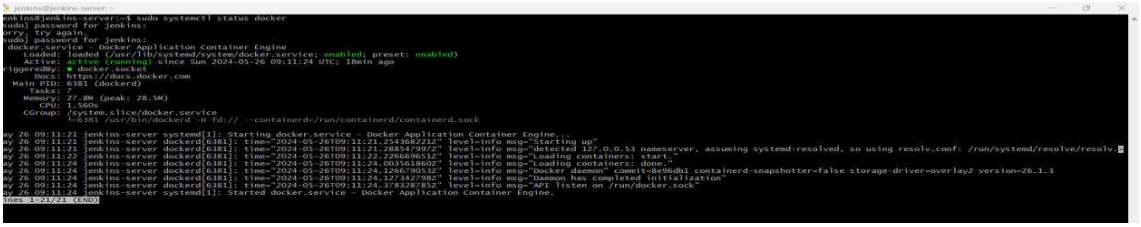
sudo systemctl start docker

Check Docker Status

Check if Docker is installed, the daemon started, and the process is enabled to start on boot. Run the following command:

sudo systemctl status docker

Checking the Docker daemon status.



The output states that the Docker daemon is up and running.

Set required permission for your Ubuntu user id to use Docker

Run the below command to add the user “devops” (it can be anything as you wish i.e. you had created in Ubuntu guest OS) to the “docker” group -

sudo usermod -a -G docker devops

How to restart docker

Do only if needed.

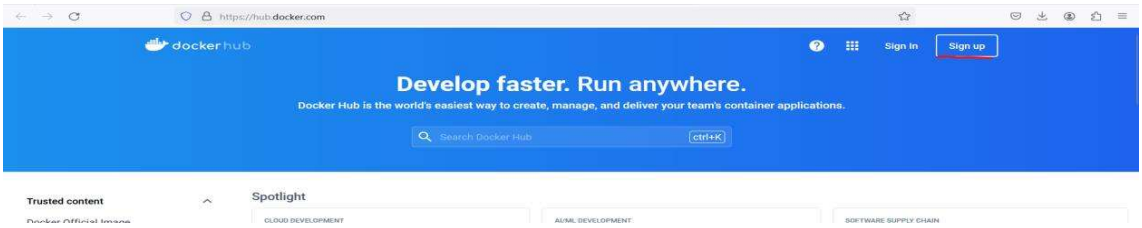
sudo systemctl daemon-reload

sudo systemctl restart docker

Create / manage Docker Hub account

Sign Up for Docker Hub account

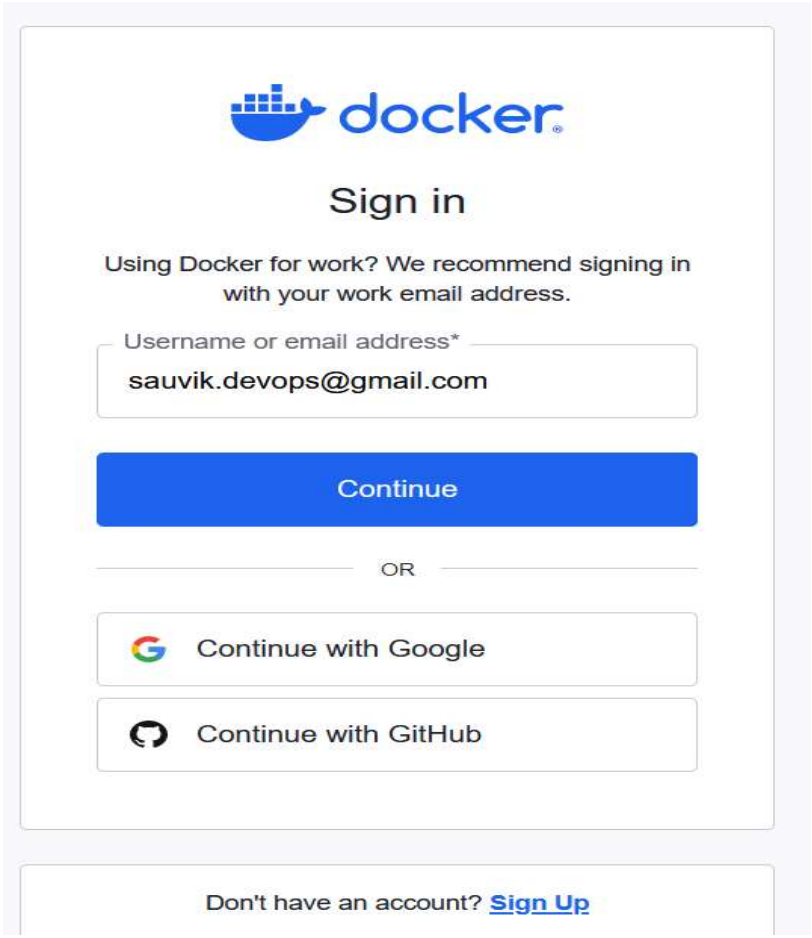
Visit https://hub.docker.com/ and click on “Sign up”



You can use your existing gmail account to signup for Docker Hub.

Sign In to Docker Hub account

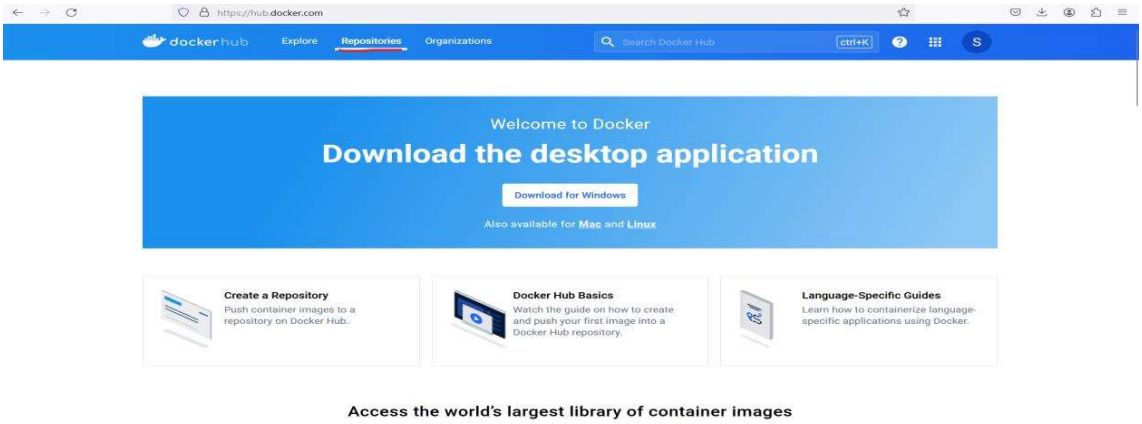
Once signup is done, please login to Docker Hub account using your gmail account.



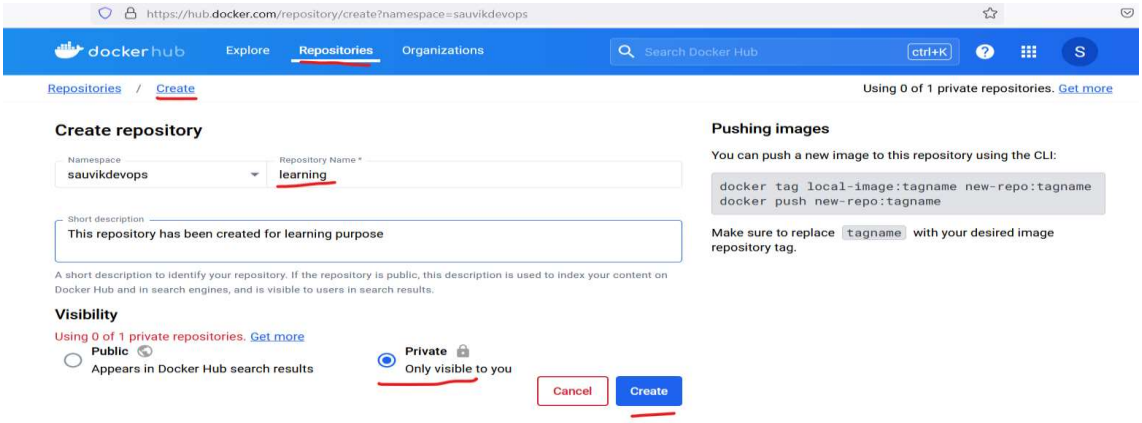
After sign in you will be able to see the home page.

Create repository in Docker Hub

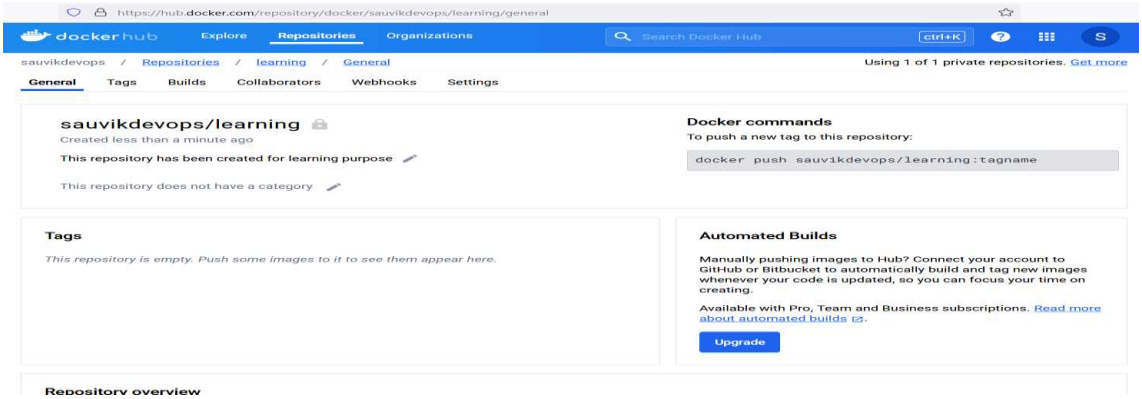
From Docker Hub home page after you login, click on '**Repositories**'



Give a name of your repository and select 'private' so that no anonymous login can happen. Now click on create to proceed.



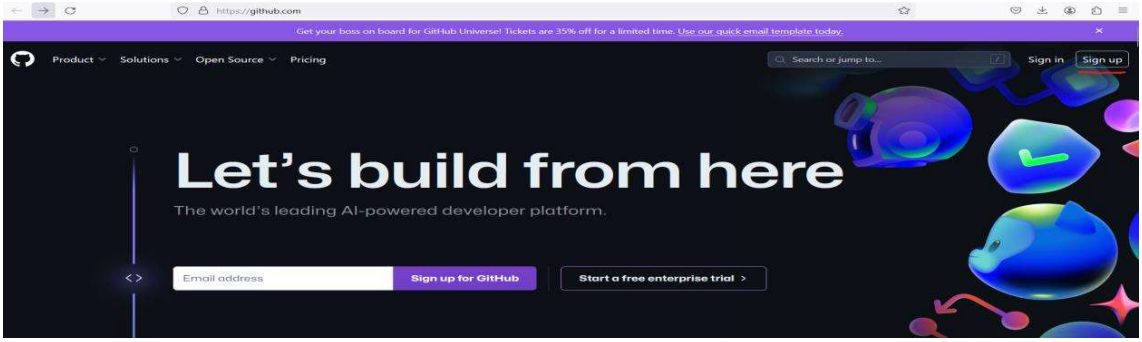
Your container registry would be created like below



Create / manage Git Hub account

Signup for GitHub account

Visit https://github.com/ and click on “Sign up”

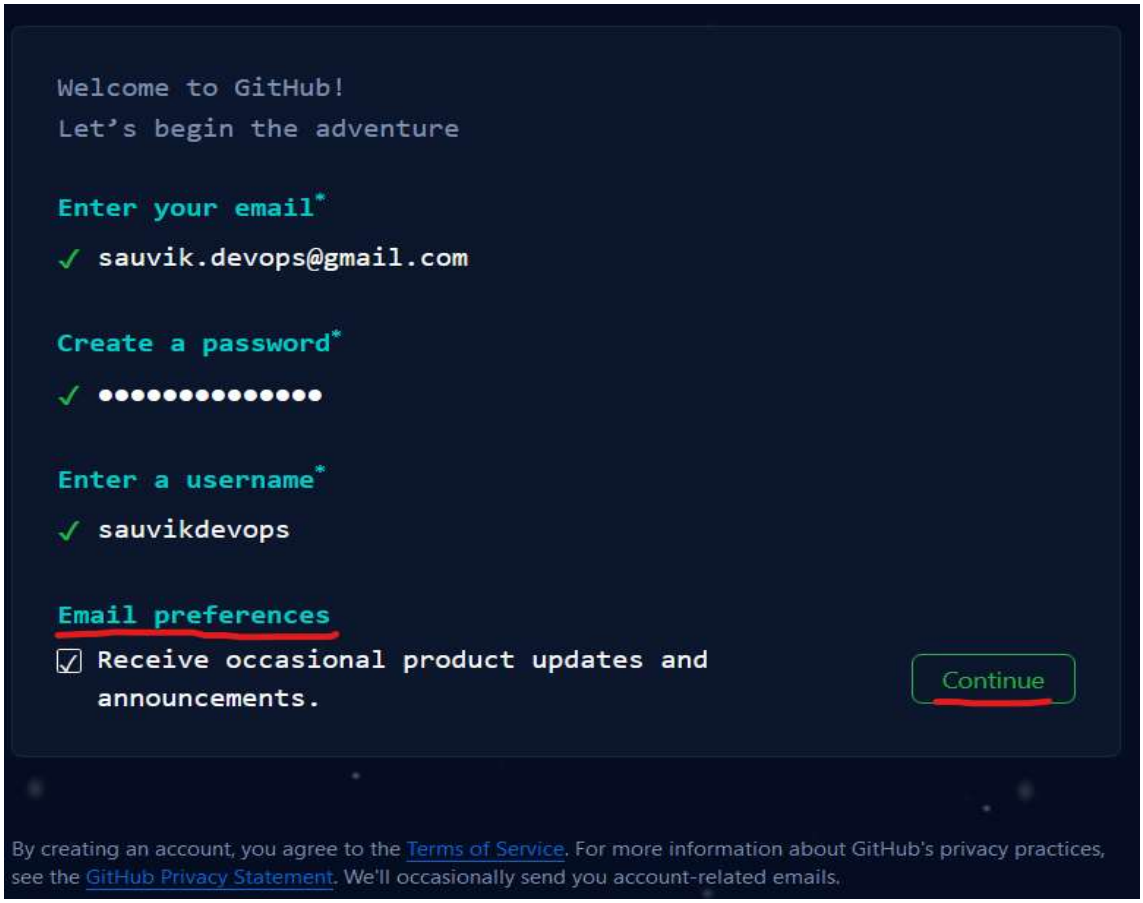


Put your e-mail ID and click on continue

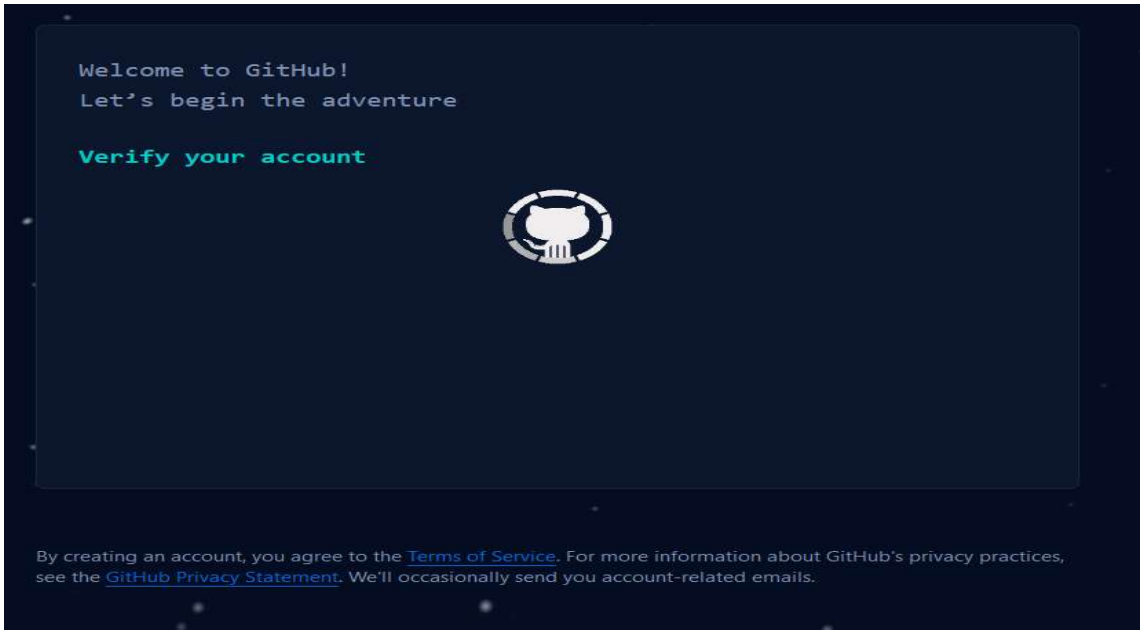
Set a strong password and click on Continue.

Pick a username for Docker Hub account and click on Continue.

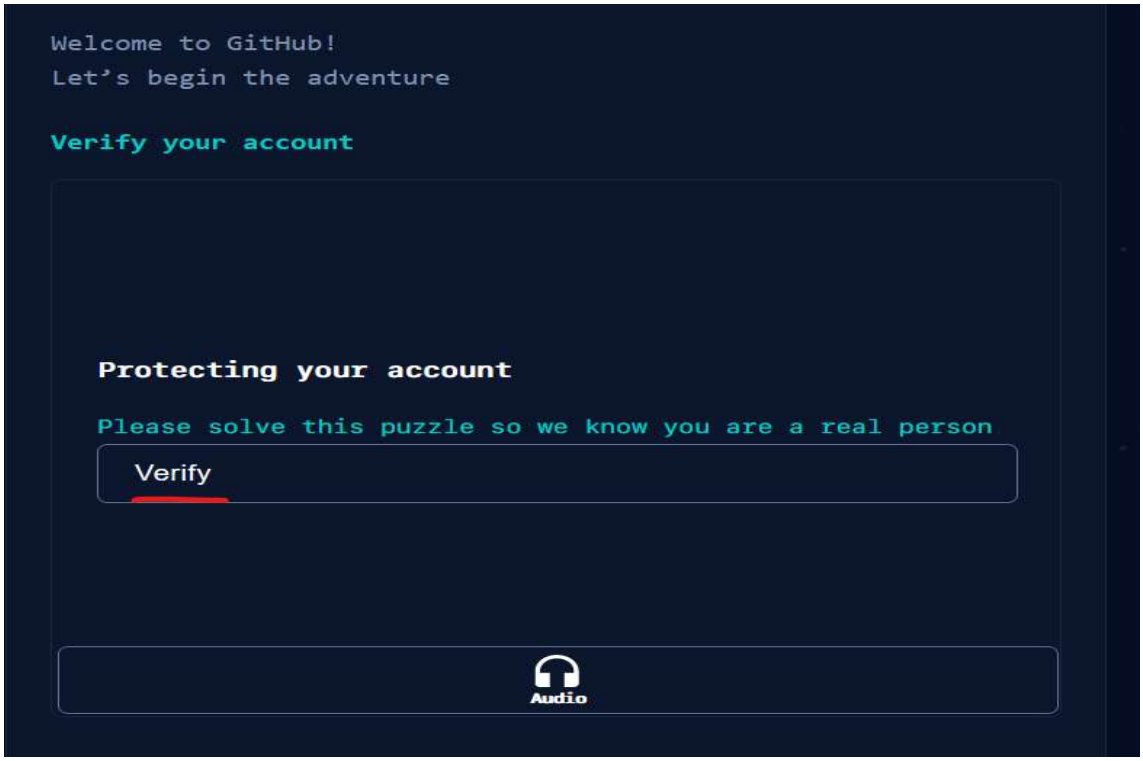
Click on Continue to proceed.



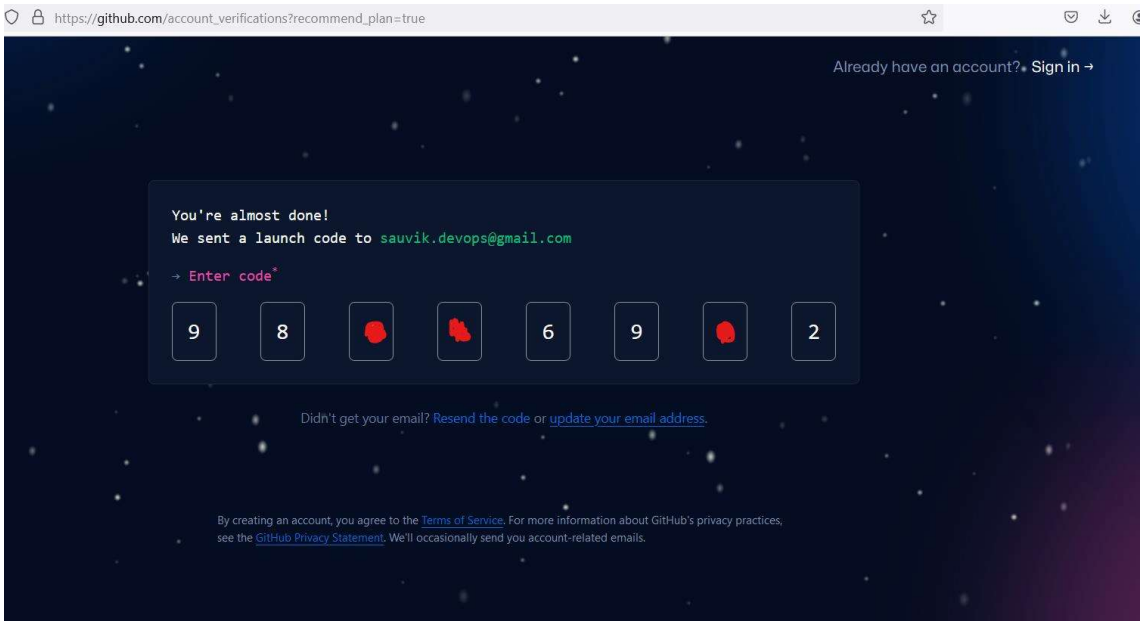
Your Github account will now get created.



Solve the puzzle for "Verify your account"



Check your email for launch code received from GitHub and then put on the GitHub account creation screen.

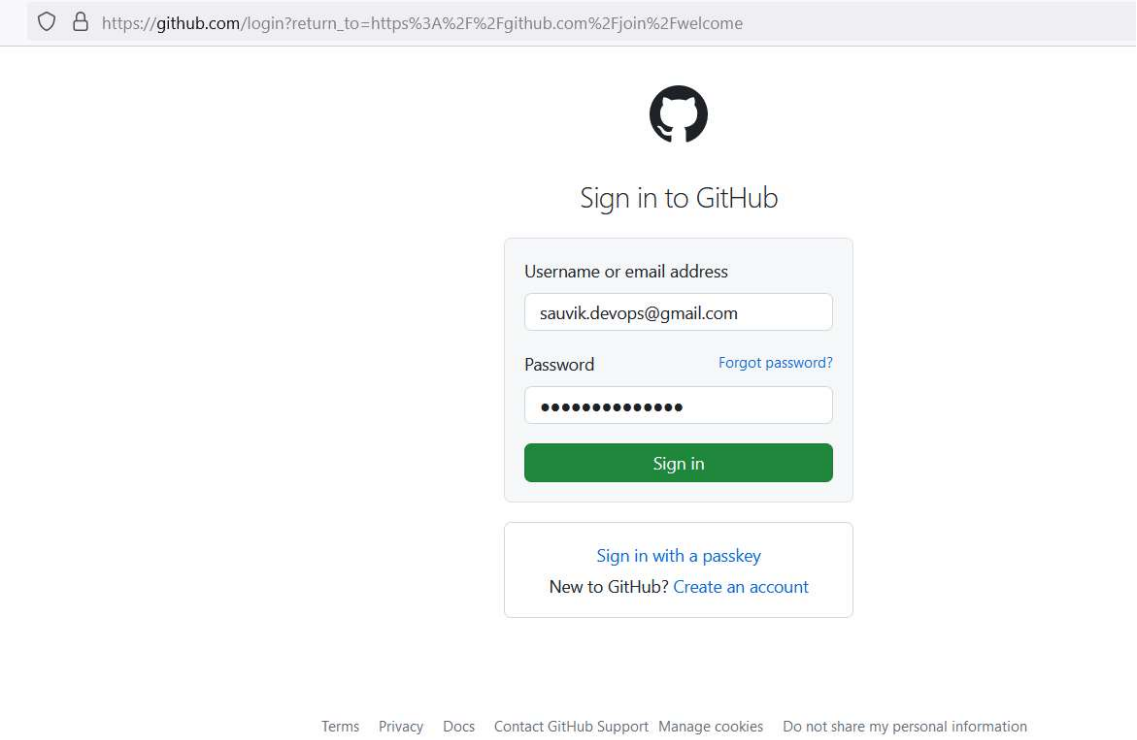


You are all set now.

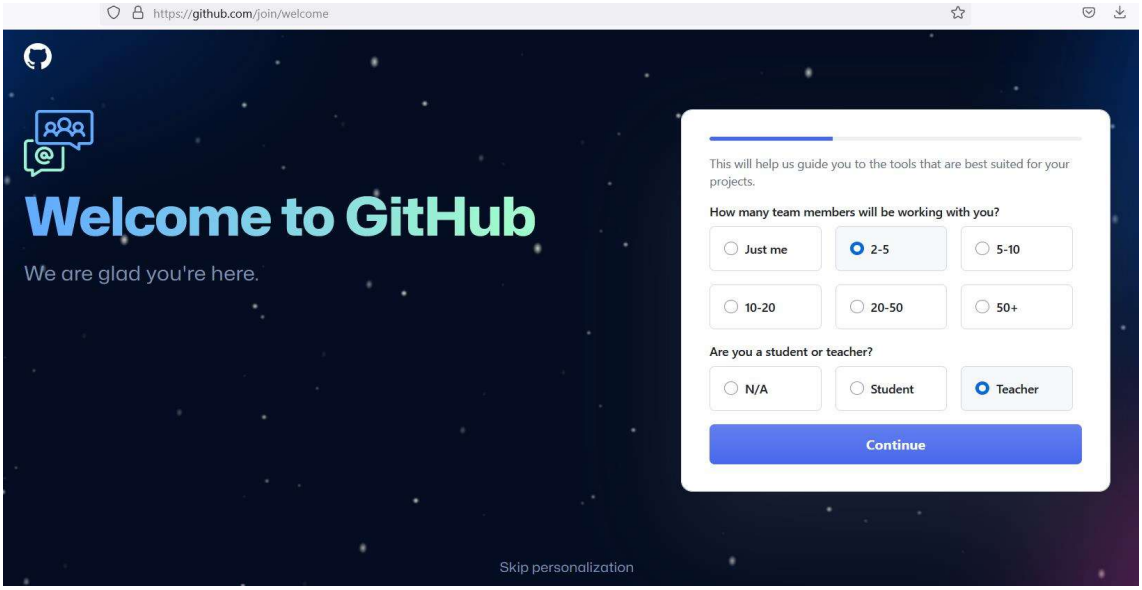
Login to GitHub account

Visit https://github.com/ and click on “Sign In”

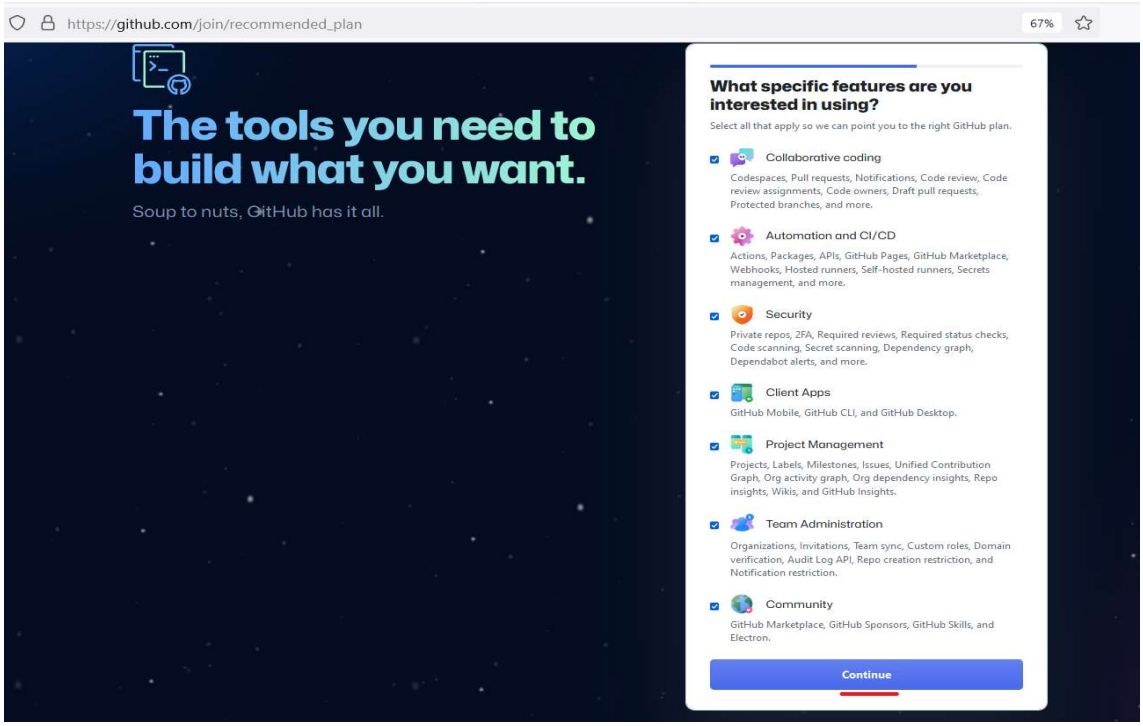
Put your GitHub account credential to login



Just fill the few questionaries to set the account as shown in below screen prints.

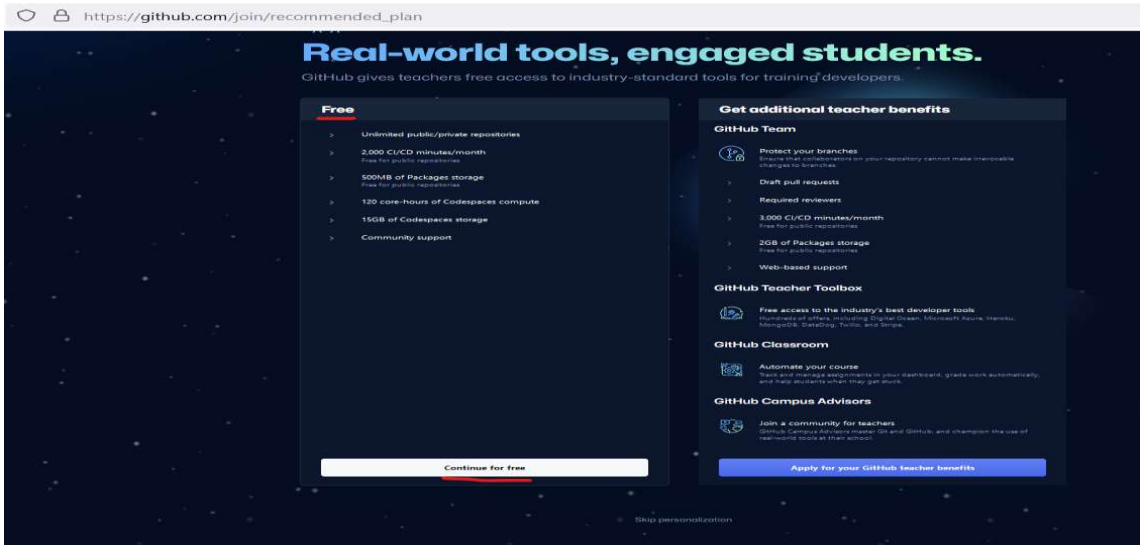


Click on continue

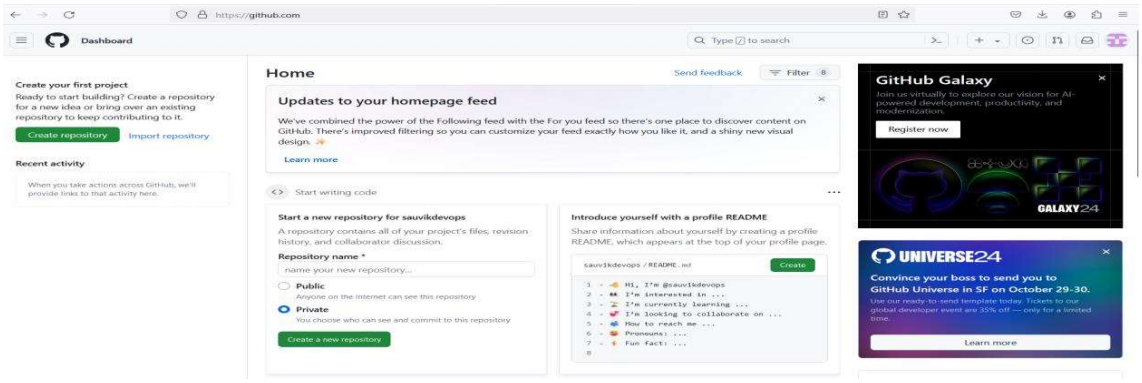


Select all options and click on – Continue

In the next screen, select “Free” account type.



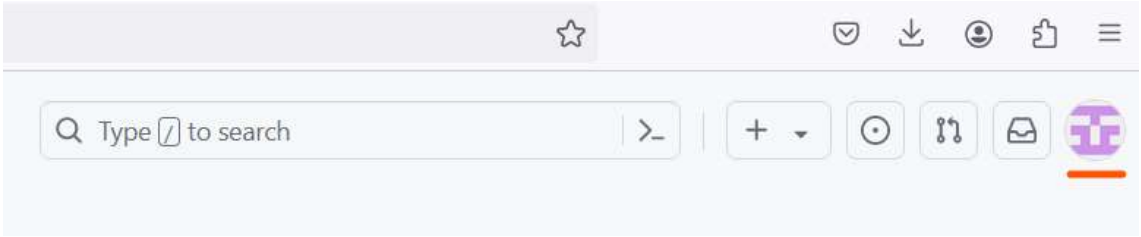
Your GitHub account setup is done and you will see the blow screen.



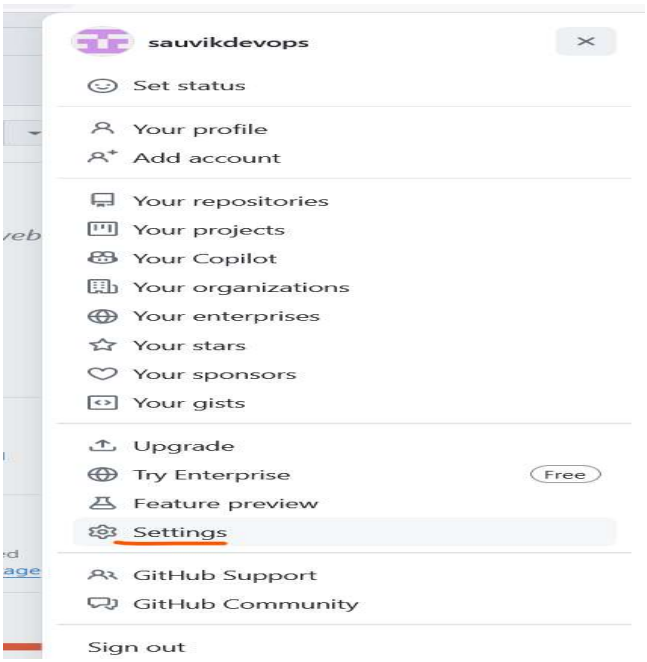
Generate Personal Access Token in GitHub

Personal access tokens are an alternative to using passwords for authentication to GitHub when using the GitHub API or the command line. Personal access tokens are intended to access GitHub resources on behalf of yourself.

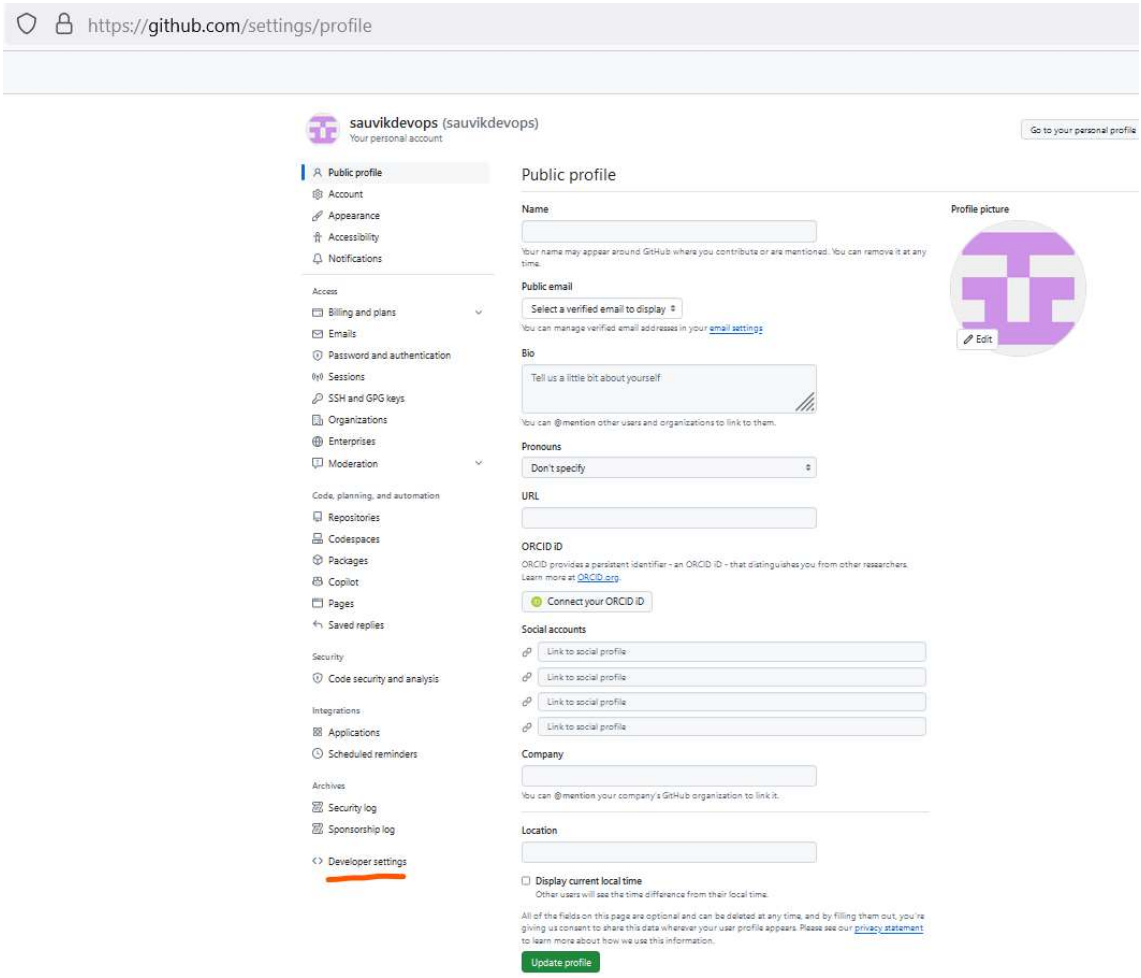
After you login to Click on the profile image (top right corner) of your GitHub account



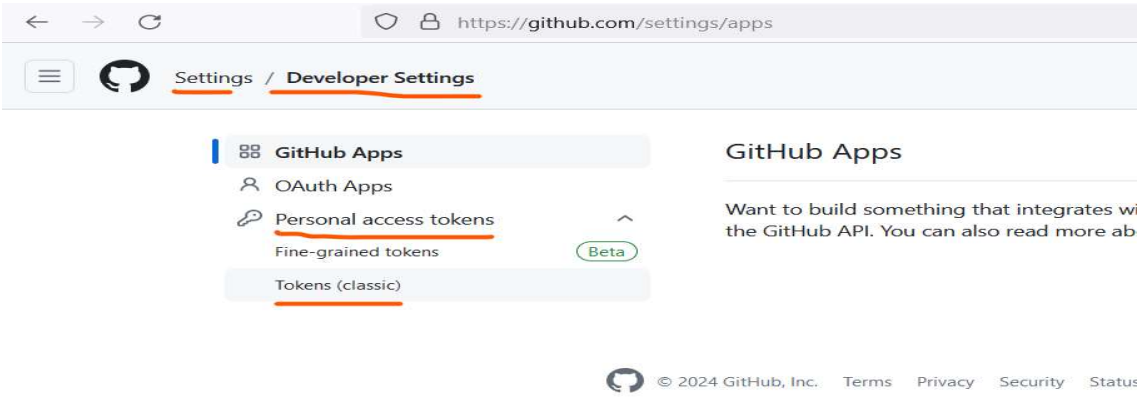
Now scroll down and click on Settings



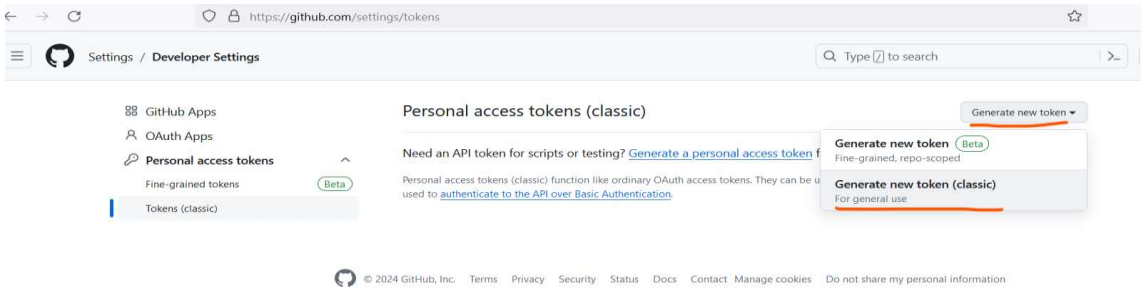
Scroll down and at bottom left Click on <> Developer Settings



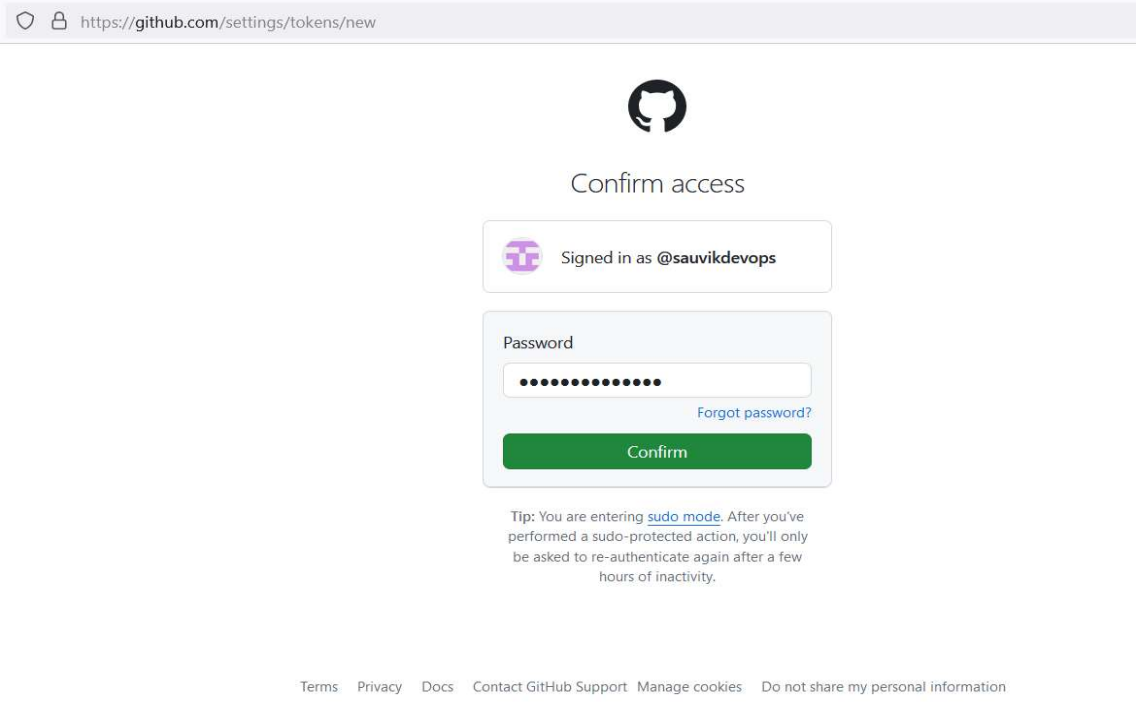
Click on Personal access tokens → Tokens (classic)



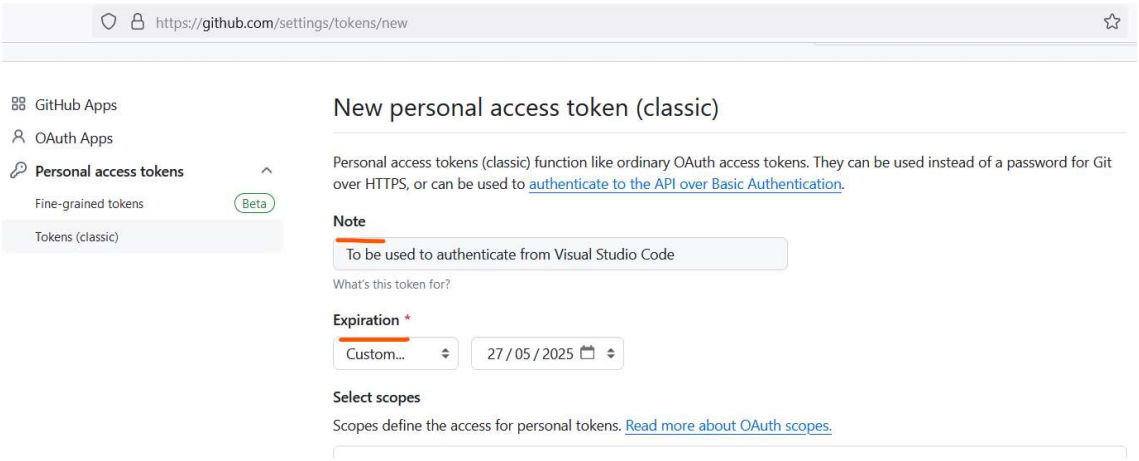
Click on Generate new token → Generate new token (classic)



Login if asked –

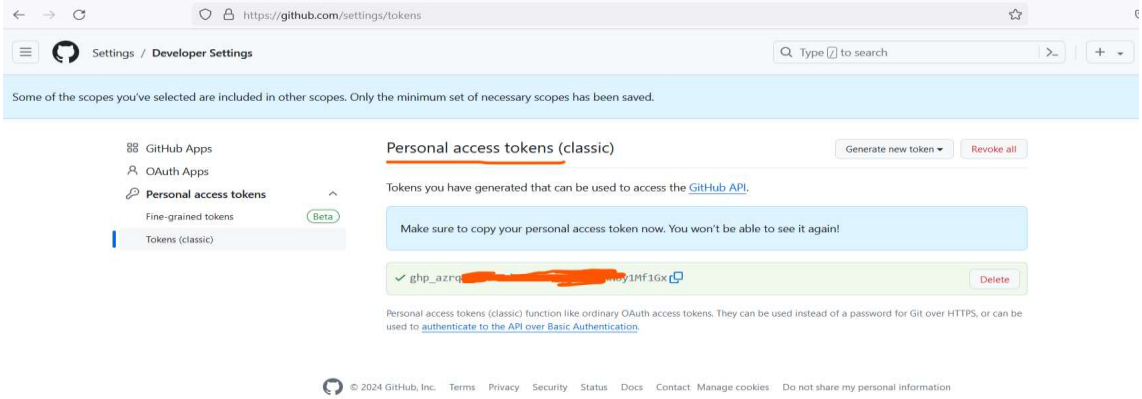


Add Note and Expiration for your token



Select the required scopes (if not sure, select all scopes) and click on Generate Token (at bottom of page)

Your Personal access token is created successfully



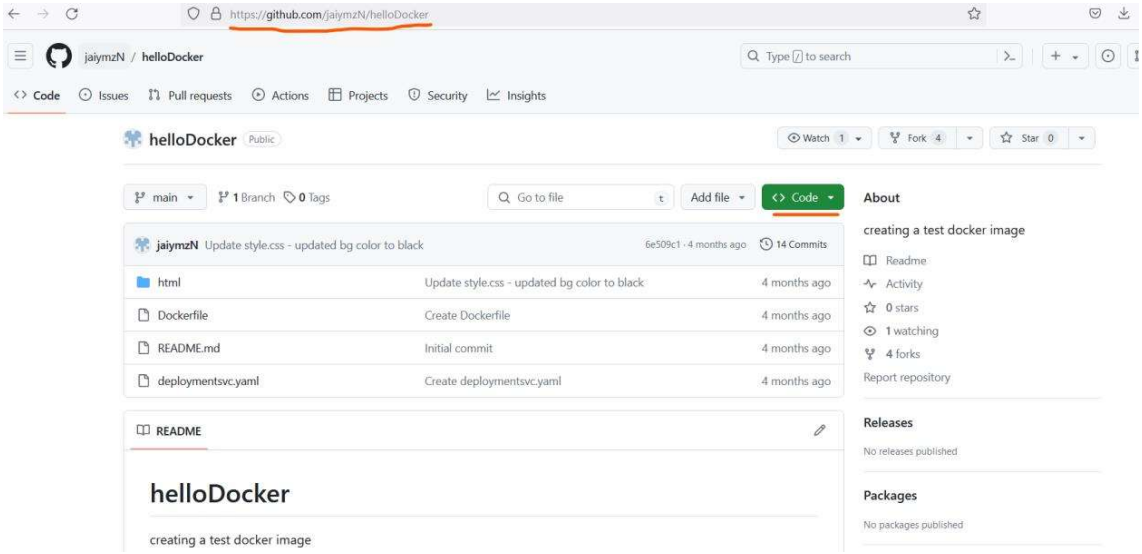
Note: Make sure to copy your personal access token now. You won't be able to see it again!

Clone a Git repository

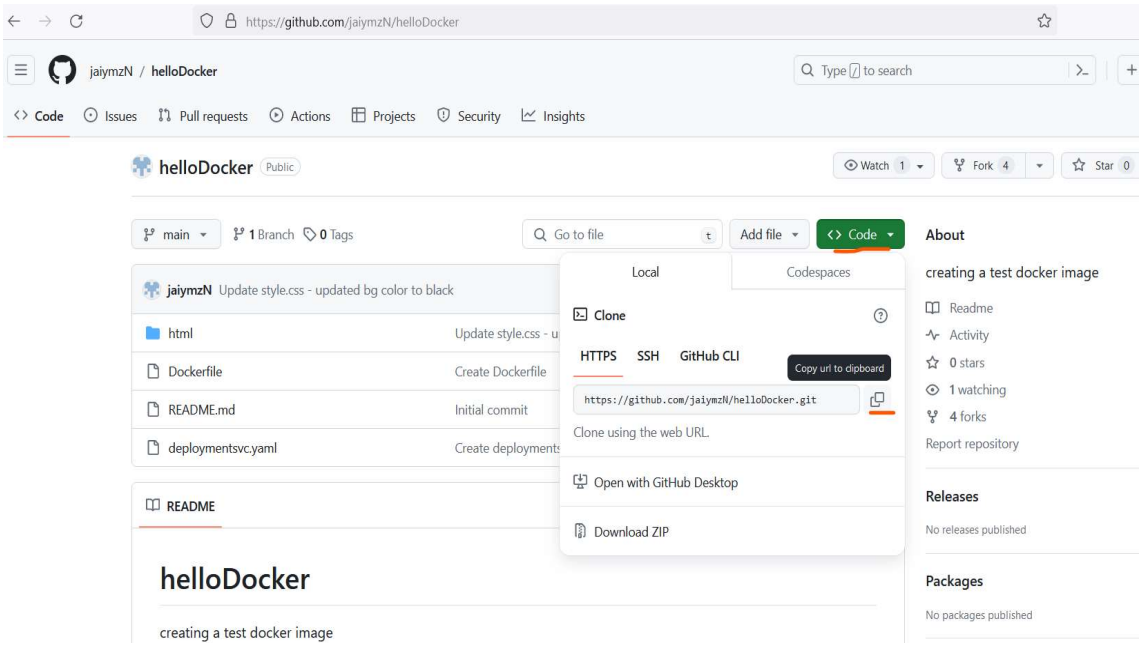
Go to any Git repo (in case of public repo, you would not need any credential of source repo, otherwise it is needed. So ask for credential / token for source repo to the owner).

For example, to clone the below public repo (we would not need any credential for this), open this URL in browser:

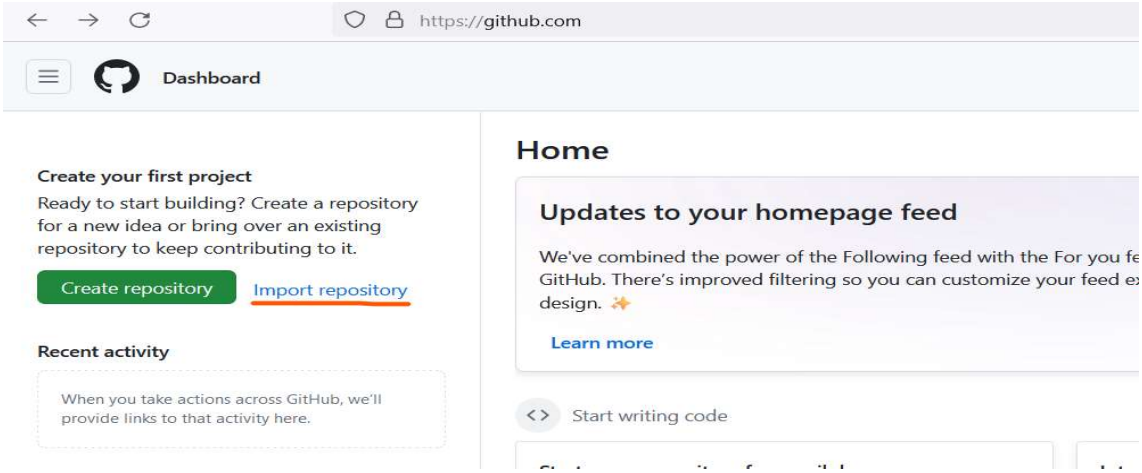
https://github.com/jaiymzN/helloDocker



Click on “Code” followed by copy sign → 

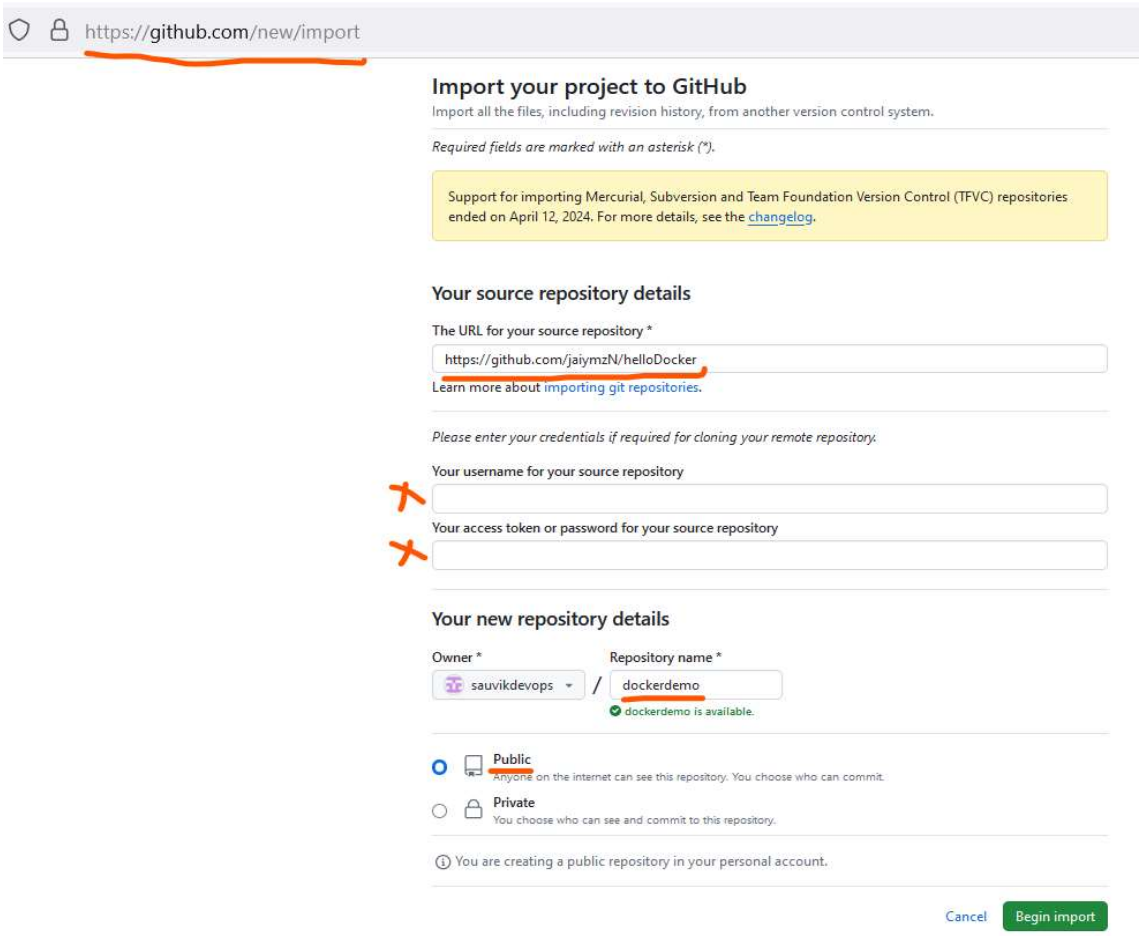


Now, login to your GitHub account (https://github.com/), and click on **“Import repository”**

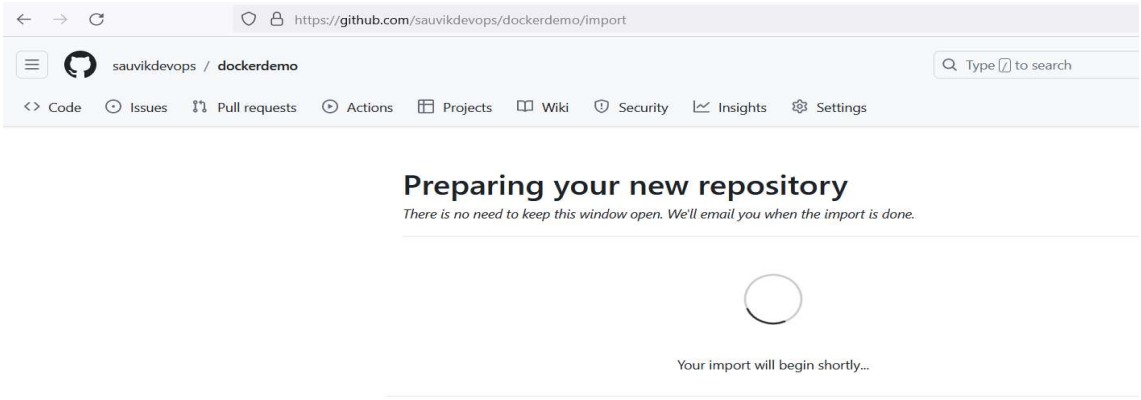


Now, do the following –

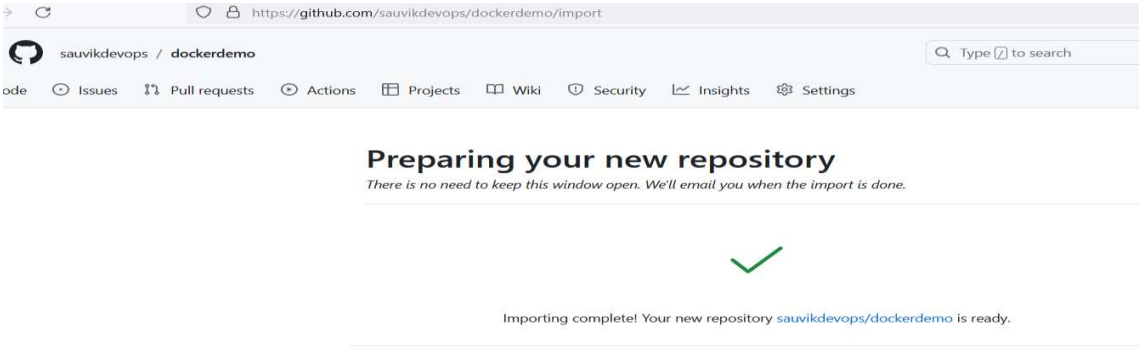
1. put the copied URL of your source repository. In this case, it is https://github.com/jaiymzN/helloDocker
2. Remove username and password of source repository since this is a public repo
3. Give a new "repo name" inside your GitHub account
4. Select the repo type as "Public"



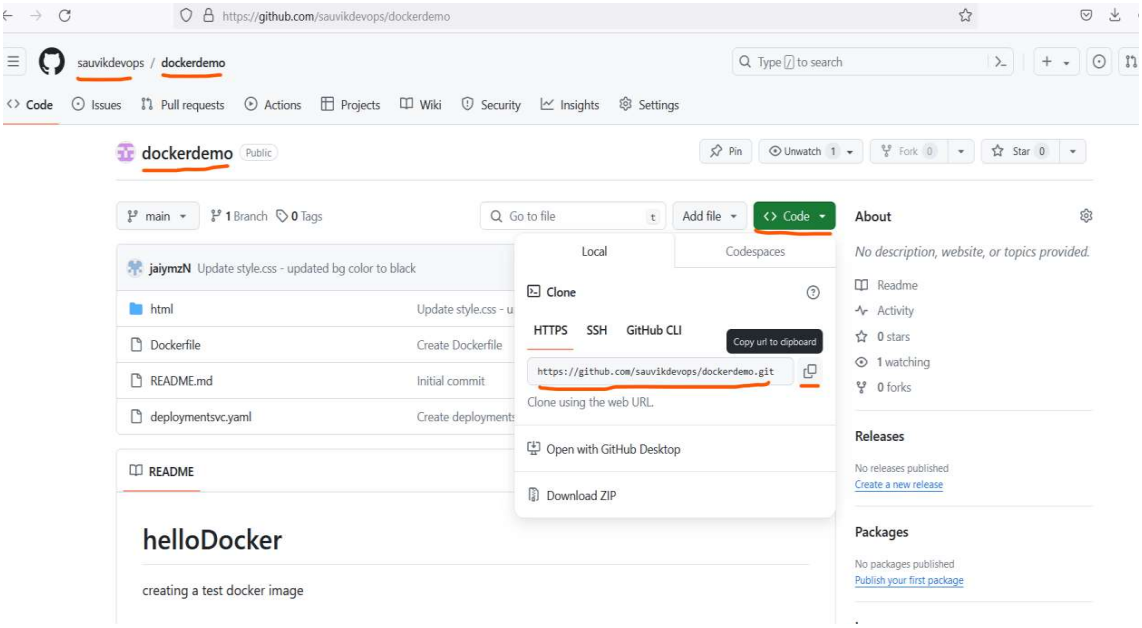
5. Click on "Begin Import"



Once import is complete, you will get confirmation as below:



Click on the newly created repo link (for example sauvikdevops/dockerdemo here) to go to there.



Note the repo URL which we would need later for CICD pipeline.

Install Minikube (for Kubernetes)

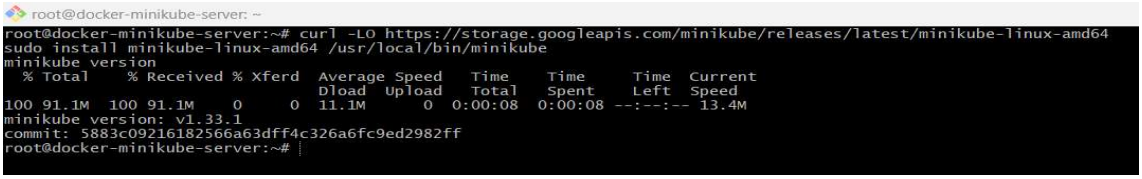
Minikube is a tool that allows you to run Kubernetes clusters locally for development and testing purposes.

Here's how you can install Minikube on Ubuntu:

Run the below commands:

curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

sudo install minikube-linux-amd64 /usr/local/bin/minikube



Run the below command to check if minikube is properly installed or not.

minikube version

Install kubectl

Update the apt package index and install packages needed to use the Kubernetes apt repository

sudo apt update

sudo apt-get install -y apt-transport-https ca-certificates curl

Download the public signing key for Kubernetes

curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key | sudo gpg — dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

Add the appropriate Kubernetes apt repository

echo ‘deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.29/deb/ /’ | sudo tee /etc/apt/sources.list.d/kubernetes.list

Update apt package index, then install kubectl

sudo apt update

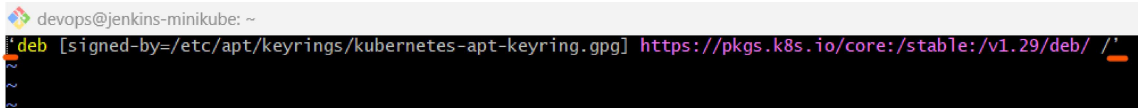
You might get this error:

```
devops@jenkins-minikube:~$ sudo apt update
E: Type ''deb' is not known on line 1 in source list /etc/apt/sources.list.d/kubernetes.list
E: The list of sources could not be read.
```

To fix the above error, please do this:

sudo vim /etc/apt/sources.list.d/kubernetes.list

Remove “ ‘ “ from beginning and end of the line.



Update apt package index,

sudo apt update

then install kubectl

sudo snap install kubectl –classic

```
root@docker-minikube-server:~# sudo snap install kubectl --classic
2024-05-26T15:54:10Z INFO Waiting for automatic snapd restart...
kubectl 1.29.5 from Canonical✓ installed
root@docker-minikube-server:~#
```

kubectl version --client

```
root@docker-minikube-server:~# kubectl version --client
Client Version: v1.29.5
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
root@docker-minikube-server:~#
```

Change permission for docker

sudo chmod 666 /var/run/docker.sock

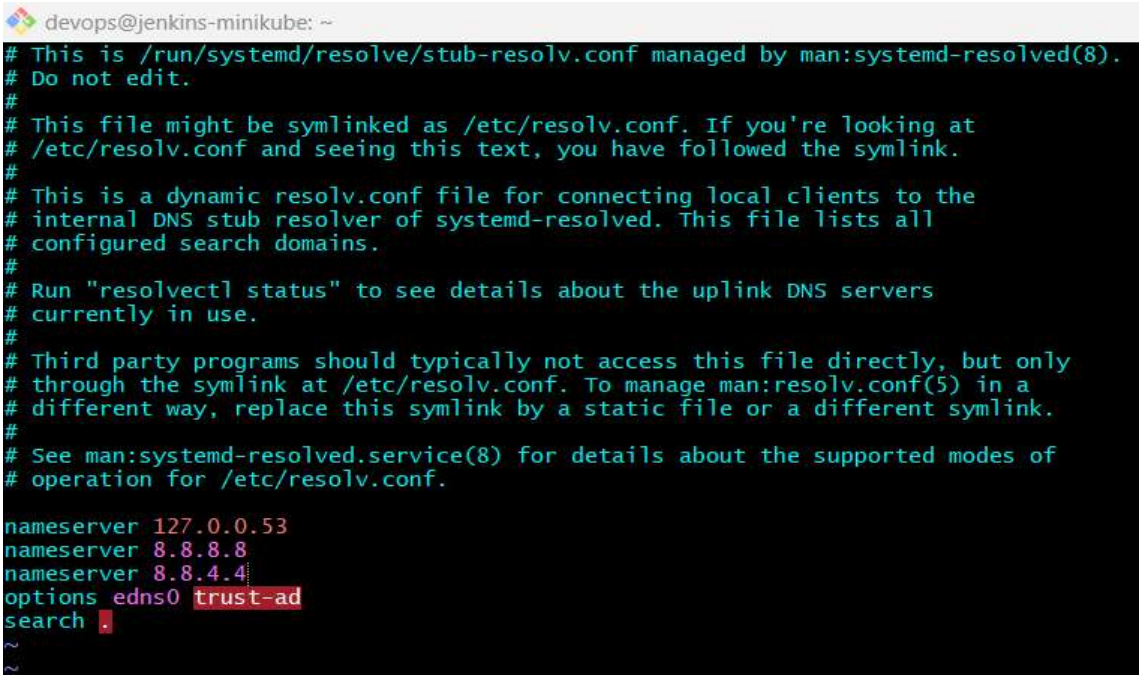
Add nameserver in Ubuntu server

To fix many issues to connectivity with external sites like GitHub or DockerHub account from Ubuntu server, edit /etc/resolv.conf & add these two lines

```
devops@jenkins-minikube:~$ sudo vi /etc/resolv.conf
devops@jenkins-minikube:~$
```

nameserver 8.8.8.8

nameserver 8.8.4.4



Run the below command to restart docker:

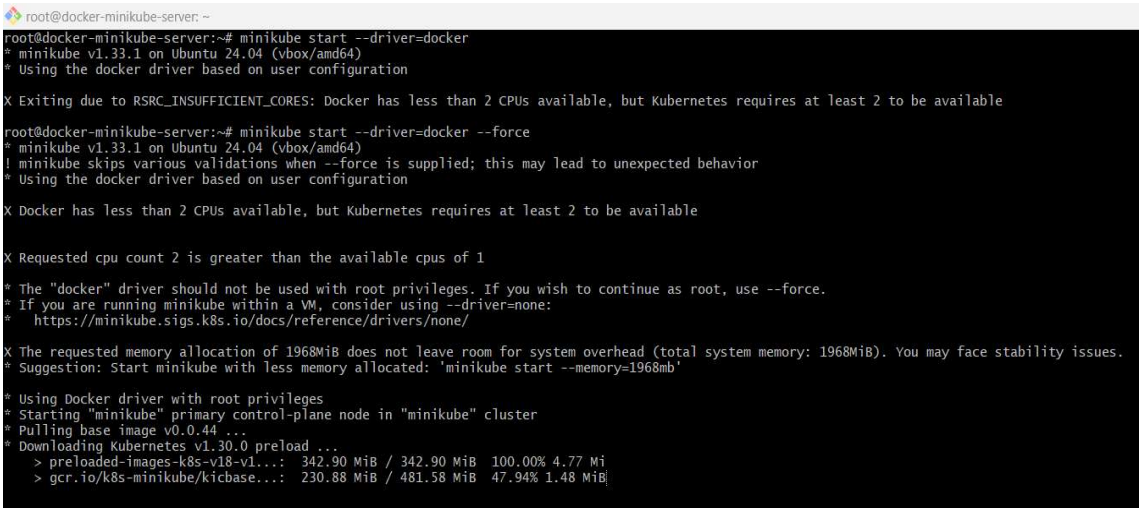
sudo systemctl daemon-reload

sudo systemctl restart docker

Start Minikube (with Docker Driver)

Command to start minikube

minikube start --driver=docker --force



[**Optional way** to start minikube using Virtual box driver. Please ignore if docker driver is working.]

Install Virtualbox on Ubuntu

sudo apt-get install virtualbox

Now that we have Minikube installed, let's start a Minikube cluster using the virtualbox driver from a normal user prompt (not root prompt):

minikube start --driver=virtualbox –force

Run these minikube and kubectl commands to play around your minikube / Kubernetes cluster

Check minikube status

```
root@docker-minikube-server: ~
root@docker-minikube-server:~# minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured

root@docker-minikube-server:~#
```

kubectl cluster-info

```
root@docker-minikube-server:~# kubectl cluster-info
Kubernetes control plane is running at https://192.168.49.2:8443
CoreDNS is running at https://192.168.49.2:8443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
root@docker-minikube-server:~#
```

kubectl config view

```
root@docker-minikube-server:~# kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority: /root/.minikube/ca.crt
    extensions:
    - extension:
        last-update: Sun, 26 May 2024 16:49:20 UTC
        provider: minikube.sigs.k8s.io
        version: v1.33.1
      name: cluster_info
    server: https://192.168.49.2:8443
  name: minikube
contexts:
- context:
    cluster: minikube
    extensions:
    - extension:
        last-update: Sun, 26 May 2024 16:49:20 UTC
        provider: minikube.sigs.k8s.io
        version: v1.33.1
      name: context_info
    namespace: default
    user: minikube
  name: minikube
current-context: minikube
kind: Config
preferences: {}
users:
- name: minikube
  user:
    client-certificate: /root/.minikube/profiles/minikube/client.crt
    client-key: /root/.minikube/profiles/minikube/client.key
root@docker-minikube-server:~#
```

kubectl get nodes

```
root@docker-minikube-server:~# kubectl get nodes
NAME       STATUS   ROLES           AGE   VERSION
minikube   Ready    control-plane   16m   v1.30.0
root@docker-minikube-server:~#
```

kubectl get pods --all-namespaces

```
root@docker-minikube-server: ~
root@docker-minikube-server:~# kubectl get pods --all-namespaces
NAMESPACE     NAME                                   READY   STATUS    RESTARTS      AGE
kube-system   coredns-7db6d8ff4d-ntgzn               1/1     Running   0             16m
kube-system   etcd-minikube                          1/1     Running   0             17m
kube-system   kube-apiserver-minikube                1/1     Running   0             17m
kube-system   kube-controller-manager-minikube       1/1     Running   1 (17m ago)   17m
kube-system   kube-proxy-6nx7p                       1/1     Running   0             16m
kube-system   kube-scheduler-minikube                1/1     Running   0             17m
kube-system   storage-provisioner                    1/1     Running   1 (15m ago)   16m
root@docker-minikube-server:~#
```

minikube addons enable metrics-server

```
root@docker-minikube-server:~# minikube addons enable metrics-server
* metrics-server is an addon maintained by Kubernetes. For any concerns contact minikube on GitHub.
You can view the list of minikube maintainers at: https://github.com/kubernetes/minikube/blob/master/OWNERS
  - Using image registry.k8s.io/metrics-server/metrics-server:v0.7.1
* The 'metrics-server' addon is enabled
root@docker-minikube-server:~#
```

minikube dashboard

```
root@docker-minikube-server:~# minikube dashboard
* Verifying dashboard health ...
* Launching proxy ...
* Verifying proxy health ...
panic: send on closed channel

goroutine 61 [running]:
k8s.io/minikube/cmd/minikube/cmd.readByteWithTimeout.func2()
        /app/cmd/minikube/cmd/dashboard.go:192 +0x76
created by k8s.io/minikube/cmd/minikube/cmd.readByteWithTimeout in goroutine 1
        /app/cmd/minikube/cmd/dashboard.go:187 +0x145
root@docker-minikube-server:~#
```

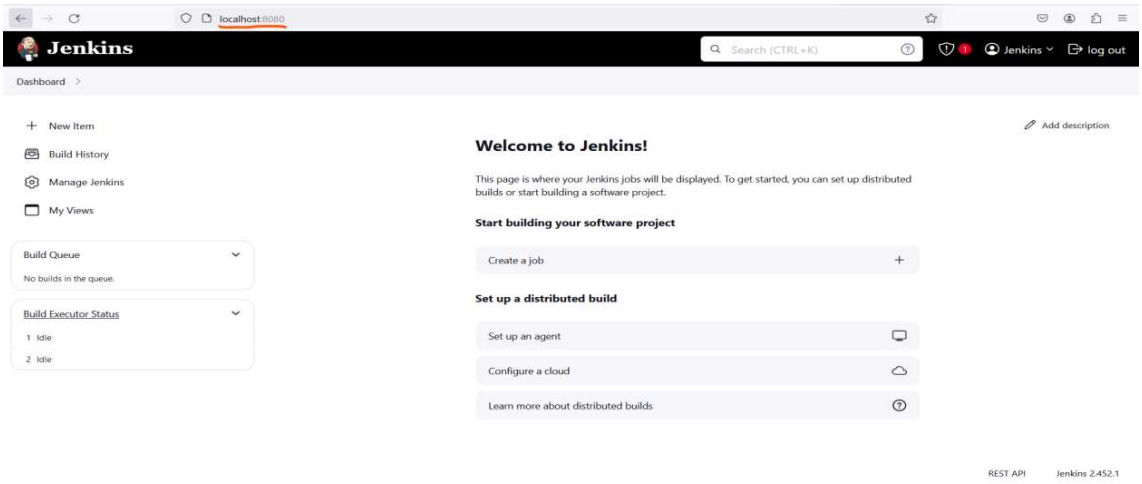
Setting up Jenkins CI/CD pipeline

You need to create your GitHub repository or clone my repository so you can use it for this project. For your repository, you will need an application or a website file. For this project, I used a simple webpage HTML, and CSS files for my application. You will also need a Dockerfile to create the image and a Kubernetes definition file that defines the deployment and service to expose the deployment

First, we need to configure authentication credentials that will allow this process to be possible. The credentials we will configure are as follows:

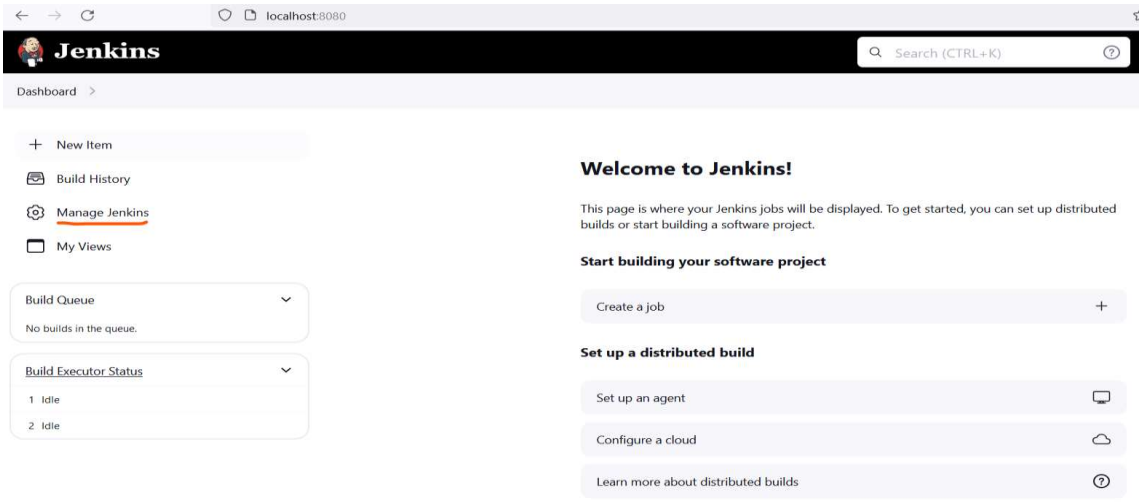
Open Jenkins URL

http://localhost:8080/

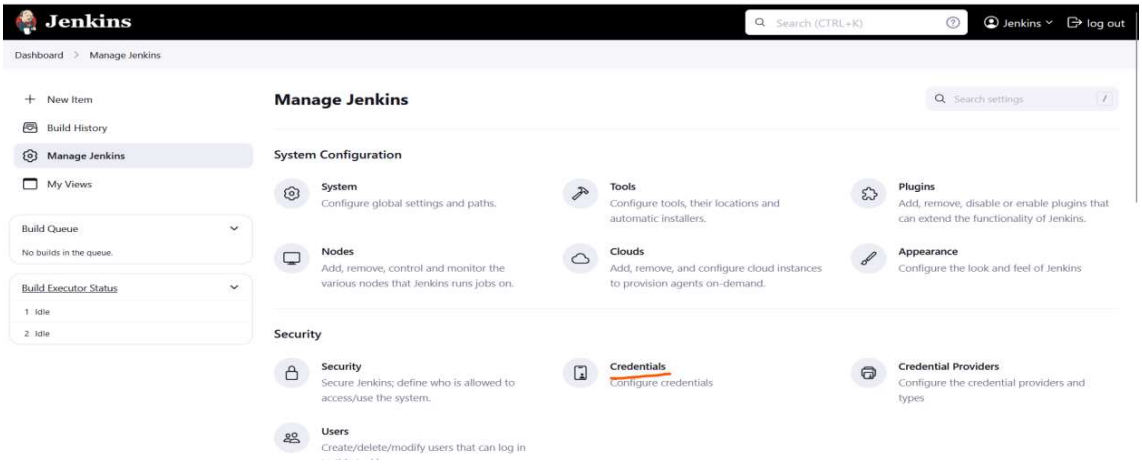


Configuring Docker Hub authentication

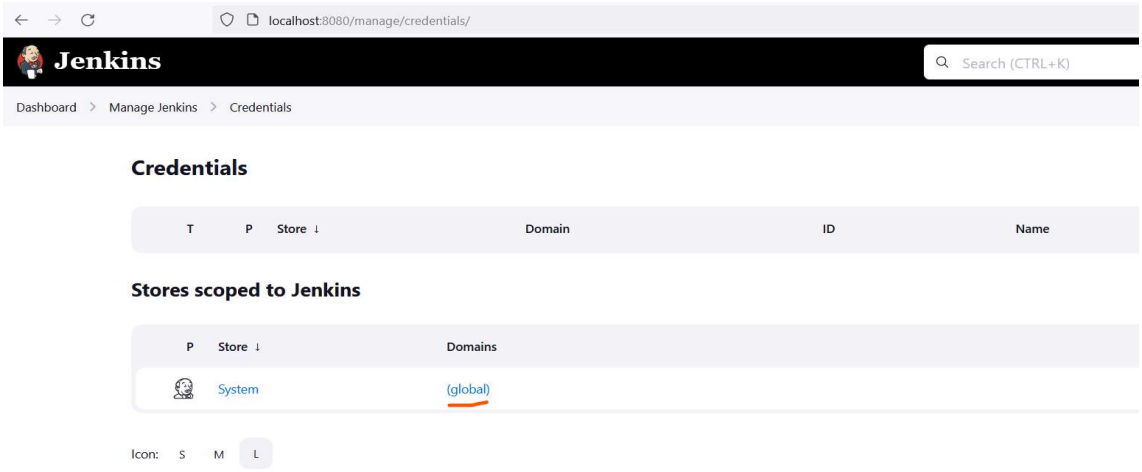
Go to manage Jenkins (at the left hand side) ->



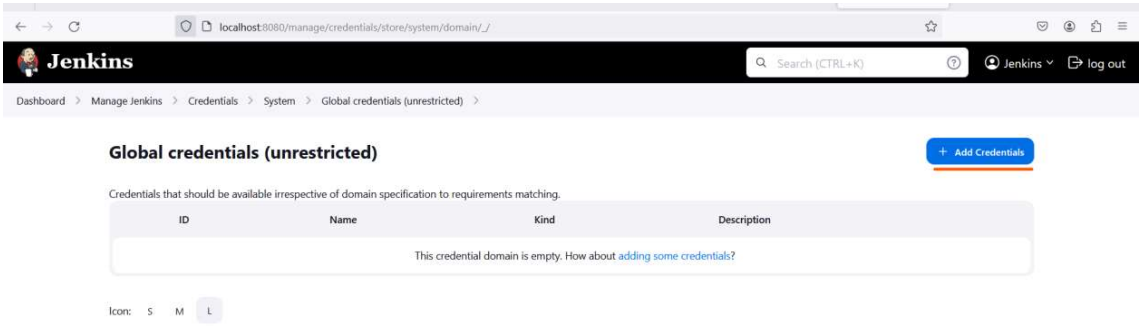
Click on Credentials ->



Click on “(global) ->



Click on Add Credentials

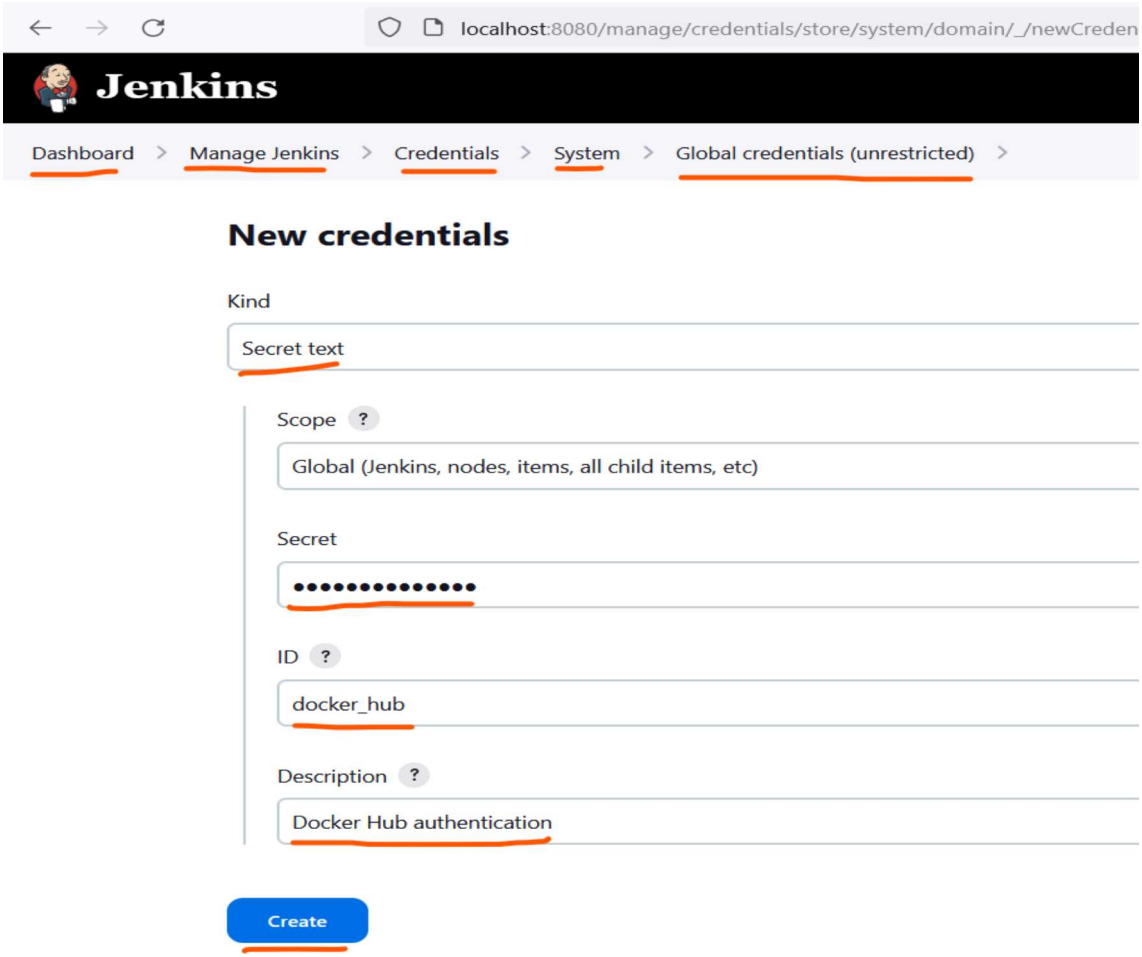


For Kind, select “Secret text”

Under “Secret”, type in the password for the docker hub

Give the secret an ID for reference

Click on Create

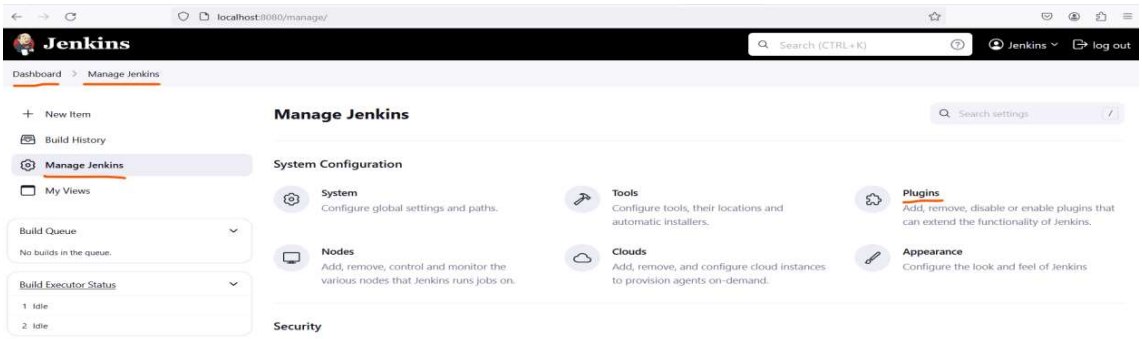


Configure Kubernetes authentication for Jenkins

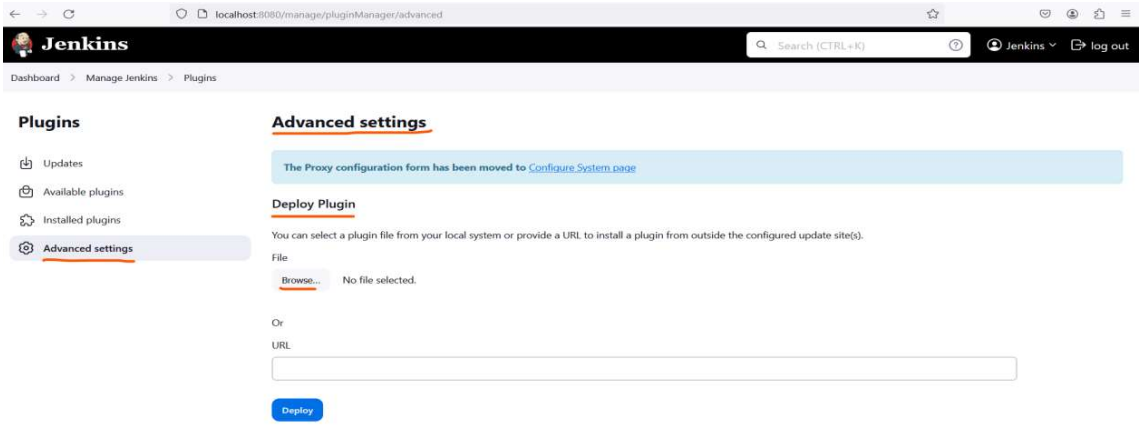
Download “Kubernetes-cd.hpi” plugin for Kubernetes from Internet to your local machine.,

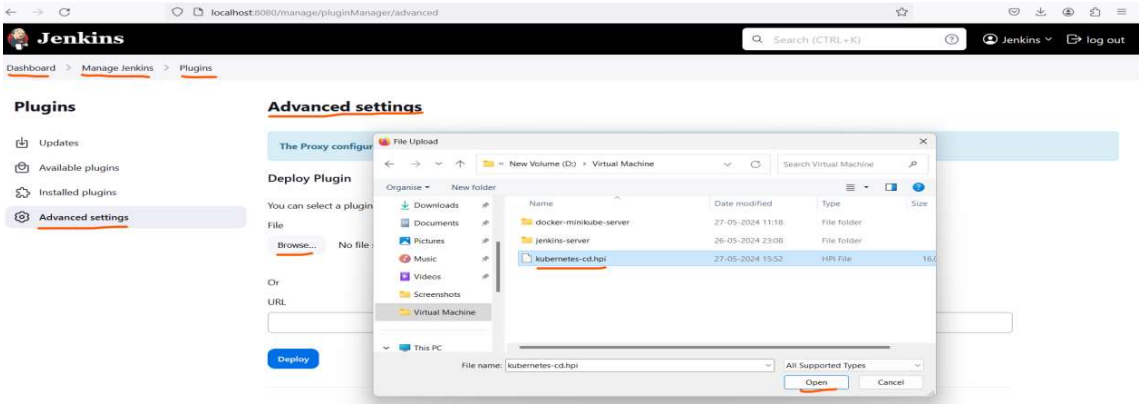
https://updates.jenkins.io/download/plugins/kubernetes-cd/1.0.0/kubernetes-cd.hpi

Go to “manage Jenkins” -> plugins

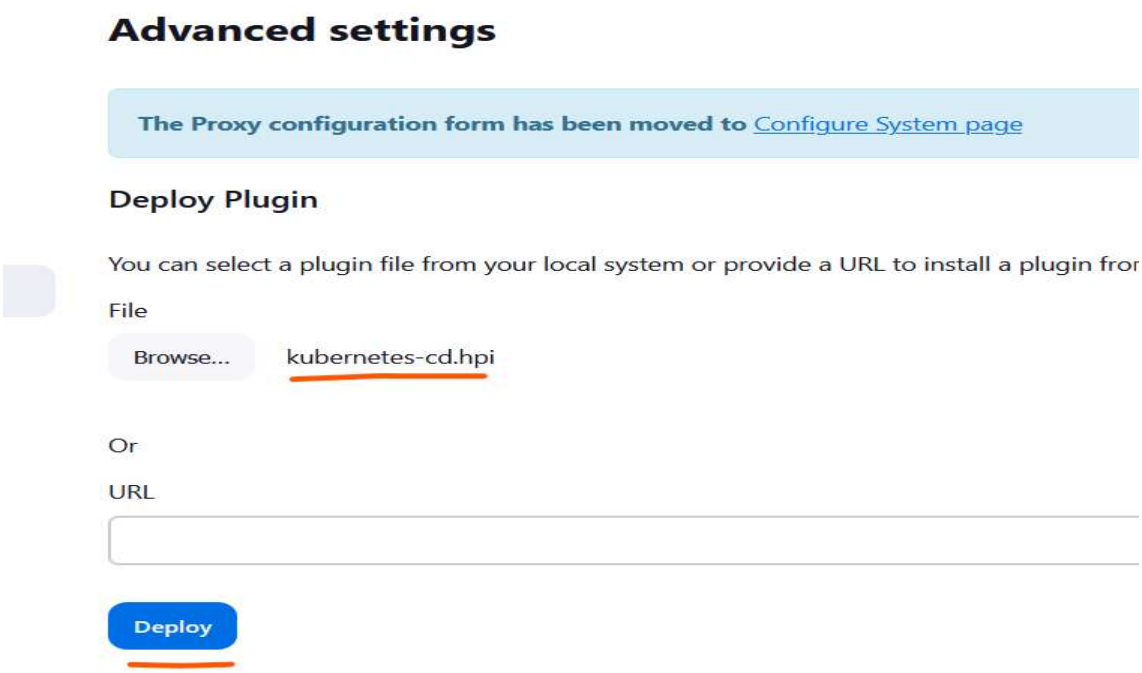


Advanced settings → go to “Deploy Plugin” section and select “Browse”

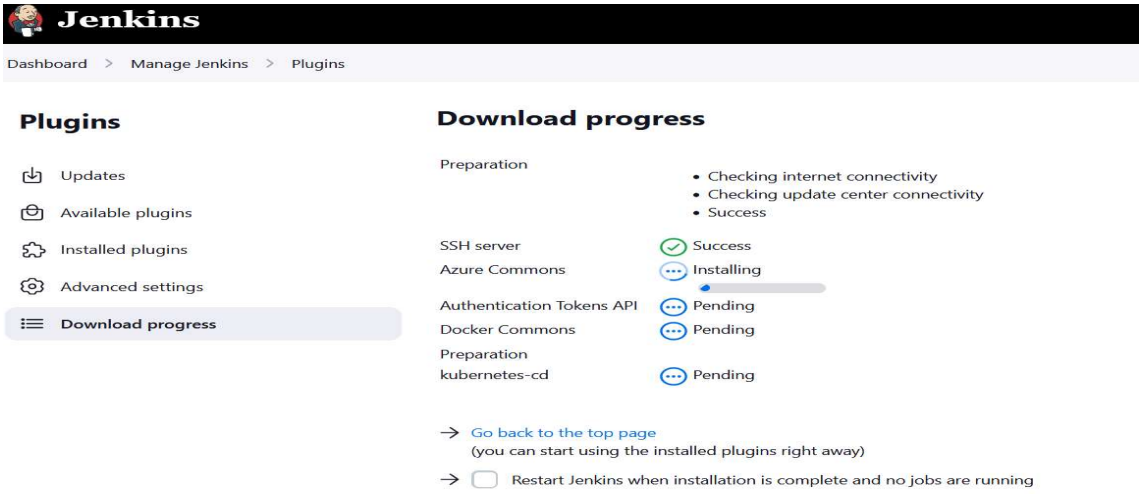




Upload the “Kubernetes-cd.hpi” plugin for Kubernetes



Select “Deploy”



Restart the Jenkins service (from Ubuntu server) —

sudo systemctl restart jenkins

If you get this warning,

Warning: The unit file, source configuration file or drop-ins of jenkins.service changed on disk. Run 'systemctl daemon-reload' to reload units.

Run this command:

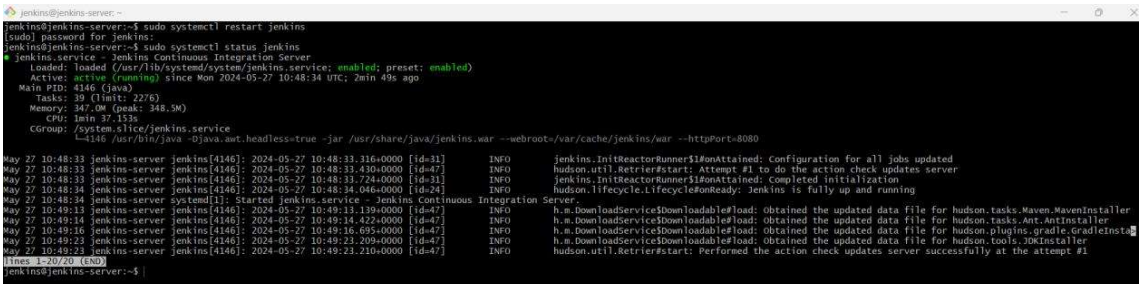
systemctl daemon-reload

Then restart Jenkins service.

sudo systemctl restart jenkins

Then you can check Jenkins status:

sudo systemctl status jenkins

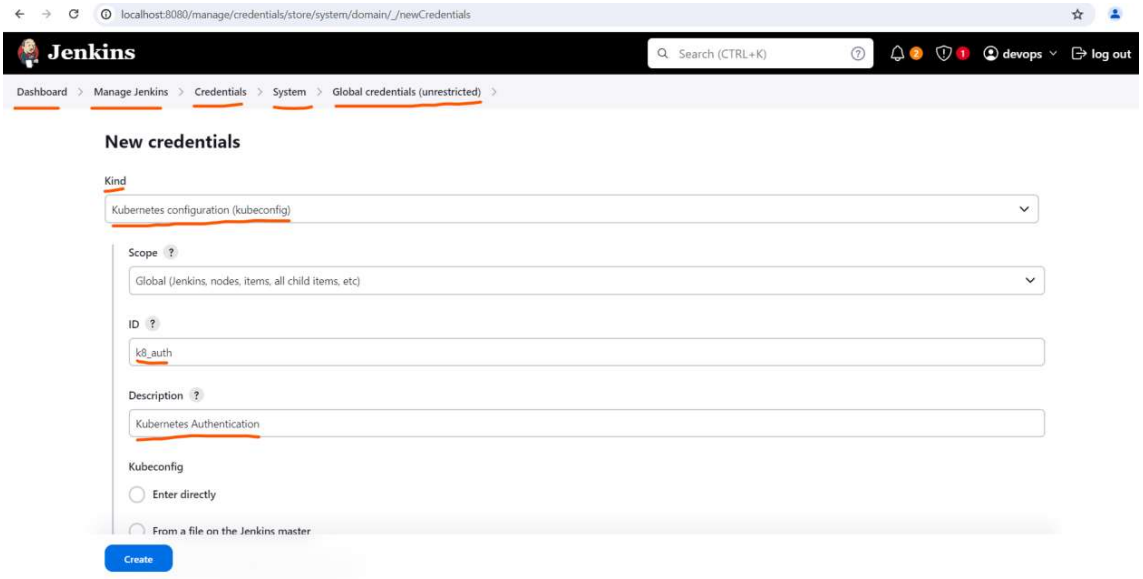


Then run the below command and copy the output

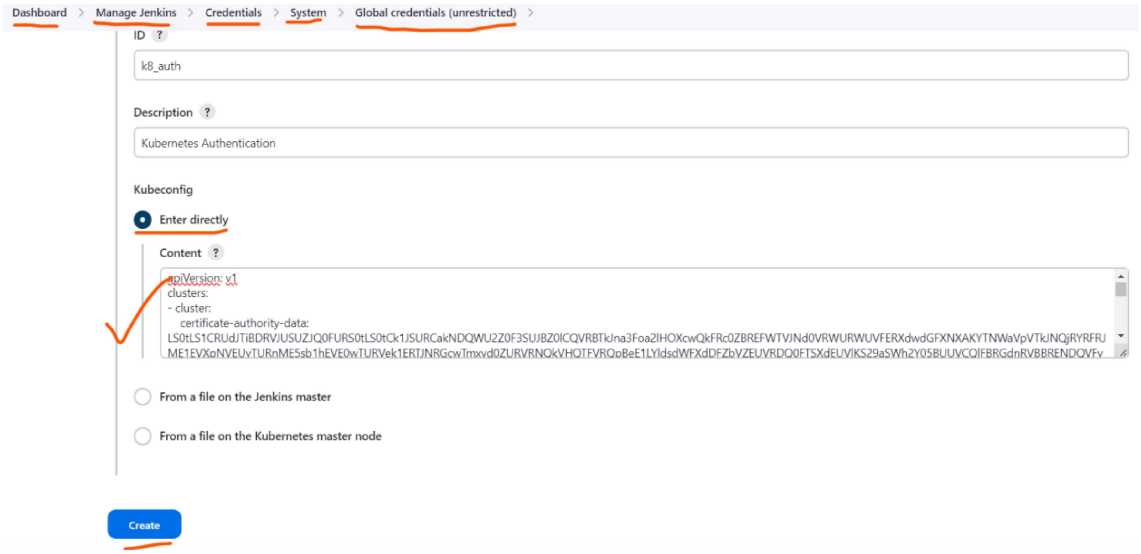
kubectl config view –flatten

Now, go back to Jenkins URL (http://localhost:8080/):

1. Go to “Manage Jenkins” -> Credentials -> (global) -> Add Credentials
2. For “Kind”, select “Kubernetes configuration (kubeconfig)

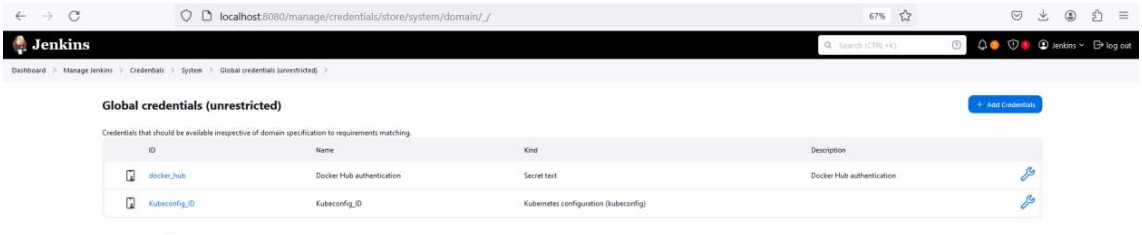


3. Give the Kubeconfig an ID
4. Select “Enter Directly”
5. Put the output of command – “kubectl config view –flatten” which you took earlier.



Click on - Create

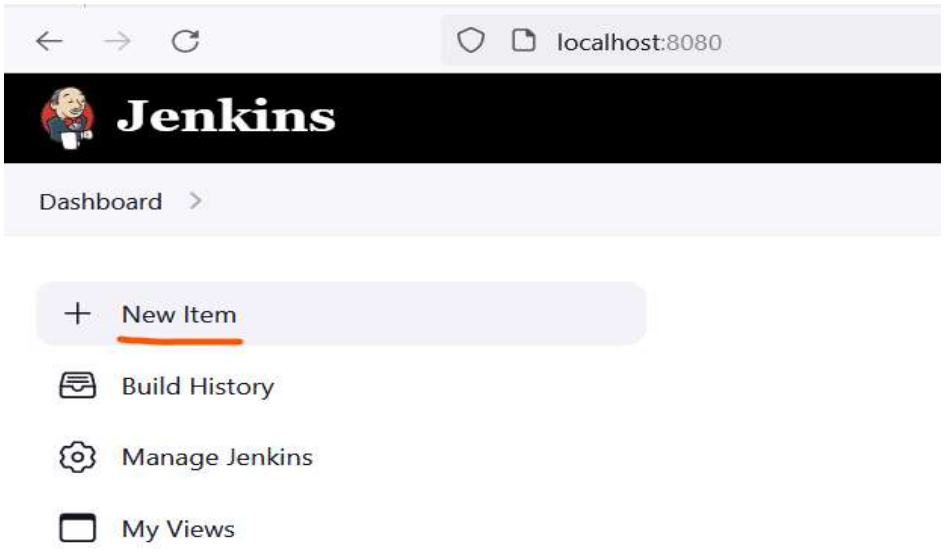
Once this is done, you will get screen like below:



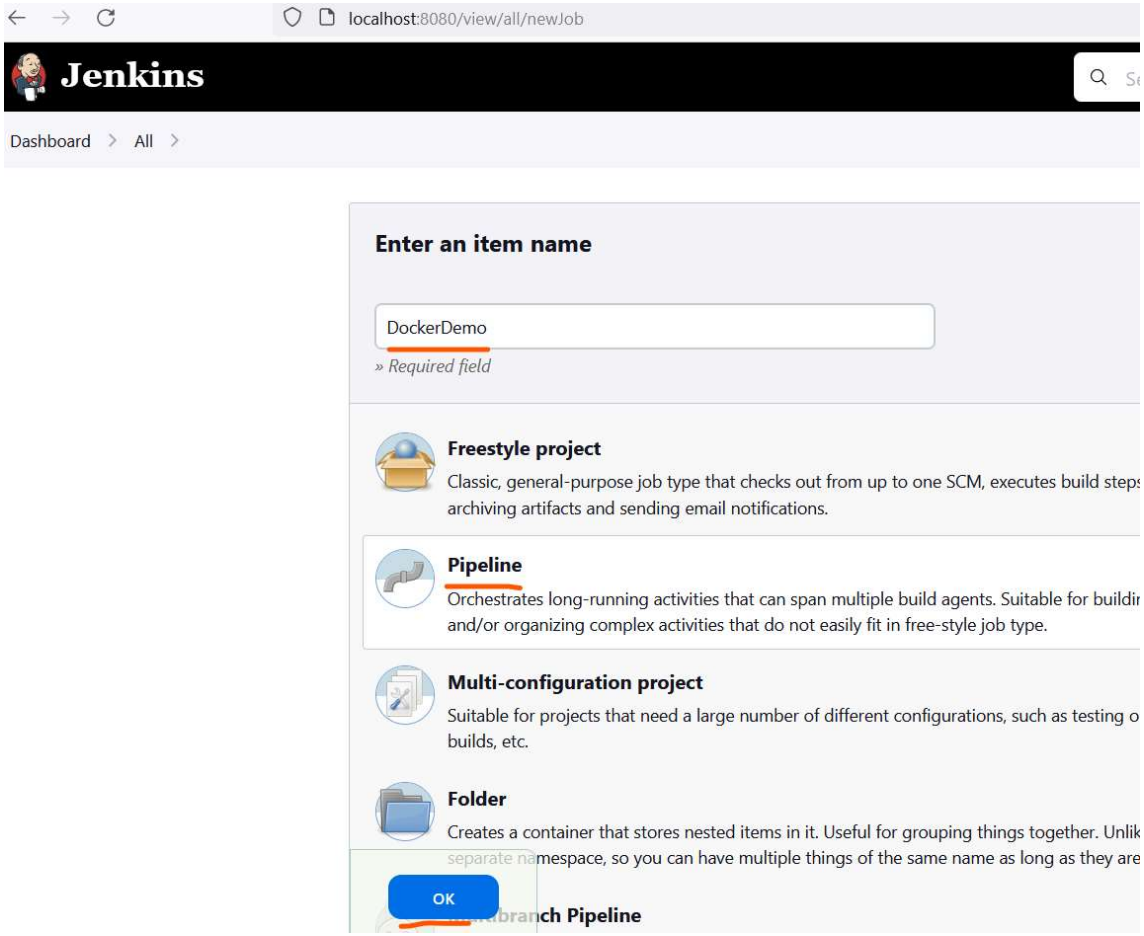
Create CICD pipeline in Jenkins

Go to your Jenkins URL / console and configure the following

Create a new item

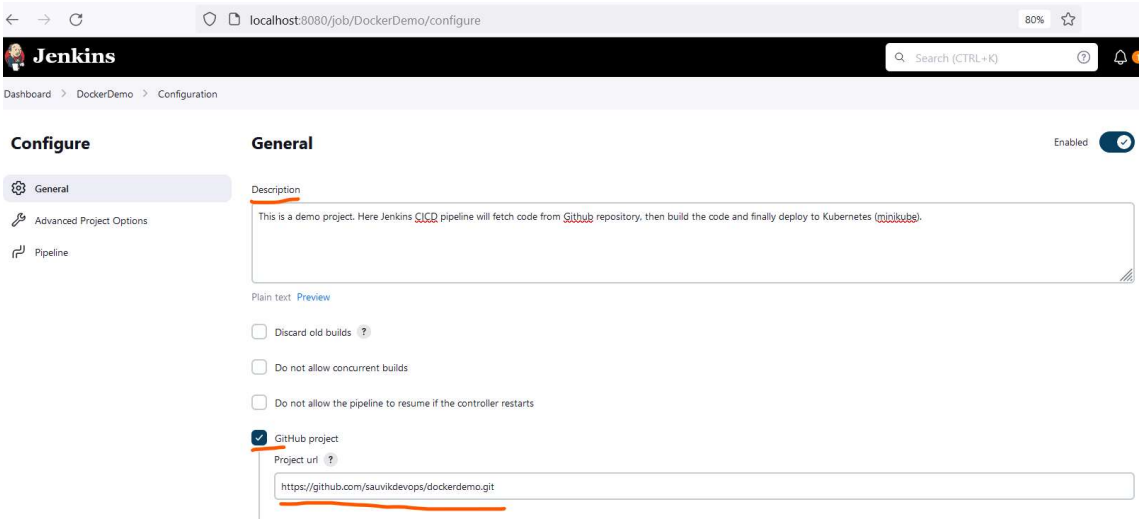


Select the Pipeline Option

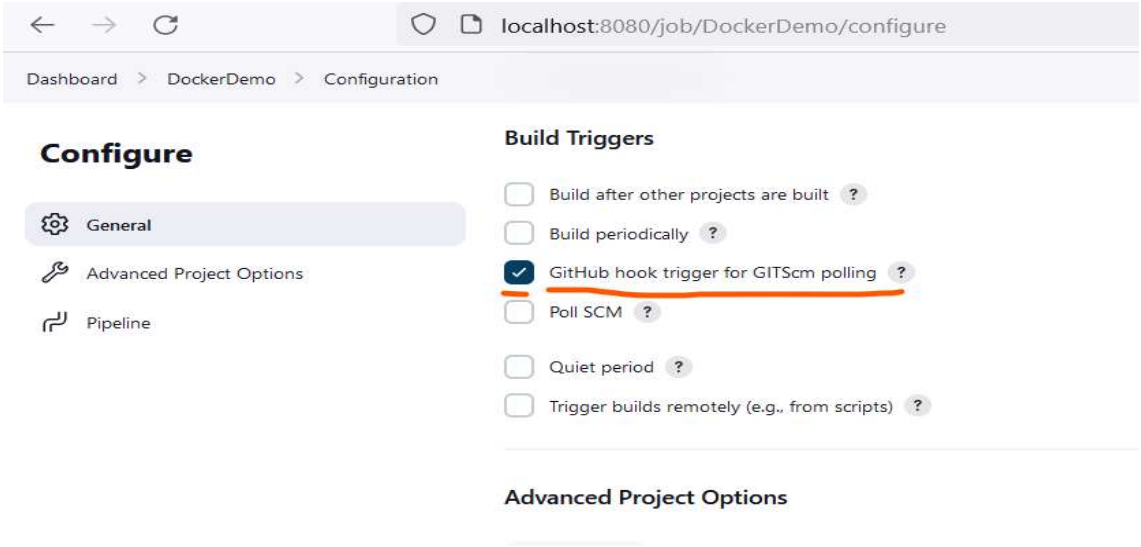


Enter a Description (optional)

Select the “GitHub project” option and paste the GitHub browser URL



Go to the “Build triggers” section and choose the option - GitHub hook trigger for GITscm polling



Go to the pipeline section and fill the following to add CICD script from GitHub repository -

Definition = Pipeline script from SCM

SCM = Git

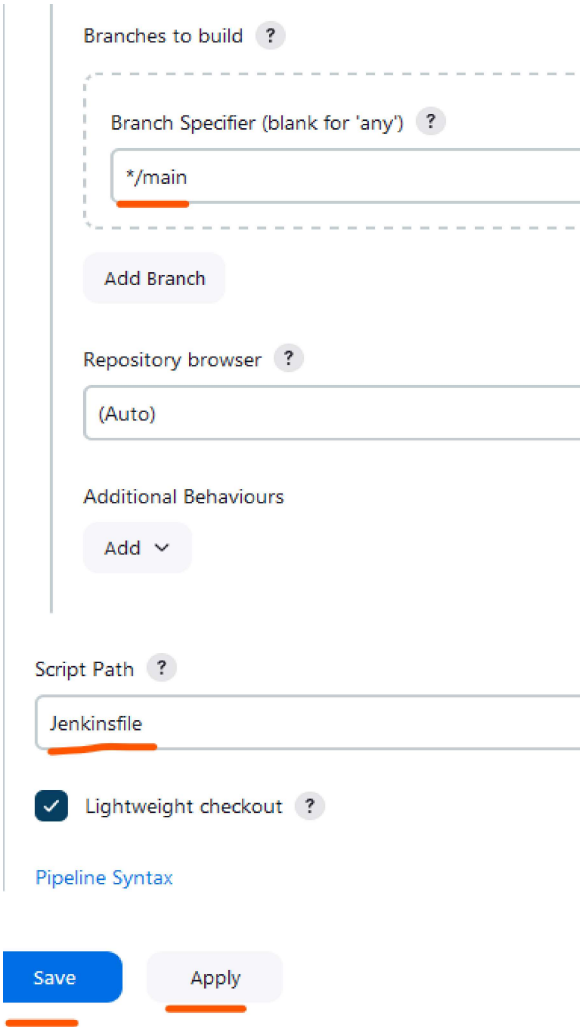
Repository URL = your git repo URL. For example, here it is –

https://github.com/sauvikdevops/dockerdemo.git

Credentials = - none - (as we are using a public repo. Otherwise, select 'Jenkins' as Jenkins Credentials Provider followed by username and password)

Branch Specifier (blank for 'any') = */main

Script Path = Jenkinsfile (remember to create a file called Jenkinsfile in main branch of your Git repository)

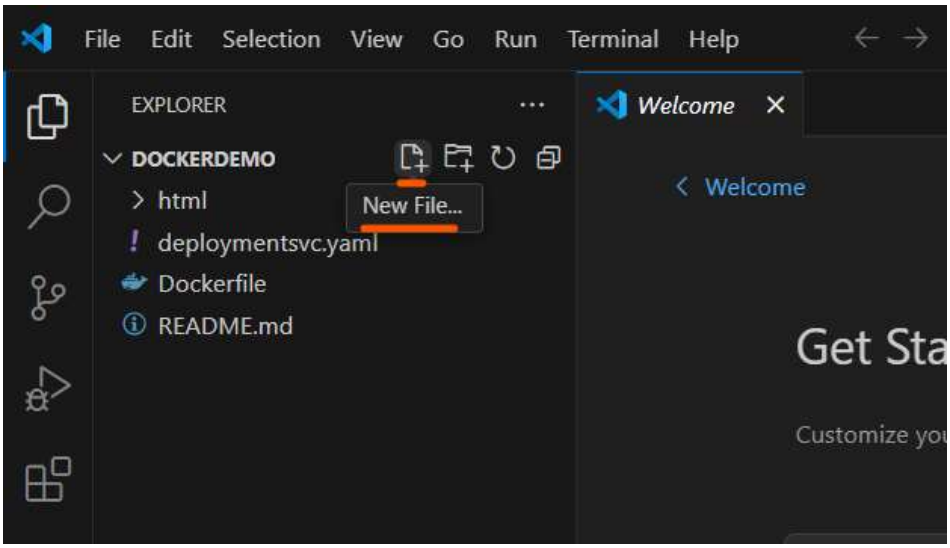


Click Apply then Save.

Note: You already know how to install & configure Visual Studio Code on local machine. If not, please refer the Table of Content for details.

Now while in Visual Studio Code from your Windows machine, create and update Git Repo with CICD pipeline script

From Visual Studio code, click on “New File” icon 



From the visual studio code window –

- at left hand side, put the new file name as – Jenkinsfile
- at right hand side put the below script

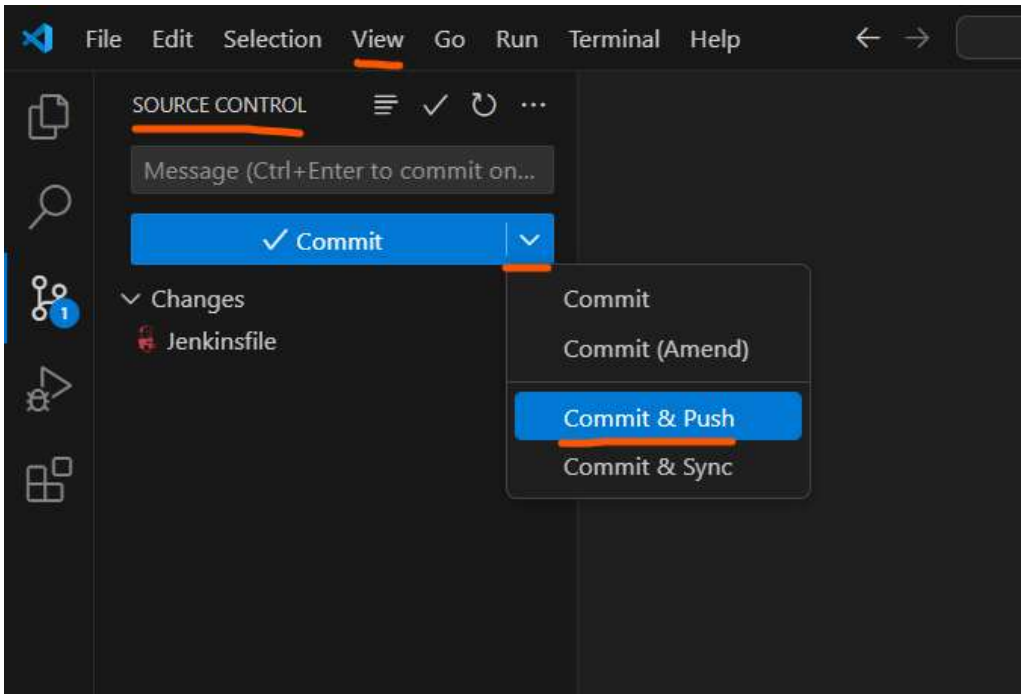
```
pipeline {
  agent any

  stages {
    stage('Checkout GitHub repo') {
      steps {
        checkout scmGit(branches: [[name: '*/main']], extensions: [], userRemoteConfigs: [[url: 'https://github.com/sauvikdevops/dockerdemo']])
      }
    }
    stage('Build and Tag Docker Image') {
      steps {
        script {
          sh 'docker build . -t hellodocker'
          sh 'docker tag hellodocker sauvikdevops/learning'
        }
      }
    }
    stage('Push the Docker Image to DockerHUb') {
```

```
-            steps {
-               script {
-                  withCredentials([string(credentialsId: 'docker_hub', variable: 'docker_hub')]) {
-                  sh 'docker login -u sauvik.devops@gmail.com -p ${docker_hub}'
-   }
-                  sh 'docker push sauvikdevops/learning'
-               }
-            }
-         }
-
-         stage('Deploy deployment and service file') {
-            steps {
-               script {
-                  kubernetesDeploy configs: 'deploymentsvc.yaml', kubeconfigId: 'k8_auth'
-               }
-            }
-         }
-      }
-   }
```

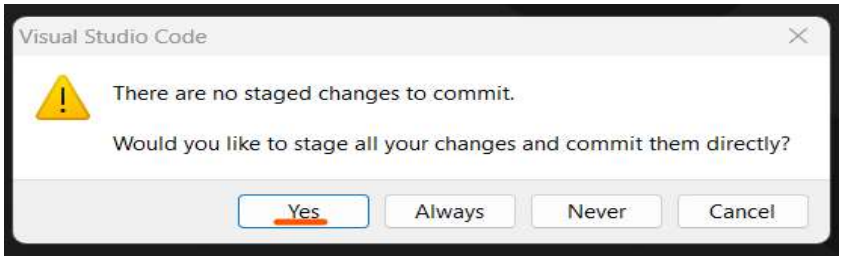
After that save the changes using Ctrl+ S or File menu → Save

Now, look at the bottom left corner of Visual Studio Code window go to View menu → Source Control

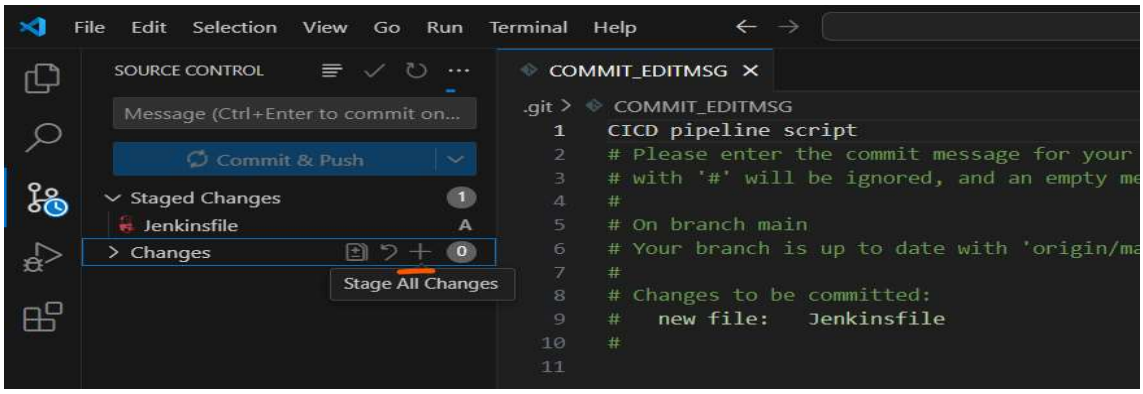
Now, click on the downward pointing arrow at right side of Commit and click on Commit & Push.



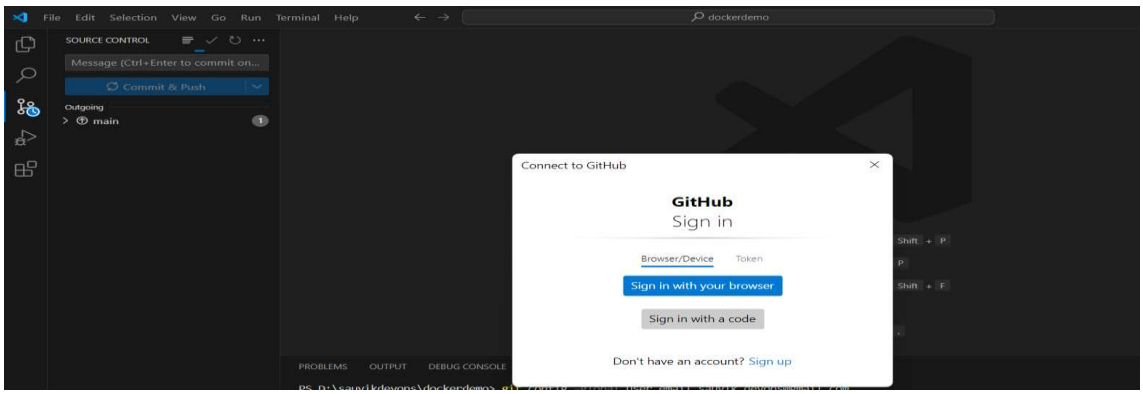
Click on Yes to continue



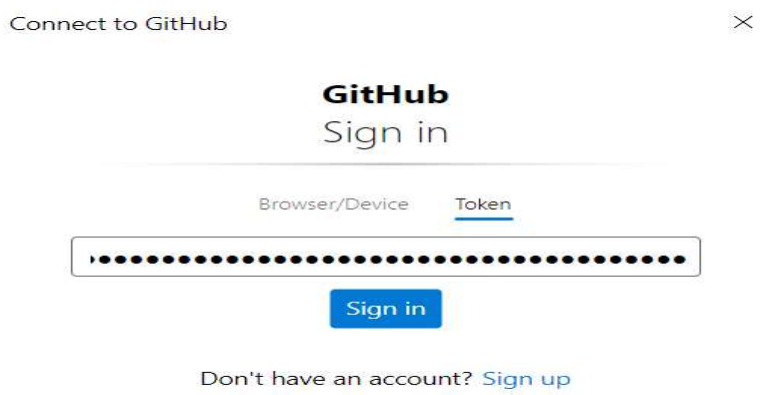
Now, you will see the changes and sync in progress.



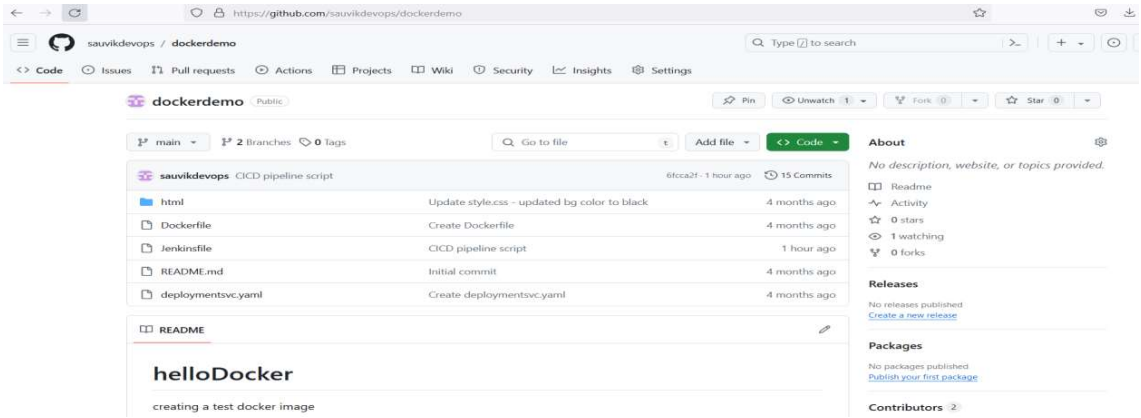
Please enter the commit message for your changes and close it to proceed.



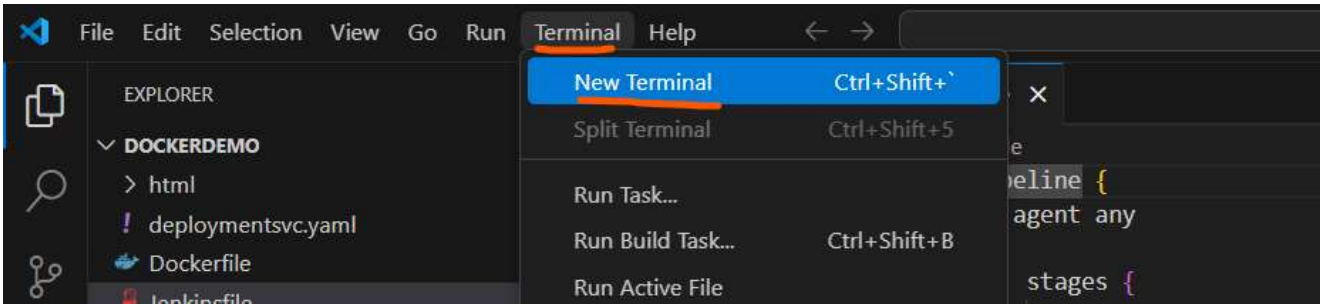
Click on “Token” and put the previously created token.



Now you can check GitHub URL to see that the changes reflected.



Note: alternatively, when you are dome with changes, instead of using Commit & Push from GUI, you can open a terminal from Visual Studio Code Terminal menu. Click on new terminal to proceed.



Now, from the VS Code terminal, run the below commands to Commit & Push to GitHub –

git status

git add .

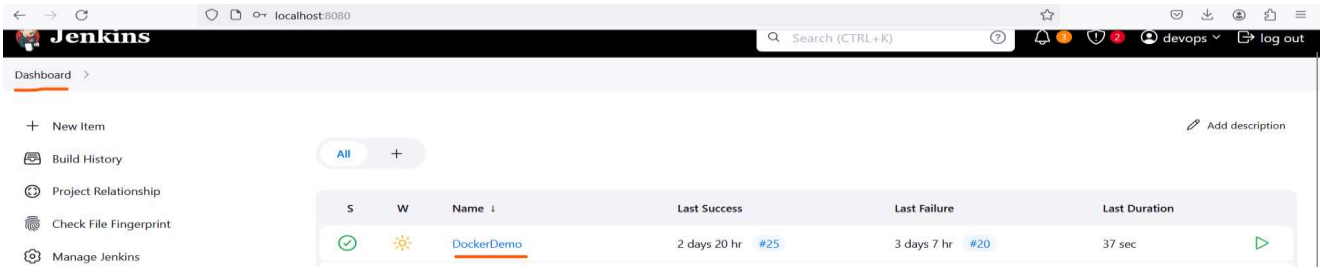
git commit -m “your commit message”

git push

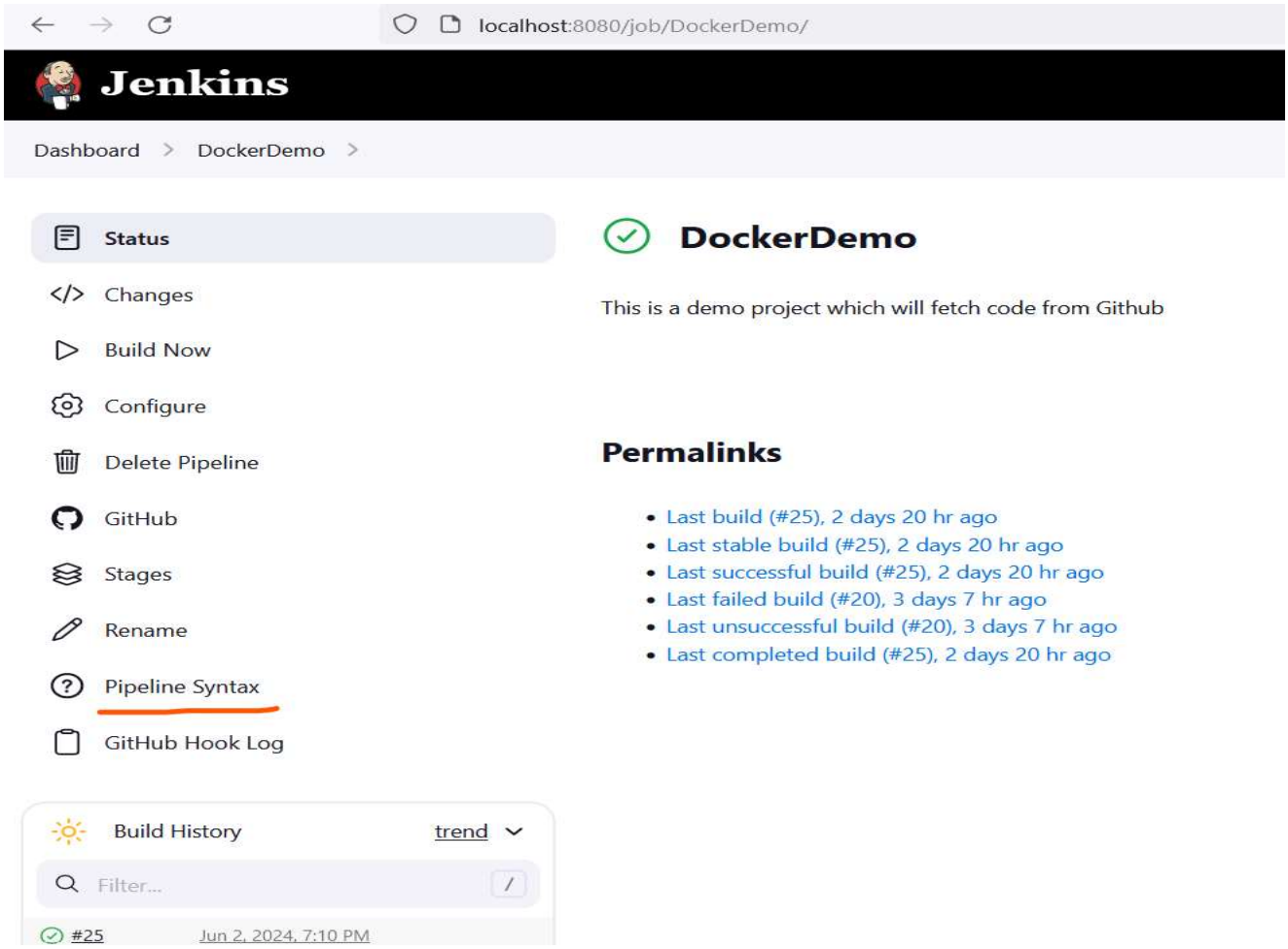
Pipeline syntax

Declarative Pipeline is a relatively recent addition to Jenkins Pipeline which presents a more simplified and opinionated syntax on top of the Pipeline sub-systems.

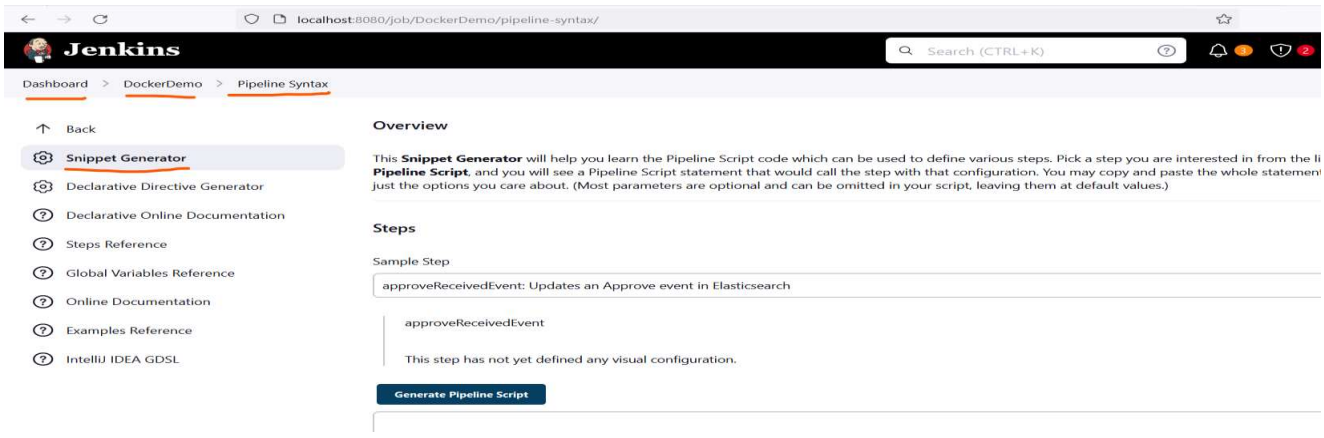
To access Jenkins Pipeline Syntax, go to any job from Jenkins dashboard.



Scroll down and look at the bottom left for – ‘Pipeline Syntax’

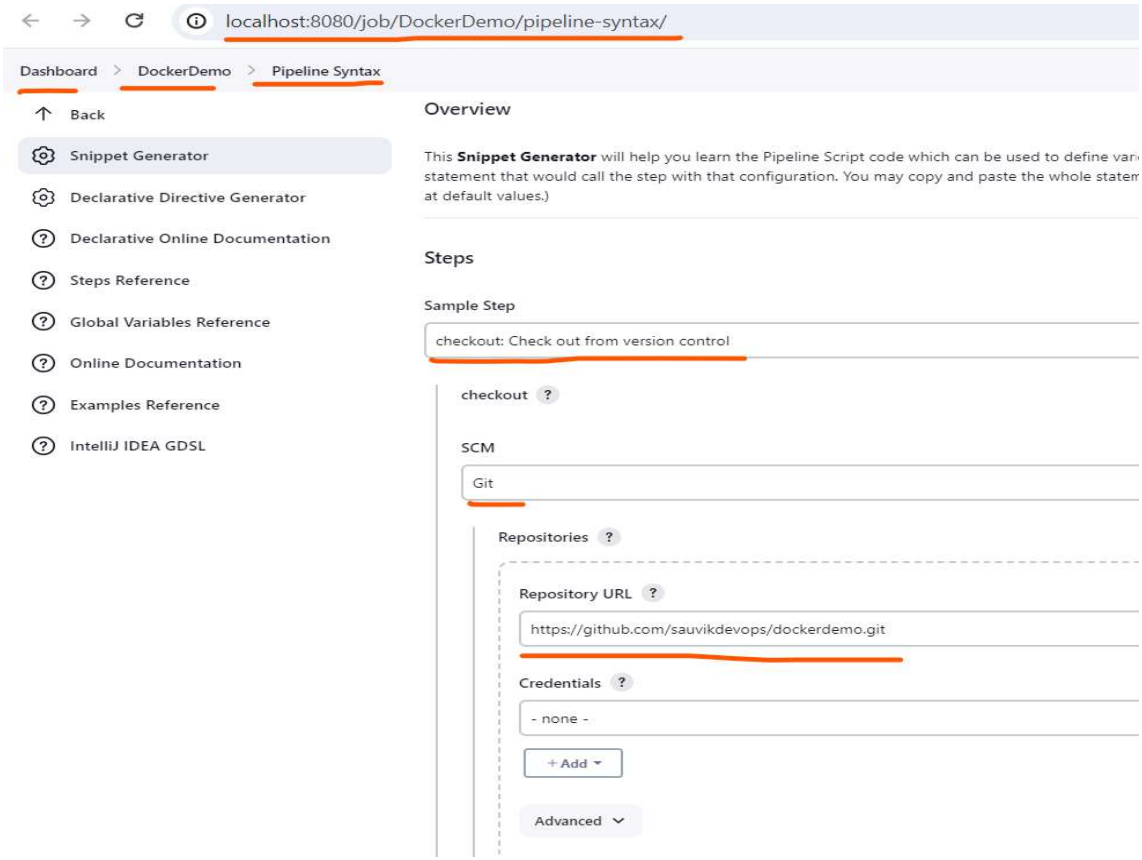


Snippet Generator will come up



Pipeline script to check out the GitHub repository

In the sample step, select – checkout: Check out from version control

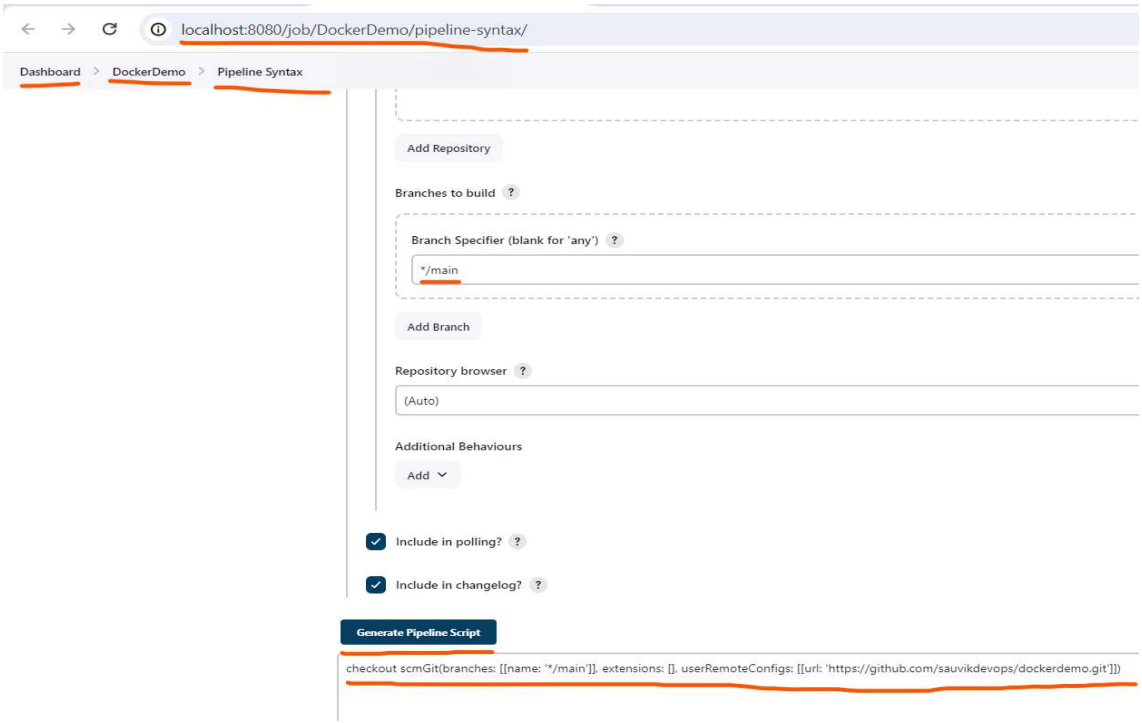


Select SCM = Git

Repository URL = URL of your GitHub repo

Scroll down and change Branch specifier = */main

Click on – “Generate Pipoeline Script”



Now, copy this code and go to Visual Studio Code → Jenkinsfile → in the steps of below stage, paste it:

```
stage('Checkout GitHub repo')
```

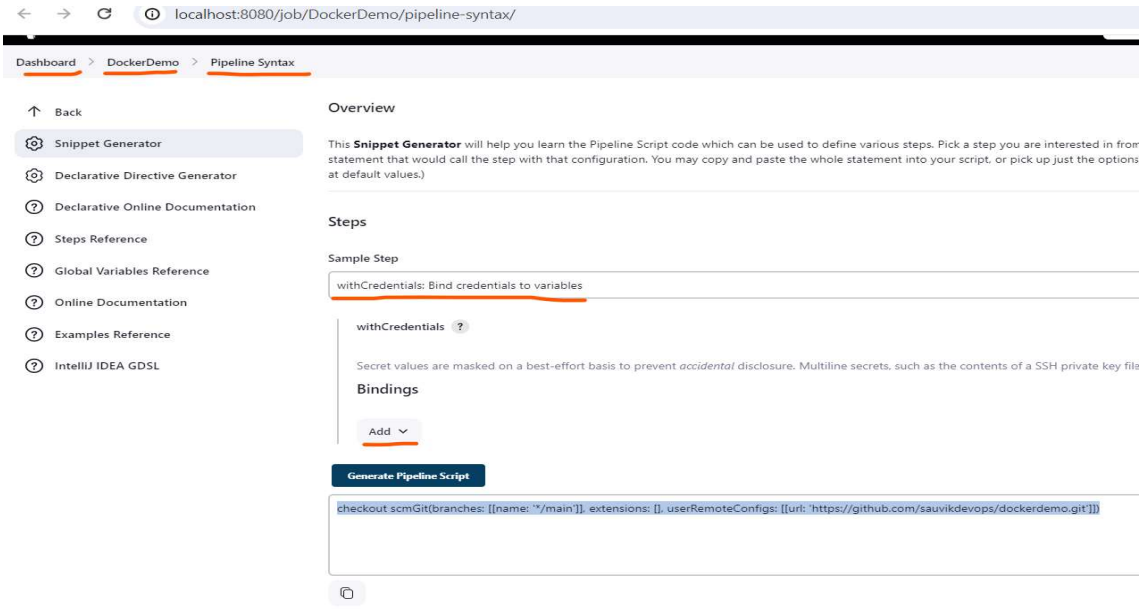
It should look like:

Now, save the file. Commit and Push to GitHub.

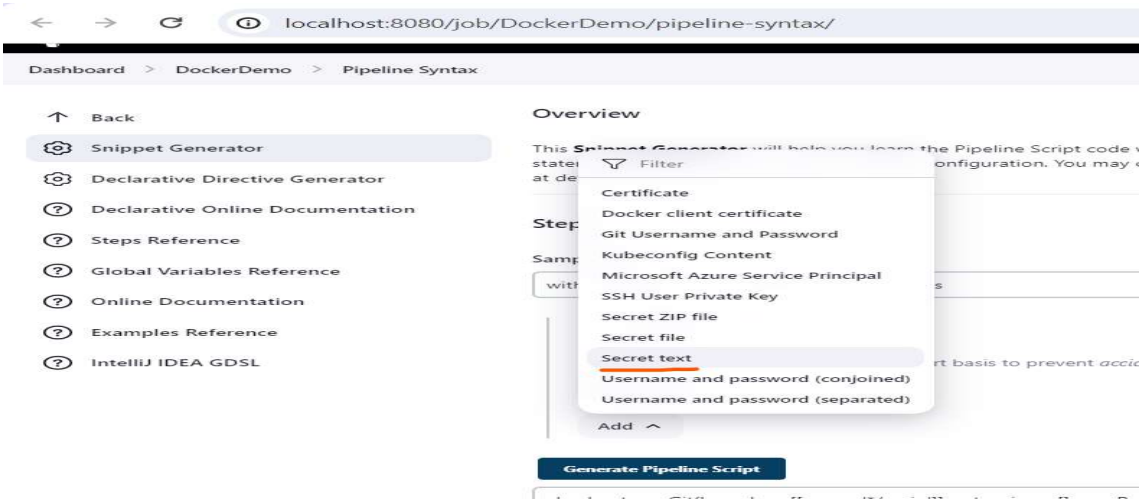
Pipeline script for Docker hub authentication

In Pipeline Syntax → Snippet Generator, Select Sample Step = with Credential Bind credential to variables

In the binding section, click on Add



From the dropdown, select - Secret text

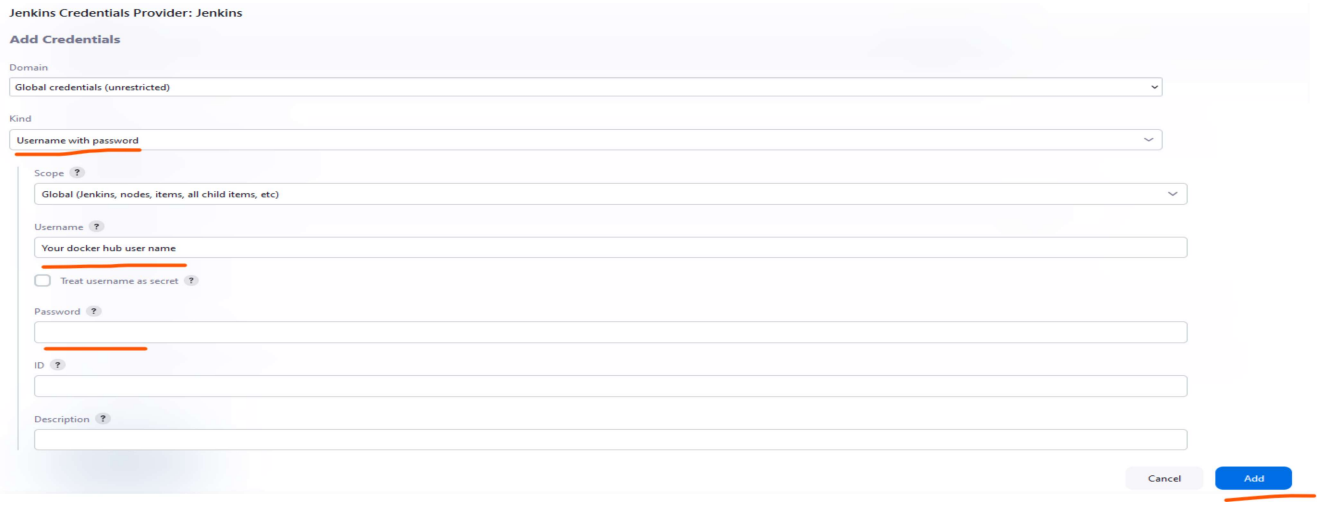


Now, put the variable = docker_hub

For credential, click on + Add

Select Jenkins

Now, put kind = Username with password. Provide your docker hub username and password and click on Add.



\

Now, copy this code and go to Visual Studio Code → Jenkinsfile → in the steps of below stage, paste it:

```
stage('Push the Docker Image to DockerHUb')
```

It should look like:

```
stage('Push the Docker Image to DockerHUb') {
    steps {
        script {
            withCredentials([string(credentialsId: 'docker hub', variable: 'docker_hub')]) {
            sh 'docker login -u sauvik.devops@gmail.com -p ${docker_hub}

            sh 'docker push sauvikdevops/learning'
        }
    }
}
```

Now, save the file. Commit and Push to GitHub.

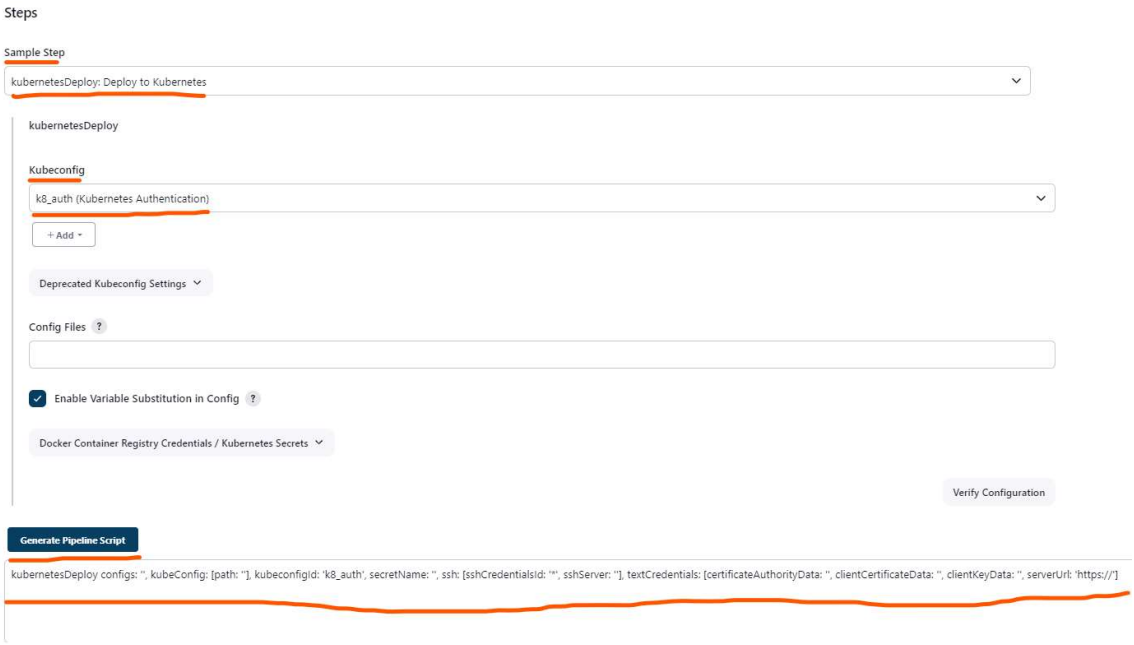
Pipeline script to allow authentication for Kubernetes deployment:

In Pipeline Syntax → Snippet Generator, Select Sample Step = kubernetesDeploy: Deploy to kubernetes

Select Kubeconfig = The Kubernetes authentication you had created in past. For our example it is K8_auth (refer the section - Configure Kubernetes authentication for Jenkins).

Click on “Generate Pipeline Script”

This will allow you to generate a syntax to deploy in Kubernetes.



Now, copy this code and go to Visual Studio Code → Jenkinsfile → in the steps of below stage, paste it:

```
stage('Deploy deployment and service file')
```

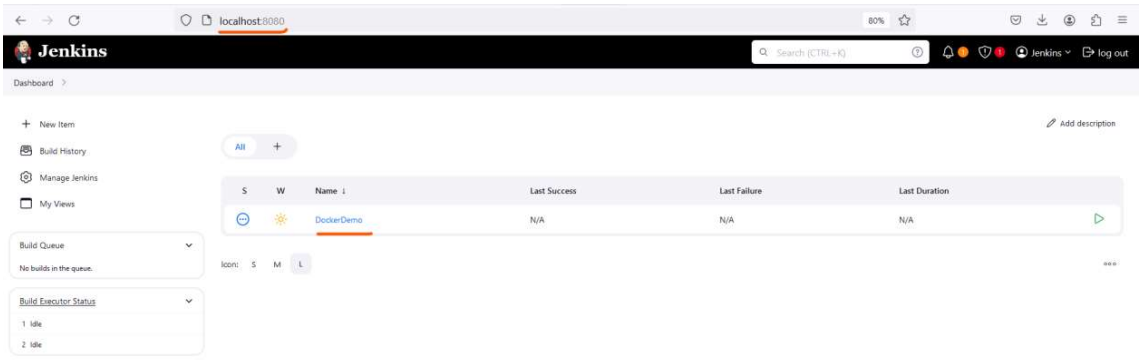
Remove the unnecessary sections so that your code will look like this:

```
stage('Deploy deployment and service file') {
    steps {
        script {
            kubernetesDeploy configs: 'deploymentsvc.yaml', kubeconfigId: 'k8_auth'
        }
    }
}
```

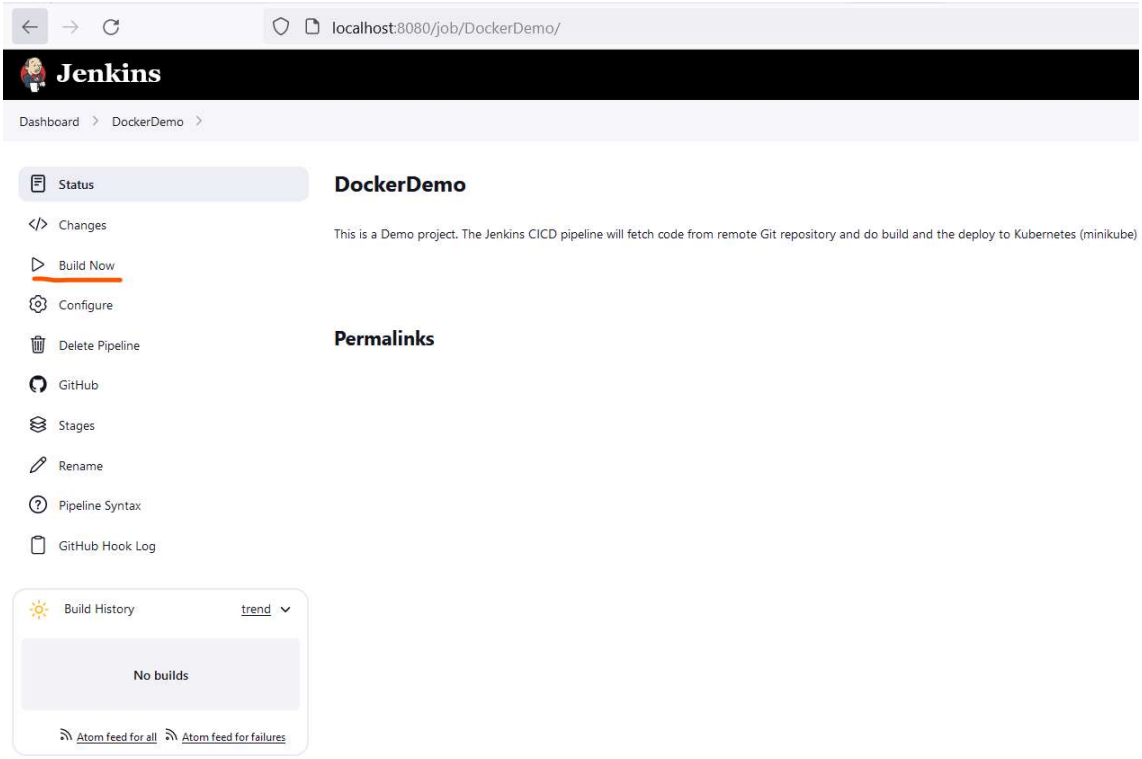
Now, save the file. Commit and Push to GitHub.

Run Jenkins Pipeline

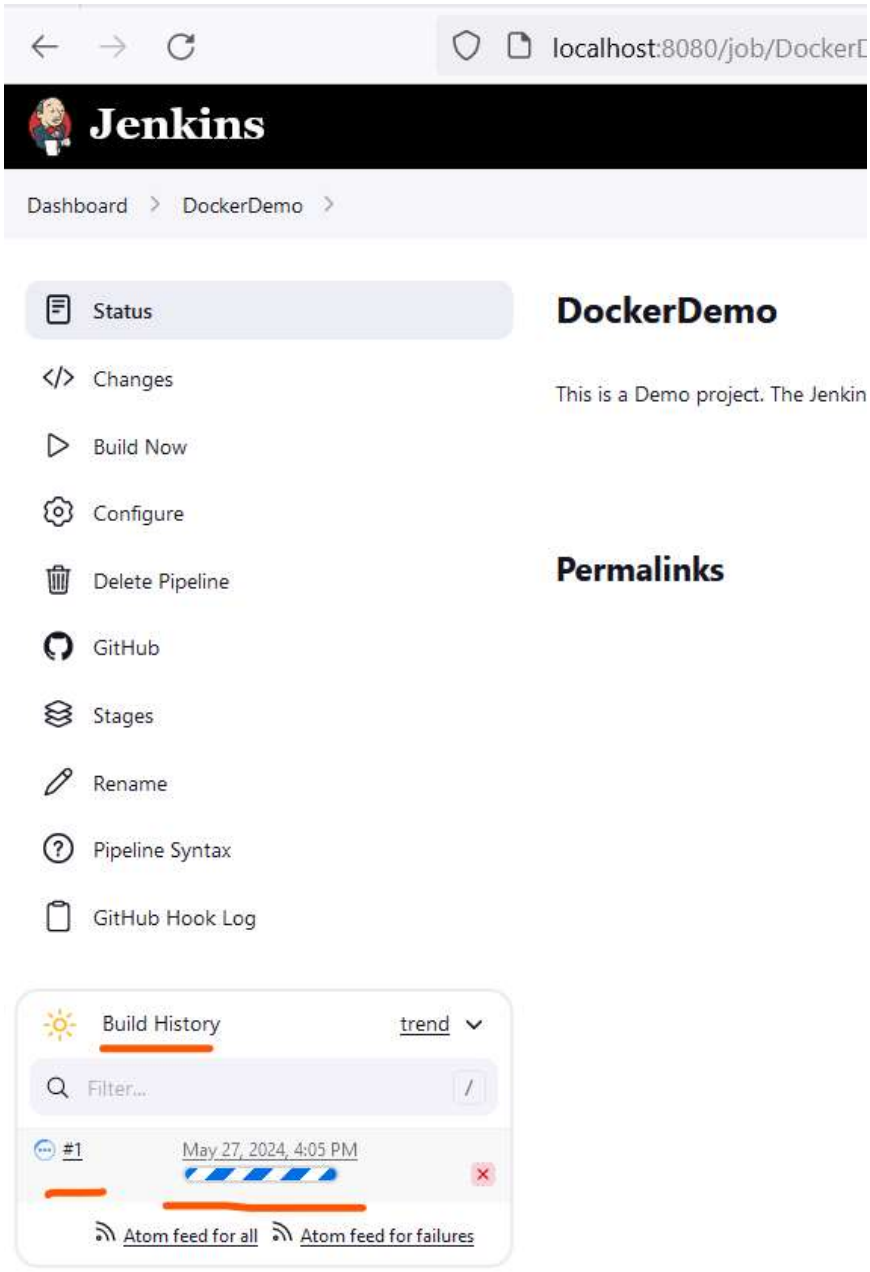
From The Jenkins Dashboard, click on the project name. For our example here, it is “Docker Demo”



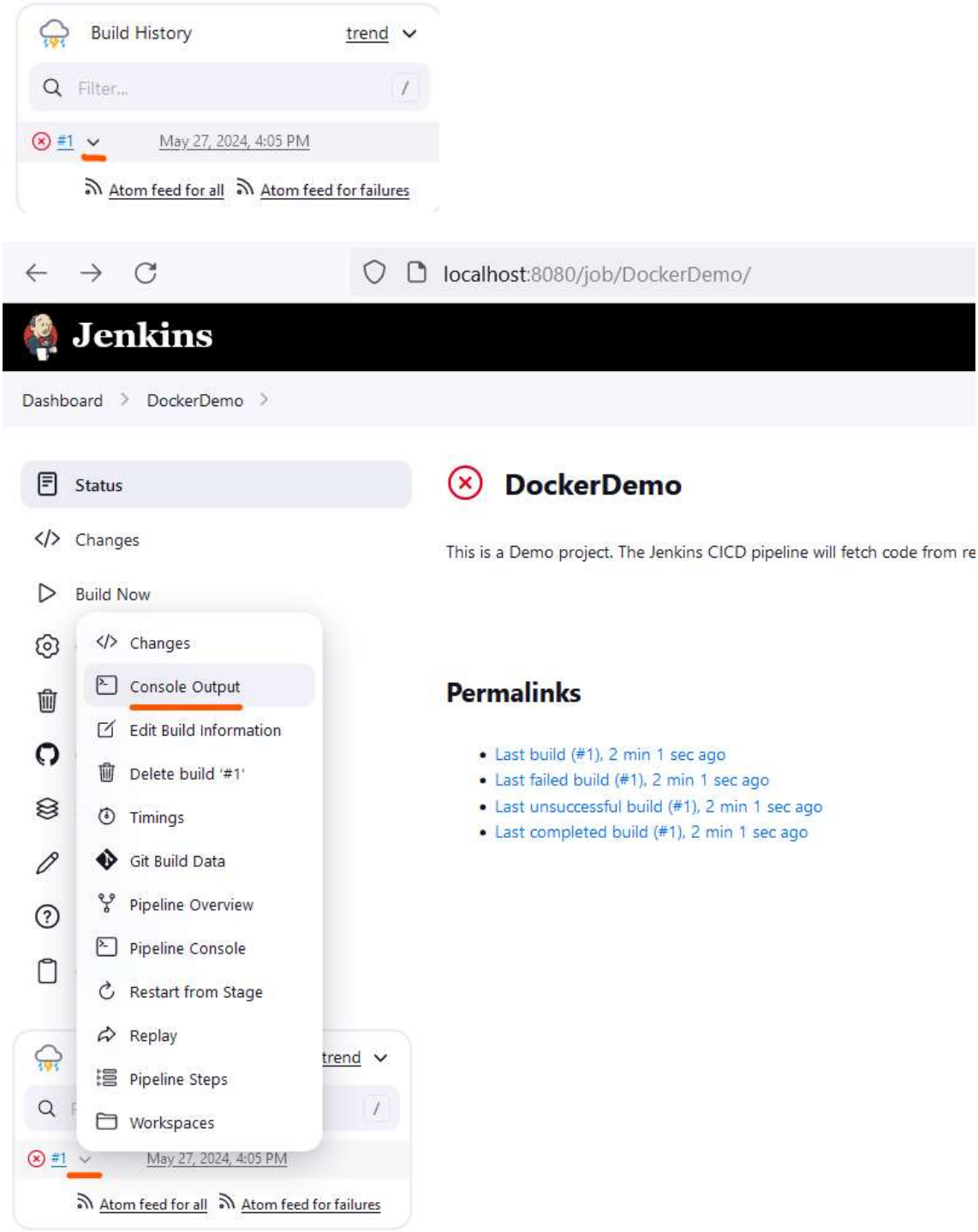
Look at the left side, click on “Build Now” –



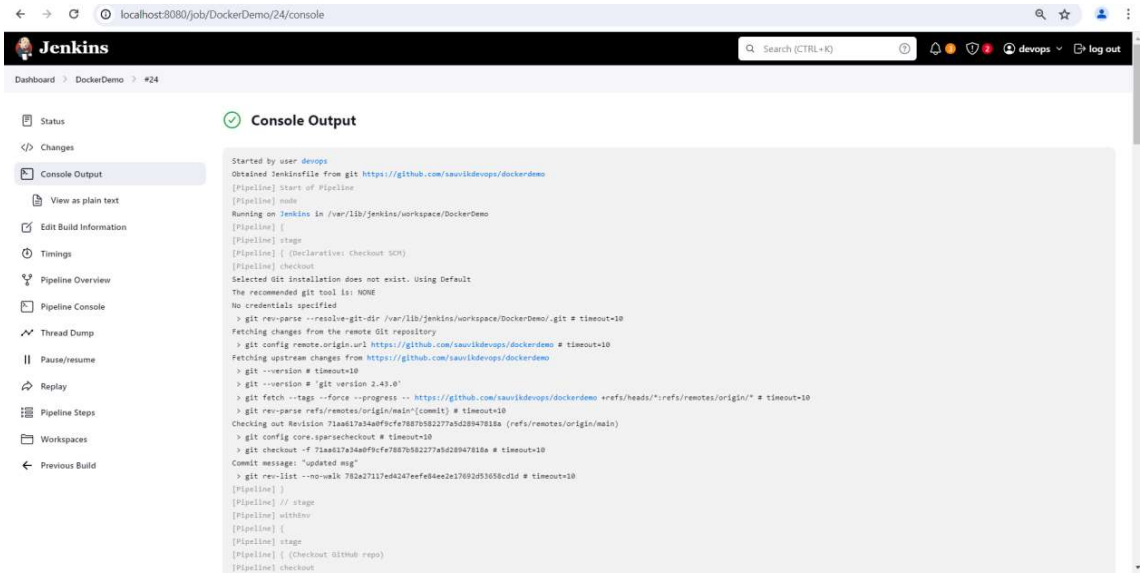
At the bottom left corner, you can see the build in progress status



Click on the downward pointing arrow (beside build number)  to see the menu, select Console Output here.



You can see CICD pipeline build and deploy log here.

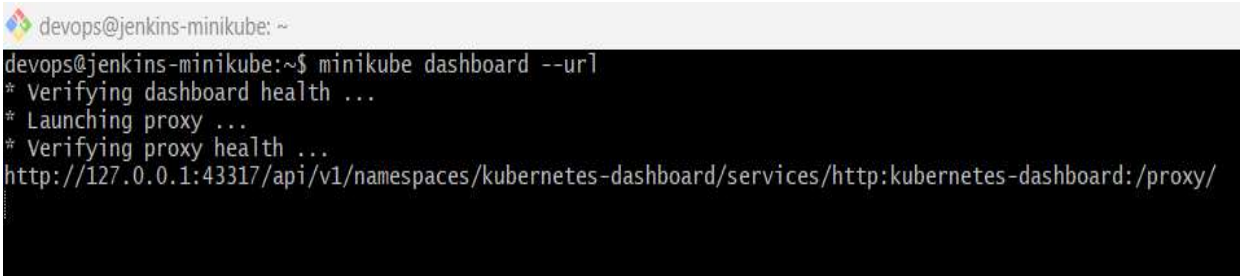


Accessing the Kubernetes Dashboard

If you're running minikube on a remote server where you can't easily access a web browser, you can run minikube dashboard with the --url option appended. This option will start the port forwarding process and provide a URL that you can use to access the dashboard, rather than opening a browser directly.

Run the below command in your Ubuntu server:

minikube dashboard –url



Copy this URL and this to be used from another terminal window later.

Note the port number that was returned by this command, as it will be different on your system and also might be different each time you run this command.

However, Kubernetes' default security configuration will prevent this URL from being accessible on a remote machine. You will need to create an SSH tunnel to access the dashboard URL. To create a tunnel from your local machine to your server, run ssh with the -L flag. Provide the port number that you noted from the forwarding process output along with the IP address of your remote server:

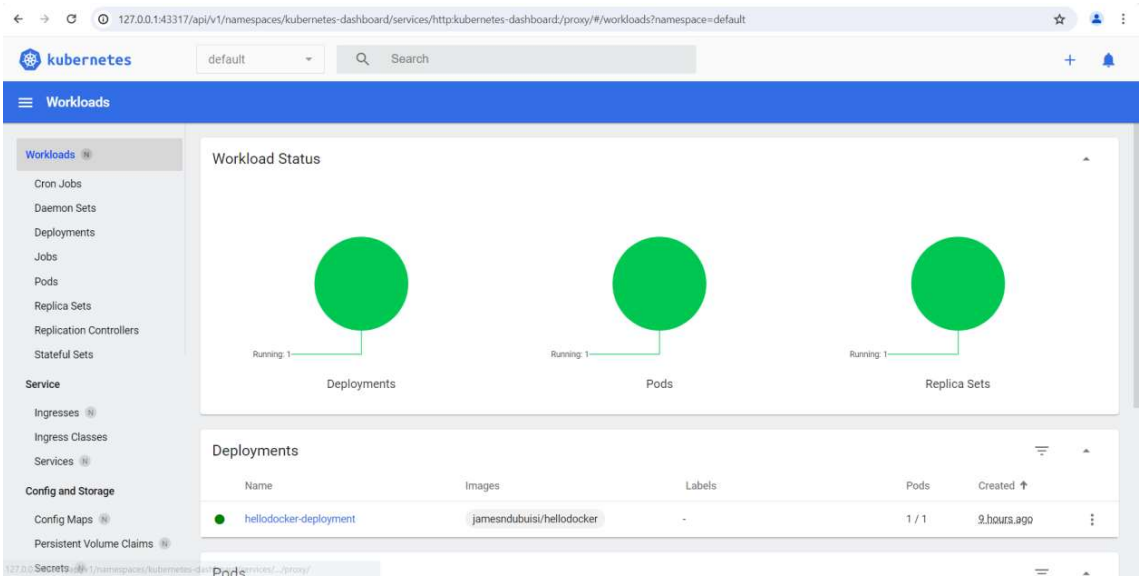
In another terminal window (Gitbash) from local machine, run this command to connect to Ubuntu server:

ssh -L 43317:127.0.0.1:43317 -p 2222 devops@127.0.0.1

After login, please hit the URL in a web browser:

http://127.0.0.1:43317/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/

You should then be able to access the dashboard now as below:



Note:

1. This URL and port number might change every time you run this command - minikube dashboard –url
2. You need to run ‘minikube dashboard –url’ in one terminal window from your local machine and keep running the command. Else it would not work. Press Ctrl+C to abort.

Check deployed application in Kubernetes

In your ubuntu server, run this command to get service list:

kubectl get svc

```
devops@jenkins-minikube:~$ kubectl get svc
NAME                  TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
hellodocker-service   NodePort    10.111.25.246   <none>        80:31421/TCP   9h
kubernetes            ClusterIP   10.96.0.1       <none>        443/TCP        26h
devops@jenkins-minikube:~$ kubectl port-forward --address 0.0.0.0 service/hellodocker-service 31421:80
Forwarding from 0.0.0.0:31421 -> 80
```

Note down the port details for your service and service name.

Run this command to show service detail which include name, url, nodePort, targetPort.

minikube service list

```
devops@jenkins-minikube: ~
devops@jenkins-minikube:~$ minikube service list
|----------------------|---------------------------|--------------|---------------------------|
|      NAMESPACE       |           NAME            | TARGET PORT  |            URL            |
|----------------------|---------------------------|--------------|---------------------------|
| default              | hellodocker-service       |           80 | http://192.168.49.2:31421 |
| default              | kubernetes                | No node port |                           |
| kube-system          | kube-dns                  | No node port |                           |
| kube-system          | metrics-server            | No node port |                           |
| kubernetes-dashboard | dashboard-metrics-scraper | No node port |                           |
| kubernetes-dashboard | kubernetes-dashboard      | No node port |                           |
|----------------------|---------------------------|--------------|---------------------------|
```

Try this command to know URI for your service:

minikube service --url <service name>

e.g. -

minikube service --url hellodocker-service

```
devops@jenkins-minikube:~$ minikube service --url hellodocker-service
http://192.168.49.2:31421
```

Create configuration to access application

Since local machine and minikube are not in the same network segment, you must do something more to access minikube service on windows. You can use command kubectl port-forward to expose service on host port, like:

kubectl port-forward --address 0.0.0.0 service/hellodocker-service 31421:80

Optionally from a different terminal (please don't interrupt the terminal where port forwarding is running) in your windows machine, run the below command to verify the listening ports and applications on Linux:

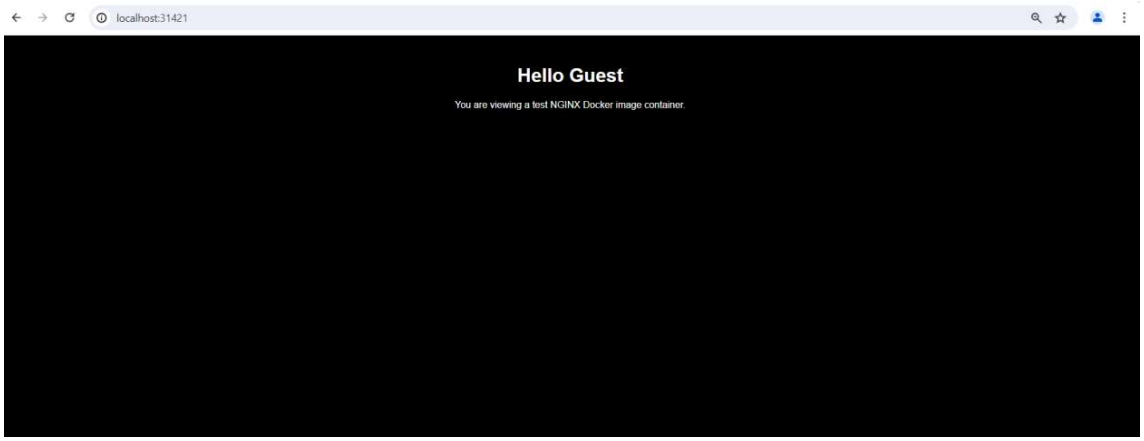
sudo lsof -i -P -n | grep LISTEN

```
devops@jenkins-minikube:~$ sudo lsof -i -P -n | grep LISTEN
[sudo] password for devops:
systemd        1            root  204u  IPv6   19043      0t0  TCP *:22 (LISTEN)
systemd-r    587 systemd-resolve   15u  IPv4    5789      0t0  TCP 127.0.0.53:53 (LISTEN)
systemd-r    587 systemd-resolve   17u  IPv4    5791      0t0  TCP 127.0.0.54:53 (LISTEN)
sshd        3478            root    3u  IPv6   19043      0t0  TCP *:22 (LISTEN)
java       34014         jenkins    8u  IPv6  145166      0t0  TCP *:8080 (LISTEN)
docker-pr 189634            root    4u  IPv4 1054908      0t0  TCP 127.0.0.1:32768 (LISTEN)
docker-pr 189647            root    4u  IPv4 1054720      0t0  TCP 127.0.0.1:32769 (LISTEN)
docker-pr 189660            root    4u  IPv4 1054958      0t0  TCP 127.0.0.1:32770 (LISTEN)
docker-pr 189673            root    4u  IPv4 1055765      0t0  TCP 127.0.0.1:32771 (LISTEN)
docker-pr 189687            root    4u  IPv4 1054989      0t0  TCP 127.0.0.1:32772 (LISTEN)
kubectl   300500          devops    3u  IPv4 1730173      0t0  TCP 127.0.0.1:43317 (LISTEN)
kubectl   317858          devops    8u  IPv4 1821680      0t0  TCP *:31421 (LISTEN)
devops@jenkins-minikube:~$
```

You will see that Ubuntu is listening the port for your service which is a good sign.

Application URL access from Windows machine

Now, from a web browser, open - http://localhost:31421/



Note: You will get port number when you run in kubernetes for getting service details.