

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

By
Sauvik Maiti

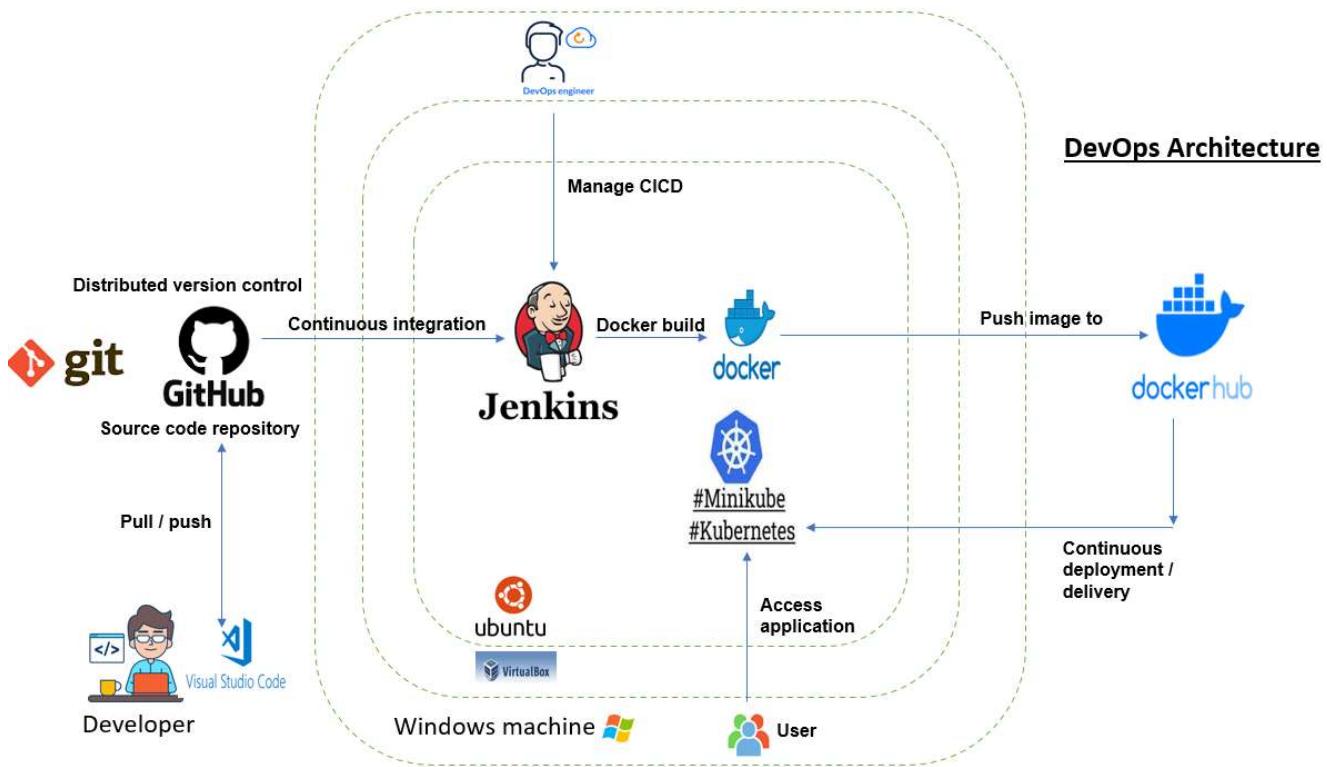
Table of Contents

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-requisites.....	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.

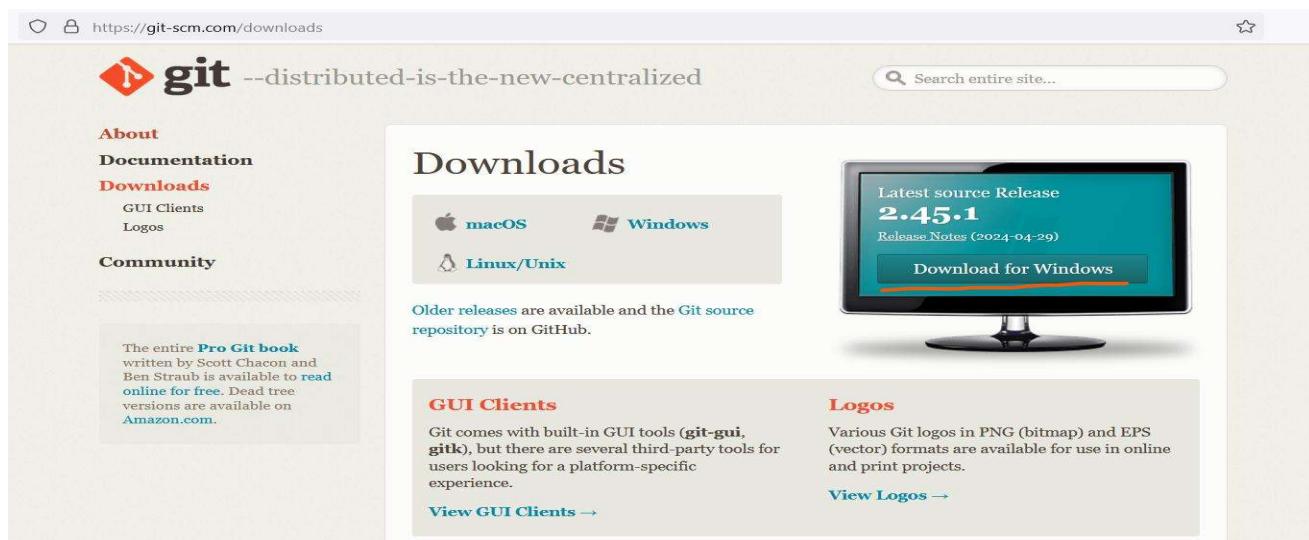
8. Users from local machine (windows) will access application deployed on Kubernetes Cluster (inside Ubuntu).

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

Install Git on local machine

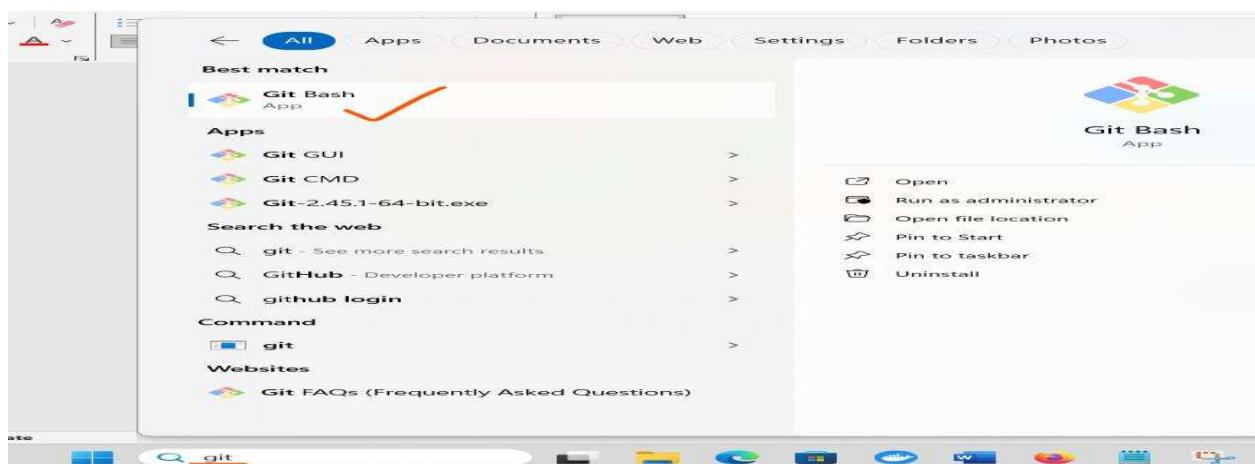
Download Git from the below URL in your local machine.

<https://git-scm.com/downloads>

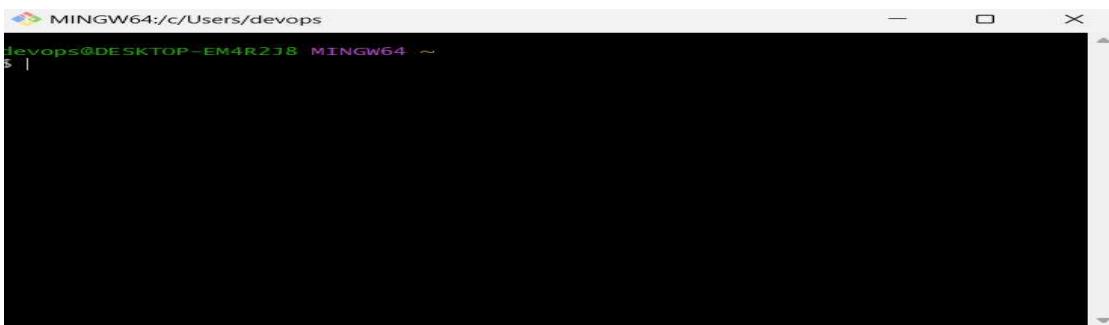


The screenshot shows the official Git website at <https://git-scm.com/>. The main navigation bar includes links for 'About', 'Documentation', 'Downloads' (selected), 'GUI Clients', 'Logos', and 'Community'. On the right, there's a search bar and a 'Latest source Release' section showing '2.45.1' with a 'Release Notes (2024-04-29)' link and a 'Download for Windows' button. Below this, there are sections for 'macOS', 'Windows', and 'Linux/Unix'. A note states: 'Older releases are available and the Git source repository is on GitHub.' To the left, there's a sidebar with information about the 'Pro Git book' and a link to 'View GUI Clients →'.

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.

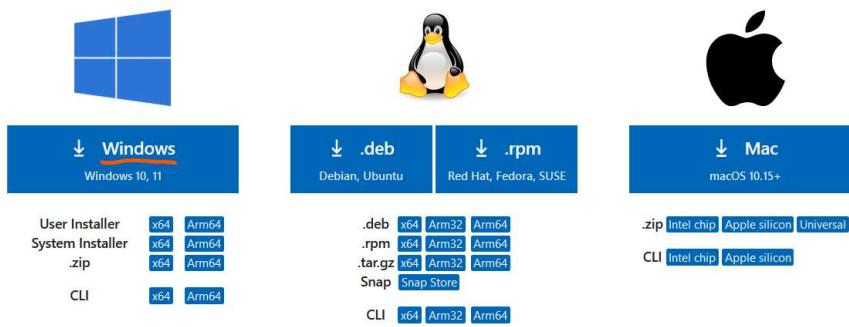


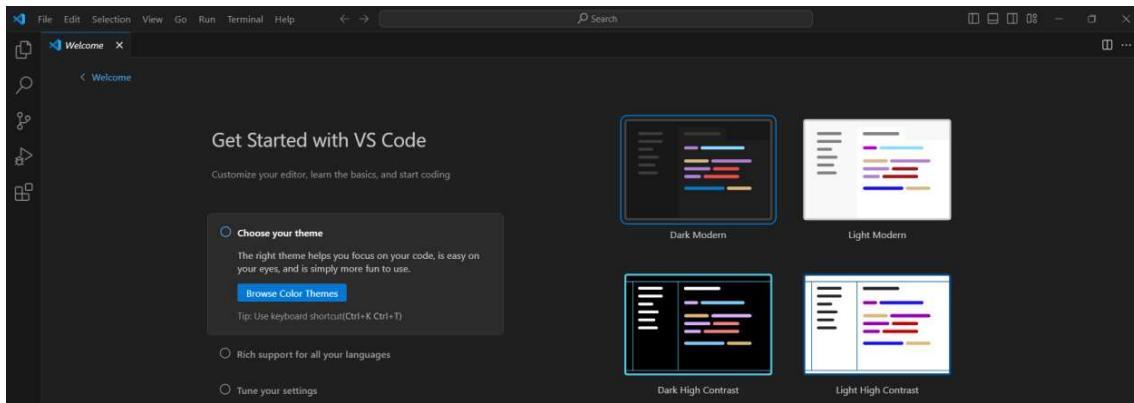
Install & configure Visual Studio Code on local machine

Download and install Visual Studio Code from the below link:

<https://code.visualstudio.com/download>

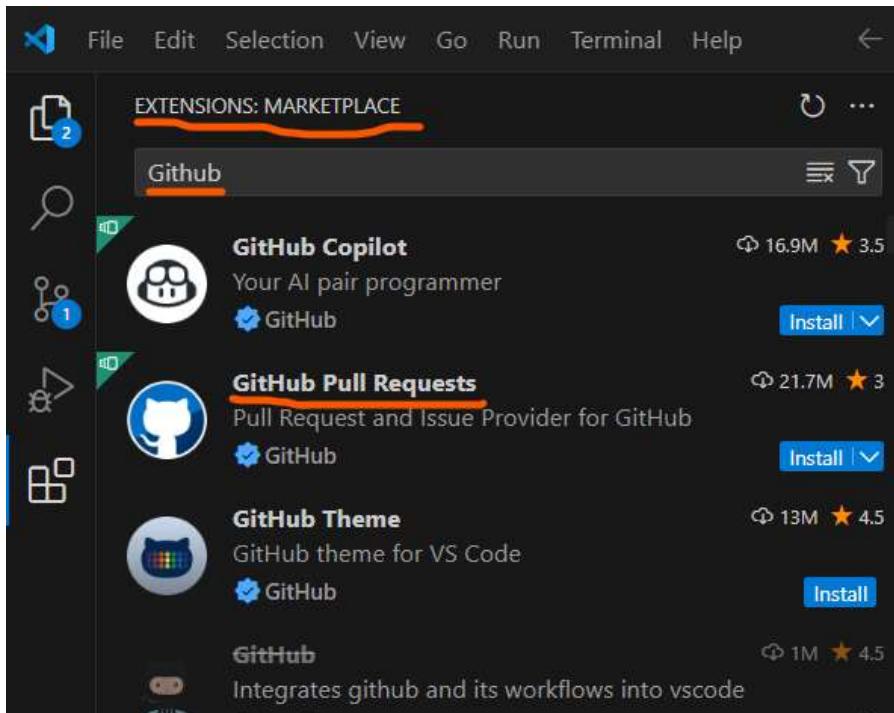
A screenshot of the Visual Studio Code download page. At the top, there is a navigation bar with links for "Visual Studio Code", "Docs", "Updates", "Blog", "API", "Extensions", "FAQ", and "Learn". To the right of the navigation bar is a search bar labeled "Search Docs" and a blue "Download" button. Below the navigation bar, a message states "Version 1.89 is now available! Read about the new features and fixes from April." The main content area has a heading "Download Visual Studio Code" and a sub-heading "Free and built on open source. Integrated Git, debugging and extensions." Below this, there are three large download buttons for "Windows", ".deb", and ".rpm". Each button has a corresponding icon (Windows logo, Tux logo, and Apple logo) and a "macOS" button below it. Under each download button, there is a table showing various installer options with their file types and supported architectures.





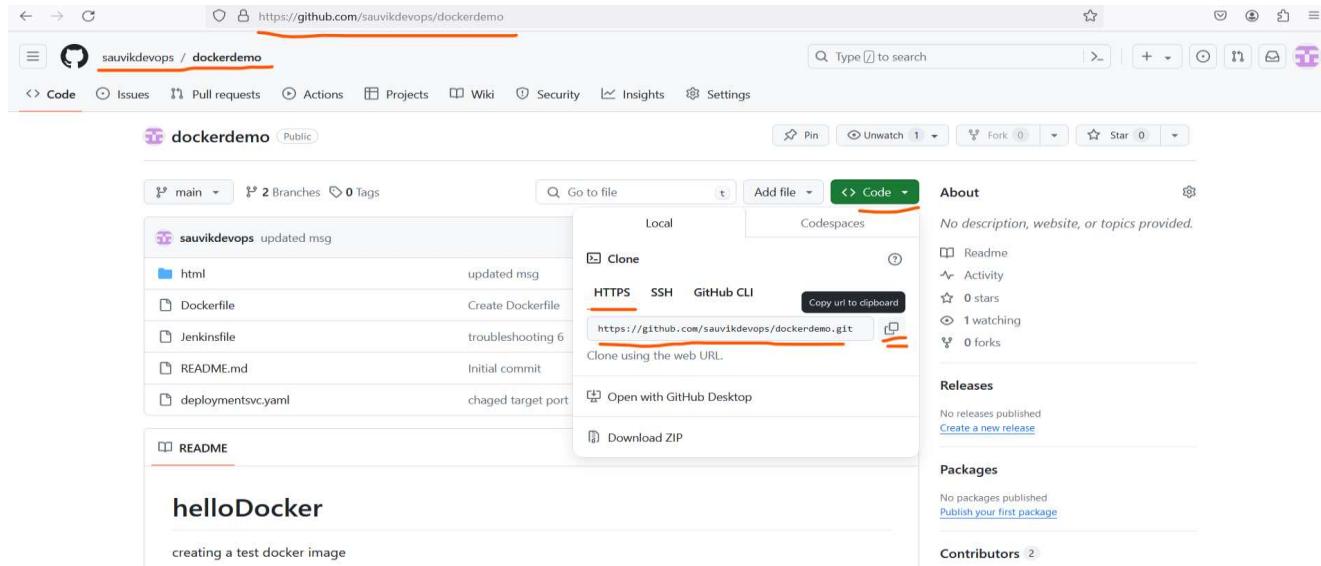
Clone a repository locally

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

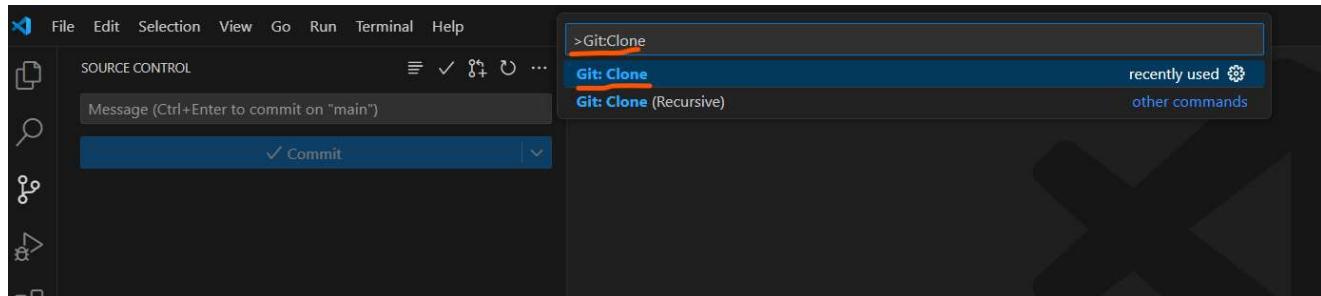


Select GitHub Pull Requests and click on install.

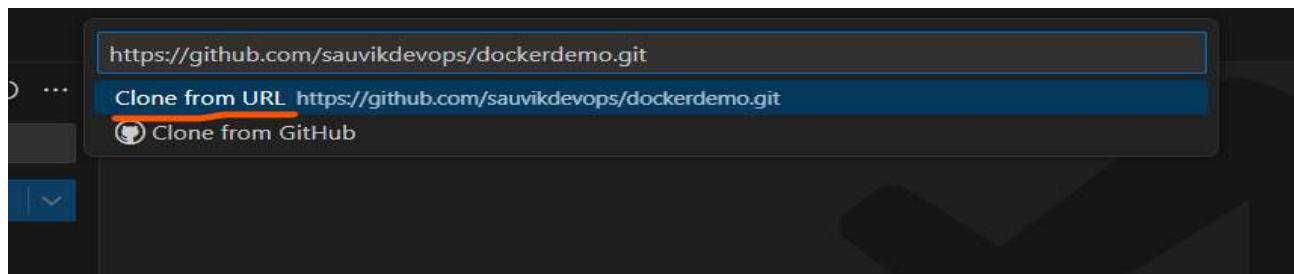
Now, we're going to add a GitHub repository to VS Code. You'll want to go back to your GitHub account (<https://github.com>) with your default browser and locate the address of the repository you want to add. Once you've navigated to the repository in question, click the Code dropdown, and copy the URL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



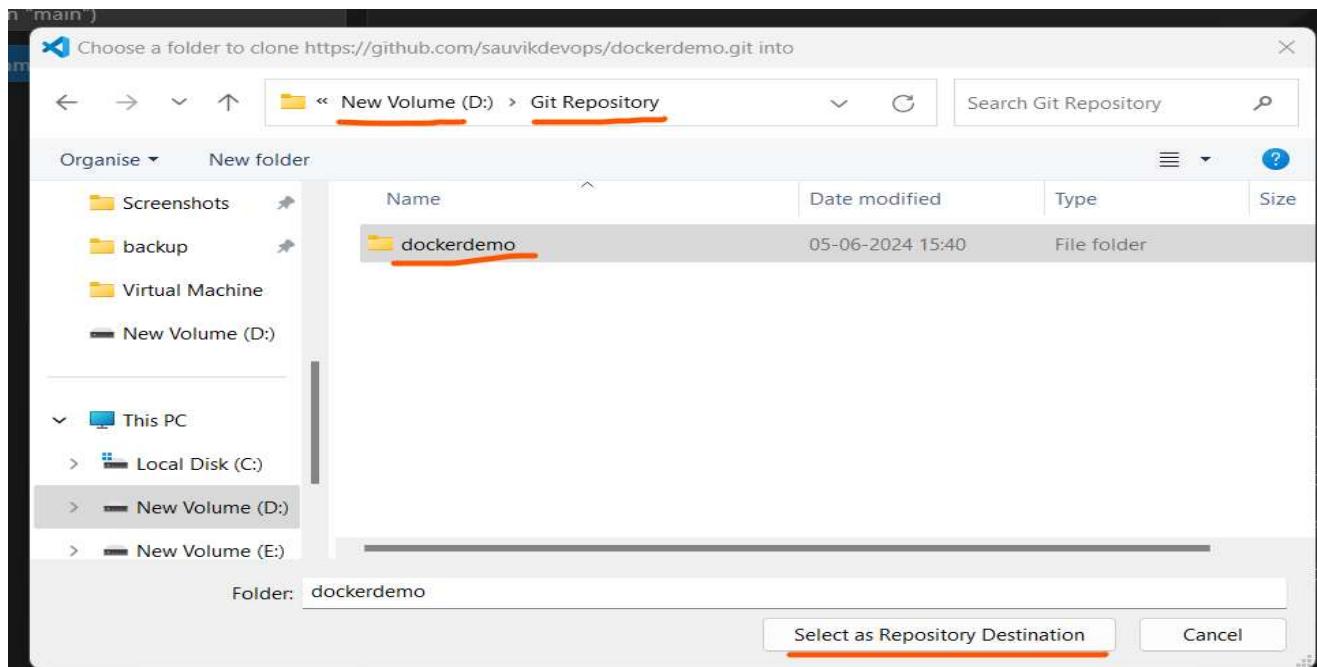
Now, come back to Visual Studio Code and press Ctrl + Shift + P to open command palette. Put Git:clone in search box:



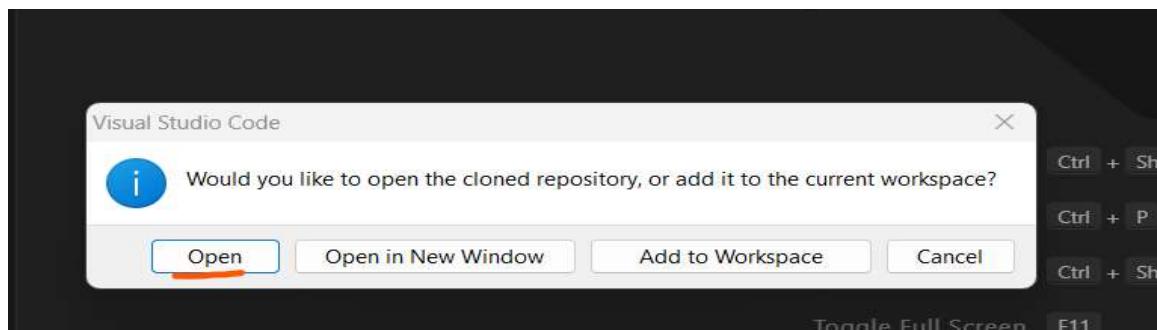
Put the copied URL of your Github repo and click on "Copy from URL ..."



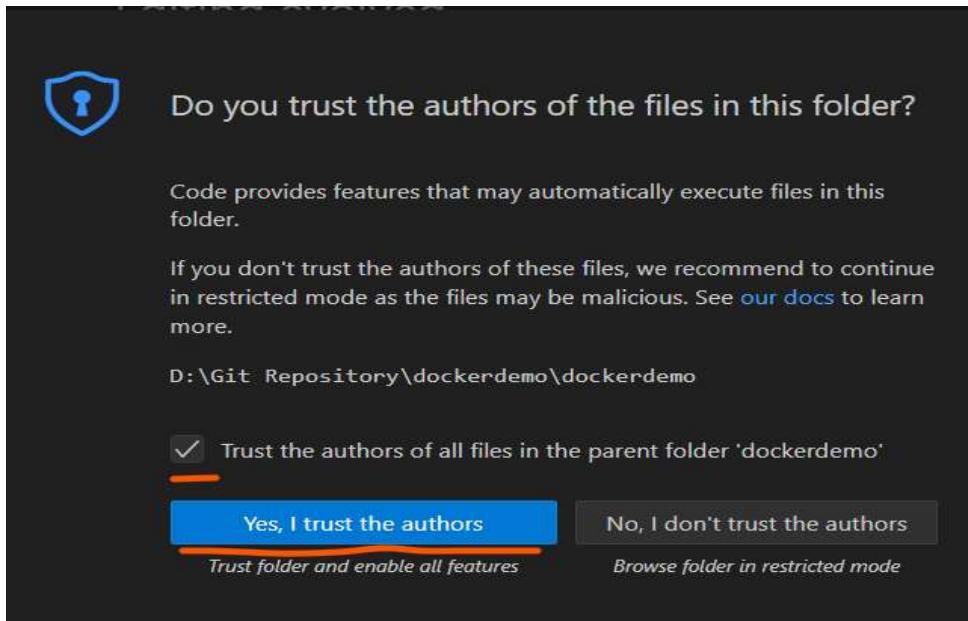
It will now ask for Repository Destination i.e. the folder in your local computer where the Github repo would be cloned. For example I have created a folder called “deckerdemo” (usually we keep it as our project name) inside D:\Git Repository folder. You can keep it anywhere you would like.



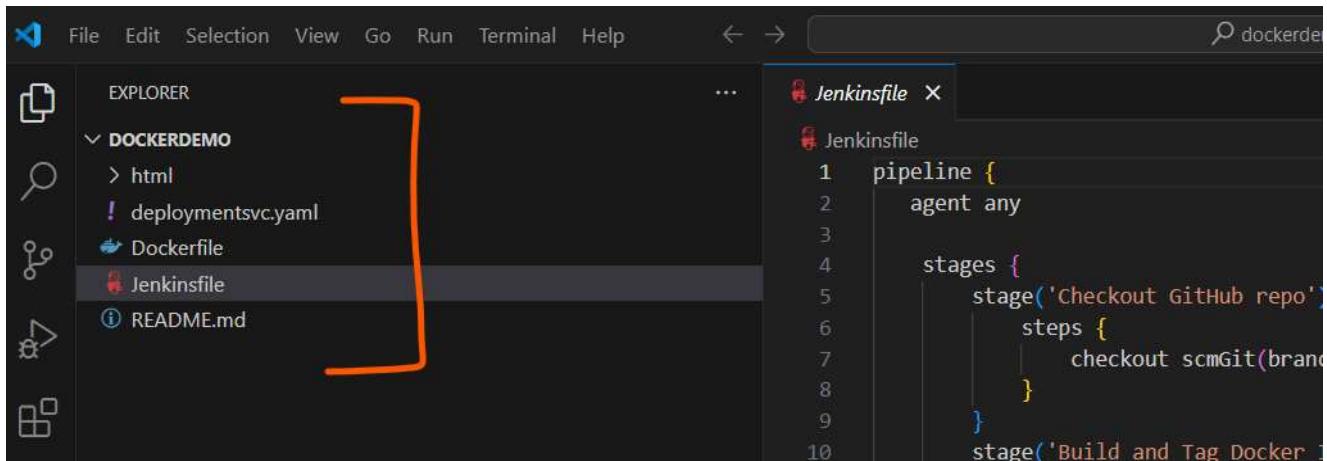
Click on “Open” if it asks. You can also add it into current VsCode workspace as well.



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

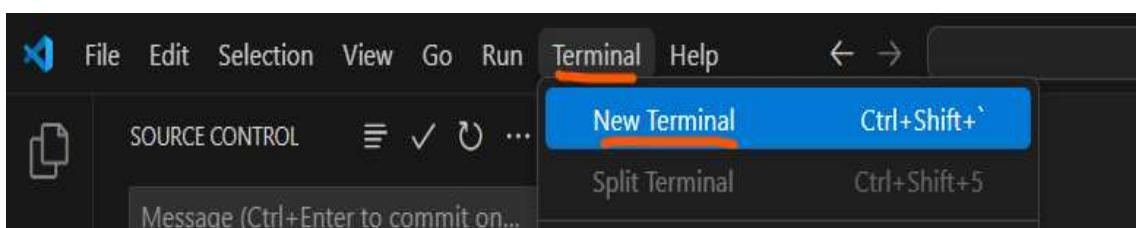


Now, you should be able to see the cloned repo in local.



[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"
```

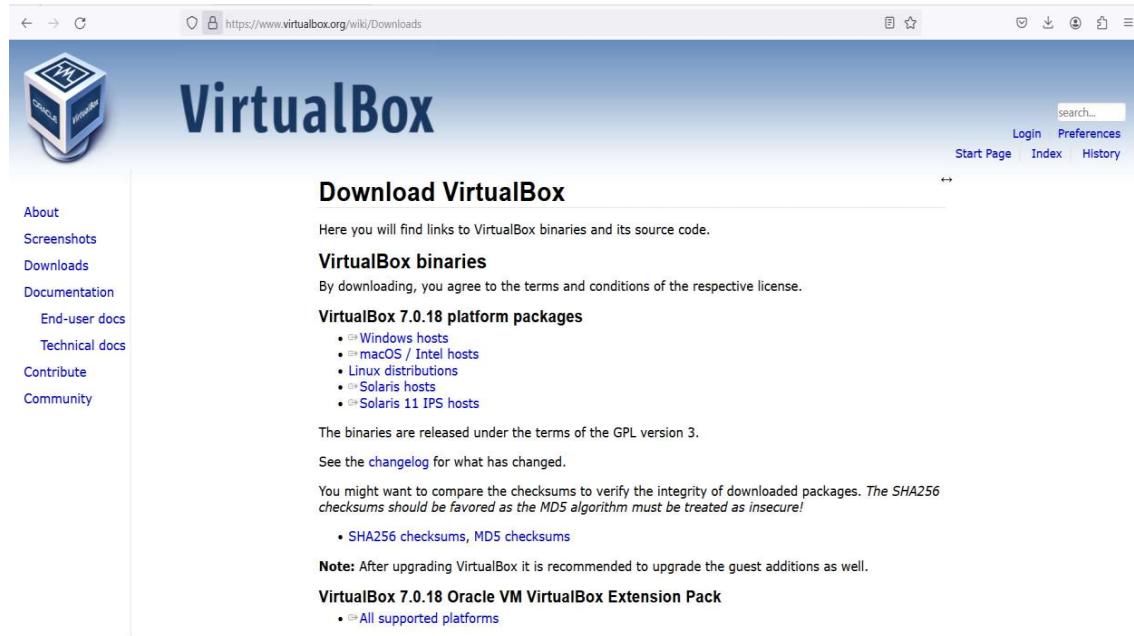
Note: Replace my email with your.

```
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

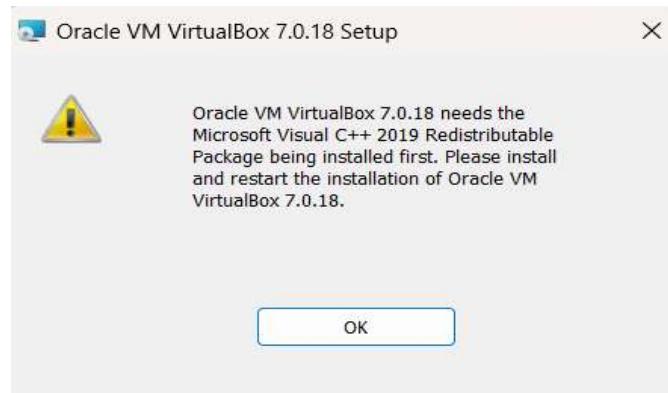
<https://www.virtualbox.org/wiki/Downloads>



The screenshot shows a web browser displaying the VirtualBox website at <https://www.virtualbox.org/wiki/Downloads>. The page title is "VirtualBox". On the left, there is a sidebar with links to "About", "Screenshots", "Downloads", "Documentation", "End-user docs", "Technical docs", "Contribute", and "Community". The main content area is titled "Download VirtualBox" and contains instructions for finding binaries and source code. It lists "VirtualBox binaries" and "VirtualBox 7.0.18 platform packages" for various host operating systems. It also mentions the GPL version 3 license and provides links for SHA256 and MD5 checksums. A note at the bottom advises upgrading guest additions after an upgrade.

Install Oracle VM VirtualBox pre-requisites

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>

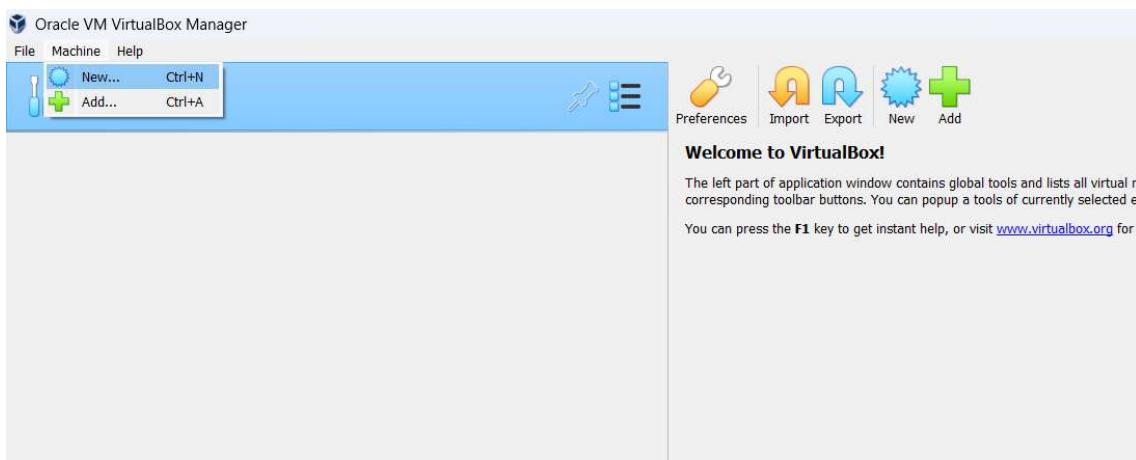
The screenshot shows the Ubuntu Server download page. At the top, there's a navigation bar with links like 'Canonical Ubuntu', 'Products', 'Use cases', 'Support', 'Community', 'Get Ubuntu', 'All Canonical', 'Sign in', and a search icon. Below the navigation is a secondary menu with tabs: 'Downloads' (highlighted), 'Desktop', 'Server' (selected), 'Core', and 'Cloud'. The main content area features a large heading 'Get Ubuntu Server'. Below it, there are three options: 'Manual installation' (selected), 'Instant VMs', and 'Automated provisioning'. A prominent section for 'Ubuntu 24.04 LTS' is shown, featuring a large orange crown icon. To the right of the icon, text explains that it's the latest LTS version, which stands for long-term support, meaning five years of free security and maintenance updates, extended to 10 years with Ubuntu Pro. A large green button labeled 'Download 24.04 LTS' is available, along with links for 'Alternative downloads' and 'Alternative architectures'. At the bottom of this section, there are links for 'What's new', 'System requirements', and 'How to install'.

Click on Download to proceed.

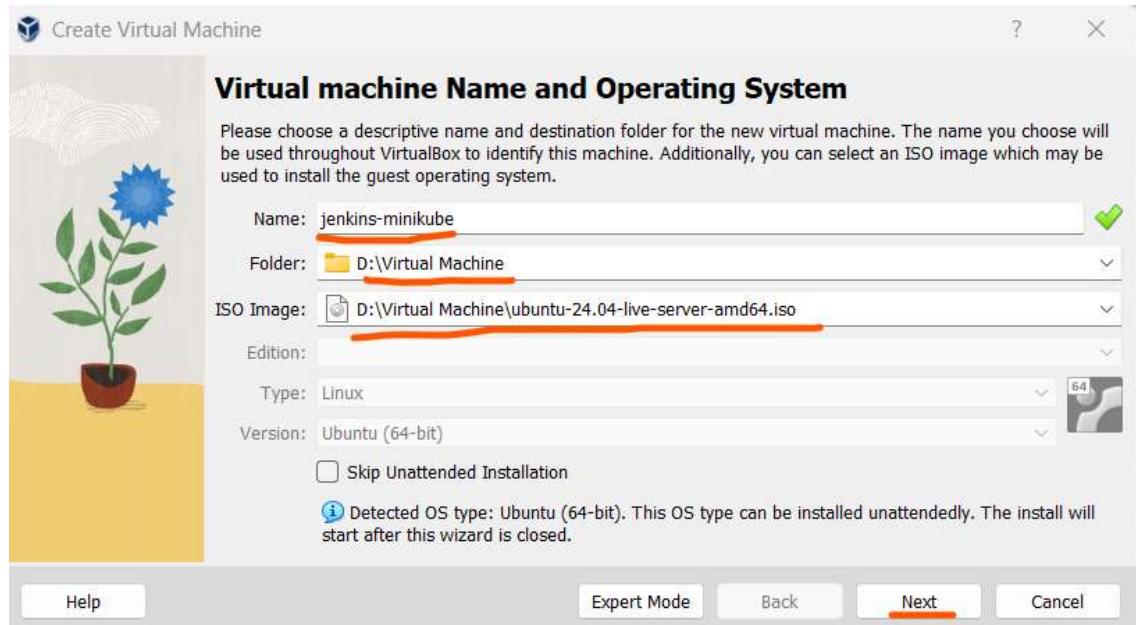
Note: For your lab, you do not need to download this Ubuntu ISO on every machine separately. Just download once and then copy to all the machines.

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.

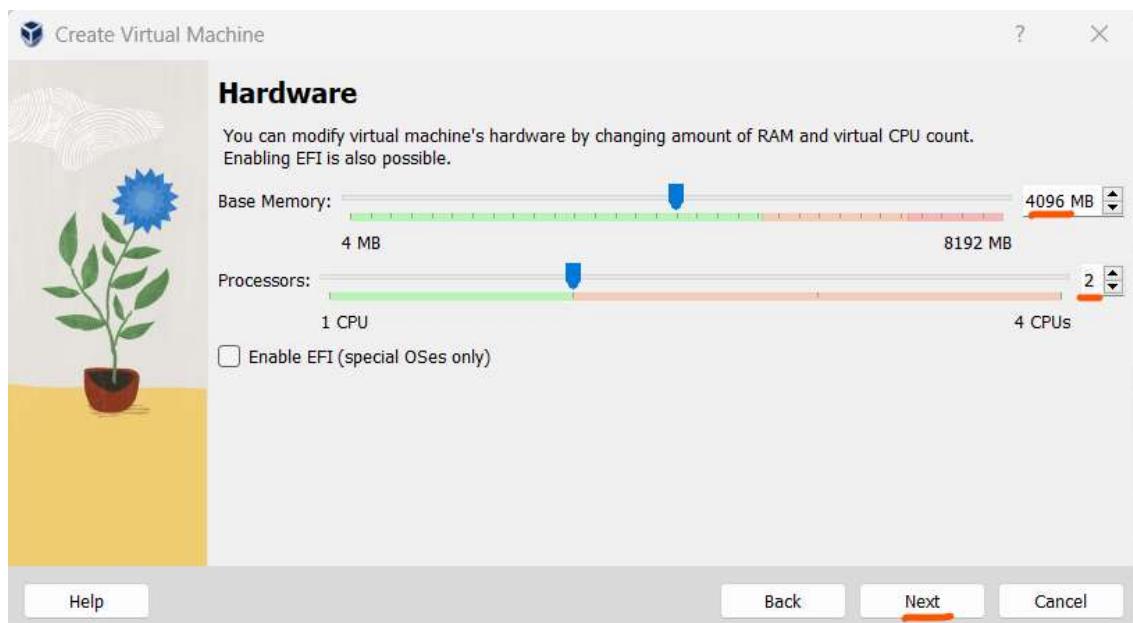


Click Next to proceed.

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

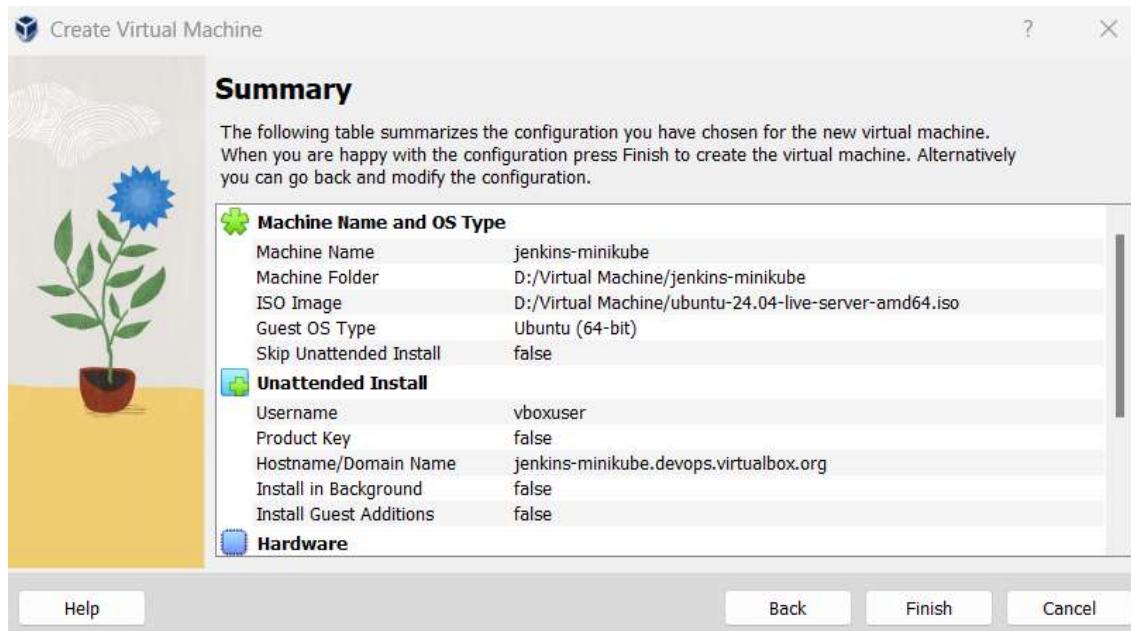


Click Next to proceed.



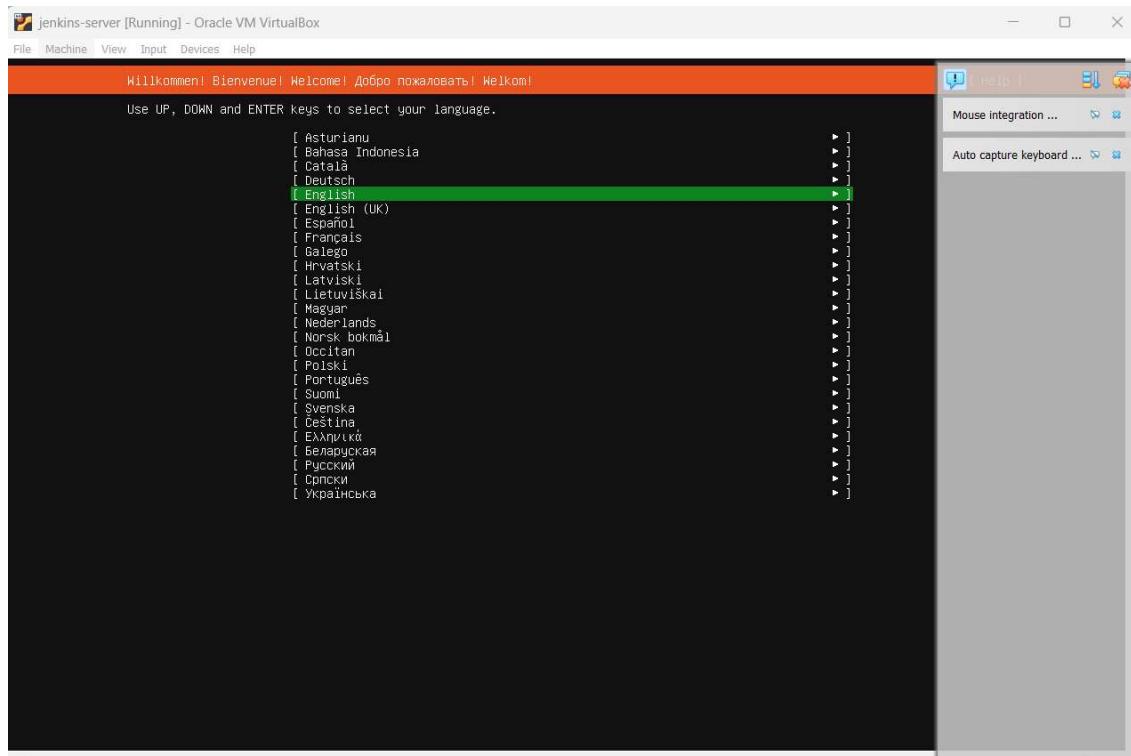
Set the virtual hard disk for this Ubuntu guest OS. I would recommend to have atleast 100 GB.

Click Next to proceed.

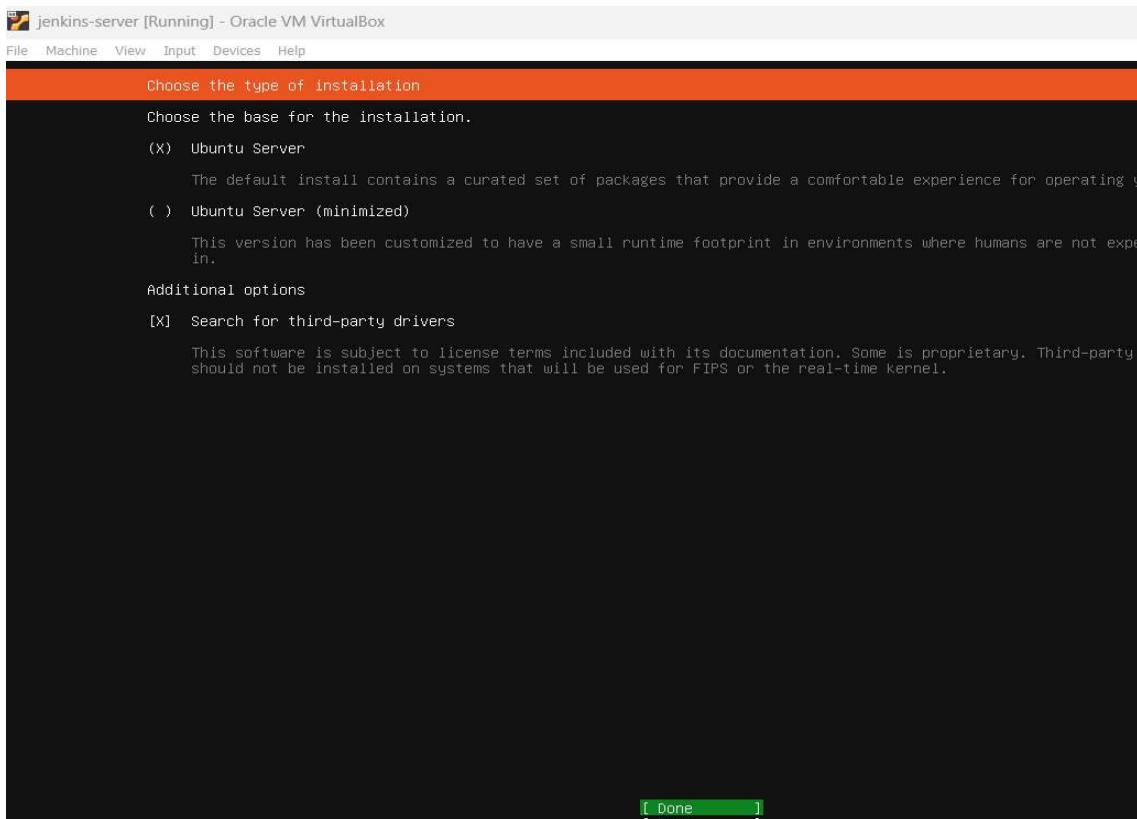
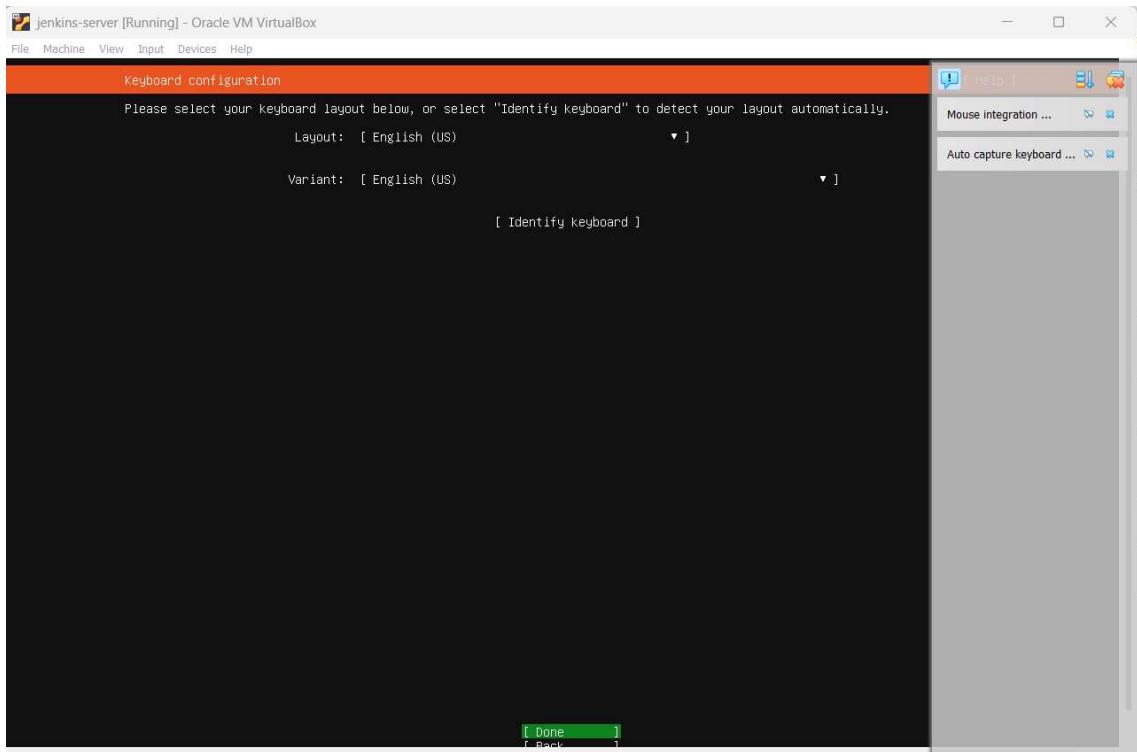


In the Summary page, just verify the settings you have set and click on Finish. It would then ask you for few preferences.

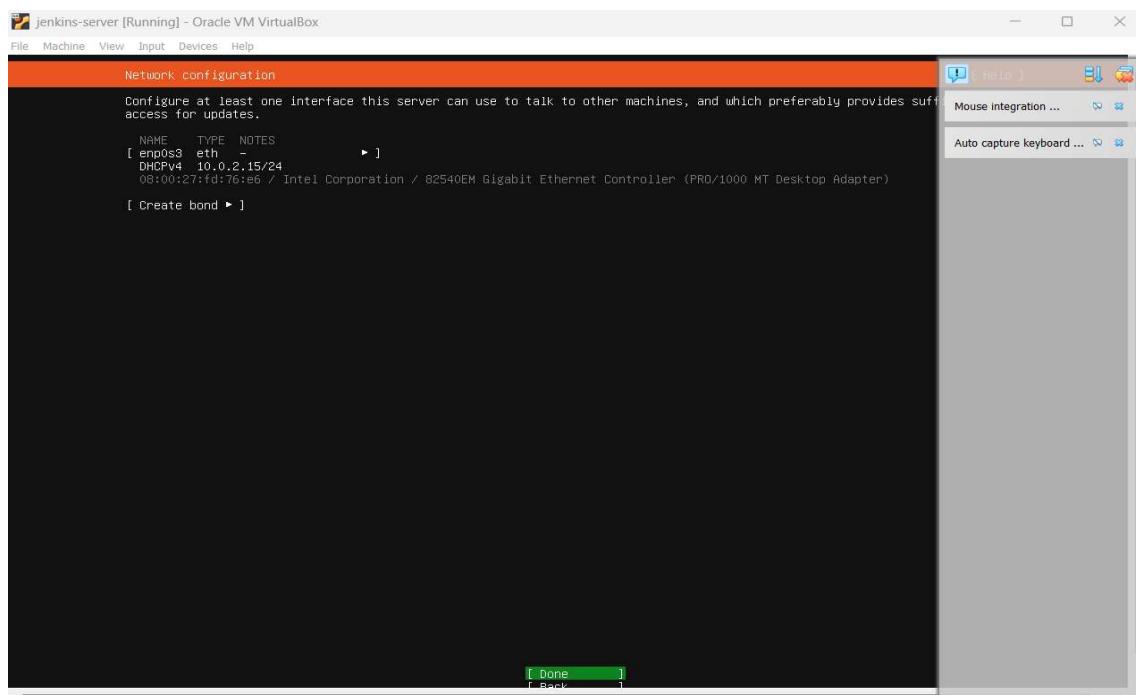
Select (or keep as it is if already selected) English as language and press enter to go to next screen.



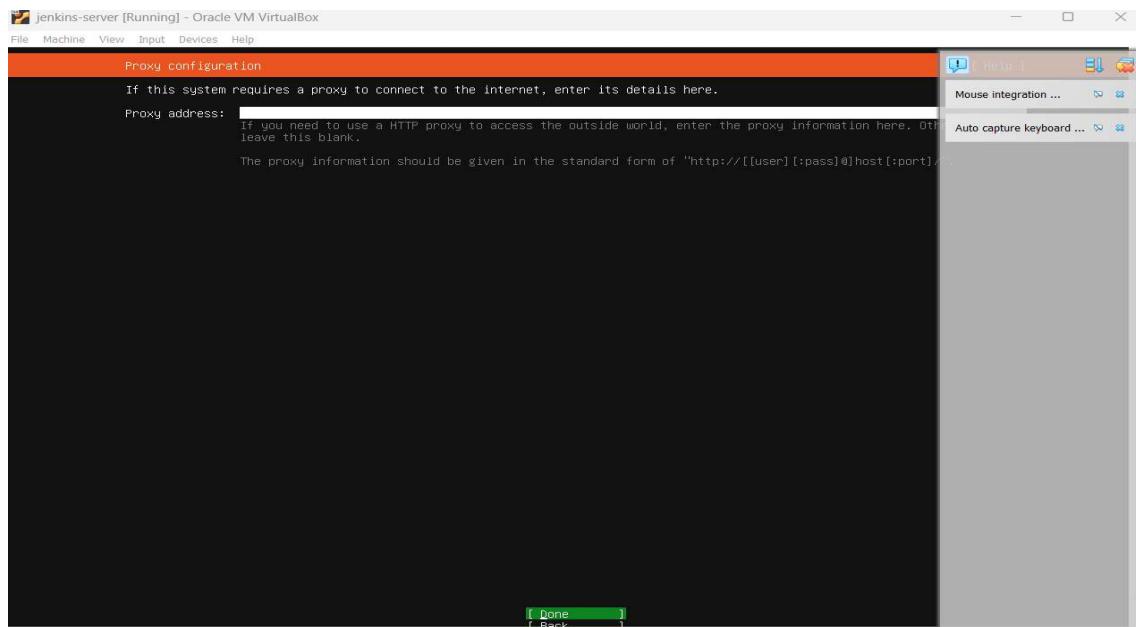
Leave the default keyboard layout and variant as it is and press enter to go to next.



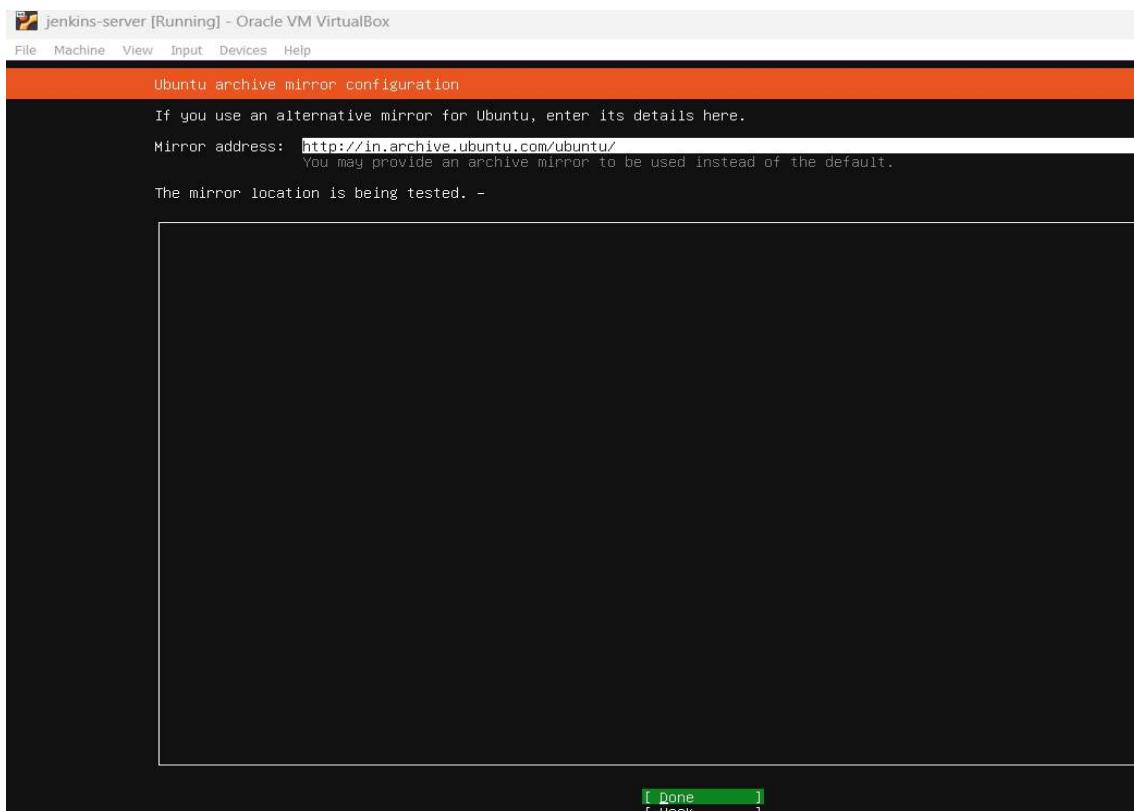
Select (or leave it if already selected) “Ubuntu Server” and press enter on “Done” to proceed.



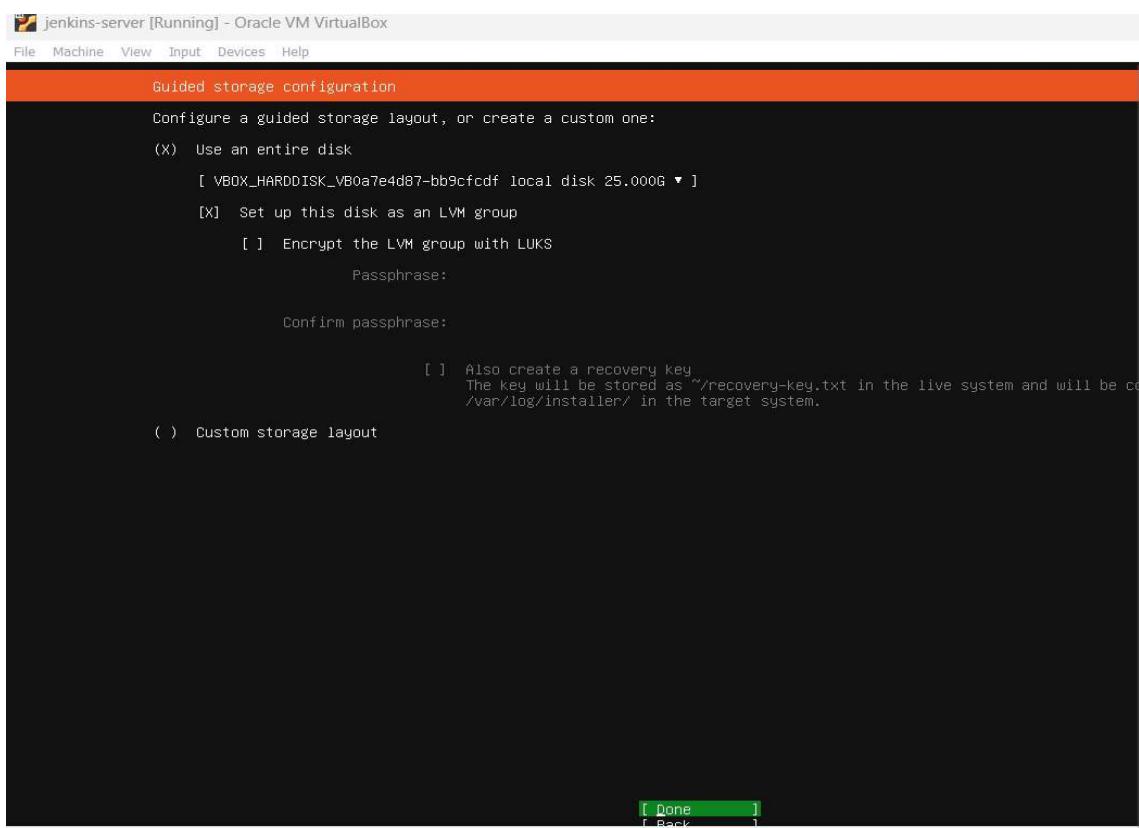
Leave the default network configuration and press enter on “Done” to proceed.



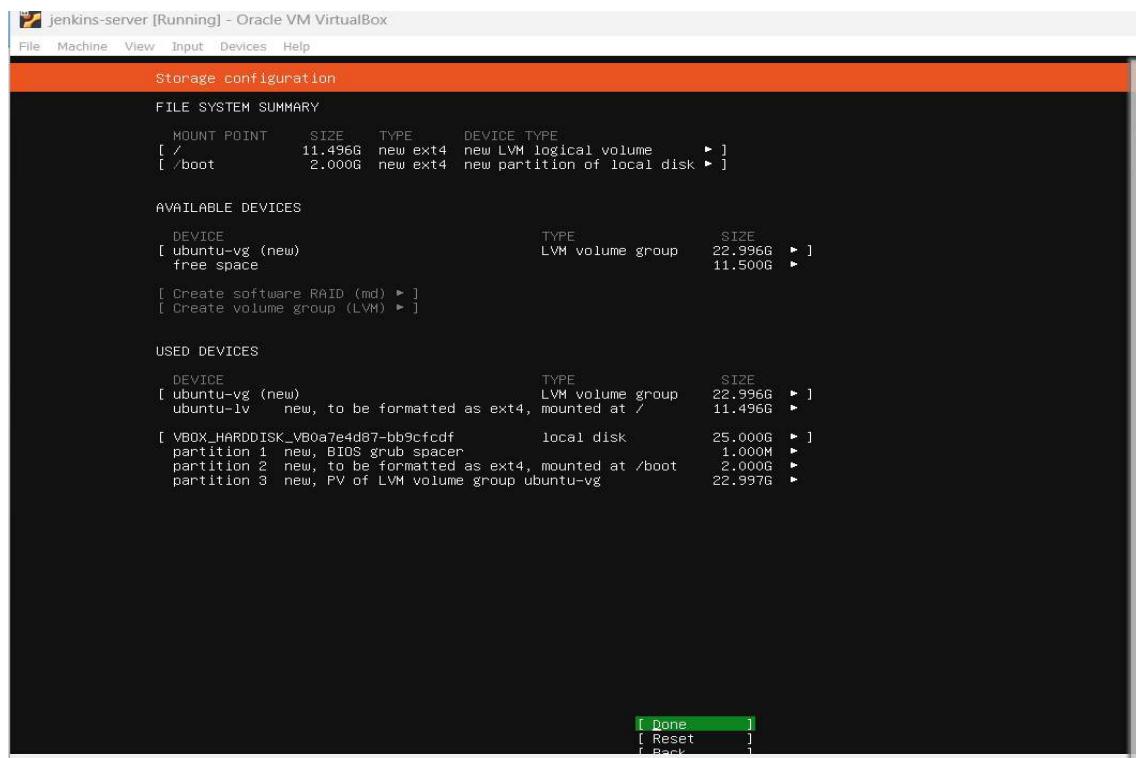
Leave the default Proxy configuration and press enter on “Done” to proceed.



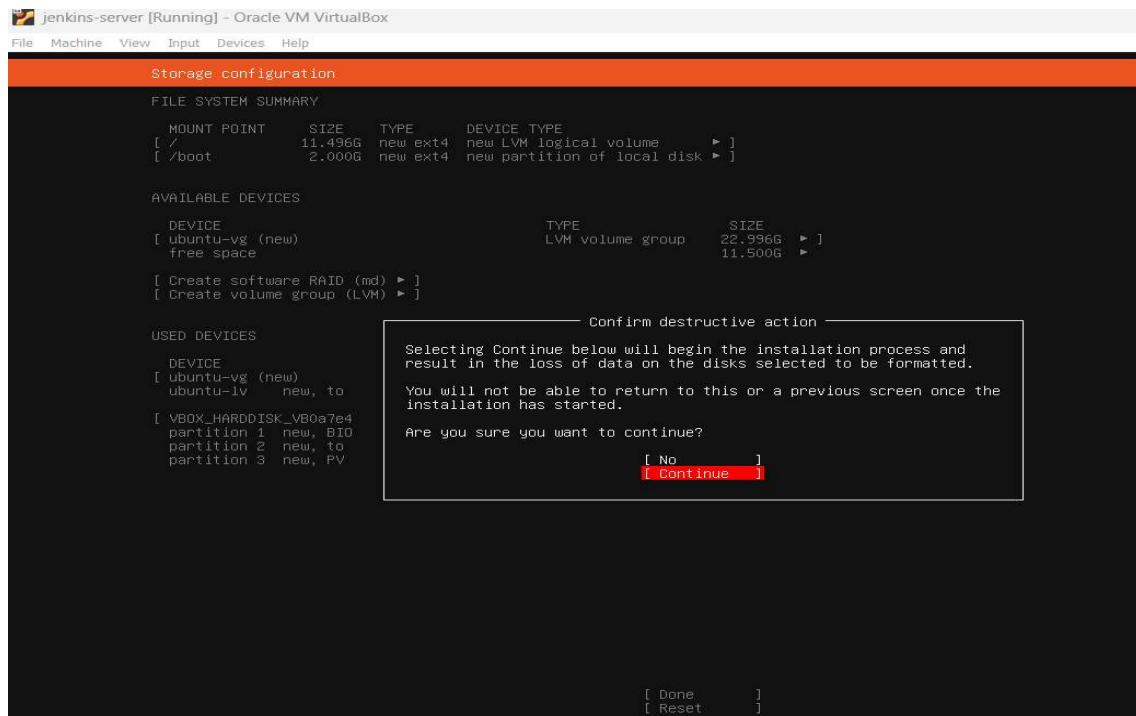
Leave the default Ubuntu archive mirror configuration and press enter on “Done” to proceed.



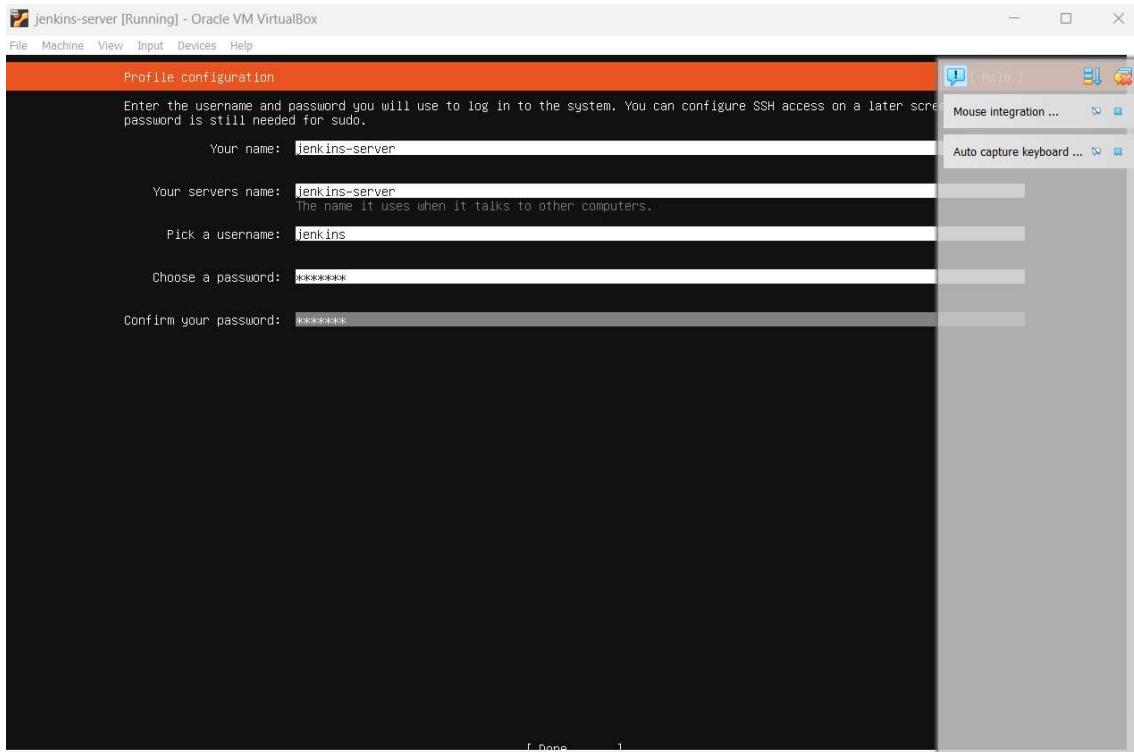
Leave the default storage configuration and press enter on “Done” to proceed.



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

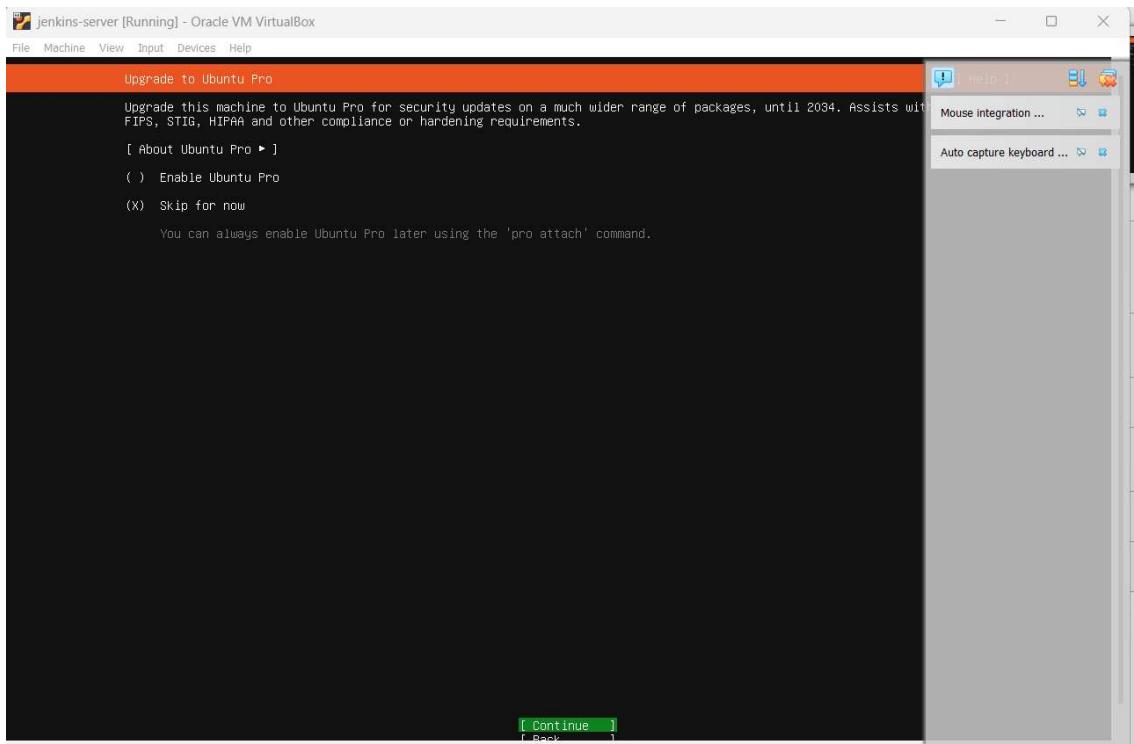


Click on Continue.

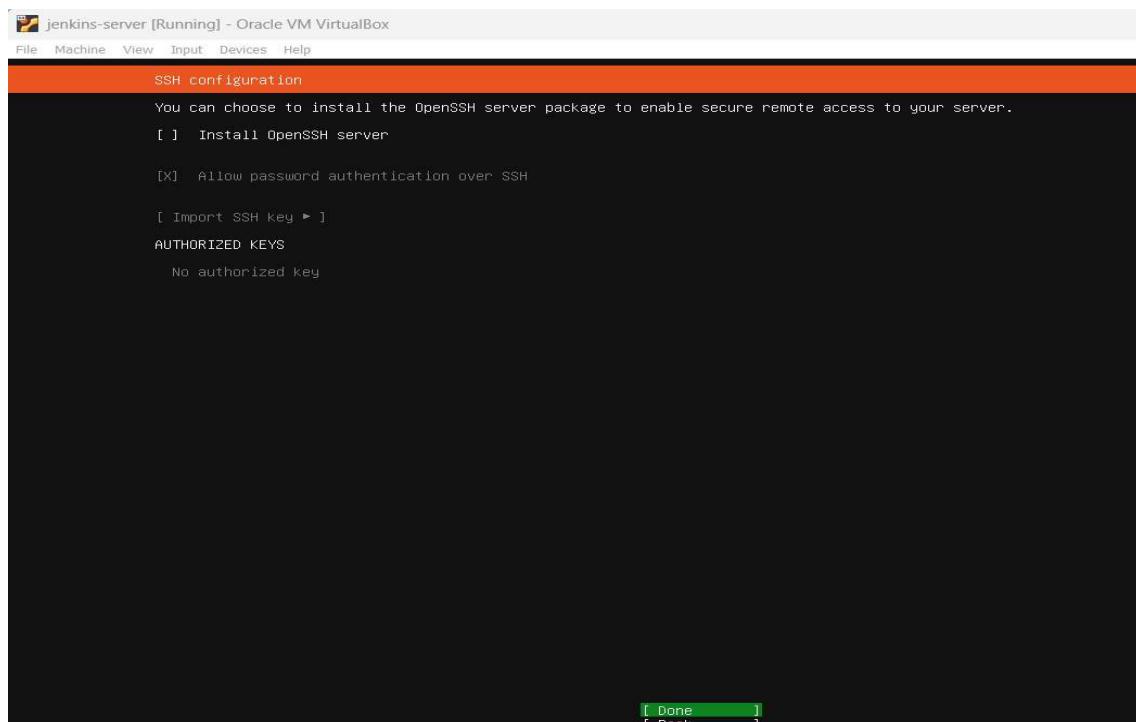


Put your name (any name), your server's name (I have put jenkins-minikube-server), give a username (I have put devops) and password.

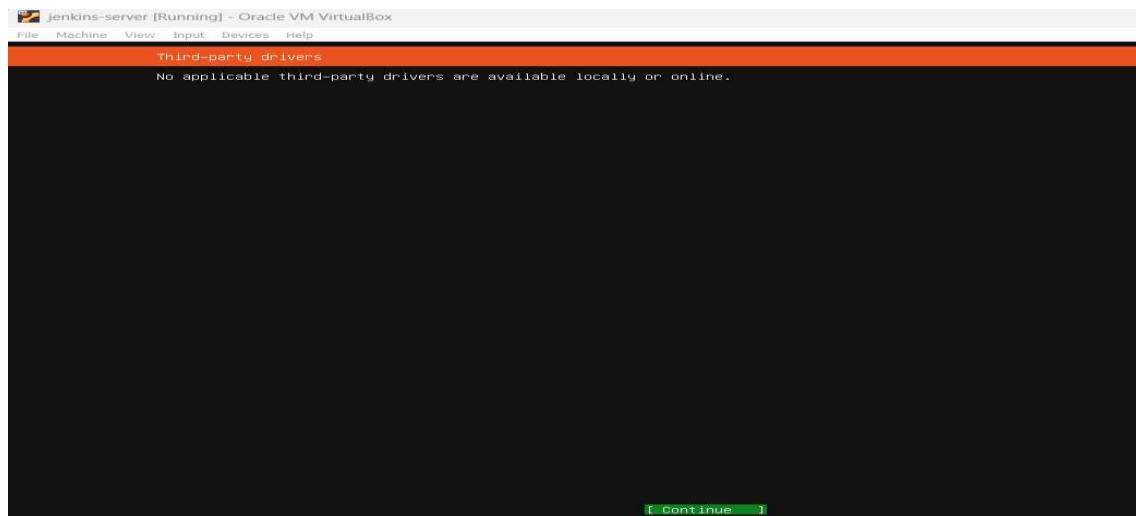
Press enter on “Done” to proceed.



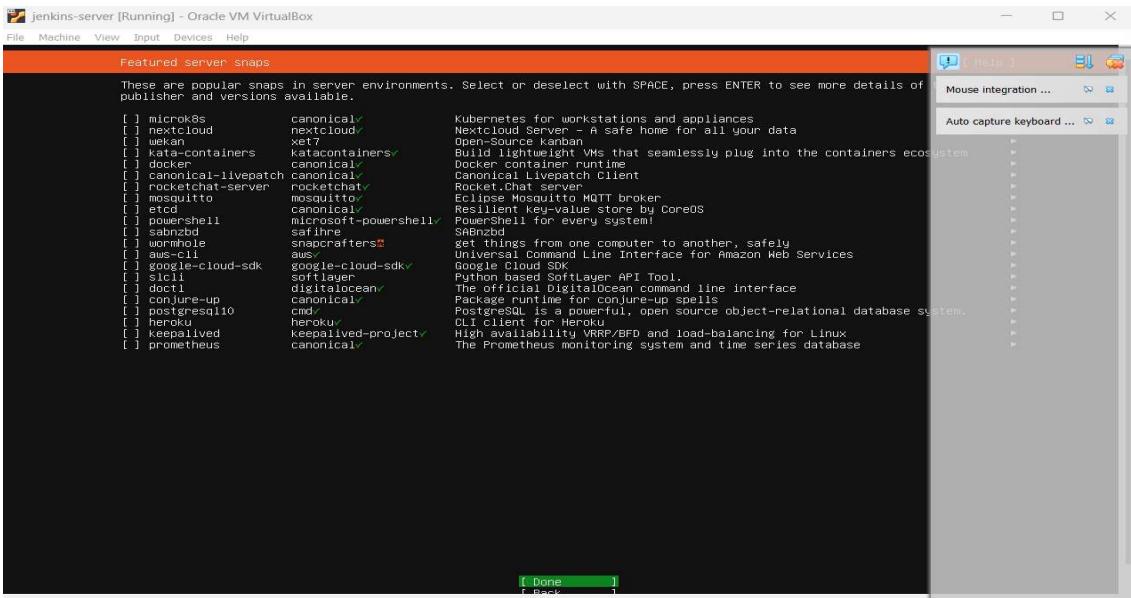
Select skip for now and press enter on “Continue” to proceed.



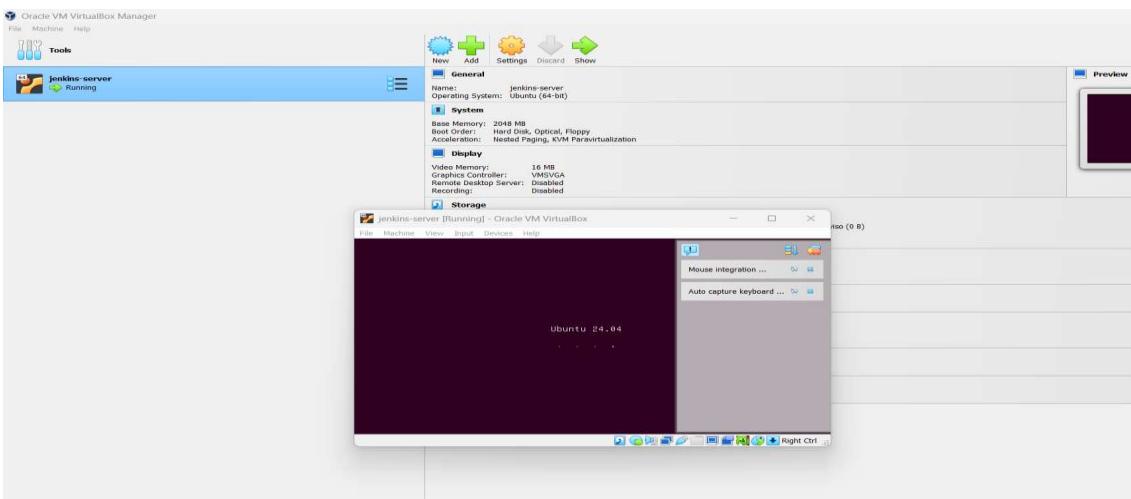
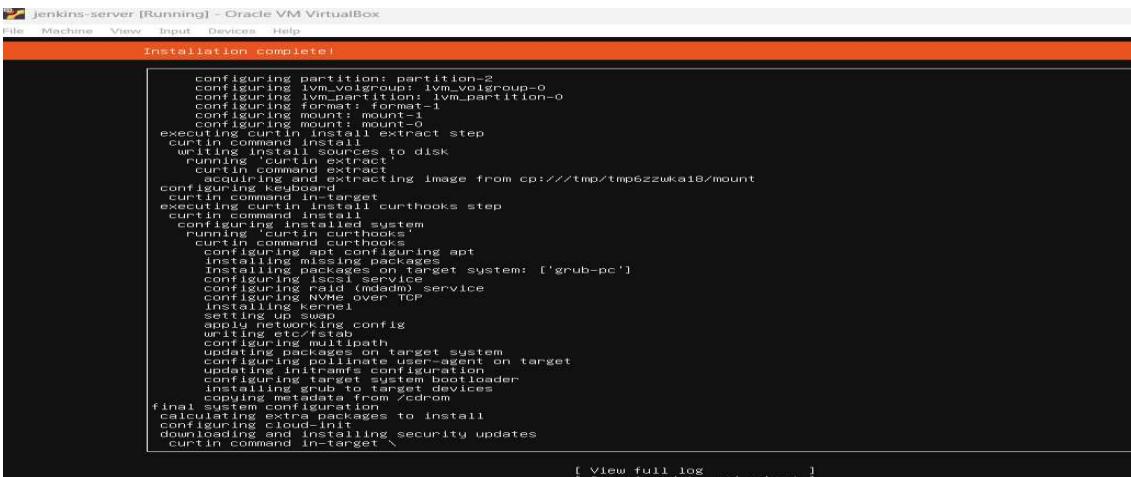
Leave the default option as black and press enter on “Done” to proceed.



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

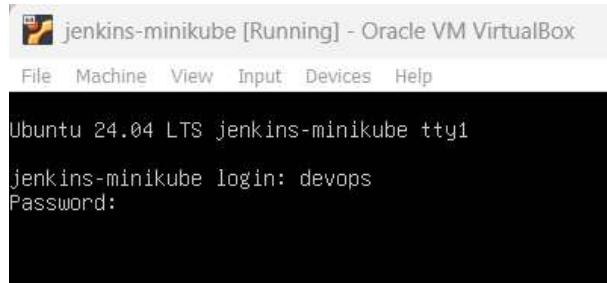


Leave all fields blank and press enter on “Done” to proceed.

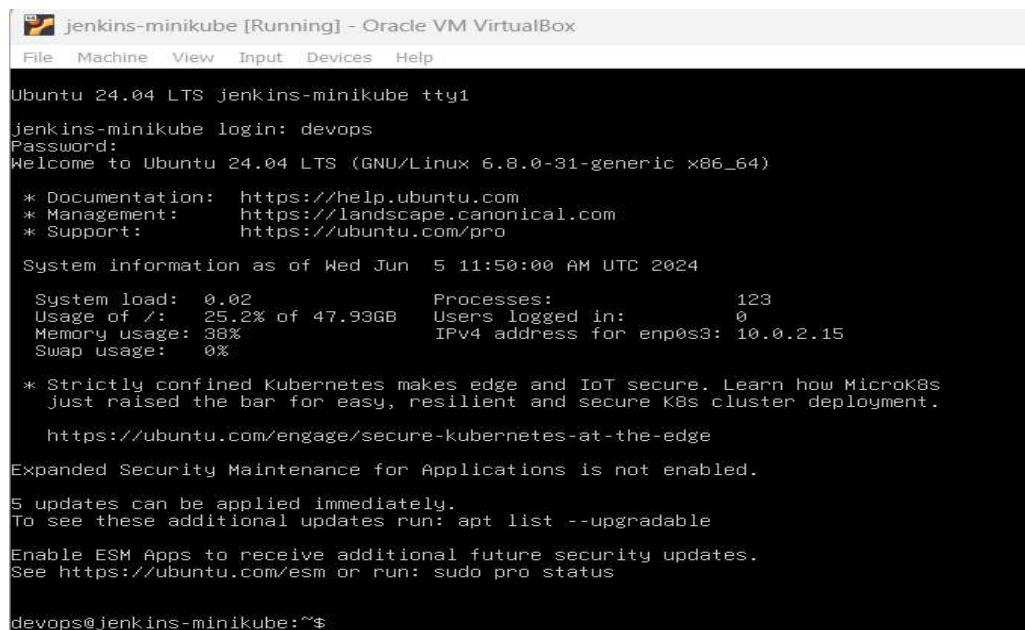


When you see installation complete, reboot the server.

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



```
jenkins-minikube [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Ubuntu 24.04 LTS jenkins-minikube tty1
jenkins-minikube login: devops
Password:
```



```
jenkins-minikube [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Ubuntu 24.04 LTS jenkins-minikube tty1
jenkins-minikube login: devops
Password:
Welcome to Ubuntu 24.04 LTS (GNU/Linux 6.8.0-31-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

System information as of Wed Jun  5 11:50:00 AM UTC 2024

 System load:  0.02      Processes:          123
 Usage of /:   25.2% of 47.93GB  Users logged in:     0
 Memory usage: 38%           IPv4 address for enp0s3: 10.0.2.15
 Swap usage:   0%           Swap:             0K

* Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
just raised the bar for easy, resilient and secure K8s cluster deployment.
  https://ubuntu.com/engage/secure-kubernetes-at-the-edge

Expanded Security Maintenance for Applications is not enabled.

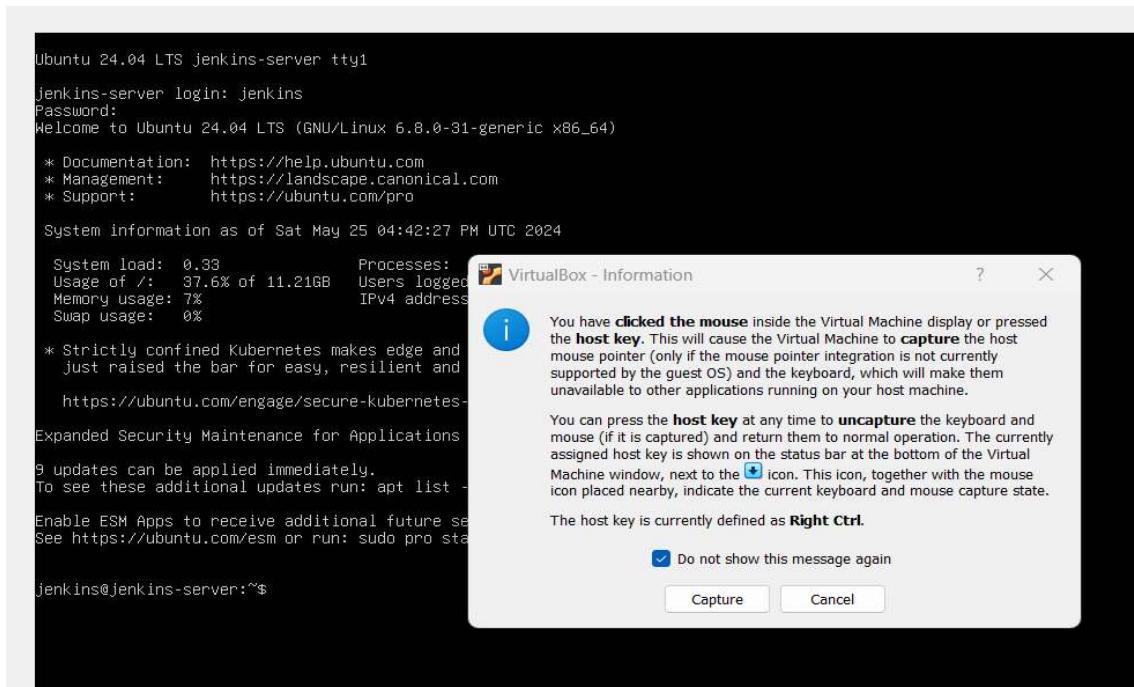
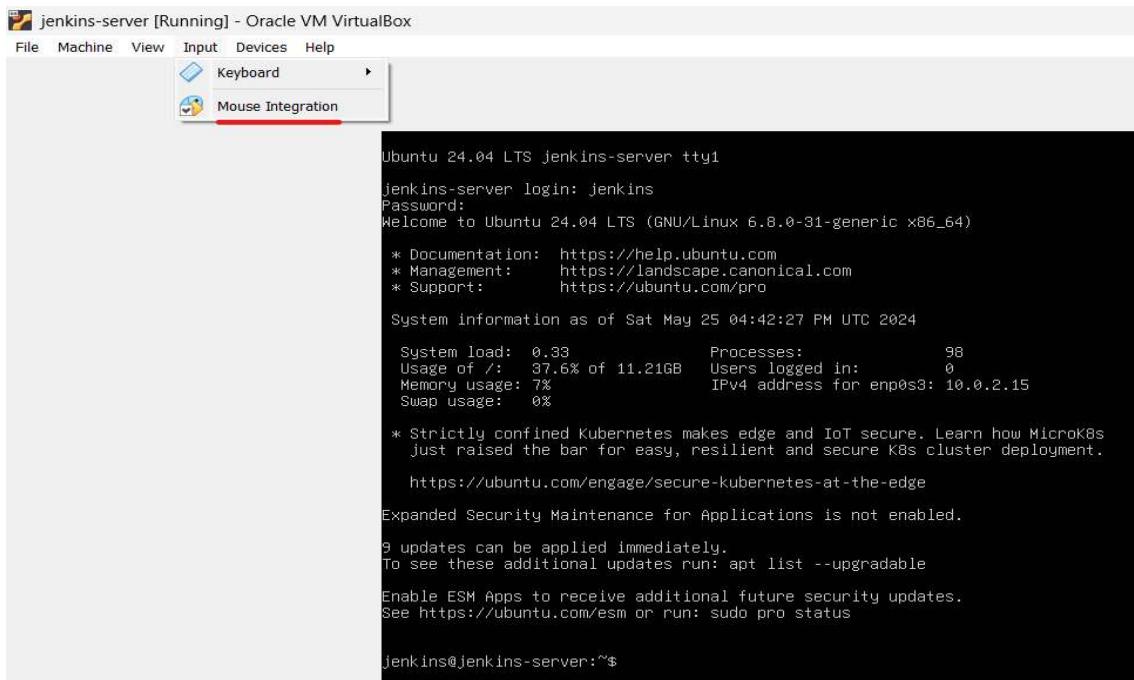
5 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

devops@jenkins-minikube:~$
```

VirtualBox mouse integration

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



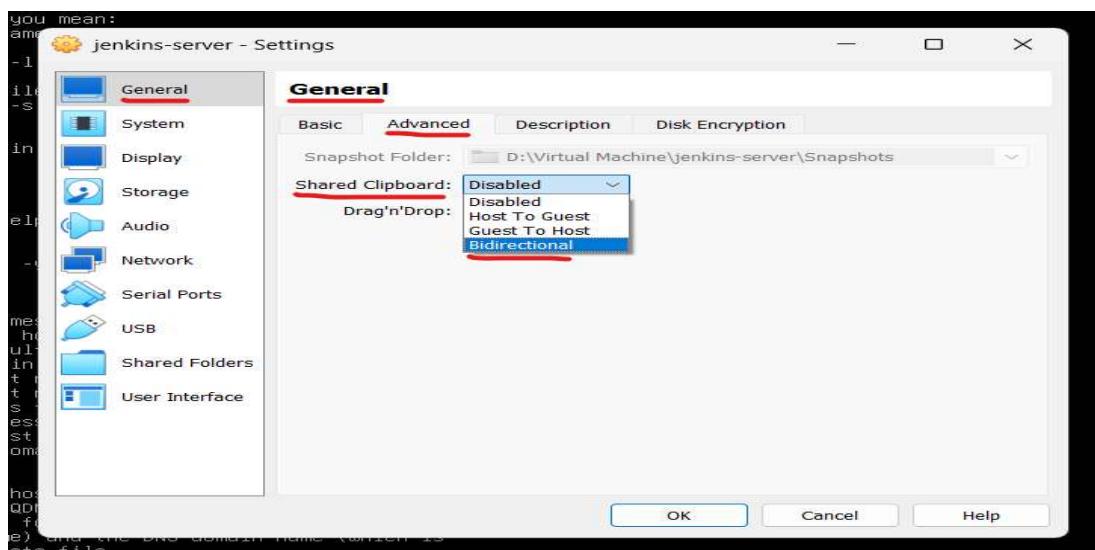
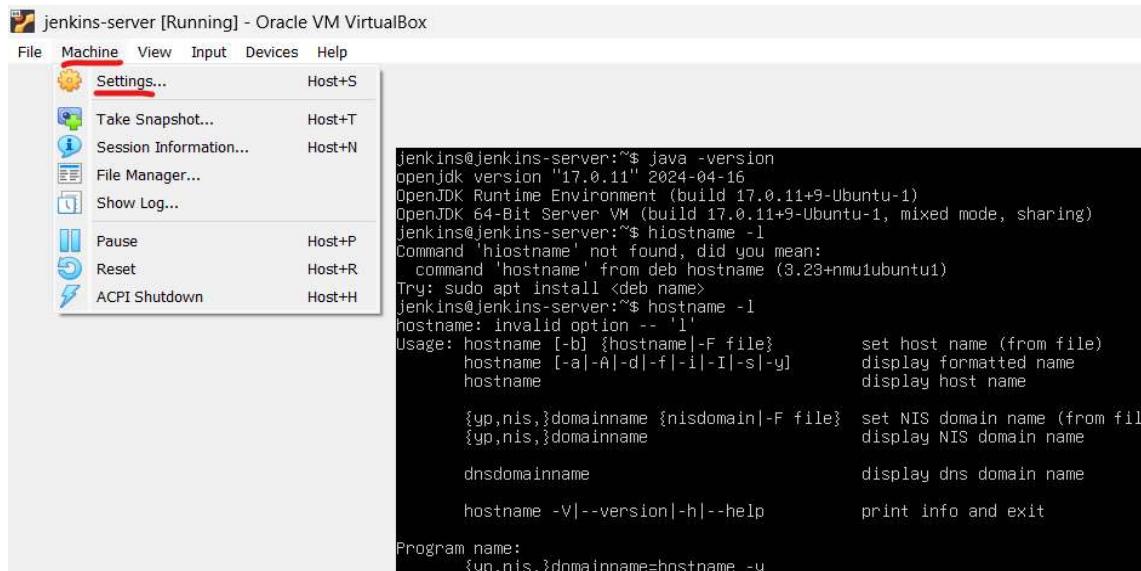
To come out the mouse cursor from guest (Ubuntu) back to the host (Windows), press CTRL from right hand side of keyboard.

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:

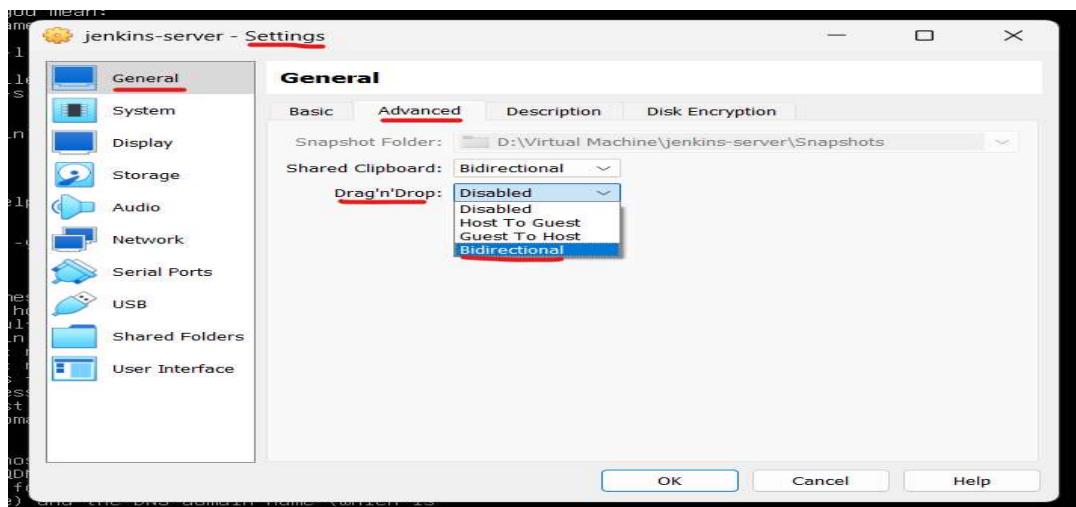
1. Launch VirtualBox.
2. Access the settings of the VM by navigating to the General section and selecting the Advanced tab.
3. Within the Advanced tab, you will find a setting to enable the “shared clipboard.” It offers three options: disabled, host to guest, and bidirectional. Opt for the bidirectional 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 192 kB in 9s (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
  libnotify0 libwayland-server0 libxcvtd0 libxfont2 virtualbox-guest-utils x11-xkb-utils x11-xserver-utils xcvt xfntns-base xfntns-encodings xfntns-utils
Suggested packages:
  lib-doc libx11-locale0 cpp-13-doc notification-daemon nckle cairo-5c xorg-docs-core xfs | xserver xfntns-100dpi | xfntns-75dpi xfntns-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
  libnotify0 libwayland-server0 libxcvtd0 libxfont2 virtualbox-guest-utils x11-xkb-utils x11-xserver-utils xcvt xfntns-base xfntns-encodings xfntns-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:

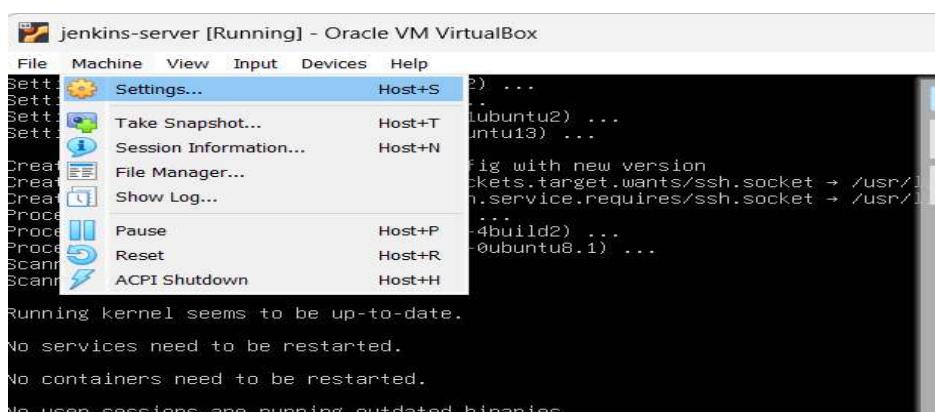
```
sudo systemctl start ssh
```

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

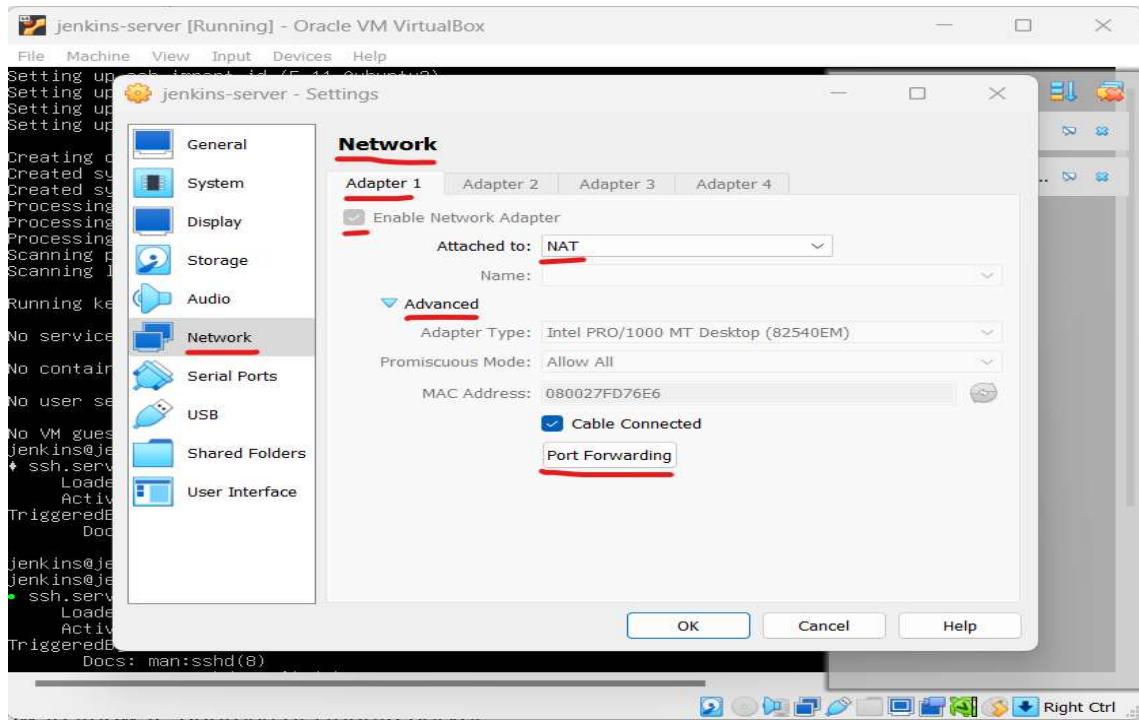
```
sudo systemctl status ssh
```

Enable port-forwarding

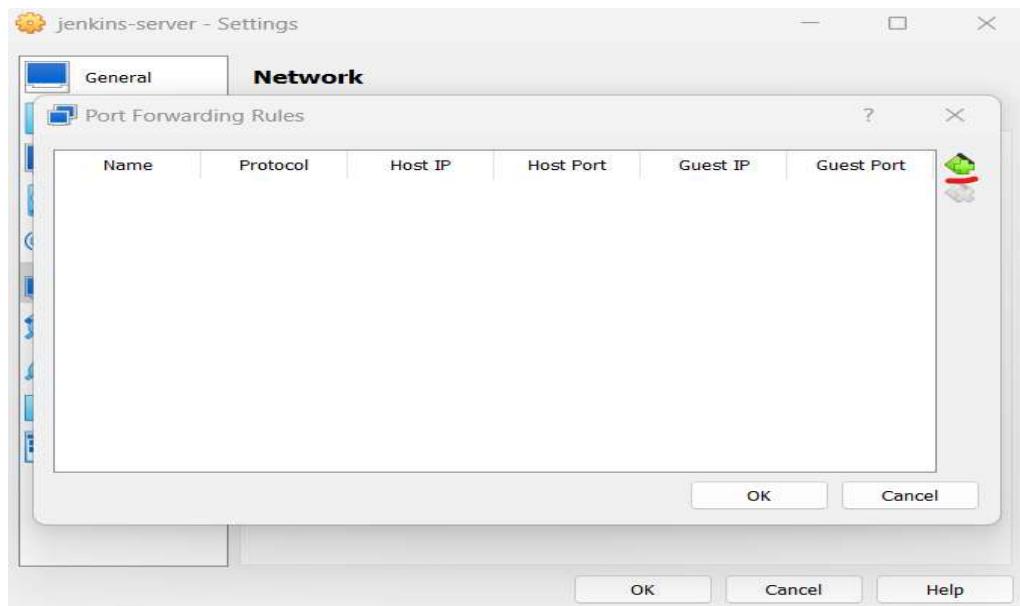
Open settings from guest OS → Machine → Settings...



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

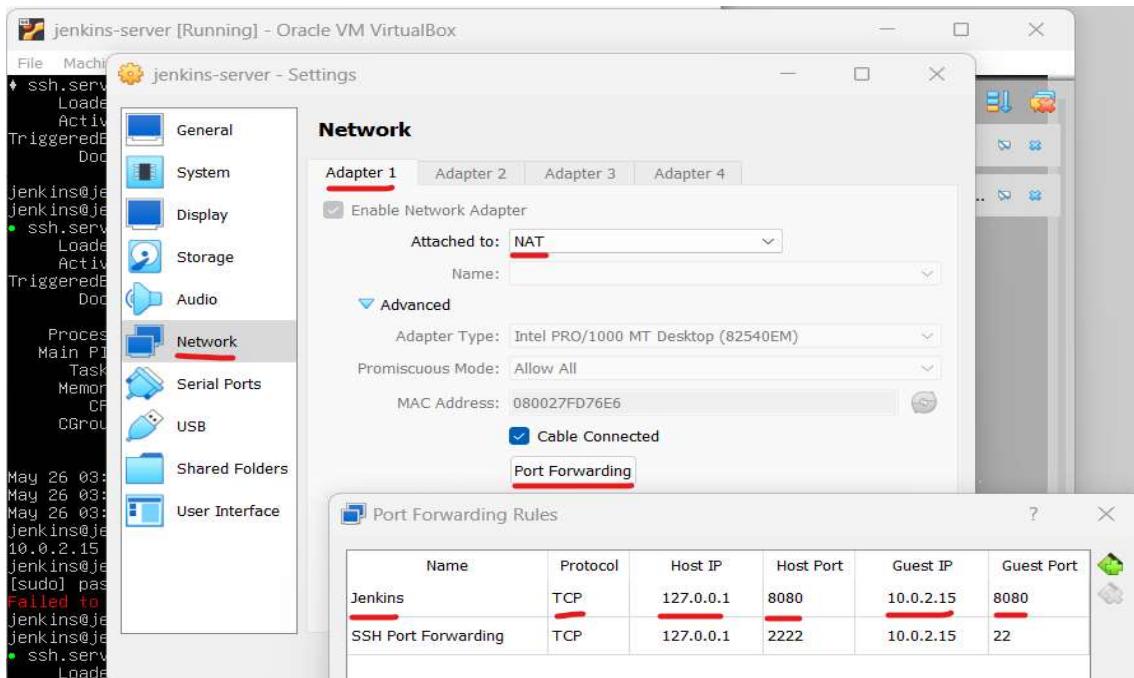


Click on Add sign as shown below:

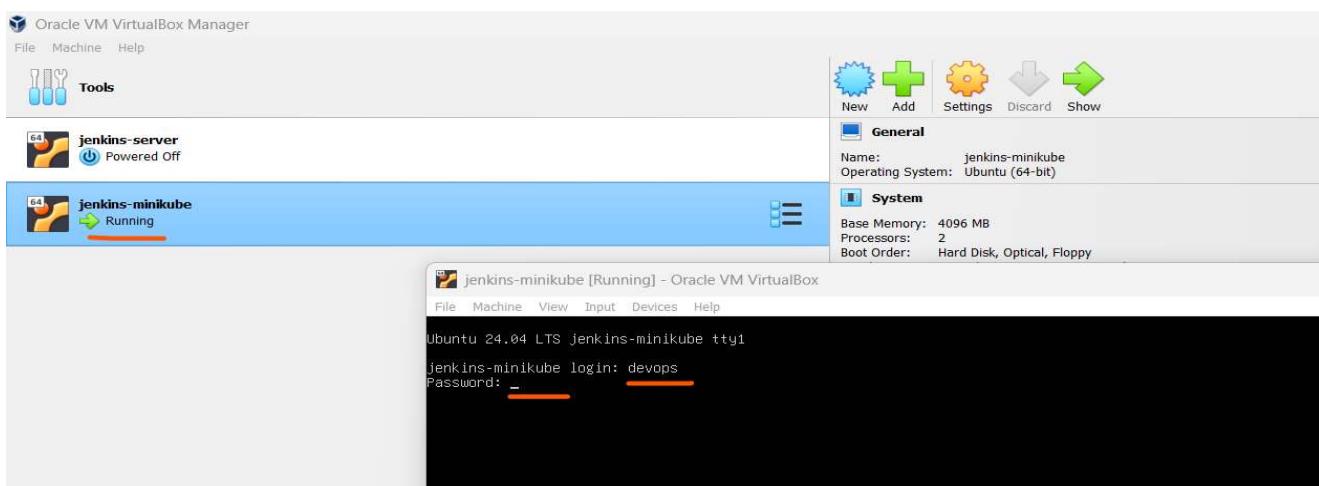


The real problem here is that the "Host IP" parameter in your VirtualBox's port forwarding rule only tells it to listen on 127.0.0.1 (loopback address). This means it will only accept connections made to 127.0.0.1

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)

```
devops@jenkins-minikube:~$ hostname -I
10.0.2.15 192.168.49.1 172.17.0.1
devops@jenkins-minikube:~$
```

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

Restart ssh server

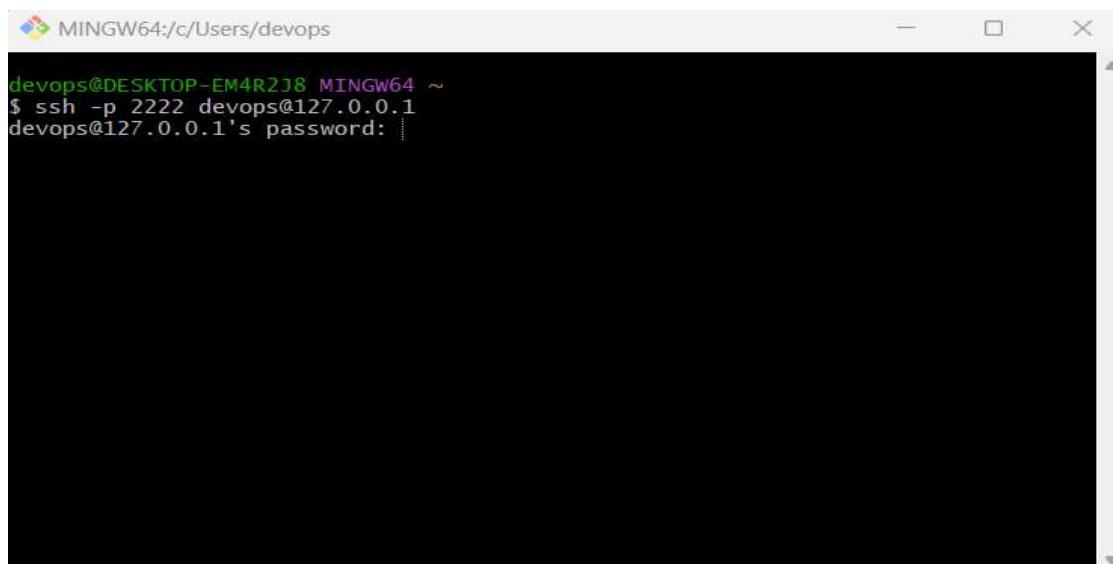
you might need to **restart ssh service** on VirtualBox for the changes to take effect.

```
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:~# 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`

```

root@docker-minikube-server:~# sudo apt install openjdk-17-jre
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  alsa-ucm-conf alsasound2-common libasound2-data libasound2t64 libatk-bridge2.0-0t64 libatk-wrapper-java libatk-wrapper-java-jni libatk1.0-0t64 libatspi2.0-0t64
  gtk-update-icon-cache hicolor-icon-themes humanity-icon-theme java-common libasound2t64 libatk-bridge2.0-0t64 libatk-wrapper-java libatk-wrapper-java-jni libatk1.0-0t64
  libavahi-client3 libavahi-common libavahi-common-data libcairo-gobject2 libcairo2 libcurl5 libdconf1 libdmz-andpu1 libdmz-nouveau2 libdmz-radeon1 libgail-common libgail18t64
  libgnome-menu2.0 libgnome-menu2.0-data libgnome2 libgnome2-common libglib2.68-0 libglib2.68-0-dbg libglib2.68-0-dbg-data libglib2.68-0-dbg-data libgnome2-0.0 libgnome2-0.0-data
  libgtk2.0-common libharfbuzz2b libice6 liblcms2-2.16 libl10n764 libpano-1.0-0 libpanoocairo-1.0-0 libpanofz2-1.0-0 libpixiaceess0 libpccs1t64 libpixman-1.0 librsvg2-2 librsvg2-common librsn6 libthai-data
  libthai0 libvulkanc libwayland-client0 libx11-xcb libxaw7 libxcb-dri3-0 libxcb-glx libxcb-present0 libxcb-randr0 libxcb-render0 libxcb-shm0 libxcb-sync1 libxcb-xfixes0
  libxcbcompositelibcursor1 libxdamage1 libxfixes3 libxft2 libxi6 libxinerama1 libxkbfile1 libxmu6 libxrandr2 libxrender1 libxshmfence1 libxt64 libxtst6 libxv1 libxf86dga1 libxf86vm1 mesa-vulkan-drivers
  openjdk-17-jre-headless session-migration ubuntu-mono x11-common x11-utils
Suggested packages:
  default-jre als-audio libasound2-plugins cups-common gvfs libcm2-utils pscsd librsvg2-bin libns-ndns fonts-ipafont-gothic fonts-ipafont-mincho fonts-way-microhei | fonts-way-zenhei fonts-indic
  mesa-utils
Recommended packages:
  luit
The following NEW packages will be installed:
  alsa-ucm-conf alsasound2-common libasound2-data libasound2t64 libatk-bridge2.0-0t64 libatk-wrapper-java libatk-wrapper-java-jni libatk1.0-0t64 libatspi2.0-0t64
  gtk-update-icon-cache hicolor-icon-themes humanity-icon-theme java-common libasound2t64 libatk-bridge2.0-0t64 libatk-wrapper-java libatk-wrapper-java-jni libatk1.0-0t64
  libavahi-client3 libavahi-common libavahi-common-data libcairo-gobject2 libcairo2 libcurl5 libdconf1 libdmz-andpu1 libdmz-nouveau2 libdmz-radeon1 libgail-common libgail18t64
  libgnome-menu2.0 libgnome-menu2.0-data libgnome2 libgnome2-common libglib2.68-0 libglib2.68-0-dbg libglib2.68-0-dbg-data libgnome2-0.0 libgnome2-0.0-data
  libgtk2.0-common libharfbuzz2b libice6 liblcms2-2.16 libl10n764 libpano-1.0-0 libpanoocairo-1.0-0 libpanofz2-1.0-0 libpixiaceess0 libpccs1t64 libpixman-1.0 librsvg2-2 librsvg2-common librsn6 libthai-data
  libthai0 libvulkanc libwayland-client0 libx11-xcb libxaw7 libxcb-dri3-0 libxcb-glx libxcb-present0 libxcb-randr0 libxcb-render0 libxcb-shm0 libxcb-sync1 libxcb-xfixes0
  libxcbcompositelibcursor1 libxdamage1 libxfixes3 libxft2 libxi6 libxinerama1 libxkbfile1 libxmu6 libxrandr2 libxrender1 libxshmfence1 libxt64 libxtst6 libxv1 libxf86dga1 libxf86vm1 mesa-vulkan-drivers
  openjdk-17-jre openjdk-17-jre-headless session-migration ubuntu-mono x11-common x11-utils
0 upgraded, 13 newly installed, 0 to remove and 3 not upgraded.
Need to get 113 MB of archives.
After this operation, 501 MB of additional disk space will be used.

```

`sudo apt install openjdk-17-jdk`

```

root@docker-minikube-server:~# sudo apt install openjdk-17-jdk
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libice-dev libpthread-stubs0-dev libsm-dev libxi-dev libxau-dev libxcb1-dev libxdmcp-dev libxt-dev openjdk-17-jdk-headless x11proto-dev xorg-sgml-doctools xtrans-dev
Suggested packages:
  libice-dev libpthread-stubs0-dev libsm-dev libxi-dev libxau-dev libxcb1-dev libxdmcp-dev libxt-dev openjdk-17-jdk-headless x11proto-dev xorg-sgml-doctools xtrans-dev
The following NEW packages will be installed:
  libice-dev libpthread-stubs0-dev libsm-dev libxi-dev libxau-dev libxcb1-dev libxdmcp-dev libxt-dev openjdk-17-jdk openjdk-17-jdk-headless x11proto-dev xorg-sgml-doctools xtrans-dev
0 upgraded, 13 newly installed, 0 to remove and 3 not upgraded.
Need to get 73.2 MB of archives.
After this operation, 85.8 MB of additional disk space will be used.
Do you want to Continue? [y/n] y

```

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
```

```
root@jenkins-server:~# sudo systemctl enable jenkins
Synchronizing state of jenkins.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Exception: /usr/lib/systemd/systemd-sysv-install is not available.
root@jenkins-server:~# sudo systemctl start jenkins
root@jenkins-server:~# sudo systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
   Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; preset: enabled)
     Active: active (running) since Sun 2024-05-26 03:14:31 UTC; 6min ago
       Main PID: 2300 (java)
          Tasks: 4 (limit: 276)
         Memory: 44.9M (limit: 863.1M)
            CPU: 1min 4.396s
           CGroup: /system.slice/jenkins.service
                   └─2300 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

May 26 03:13:44 jenkins-server jenkins[2300]: 7f9eef1d0134b7eb006fb73ab6c9
May 26 03:13:44 jenkins-server jenkins[2300]: This may also be found at: /var/lib/jenkins/secrets/initialAdminPassword
May 26 03:13:44 jenkins-server jenkins[2300]: ****
May 26 03:13:44 jenkins-server jenkins[2300]: ****
May 26 03:13:44 jenkins-server jenkins[2300]: ****
May 26 03:14:31 jenkins-server jenkins[2300]: 2024-05-26 03:14:31,282+0000 [id=31]      INFO  jenkins.InitiateOrRun@1#onAttained: Completed initialization
May 26 03:14:31 jenkins-server jenkins[2300]: 2024-05-26 03:14:31,282+0000 [id=31]      INFO  hudson.lifecycle.Lifecycle@1#onReady: Jenkins is fully up and running
May 26 03:14:32 jenkins-server systemd[1]: Started jenkins.service - Jenkins Continuous Integration Server.
May 26 03:14:32 jenkins-server jenkins[2300]: 2024-05-26 03:14:32,738+0000 [id=47]      INFO  h.m.DownloadService$Downloadable#load: Obtained the updated data file for hudson.tasks.Maven.MavenInstaller
May 26 03:14:32 jenkins-server jenkins[2300]: 2024-05-26 03:14:32,746+0000 [id=47]      INFO  hudson.util.Retriger#start: Performed the action check updates server successfully at the attempt #1
root@jenkins-server:~#
```

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

```
sudo ufw status
```

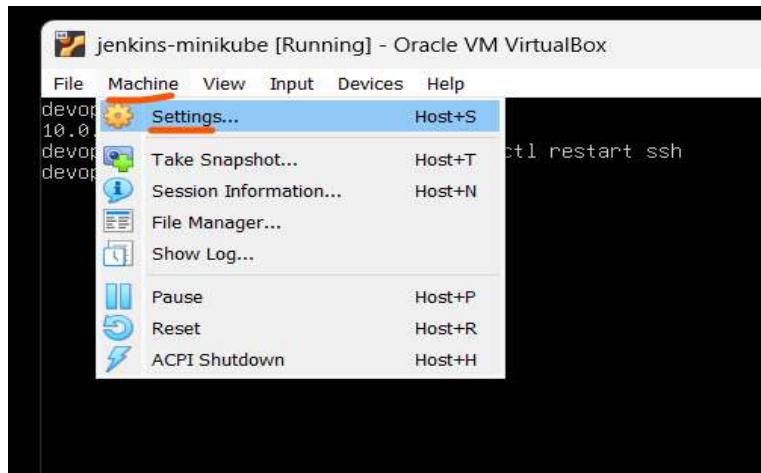
```
sudo ufw disable
```

```
jenkins@jenkins-server:~$ sudo ufw status
Status: inactive
jenkins@jenkins-server:~$ sudo ufw disable
Firewall stopped and disabled on system startup
jenkins@jenkins-server:~$ |
```

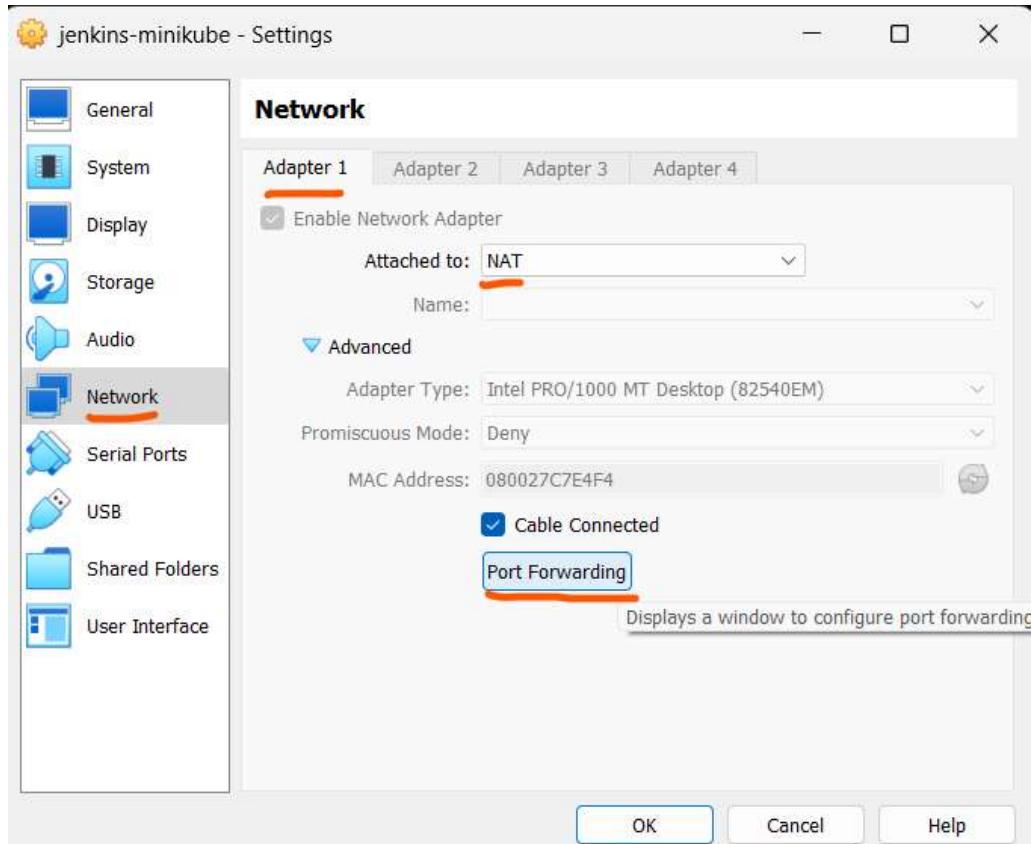
Set port forwarding for Jenkins (guest / Ubuntu to host / Windows)

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

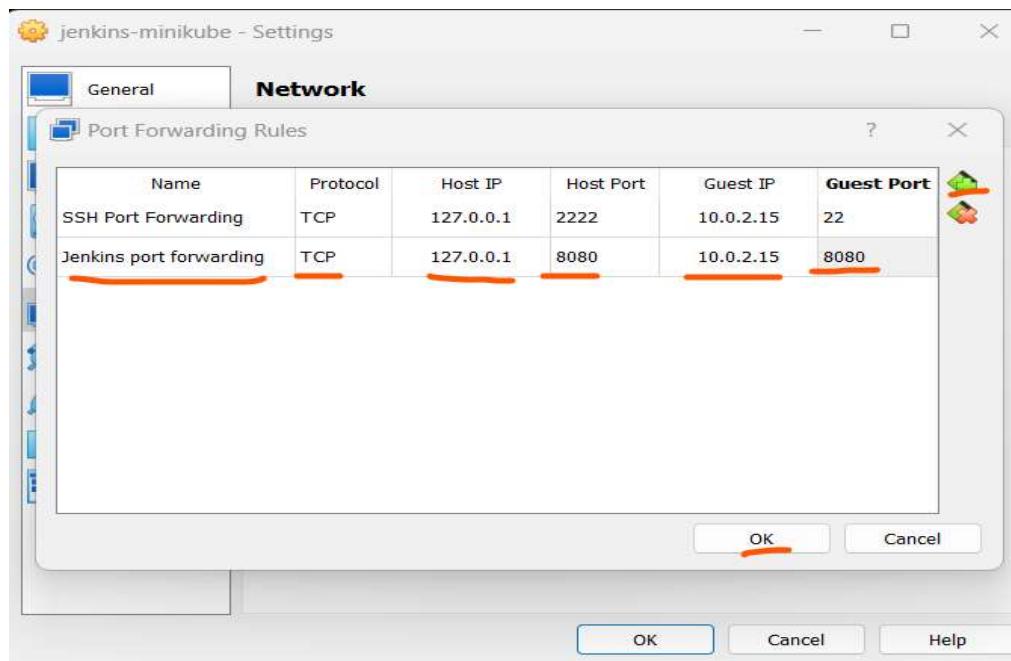


Click on network → Adapter 1 → NAT → click on Port Forwarding



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.

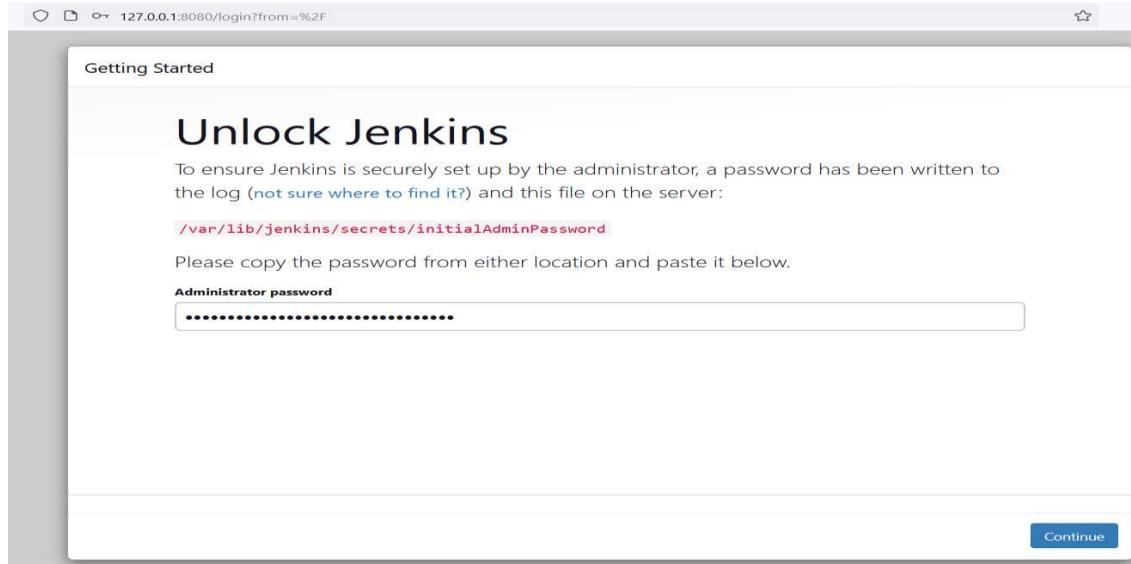
To ensure Jenkins is securely set up by the administrator, a password has been written to the log:

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

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

```
jenkins@jenkins-server: ~
jenkins@jenkins-server:~$ cat /var/lib/jenkins/secrets/initialAdminPassword
7fc9eef1d01[REDACTED]006fb73[REDACTED]
jenkins@jenkins-server:~$ |
```

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



Click on Continue.

Customize Jenkins

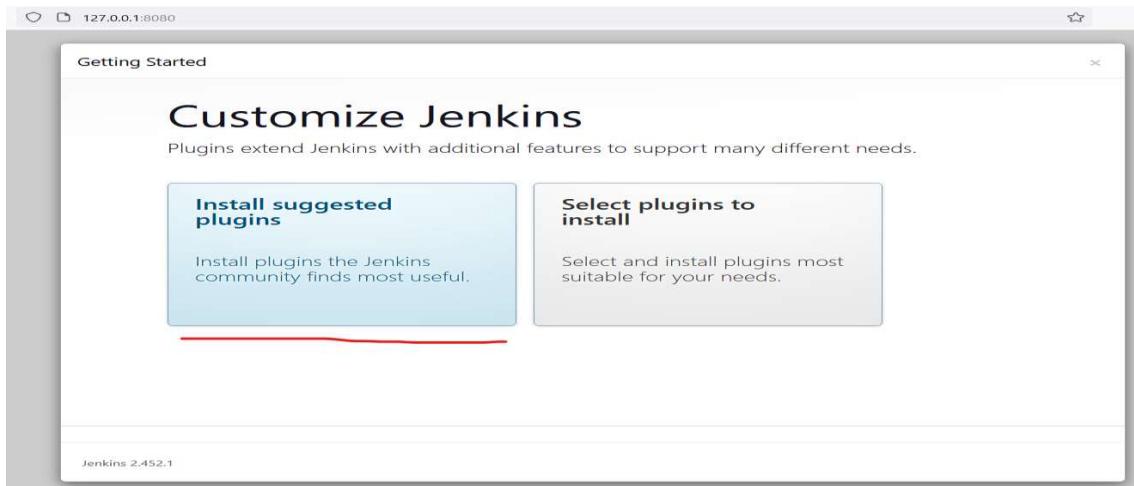
Initial Jenkins configuration

First time installation – default of set plugins.

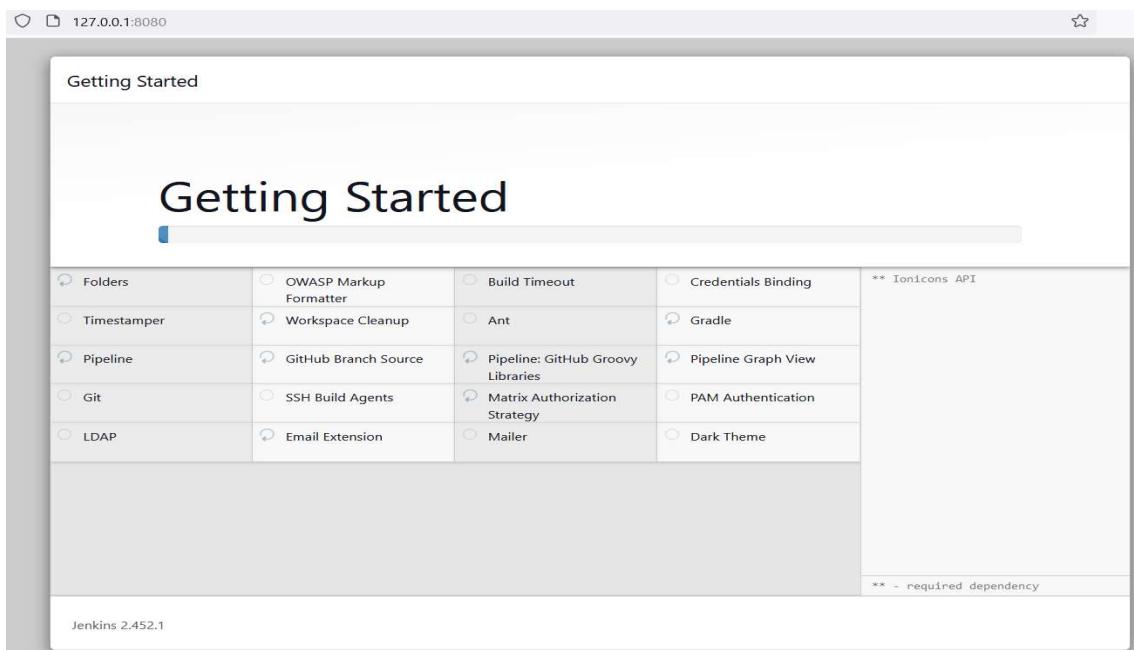
Plugins extend Jenkins with additional features to support many different needs.

For first time, you will get these options.

Please select – “Install suggested plugins”



You will be 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

Getting Started

Create First Admin User

Username

Password

Confirm password

Full name

E-mail address

Jenkins 2.452.1 [Skip and continue as admin](#) [Save and Continue](#)



Instance Configuration

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

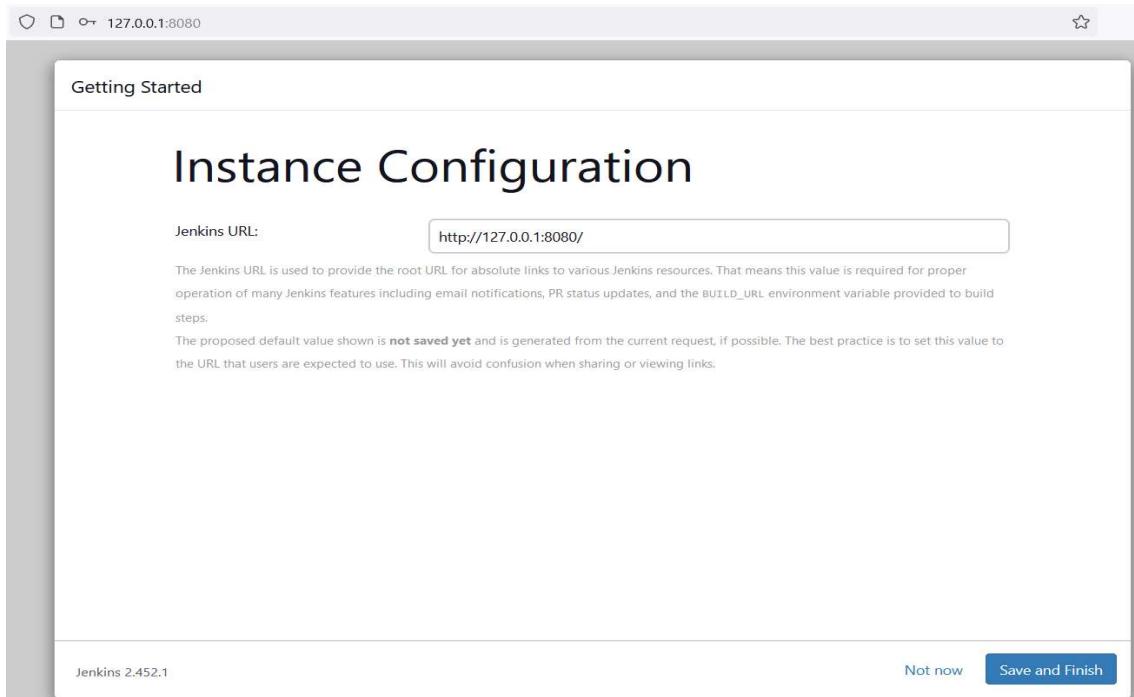
Getting Started

Instance Configuration

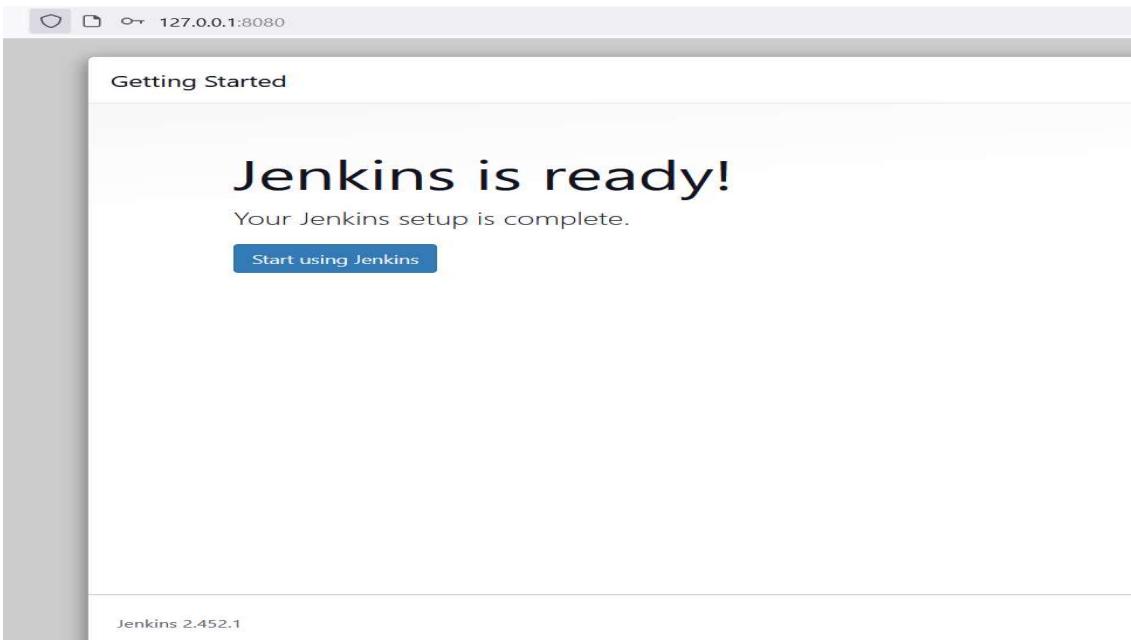
Jenkins URL:

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the `BUILD_URL` environment variable provided to build steps.
The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

Jenkins 2.452.1 [Not now](#) [Save and Finish](#)



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



You will get the Jenkins home page like below:

A screenshot of the Jenkins home page. The URL in the address bar is 127.0.0.1:8080. The page title is 'Jenkins'. The main content includes a 'Welcome to Jenkins!' message, a 'Start building your software project' call-to-action, and sections for 'Build Queue' (empty), 'Build Executor Status' (2 Idle), and 'Set up a distributed build' (links to 'Create a job', 'Set up an agent', 'Configure a cloud', and 'Learn more about distributed builds'). The footer links to 'REST API' and 'Jenkins 2.452.1'.

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.

The screenshot shows the Jenkins Plugin Manager interface. The search bar at the top contains the text 'docker'. Below the search bar, there is a list of available plugins. The first few items in the list are:

- Docker Pipeline 5.0.0.v0c0340660b, 54 releases ago. Includes Pipeline, DevOps, Deployment, docker. Description: Build and use Docker containers from pipelines. Status: Released, last updated 5 days 15 hr ago.
- Docker API 3.5.4-06.v591_a_5e7e342c, 6 months 0 days ago. Includes Library plugin (for use by other plugins), docker. Description: This plugin provides docker-Java API for other plugins. Status: Released, last updated 6 months 0 days ago. A note says 'This plugin is up for adoption! We are looking for new maintainers. Visit our [Adopt a Plugin](#) initiative for more information.'
- Docker 1.6.1, 14 days ago. Includes Cloud Providers, Cluster Management, docker. Description: This plugin integrates Jenkins with Docker. Status: Released, last updated 14 days ago.
- docker-build-step 2.11, 4 months 24 days ago. Includes Build Tools, docker. Description: This plugin allows to add various docker commands to your job as build steps. A warning message says 'Warning: This plugin version may not be safe to use. Please review the following security notices: + CSRF vulnerability and missing permission check.' Status: Released, last updated 4 months 24 days ago.
- CloudBees Docker Build and Publish 1.4.0, 1 year 9 months ago. Includes Build Tools, docker. Description: This plugin enables building Dockerfile based projects, as well as publishing of the built images/repos to the docker registry. Status: Released, last updated 1 year 9 months ago.

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

The screenshot shows the Jenkins Plugins page with the 'Download progress' tab selected. On the left, there is a sidebar with links: Updates, Available plugins, Installed plugins, Advanced settings, and Download progress. The main area is titled 'Download progress' and shows a table of Jenkins plugins and their current download status. Most entries show 'Pending' with a blue circular progress indicator.

Plugin	Status
Docker Pipeline	Pending
Apache HttpComponents Client 5.x API	Pending
Docker API	Pending
Cloud Statistics	Pending
Docker	Pending
Javadoc	Pending
JSch dependency	Pending
Maven Integration	Pending
docker-build-step	Pending
CloudBees Docker Build and Publish	Pending
Docker Compose Build Step	Pending
Docker Slaves	Pending
CloudBees Docker Custom Build Environment	Pending
CloudBees Docker Hub/Registry Notification	Pending
Image Tag Parameter	Pending
Docker Swarm	Pending
CloudBees Docker Traceability	Pending
Loading plugin extensions	Pending

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.

The screenshot shows the Jenkins 'Available plugins' page with a search bar containing 'kubernetes'. A blue 'Install' button is highlighted at the top right of the search results table. The table lists several Jenkins plugins related to Kubernetes:

Install	Name	Released
<input checked="" type="checkbox"/>	Kubernetes Client API 6.10.0-240.v57800e80_0n_2	4 mo 2 days ago
<input checked="" type="checkbox"/>	Kubernetes - Library plugin (for use by other plugins)	
<input checked="" type="checkbox"/>	Kubernetes Credentials 173.v04ed9c17md7	5 days 2 hr ago
<input checked="" type="checkbox"/>	Kubernetes - credentials	
<input checked="" type="checkbox"/>	Kubernetes - 4233.vb_67a_0e11a_099	4 days 5 hr ago
<input checked="" type="checkbox"/>	Cloud Providers - ClusterManagement - kubernetes - Agent Management	
<input checked="" type="checkbox"/>	Kubernetes CLI 1.12.1	9 mo 1 day ago
<input checked="" type="checkbox"/>	kubernetes - Configure kubectl for Kubernetes	
<input checked="" type="checkbox"/>	Kubernetes Credentials Provider 1.262.v2670effea_c5	2 mo 28 days ago
<input checked="" type="checkbox"/>	kubernetes - credentials	
<input checked="" type="checkbox"/>	Kubernetes :: Pipeline :: DevOps Steps 1.6	5 yr 4 mo ago
<input checked="" type="checkbox"/>	pipeline - kubernetes	
<input type="checkbox"/>	GitLab Credentials - Kubernetes Integration 259.v7cf2899bd930	3 mo 5 days ago
	gitlab - gitlab	

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

The screenshot shows the Jenkins 'Download progress' page. The left sidebar has 'Download progress' selected. The main area displays the 'Preparation' section with a list of installed Kubernetes-related plugins: Kubernetes Client API, Kubernetes Credentials, Kubernetes, Kubernetes CLI, Kubernetes Credentials Provider, and Kubernetes :: Pipeline :: DevOps Steps. To the right, a 'Download progress' chart shows the status of each plugin's download: most are marked as 'Success' with green checkmarks, while one is 'Running'.

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.

```
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
Ign:5 https://pkg.jenkins.io/debian-stable binary/ InRelease
Hit:6 https://pkg.jenkins.io/debian-stable binary/ Release
Fetched 89.7 kB in 5s (19.9 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
0 packages can be upgraded. Run 'apt list --upgradable' to see them.
jenkins@jenkins-server:~$ |
```

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.

```
jenkins@jenkins-server:~$ sudo apt install apt-transport-https ca-certificates curl software-properties-common -y
[sudo] password for jenkins:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
apt-transport-https 2.7.14build2
ca-certificates 0.980.1-0ubuntu10.1
curl 8.5.0-0ubuntu10.1
software-properties-common 0.99.48
0 upgraded, 1 newly installed, 0 to remove and 10 not upgraded.
Need to get 3,974 B of archives.
After this operation, 3,784 B of additional disk space will be used.
Get:1 http://in.archive.ubuntu.com/ubuntu noble/universe amd64 apt-transport-https all 2.7.14build2 [3,974 B]
Preparing to unpack .../apt-transport-https_2.7.14build2_all.deb ...
Unpacking apt-transport-https (2.7.14build2) ...
Selecting previously unselected package apt-transport-https.
Scanning for hardware changes...
Scanning /var/lib/dkms...
Running kernel seems to be up-to-date.
No services need to be restarted.
No containers need to be restarted.
No user sessions are running outdated binaries.
No VM guests are running outdated hypervisor (QEMU) binaries on this host.
jenkins@jenkins-server:~$ |
```

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.

```
jenkins@jenkins-server: ~
jenkins@jenkins-server:~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
Warning: apt-key is deprecated. Manage keyring files in trusted.gpg.d instead (see apt-key(8)).
OK
jenkins@jenkins-server:~$
```

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.

```
jenkins@jenkins-server: ~
jenkins@jenkins-server:~$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
Reading package lists...
Adding repository.
Press [ENTER] to continue or Ctrl+C to cancel.
Adding deb entry to /etc/apt/sources.list.d/archive.uri-https_download_docker_com_linux_ubuntu-noble.list
Adding disabled deb-src entry to /etc/apt/sources.list.d/archive.uri-https_download_docker_com_linux_ubuntu-noble.list
Get:1 http://archive.ubuntu.com/ubuntu/noble InRelease
Ign:2 https://pkgs.jenkins.io/debian-stable binary/ InRelease
Hit:3 http://in.archive.ubuntu.com/ubuntu/noble InRelease
Hit:4 http://archive.ubuntu.com/ubuntu/noble-security InRelease
Get:5 http://security.ubuntu.com/ubuntu/noble-security InRelease
Get:6 http://archive.ubuntu.com/ubuntu/noble-updates InRelease
Get:7 http://archive.ubuntu.com/ubuntu/noble-updates InRelease [89.7 kB]
Get:8 https://download.docker.com/linux/ubuntu/noble/stable amd64 Packages [6,952 B]
Get:9 https://download.docker.com/linux/ubuntu/noble/repositories/main amd64 Packages [10.1 kB]
Fetched 258 kB in 5s (50.5 kB/s)
Reading package lists...
W: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: Key is stored in legacy trusted.gpg keyring (/etc/apt/trusted.gpg). see the DEPRECATION section in apt-key(8) for details.
jenkins@jenkins-server:~$
```

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.

```
jenkins@jenkins-server: ~$ apt-cache policy docker-ce
docker-ce:
  Installed: (none)
  Candidate: 5:26.1.3-1~ubuntu.24.04~noble
  Version table:
    5:26.1.3-1~ubuntu.24.04~noble 500
      500 https://download.docker.com/linux/ubuntu/noble/stable amd64 Packages
    5:26.1.2-1~ubuntu.24.04~noble 500
      500 https://download.docker.com/linux/ubuntu/noble/stable amd64 Packages
    5:26.1.1-1~ubuntu.24.04~noble 500
      500 https://download.docker.com/linux/ubuntu/noble/stable amd64 Packages
    5:26.1.0-1~ubuntu.24.04~noble 500
      500 https://download.docker.com/linux/ubuntu/noble/stable amd64 Packages
    5:26.0.2-1~ubuntu.24.04~noble 500
      500 https://download.docker.com/linux/ubuntu/noble/stable amd64 Packages
    5:26.0.1-1~ubuntu.24.04~noble 500
      500 https://download.docker.com/linux/ubuntu/noble/stable amd64 Packages
    5:26.0.0-1~ubuntu.24.04~noble 500
      500 https://download.docker.com/linux/ubuntu/noble/stable amd64 Packages
jenkins@jenkins-server: ~$
```

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.

```
jenkins@jenkins-server: ~$ sudo apt install docker-ce -y
Reading package lists... Done
Building dependency tree... Done
Reading status information... Done
The following additional packages will be installed:
  containerd.io docker-buildx-plugin docker-ce-cli docker-ce-rootless-extras docker-compose-plugin libltdl7 libsllrp0 pigz slirp4netns
Suggested packages:
  aufs-tools cgroupfs-mount coruport-lite
The following NEW packages will be installed:
  containerd.io docker-buildx-plugin docker-ce docker-ce-rootless-extras docker-compose-plugin libltdl7 libsllrp0 pigz slirp4netns
0 upgraded, 10 newly installed, 0 to remove and 10 not upgraded.
Need to get 122 MB of archives.
After this operation, 434 MB of additional disk space will be used.
Get:1 https://download.docker.com/linux/ubuntu/noble/stable amd64 containerd.io amd64 1:1.6.32-1 [30.0 MB]
Get:2 https://in.archive.ubuntu.com/ubuntu/noble/universe amd64 libltdl7 amd64 2.4.1-2~build1 [40.3 kB]
Get:3 https://in.archive.ubuntu.com/ubuntu/noble/main amd64 libslirp0 amd64 4.7.0-1ubuntu1 [63.8 kB]
Get:4 https://download.docker.com/linux/ubuntu/noble/stable amd64 docker-buildx-plugin amd64 0.14.0-1~ubuntu.24.04~noble [29.7 MB]
Get:5 https://in.archive.ubuntu.com/ubuntu/noble/universe amd64 slirp4netns amd64 1.2.1-1build1 [34.9 kB]
Get:6 https://in.archive.ubuntu.com/ubuntu/noble/universe amd64 slirp4netns amd64 1.2.1-2~build3-3 [24.04~noble [14.6 MB]
Get:7 https://download.docker.com/linux/ubuntu/noble/stable amd64 docker-ce amd64 5:26.1.3-1~ubuntu.24.04~noble [25.3 kB]
Get:8 https://download.docker.com/linux/ubuntu/noble/stable amd64 docker-ce-rootless-extras amd64 5:26.1.3-1~ubuntu.24.04~noble [9.319 kB]
Get:9 https://download.docker.com/linux/ubuntu/noble/stable amd64 docker-compose-plugin amd64 2.27.0-1~ubuntu.24.04~noble [12.5 MB]
Get:10 https://download.docker.com/linux/ubuntu/noble/stable amd64 docker-compose-plugin amd64 2.27.0-1~ubuntu.24.04~noble [12.5 MB]
Fetched 122 MB in 10s (12.1 MB/s)
Selecting previously unselected package pigz.
(Reading database... 95882 files and directories currently installed.)
Preparing to unpack .../0-pigz_2.8-1_amd64.deb ...
Unpacking pigz (2.8-1) ...
Selecting previously unselected package containerd.io.
Preparing to unpack .../1-containerd.io_1.6.32-1_amd64.deb ...
Unpacking containerd.io (1:1.6.32-1) ...
```

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:

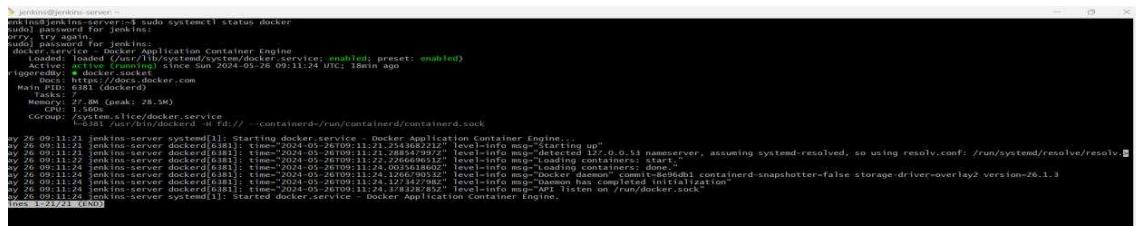
```
sudo systemctl start docker
```

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.



```
[root@jenkins-server ~]# sudo systemctl status docker
[sudo] password for jenkins:
[sudo] password for jenkins:
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; preset: enabled)
   Active: active (running) since Sun 2024-05-26 09:11:24 UTC; 1min ago
     Tasks: 27 (limit: 28311)
   Memory: 27.8M (peak: 28.3M)
      CPU: 3.968s
     CGroup: /docker.service
             └─=381 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

May 26 09:11:21 jenkins-server systemd[1]: Starting docker.service - Docker Application Container Engine...
May 26 09:11:21 jenkins-server dockerd[381]: time="2024-05-26T09:11:21.288547997Z" level=info msg="Starting up"
May 26 09:11:21 jenkins-server dockerd[381]: time="2024-05-26T09:11:21.288547997Z" level=info msg="detected /etc/resolv.conf, assuming system-resolved, so using resolv.conf: /run/systemd/resolve/resolv.conf"
May 26 09:11:21 jenkins-server dockerd[381]: time="2024-05-26T09:11:21.288547997Z" level=info msg="loading containerd containers start"
May 26 09:11:24 jenkins-server dockerd[381]: time="2024-05-26T09:11:24.003561860Z" level=info msg="Loading containers: done."
May 26 09:11:24 jenkins-server dockerd[381]: time="2024-05-26T09:11:24.003561860Z" level=info msg="Containerd pid=381 containerd-snapshottor=false storage-driver=overlay2 version=26.1.3"
May 26 09:11:24 jenkins-server dockerd[381]: time="2024-05-26T09:11:24.127382798Z" level=info msg="Daemon has completed initialization"
May 26 09:11:24 jenkins-server systemd[1]: Started docker.service - Docker Application Container Engine.
[done 1/23/21 (C)2024]
```

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.

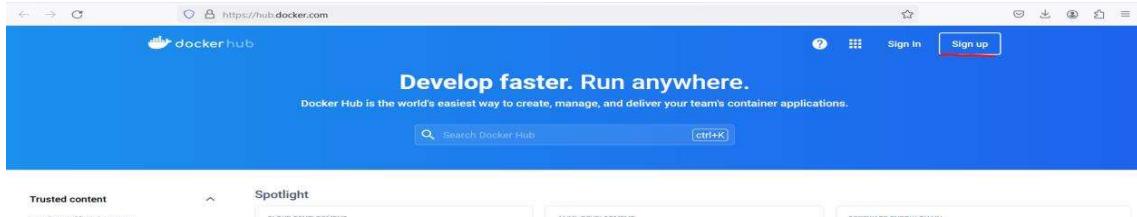
```
sudo systemctl daemon-reload
```

```
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.

A screenshot of the Docker Hub sign-in page. At the top is the Docker logo. Below it is the heading "Sign in". A subtext message reads: "Using Docker for work? We recommend signing in with your work email address." There is a text input field labeled "Username or email address*" containing "sauvik.devops@gmail.com". Below the input field is a large blue "Continue" button. Underneath the button is the word "OR". There are two additional "Continue" buttons: one for "Continue with Google" featuring the Google "G" logo, and one for "Continue with GitHub" featuring the GitHub logo. At the bottom of the page is a link "Don't have an account? [Sign Up](#)".

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

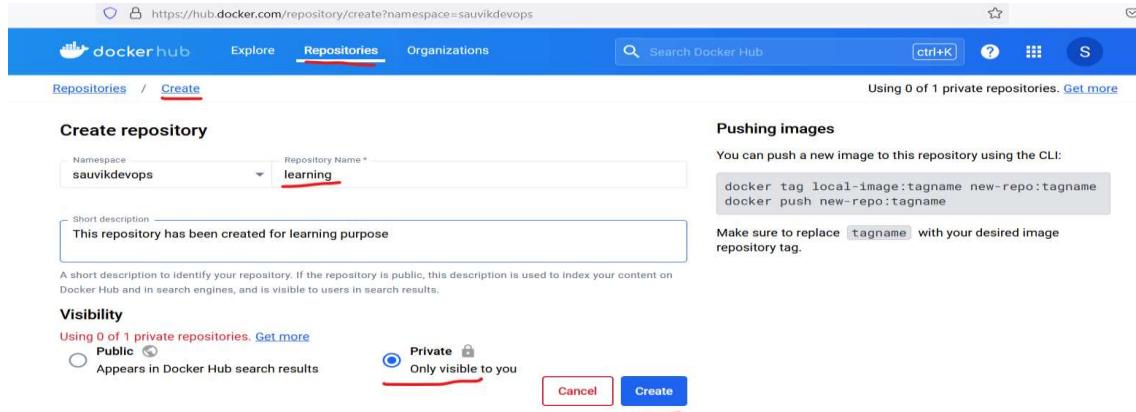
Create repository in Docker Hub

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



The screenshot shows the Docker Hub homepage. At the top, there are navigation links: 'docker hub', 'Explore', 'Repositories' (which is highlighted in red), 'Organizations', and a search bar 'Search Docker Hub'. Below the header, a large blue banner says 'Welcome to Docker' and 'Download the desktop application', with a 'Download for Windows' button and a note 'Also available for Mac and Linux'. Underneath the banner, there are three cards: 'Create a Repository' (Push container images to a repository on Docker Hub.), 'Docker Hub Basics' (Watch the guide on how to create and push your first image into a Docker Hub repository.), and 'Language-Specific Guides' (Learn how to containerize language-specific applications using Docker.). At the bottom of the page, it says 'Access the world's largest library of container images'.

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



The screenshot shows the 'Create repository' form on Docker Hub. At the top, it says 'https://hub.docker.com/repository/create?namespace=sauvikdevops'. The 'Repositories' tab is selected, and the 'Create' link is underlined. The form has fields for 'Namespace' (set to 'sauvikdevops') and 'Repository Name' (set to 'learning'). There is a 'Short description' field containing 'This repository has been created for learning purpose'. On the right, there is a 'Pushing images' section with CLI instructions: 'docker tag local-image:tagname new-repo:tagname' and 'docker push new-repo:tagname'. It also says 'Make sure to replace tagname with your desired image repository tag.' Below the form, it says 'Using 0 of 1 private repositories. [Get more](#)'. Under 'Visibility', there are two options: 'Public' (unchecked) and 'Private' (checked, with a lock icon). A note says 'Appears in Docker Hub search results'. At the bottom are 'Cancel' and 'Create' buttons, with 'Create' being highlighted in blue.

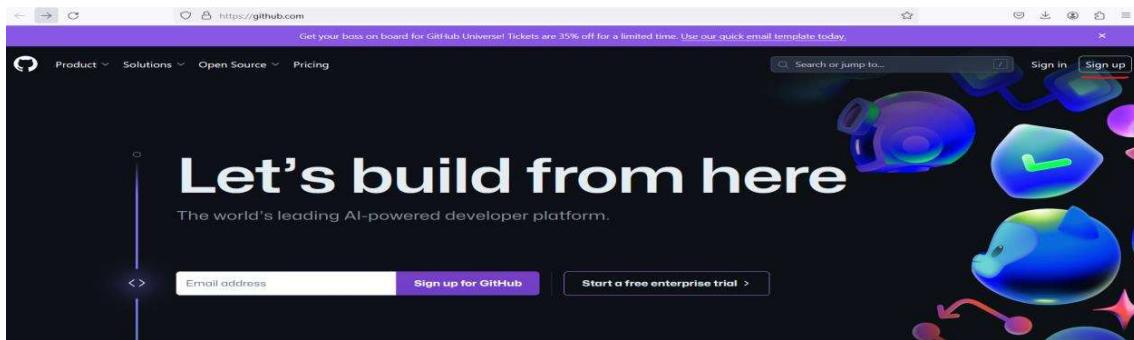
Your container registry would be created like below

The screenshot shows a DockerHub repository page for 'sauvikdevops/learning'. At the top, there's a navigation bar with 'Explore', 'Repositories', 'Organizations', a search bar, and a message about using 1 of 1 private repositories. Below the header, the repository name 'sauvikdevops/learning' is displayed, along with a note that it was created less than a minute ago and for learning purposes. A 'Docker commands' section shows a command to push a new tag. The 'Tags' section indicates the repository is empty. The 'Automated Builds' section is available with Pro, Team, and Business subscriptions. A blue 'Upgrade' button is present.

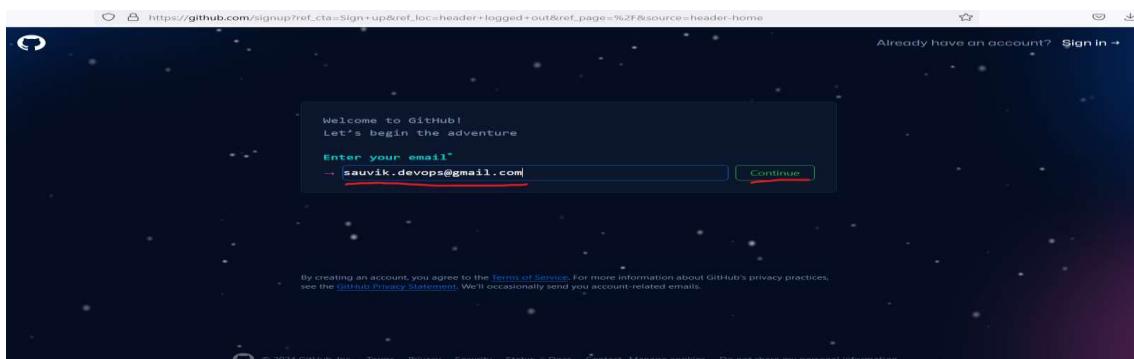
Create / manage Git Hub account

Signup for GitHub account

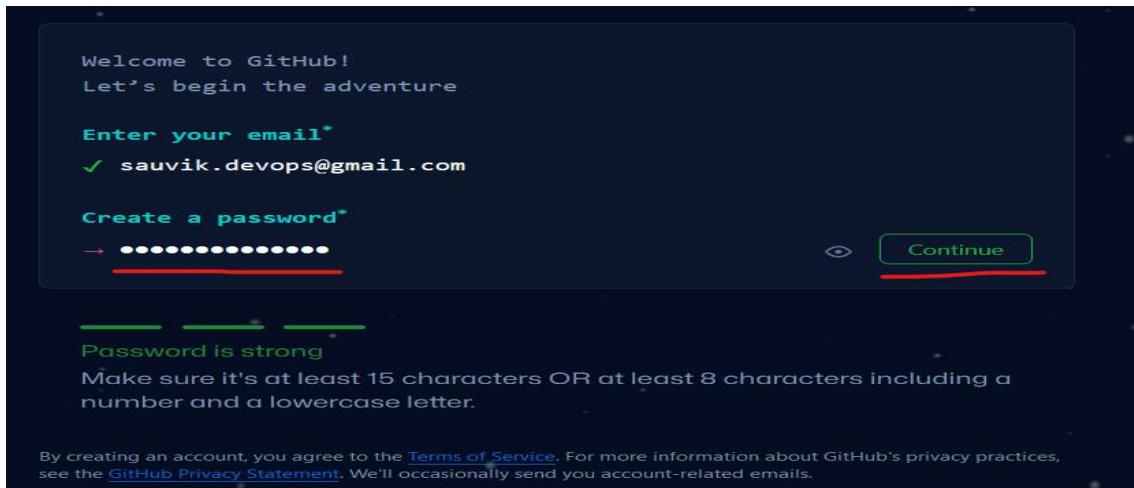
Visit <https://github.com/> and click on “Sign up”



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.

Welcome to GitHub!
Let's begin the adventure

Enter your email*

✓ sauvik.devops@gmail.com

Create a password*

✓ ••••••••••••••

Enter a username*

✓ sauvikdevops

Email preferences

Receive occasional product updates and announcements.

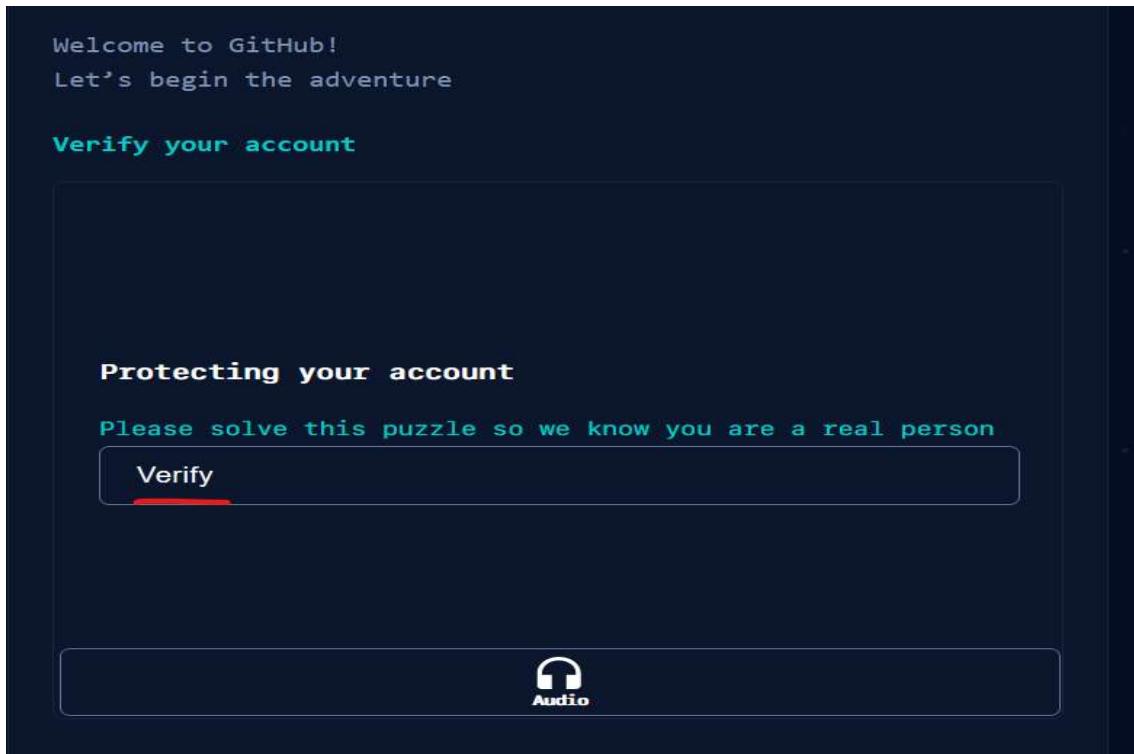
Continue

By creating an account, you agree to the [Terms of Service](#). For more information about GitHub's privacy practices, see the [GitHub Privacy Statement](#). We'll occasionally send you account-related emails.

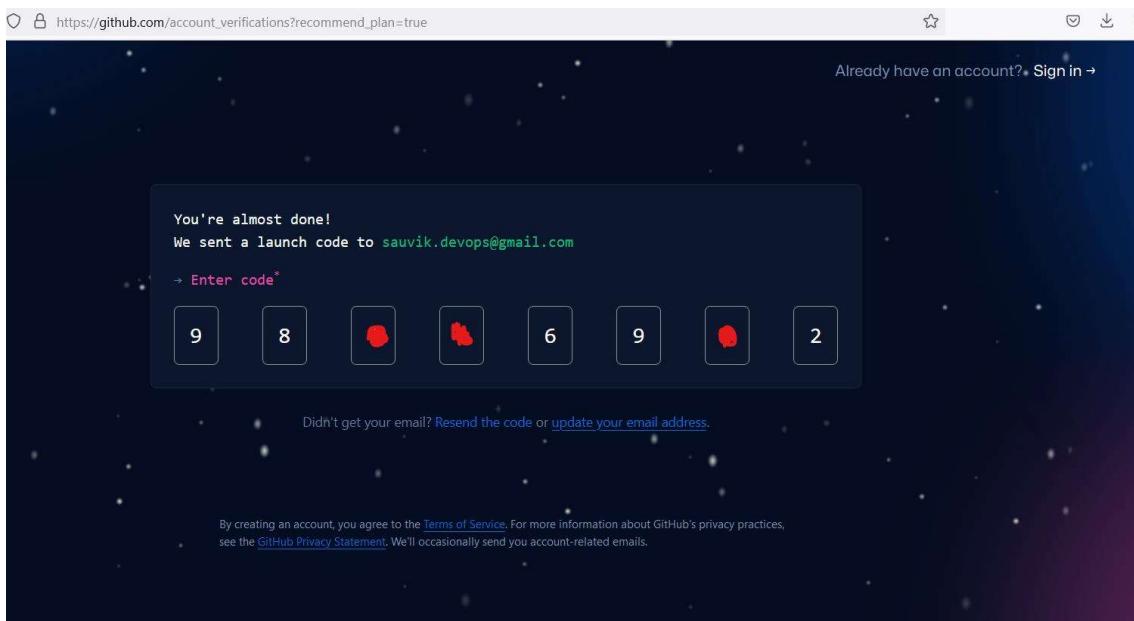
Your Github account will now get created.



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.

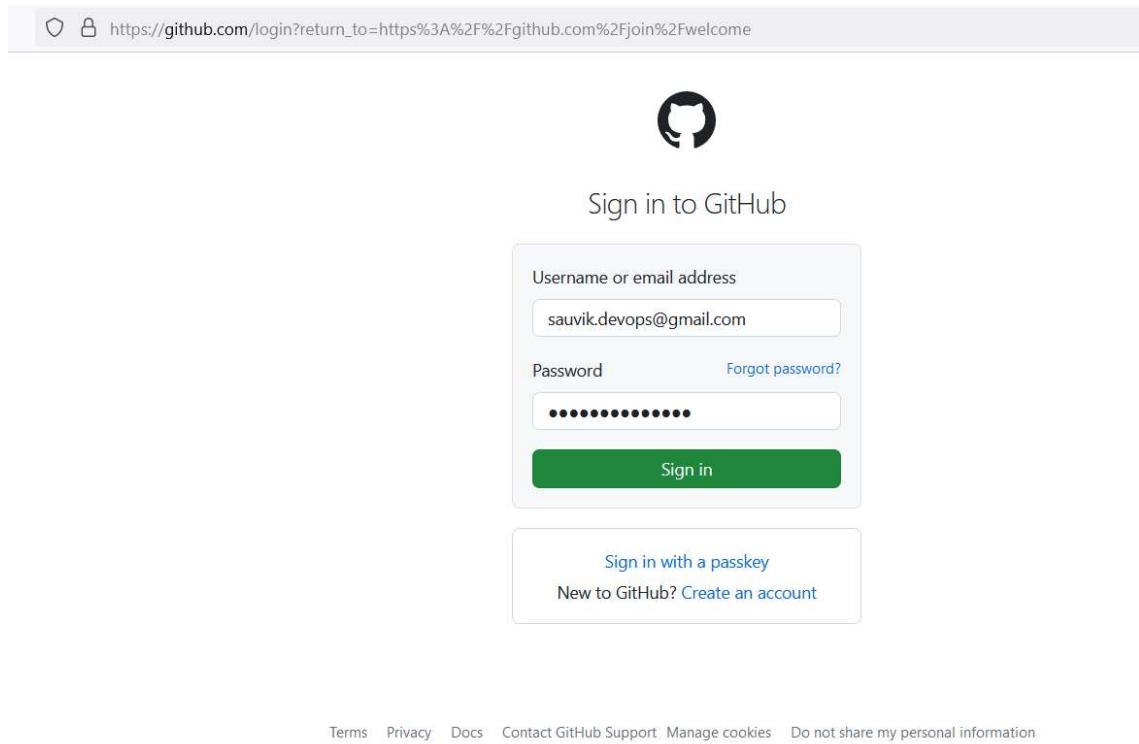


You are all set now.

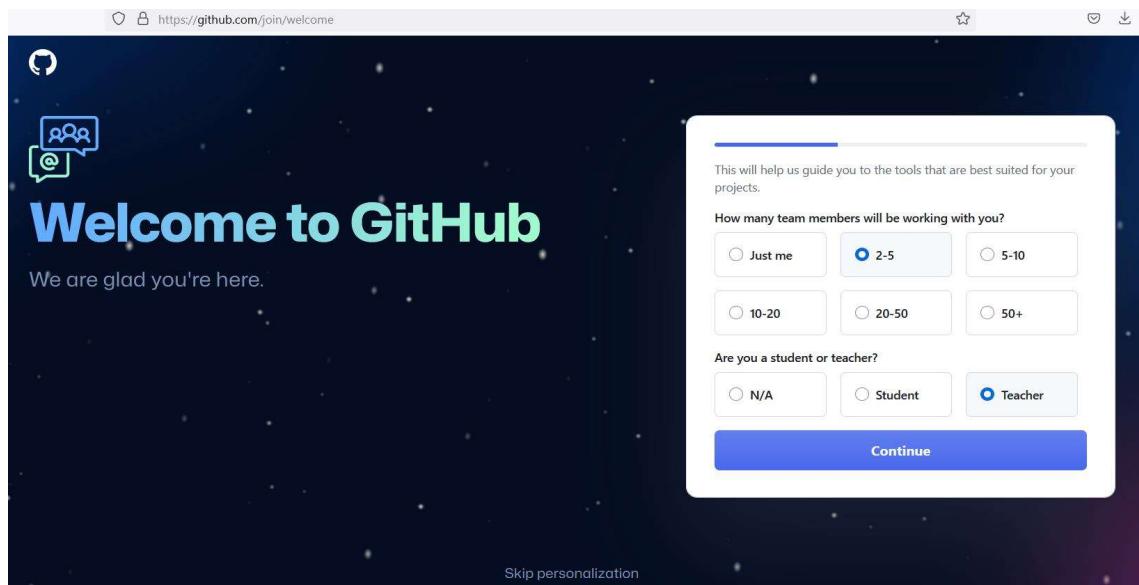
Login to GitHub account

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

The tools you need to build what you want.
Soup to nuts, GitHub has it all.

What specific features are you interested in using?

Select all that apply so we can point you to the right GitHub plan.

- Collaborative coding
Codespaces, Pull requests, Notifications, Code review, Code review assignments, Code owners, Draft pull requests, Protected branches, and more.
- Automation and CI/CD
Actions, Packages, APIs, GitHub Pages, GitHub Marketplace, Webhooks, Hosted runners, Self-hosted runners, Secrets management, and more.
- Security
Private repos, 2FA, Required reviews, Required status checks, Code scanning, Secret scanning, Dependency graph, Dependabot alerts, and more.
- Client Apps
GitHub Mobile, GitHub CLI, and GitHub Desktop.
- Project Management
Projects, Labels, Milestones, Issues, Unified Contribution Graph, Org activity graph, Org dependency insights, Repo insights, Wikis, and GitHub Insights.
- Team Administration
Organizations, Invitations, Team sync, Custom roles, Domain verification, Audit Log API, Repo creation restriction, and Notification restriction.
- Community
GitHub Marketplace, GitHub Sponsors, GitHub Skills, and Electron.

Continue

Select all options and click on – Continue

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

Real-world tools, engaged students.
GitHub gives teachers free access to industry-standard tools for training developers.

Free

- Unlimited public/private repositories
- 2,000 C/C/D minutes/month
Free for public repositories
- 500MB of Packages storage
- 120 core-hours of Codespaces compute
- 15GB of Codespaces storage
- Community support

Get additional teacher benefits

GITHUB TEAM

- Protect your branches
- Draft pull requests
- Required reviewers
- 3,000 C/C/D minutes/month
Free for public repositories
- 2GB of Packages storage
- Web-based support

GITHUB TEACHER TOOLBOX

Free access to the industry's best developer tools

GITHUB CLASSROOM

Automate your course

GITHUB CAMPUS ADVISORS

Join a community for teachers

Continue for free Apply for your GitHub teacher benefits

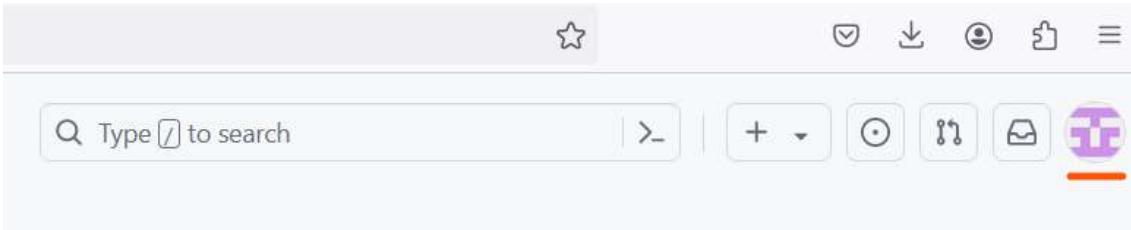
Your GitHub account setup is done and you will see the blow screen.

The screenshot shows the GitHub Home page. On the left, there's a 'Create your first project' section with a 'Create repository' button. Below it is a 'Recent activity' section. In the center, there's a form to 'Start a new repository' for a user named 'sauvikdevops'. The repository name is 'sauvikdevops', and the visibility is set to 'Public'. On the right, there are two promotional banners: one for 'GitHub Galaxy' and another for 'GitHub Universe 24'.

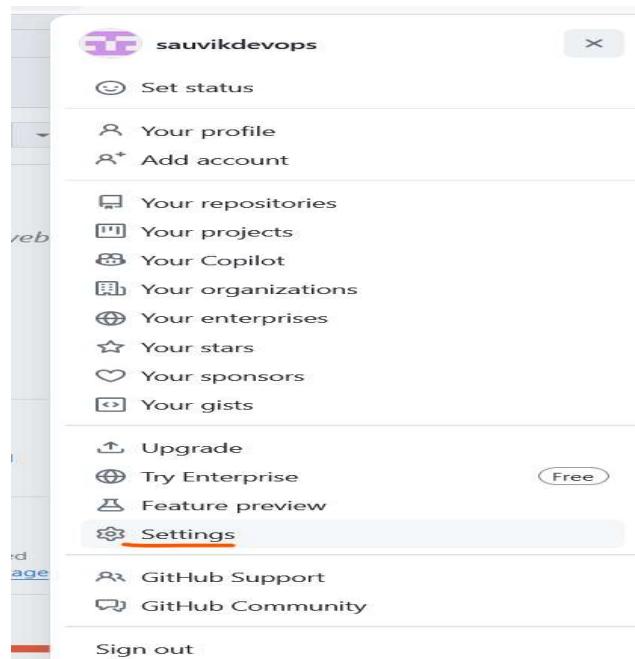
Generate Personal Access Token in GitHub

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

A screenshot of the GitHub Public profile settings page for the account 'sauvikdevops'. The page shows sections for Public profile, Account, Appearance, Accessibility, Notifications, Access, Code, planning, and automation, Security, Integrations, Archives, and Developer settings. The 'Developer settings' link is highlighted with an orange underline. The right side of the page contains fields for Name, Profile picture, Public email, Bio, Pronouns, URL, ORCID ID, Social accounts, Company, and Location, along with a 'Display current local time' checkbox and an 'Update profile' button.

Click on Personal access tokens → Tokens (classic)

The screenshot shows the GitHub Developer Settings page at <https://github.com/settings/apps>. The navigation bar includes 'Settings' and 'Developer Settings'. The left sidebar lists 'GitHub Apps', 'OAuth Apps', 'Personal access tokens' (which is highlighted with an orange underline), 'Fine-grained tokens' (Beta), and 'Tokens (classic)' (which is also highlighted with an orange underline). The main content area is titled 'GitHub Apps' with a sub-section for 'Personal access tokens (classic)'. It includes a note: 'Want to build something that integrates with the GitHub API. You can also read more about the GitHub API'. At the bottom right of the main content area, there are links for 'Terms', 'Privacy', 'Security', and 'Status'.

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

The screenshot shows the GitHub Personal access tokens (classic) page at <https://github.com/settings/tokens>. The navigation bar includes 'Settings' and 'Developer Settings'. The left sidebar lists 'GitHub Apps', 'OAuth Apps', 'Personal access tokens' (highlighted with an orange underline), 'Fine-grained tokens' (Beta), and 'Tokens (classic)'. The main content area is titled 'Personal access tokens (classic)' and includes a note: 'Need an API token for scripts or testing? [Generate a personal access token](#)'. It also notes: 'Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used to authenticate to the API over Basic Authentication.' On the right side, there is a dropdown menu with two options: 'Generate new token (Beta)' and 'Generate new token (classic) For general use' (which is highlighted with an orange underline). At the bottom right of the main content area, there are links for 'Terms', 'Privacy', 'Security', 'Status', 'Docs', 'Contact', 'Manage cookies', and 'Do not share my personal information'.

Login if asked –

The screenshot shows the GitHub 'Confirm access' page at <https://github.com/settings/tokens/new>. The page features a large GitHub logo at the top. Below it, the title 'Confirm access' is displayed. A box shows the user is signed in as '@sauvikdevops'. A password input field contains several dots, with a 'Forgot password?' link below it. A green 'Confirm' button is at the bottom of the form. A tip at the bottom of the page states: 'Tip: You are entering sudo mode. After you've performed a sudo-protected action, you'll only be asked to re-authenticate again after a few hours of inactivity.' At the very bottom of the page, there are links for 'Terms', 'Privacy', 'Docs', 'Contact GitHub Support', 'Manage cookies', and 'Do not share my personal information'.

Add Note and Expiration for your token

New personal access token (classic)

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Note

To be used to authenticate from Visual Studio Code

What's this token for?

Expiration *

Custom... 27 / 05 / 2025

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

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

copilot Full control of GitHub Copilot settings and seat assignments
 manage_billing:copilot View and edit Copilot Business seat assignments

project Full control of projects
 read:project Read access of projects

admin:gpg_key Full control of public user GPG keys
 write:gpg_key Write public user GPG keys
 read:gpg_key Read public user GPG keys

admin:ssh_signing_key Full control of public user SSH signing keys
 write:ssh_signing_key Write public user SSH signing keys
 read:ssh_signing_key Read public user SSH signing keys

Generate token **Cancel**

Your Personal access token is created successfully

Personal access tokens (classic)

Tokens you have generated that can be used to access the [GitHub API](#).

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

ghp_azrq... **Delete**

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

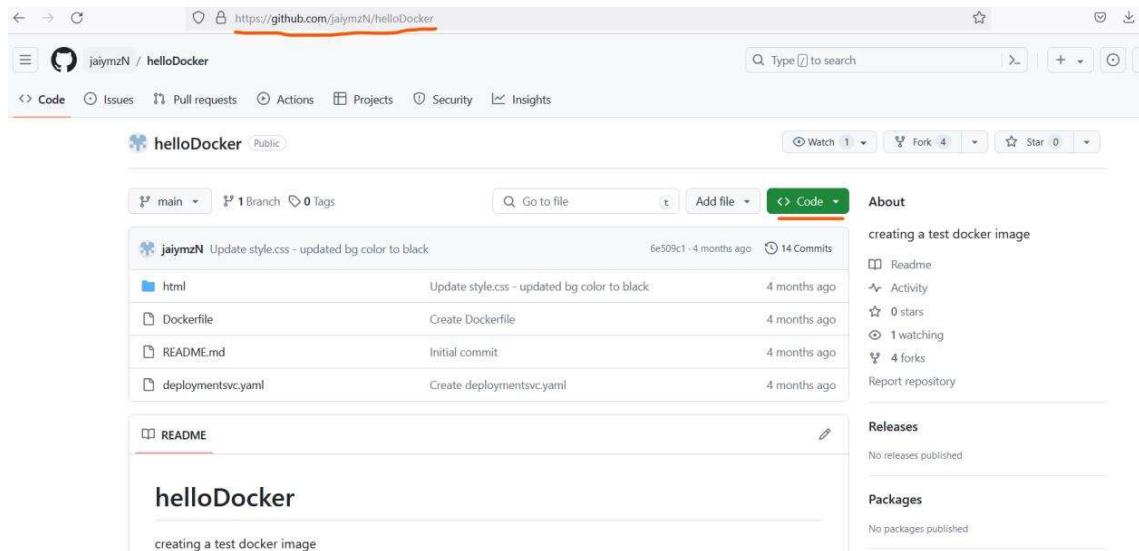
Note: Make sure to copy your personal access token now. You won't be able to see it again!

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).

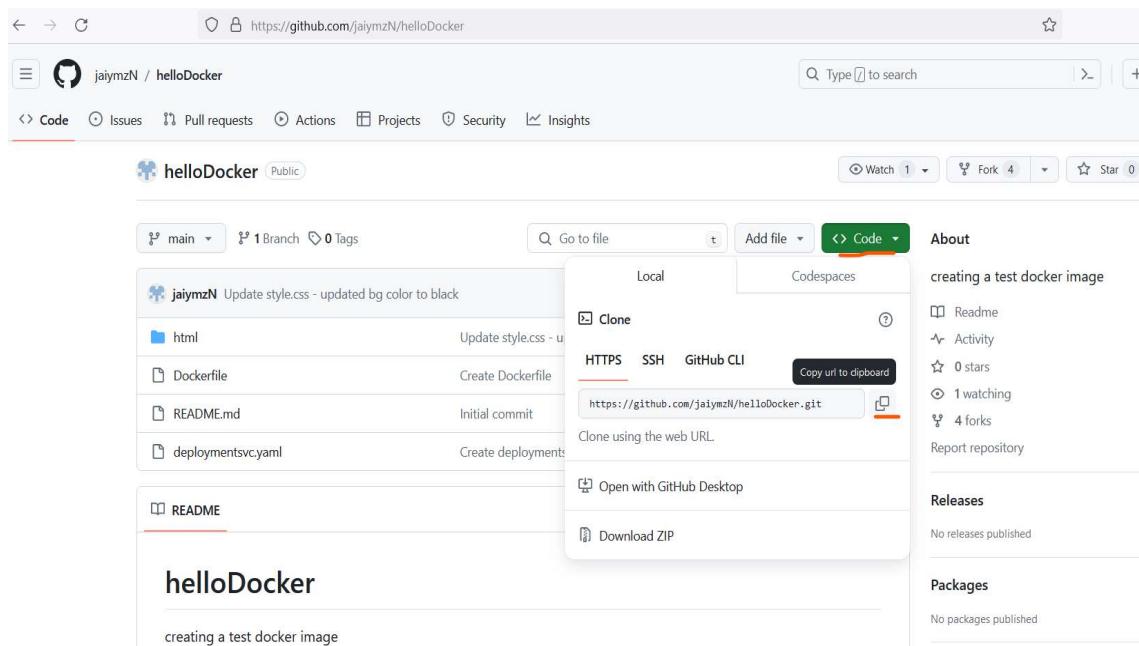
For example, to clone the below public repo (we would not need any credential for this), open this URL in browser:

<https://github.com/jaiymzN/helloDocker>



The screenshot shows the GitHub repository page for 'helloDocker'. The 'Code' tab is selected. On the left, there's a list of files: 'main' (branch), '1 Branch', '0 Tags', 'README', 'helloDocker', and 'creating a test docker image'. The 'README' file is expanded. On the right, there's an 'About' section with details like 'creating a test docker image', 'Readme', 'Activity', '0 stars', '1 watching', '4 forks', and a 'Report repository' link. Below that is a 'Releases' section stating 'No releases published'. At the bottom is a 'Packages' section stating 'No packages published'.

Click on “Code” followed by copy sign → 



The screenshot shows the same GitHub repository page for 'helloDocker'. The 'Code' tab is selected. A red box highlights the 'Clone' button under the 'Local' tab. To its right, there are options for 'HTTPS', 'SSH', and 'GitHub CLI', with 'HTTPS' selected. A 'Copy url to clipboard' button is also present. The URL 'https://github.com/jaiymzN/helloDocker.git' is copied to the clipboard. The rest of the interface is identical to the first screenshot.

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

The screenshot shows the GitHub Home page. On the left, there's a 'Create your first project' section with 'Create repository' and 'Import repository' buttons. The 'Import repository' button is highlighted with a red underline. On the right, there's a 'Updates to your homepage feed' section with a message about combining Following and For you feeds, and a 'Learn more' link. Below that is a code editor area with the placeholder 'Start writing code'.

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”

The screenshot shows the 'Import your project to GitHub' page at <https://github.com/new/import>. The URL field contains the copied URL from the previous step: <https://github.com/jaiymzN/helloDocker>. The 'Your source repository details' section includes fields for 'Your username for your source repository' and 'Your access token or password for your source repository'. The 'Your new repository details' section shows the owner as 'sauvikdevops' and the repository name as 'dockerdemo'. The 'Public' radio button is selected. At the bottom, there are 'Cancel' and 'Begin import' buttons.

5. Click on “Begin Import”

A screenshot of a GitHub import progress page. The URL is https://github.com/sauvikdevops/dockerdemo/import. The page title is "Preparing your new repository". It says "There is no need to keep this window open. We'll email you when the import is done." Below the text is a large circular progress bar. At the bottom, it says "Your import will begin shortly...".

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

A screenshot of a GitHub import confirmation page. The URL is https://github.com/sauvikdevops/dockerdemo/import. The page title is "Preparing your new repository". It says "There is no need to keep this window open. We'll email you when the import is done." Below the text is a green checkmark icon. At the bottom, it says "Importing complete! Your new repository sauvikdevops/dockerdemo is ready."

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

A screenshot of a GitHub repository page for "dockerdemo". The URL is https://github.com/sauvikdevops/dockerdemo. The repository is public. It shows a commit by "jaiymzn" and files like "html", "Dockerfile", "README.md", and "deploymentsvc.yaml". On the right, there's an "About" section with "No description, website, or topics provided.", a "Clone" section with "HTTPS", "SSH", and "GitHub CLI" options, and a "Copy url to clipboard" button. Other sections include "Releases", "Packages", and "Settings".

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

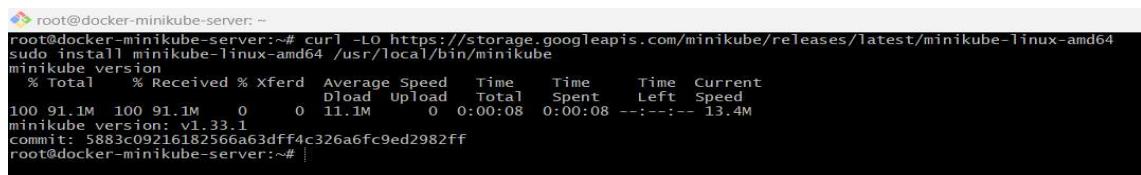
Install Minikube (for Kubernetes)

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
```



```
root@docker-minikube-server:~# curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64  
root@docker-minikube-server:~# sudo install minikube-linux-amd64 /usr/local/bin/minikube  
minikube version  
  % Total    % Received % Xferd  Average Speed   Time     Time      Time  Current  
  % Total    % Received % Xferd  Average Speed   Time     Time      Time  Current  
  100 91.1M  100 91.1M    0     0  11.1M      0  0:00:08  0:00:08  --:--:-- 13.4M  
minikube version: v1.33.1  
commit: 5883c09216182566a63dff4c326a6fc9ed2982ff  
root@docker-minikube-server:~# |
```

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.

```
devops@jenkins-minikube: ~
[deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.29/deb/ /'
```

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
```

```
devops@jenkins-minikube: ~
# This is /run/systemd/resolve/stub-resolv.conf managed by man:systemd-resolved(8).
# Do not edit.
#
# This file might be symlinked as /etc/resolv.conf. If you're looking at
# /etc/resolv.conf and seeing this text, you have followed the symlink.
#
# This is a dynamic resolv.conf file for connecting local clients to the
# internal DNS stub resolver of systemd-resolved. This file lists all
# configured search domains.
#
# Run "resolvectl status" to see details about the uplink DNS servers
# currently in use.
#
# Third party programs should typically not access this file directly, but only
# through the symlink at /etc/resolv.conf. To manage man:resolv.conf(5) in a
# different way, replace this symlink by a static file or a different symlink.
#
# See man:systemd-resolved.service(8) for details about the supported modes of
# operation for /etc/resolv.conf.

nameserver 127.0.0.53
nameserver 8.8.8.8
nameserver 8.8.4.4
options edns0 trust-ad
search .
~
```

Run the below command to restart docker:

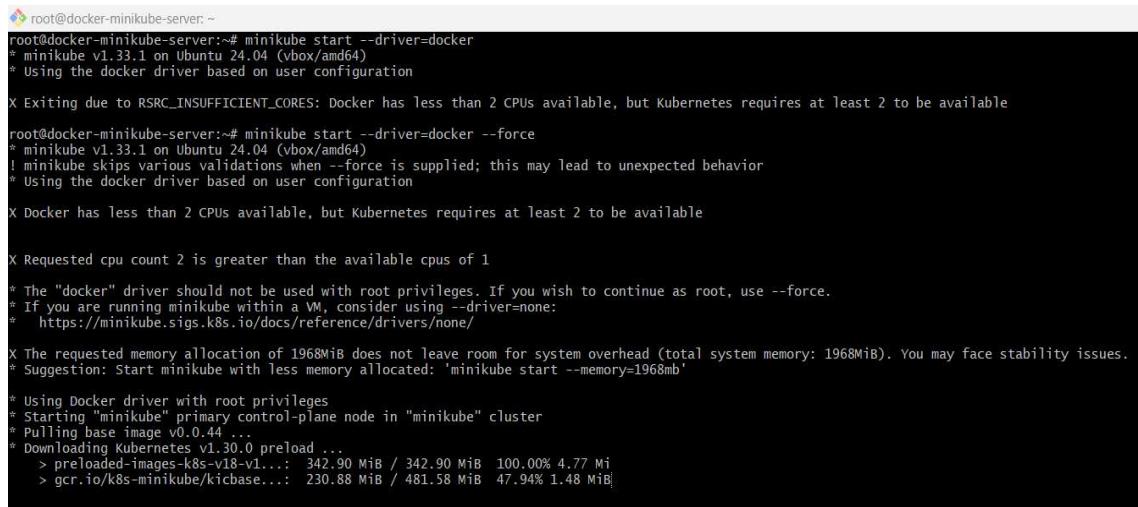
```
sudo systemctl daemon-reload
```

```
sudo systemctl restart docker
```

Start Minikube (with Docker Driver)

Command to start minikube

```
minikube start --driver=docker --force
```



A terminal window showing the output of the minikube start command. The output indicates that the Docker driver is being used, but it fails due to insufficient CPU resources (less than 2 CPUs available, while Kubernetes requires at least 2). It also shows warnings about Docker not being used with root privileges and memory allocation issues.

```
root@docker-minikube-server:~# minikube start --driver=docker
* minikube v1.33.1 on Ubuntu 24.04 (vbox/amd64)
* Using the docker driver based on user configuration

X Exiting due to RSRC_INSUFFICIENT_CORES: Docker has less than 2 CPUs available, but Kubernetes requires at least 2 to be available

root@docker-minikube-server:~# minikube start --driver=docker --force
* minikube v1.33.1 on Ubuntu 24.04 (vbox/amd64)
! minikube skips various validations when --force is supplied; this may lead to unexpected behavior
* Using the docker driver based on user configuration

X Docker has less than 2 CPUs available, but Kubernetes requires at least 2 to be available

X Requested cpu count 2 is greater than the available cpus of 1

* The "docker" driver should not be used with root privileges. If you wish to continue as root, use --force.
* If you are running minikube within a VM, consider using --driver=none:
*   https://minikube.sigs.k8s.io/docs/reference/drivers/none/

X The requested memory allocation of 1968MiB does not leave room for system overhead (total system memory: 1968MiB). You may face stability issues.
* Suggestion: Start minikube with less memory allocated: "minikube start --memory=1968mb"

* Using Docker driver with root privileges
* Starting "minikube" primary control-plane node in "minikube" cluster
* Pulling base image v0.0.44 ...
* Downloading Kubernetes v1.30.0 preload ...
  > preloaded-images-k8s-v18-v1...: 342.90 MiB / 342.90 MiB 100.00% 4.77 MiB
  > gcr.io/k8s-minikube/kicbase...: 230.88 MiB / 481.58 MiB 47.94% 1.48 MiB
```

[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:~# 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:~# 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/>

The screenshot shows the Jenkins dashboard at localhost:8080. The left sidebar has links for 'New Item', 'Build History', 'Manage Jenkins' (which is selected), and 'My Views'. The main area features a 'Welcome to Jenkins!' message, a 'Start building your software project' section with a 'Create a job' button, and a 'Set up a distributed build' section with 'Set up an agent' and 'Configure a cloud' buttons. At the bottom right are links for 'REST API' and 'Jenkins 2.452.1'.

Configuring Docker Hub authentication

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

This screenshot is identical to the one above, but the 'Manage Jenkins' link in the sidebar is highlighted with an orange underline, indicating it is the active section.

Click on Credentials ->

The screenshot shows the 'Manage Jenkins' page under the 'System Configuration' section. It includes sections for 'System', 'Nodes', 'Tools', 'Clouds', 'Plugins', 'Appearance', 'Security', 'Credentials' (which is selected and highlighted in orange), and 'User Providers'. The 'Credentials' section has a 'Configure credentials' link.

Click on "(global)" ->

The screenshot shows the Jenkins 'Credentials' page at localhost:8080/manage/credentials/. The title bar says 'Jenkins'. The main heading is 'Credentials'. Below it, a table header row has columns: T, P, Store ↓, Domain, ID, and Name. A sub-section titled 'Stores scoped to Jenkins' shows a table with columns: P, Store ↓, Domains, Icon, and L. It lists one item: 'System' under '(global)'. There are icons for S, M, and L below the table.

Click on Add Credentials

The screenshot shows the 'Global credentials (unrestricted)' page at localhost:8080/manage/credentials/store/system/domain/_/. The title bar says 'Jenkins'. The main heading is 'Global credentials (unrestricted)'. A blue button labeled '+ Add Credentials' is highlighted with a red underline. Below it, a table header row has columns: ID, Name, Kind, and Description. A message in the center says 'This credential domain is empty. How about [adding some 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

The screenshot shows the Jenkins 'New credentials' creation page. The URL is `localhost:8080/manage/credentials/store/system/domain/_/newCreden`. The navigation path is: Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted). The 'Kind' field is set to 'Secret text'. The 'Scope' is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'Secret' field contains redacted text. The 'ID' is 'docker_hub'. The 'Description' is 'Docker Hub authentication'. A blue 'Create' button is at the bottom.

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

The screenshot shows the Jenkins 'Manage Jenkins' page. The URL is `localhost:8080/manage/`. The navigation path is: Dashboard > Manage Jenkins. The 'Manage Jenkins' section is selected. On the left, there are links for 'New Item', 'Build History', 'Manage Jenkins' (which is highlighted), and 'My Views'. On the right, there are sections for 'System Configuration' (with 'System', 'Nodes', 'Clouds', and 'Security' sub-links), 'Tools' (with 'System', 'Nodes', and 'Clouds' sub-links), and 'Plugins' (with 'Add, remove, disable or enable plugins that can extend the functionality of Jenkins.' sub-link). A search bar at the top right says 'Search (CTRL+K)'.

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

The screenshot shows the Jenkins "Advanced settings" page under the "Plugins" section. The "Deploy Plugin" section is active, showing a "File" input field with a "Browse..." button and a message indicating no file has been selected. Below it is an "Or" label and a "URL" input field. A "Deploy" button is at the bottom.

The screenshot shows the Jenkins "Advanced settings" page with a file upload dialog open over the "Deploy Plugin" section. The dialog displays a file tree from "This PC" with a file named "kubernetes-cd.hpi" selected. The "Open" button is highlighted with an orange rectangle.

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

Advanced settings

The Proxy configuration form has been moved to [Configure System page](#)

Deploy Plugin

You can select a plugin file from your local system or provide a URL to install a plugin from

File

Browse...

kubernetes-cd.hpi

Or

URL

Deploy

Select “Deploy”

The screenshot shows the Jenkins 'Plugins' page under 'Manage Jenkins'. The 'Download progress' tab is selected. On the left, there's a sidebar with links: 'Updates', 'Available plugins', 'Installed plugins', 'Advanced settings', and 'Download progress' (which is highlighted). The main area is titled 'Download progress' and shows the status of several plugin installations:

Plugin	Status	Notes
SSH server	Success	
Azure Commons	Installing	Progress bar shown
Authentication Tokens API	Pending	
Docker Commons	Pending	
Preparation	Pending	
kubernetes-cd	Pending	

Below the table, there are two links: 'Go back to the top page' and 'Restart Jenkins when installation is complete and no jobs are running'.

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.

```
[sudo] password for devops:  
Warning: The unit file, source configuration file or drop-ins of jenkins.service changed on disk. Run 'systemctl daemon-reload' to reload units.  
Job for jenkins.service failed because a timeout was exceeded  
see 'systemctl status jenkins.service' and 'journalctl -xeu jenkins.service' for details.
```

Run this command:

```
systemctl daemon-reload
```

Then restart Jenkins service.

```
sudo systemctl restart jenkins
```

Then you can check Jenkins status:

```
sudo systemctl status jenkins
```

```

jenkins@jenkins-server:~$ sudo systemctl restart jenkins
[judo] password for jenkins:
jenkins@jenkins-server:~$ sudo systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
   Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: enabled)
     Active: active (running) since Mon 2024-05-27 10:48:34 UTC; 2min 49s ago
       Main PID: 4146 (java)
         Tasks: 1 (limit: 276)
        Memory: 347.0M (peak: 348.5M)
        CPU: 1min 37.153s
      CGroup: /system.slice/jenkins.service
             └─4146 /usr/java/jdk-18.0.2+10/bin/java -Djava.net.httpServer=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

May 27 10:48:33 jenkins-server jenkins[4146]: 2024-05-27 10:48:33.316+0000 [id=31] INFO jenkins.InitReactorRunner$1@#Attained: Configuration for all jobs updated
May 27 10:48:33 jenkins-server jenkins[4146]: 2024-05-27 10:48:33.317+0000 [id=31] INFO hudson.util.ReverseStartAttentioner: Attained: Configuration check completes server
May 27 10:48:34 jenkins-server jenkins[4146]: 2024-05-27 10:48:34.046+0000 [id=24] INFO hudson.lifecycle.Lifecycle$OnReady: Jenkins is fully up and running
May 27 10:48:34 jenkins-server jenkins[4146]: 2024-05-27 10:48:34.047+0000 [id=24] INFO hudson.lifecycle.Lifecycle$OnReady: Jenkins is fully up and running
May 27 10:48:34 jenkins-server jenkins[4146]: 2024-05-27 10:48:34.047+0000 [id=24] INFO hudson.lifecycle.Lifecycle$OnReady: Jenkins is fully up and running
May 27 10:49:14 jenkins-server jenkins[4146]: 2024-05-27 10:49:14.422+0000 [id=47] INFO h.m.DownloadService$Downloadable$load: Obtained the updated data file for hudson.tasks.Maven.MavenInstaller
May 27 10:49:14 jenkins-server jenkins[4146]: 2024-05-27 10:49:14.422+0000 [id=47] INFO h.m.DownloadService$Downloadable$load: Obtained the updated data file for hudson.tasks.Ant.AntInstaller
May 27 10:49:16 jenkins-server jenkins[4146]: 2024-05-27 10:49:16.695+0000 [id=47] INFO h.m.DownloadService$Downloadable$load: Obtained the updated data file for hudson.plugins.gradle.GradleInstaller
May 27 10:49:23 jenkins-server jenkins[4146]: 2024-05-27 10:49:23.209+0000 [id=47] INFO h.m.DownloadService$Downloadable$load: Obtained the updated data file for hudson.tools.JDKInstaller
May 27 10:49:23 jenkins-server jenkins[4146]: 2024-05-27 10:49:23.209+0000 [id=47] INFO hudson.util.ReverseStart: Performed the action check updates server successfully at the attempt #2
[lines 1-20/20 (END)]
jenkins@jenkins-server:~$ 

```

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)”

The screenshot shows the Jenkins 'New credentials' page. The 'Kind' dropdown is set to 'Kubernetes configuration (kubeconfig)'. Other fields include 'Scope' (Global), 'ID' (k8_auth), and 'Description' (Kubernetes Authentication). Under 'Kubeconfig', there are two options: 'Enter directly' (selected) and 'From a file on the Jenkins master'. A 'Create' button is at the bottom.

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

Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) >

ID ?
k8_auth

Description ?
Kubernetes Authentication

Kubeconfig

Enter directly
Content ?

```
apiVersion: v1
clusters:
- cluster:
  certificate-authority-data:
  LS0tLS1CRUdJTiBDRVUSUZJQ0FURS0hLS0tOk1JSURCakNDQWU2Z0F3SUJBZ0lC0VVRBTkna2Foa2lHOXewQkFrC0ZBREFWTViNj0VRWURWUVFERXdwGFXNXAKYTNWaVpVTkUNQjRYFRU
  ME1EVx0nVEUvTURnME5b1hEVE0wTURVe1ERTJNRCgwImxvd0ZURVRNQkVHQfEVROoBeE1LyldsdWFxdDFZbVZEUVRDQ0FTSxdeUvIKs29aSWh2Y05BUUVCoIFBRGdnRVBBRENDOVfV...
```

From a file on the Jenkins master

From a file on the Kubernetes master node

Create

Click on - Create

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

localhost:8080/manage/credentials/store/system/domain/_/

Jenkins

Global credentials (unrestricted)

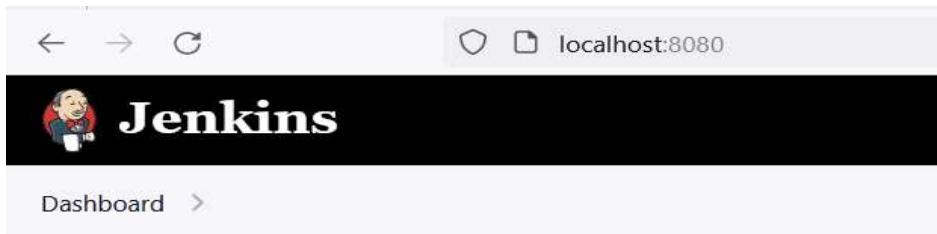
ID	Name	Kind	Description
docker_hub	Docker Hub authentication	Secret text	Docker Hub authentication
Kubeconfig_ID	Kubeconfig ID	Kubernetes configuration (kubeconfig)	

+ Add Credential

Create CI/CD pipeline in Jenkins

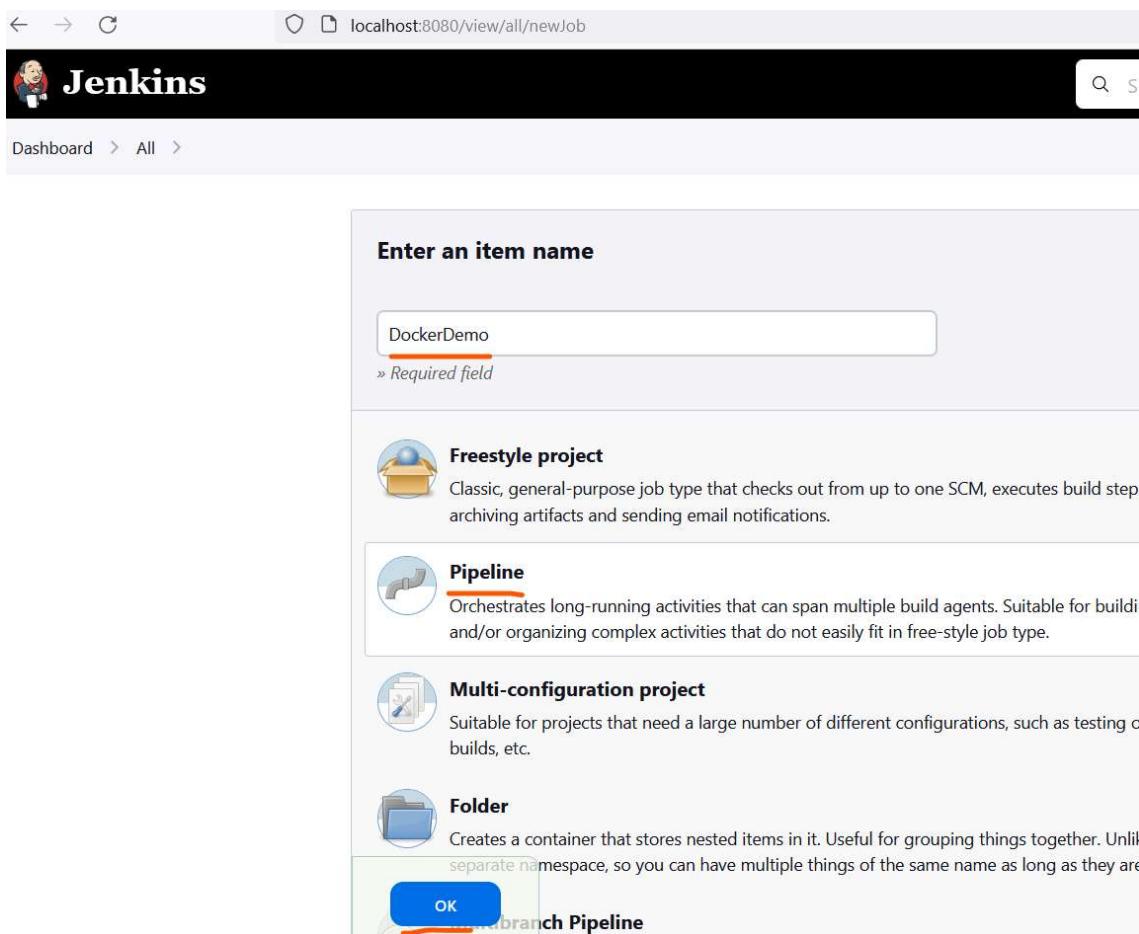
Go to your Jenkins URL / console and configure the following

Create a new item



The screenshot shows the Jenkins dashboard at localhost:8080. At the top, there are navigation icons (back, forward, search) and a URL bar showing 'localhost:8080'. Below the header is a black bar with the Jenkins logo and the word 'Jenkins' in white. A 'Dashboard' link is visible. On the left, a sidebar menu includes 'New Item' (which is underlined in red), 'Build History', 'Manage Jenkins', and 'My Views'.

Select the Pipeline Option



The screenshot shows the 'Create New Job' page at localhost:8080/view/all/newJob. The title bar says 'Jenkins'. The main area has a form titled 'Enter an item name' with 'DockerDemo' entered. Below it, there are four project types: 'Freestyle project', 'Pipeline', 'Multi-configuration project', and 'Folder'. The 'Pipeline' option is highlighted with a red underline. At the bottom right are 'OK' and 'Cancel' buttons, and a link 'Pipeline'.

Enter a Description (optional)

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

The screenshot shows the Jenkins job configuration page for 'DockerDemo'. In the 'General' section, there is a 'Description' field containing the text: 'This is a demo project. Here Jenkins CICD pipeline will fetch code from Github repository, then build the code and finally deploy to Kubernetes (minikube).'. Below this, under 'GitHub project', the 'Project url' is set to 'https://github.com/sauvikdevops/dockerdemo.git'. The 'Enabled' switch is turned on.

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

The screenshot shows the Jenkins job configuration page for 'DockerDemo'. In the 'Build Triggers' section, the 'GitHub hook trigger for GITscm polling' option is selected, indicated by a checked checkbox. Other options like 'Build after other projects are built', 'Build periodically', 'Poll SCM', 'Quiet period', and 'Trigger builds remotely' are also listed but not selected.

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

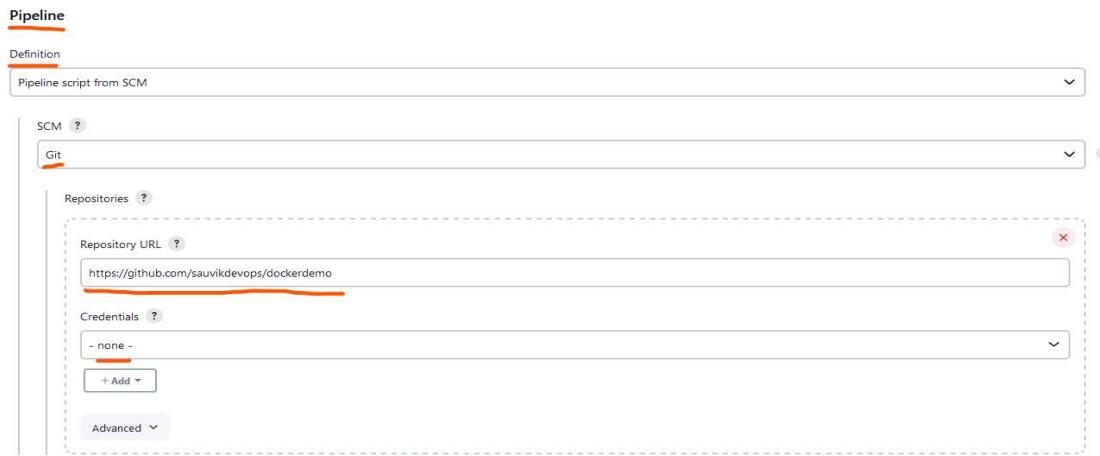
Definition = Pipeline script from SCM

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)

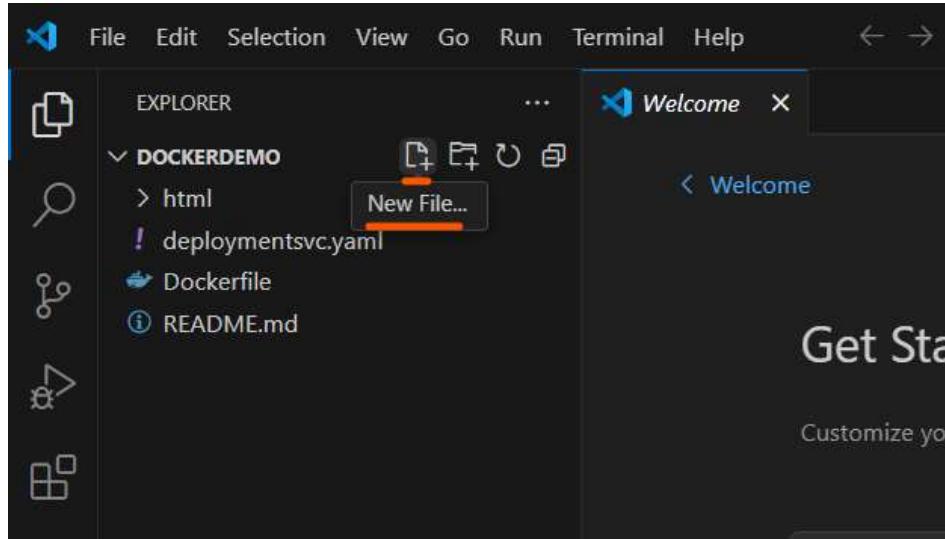
The screenshot shows the Jenkins Pipeline configuration interface. Under the 'Branches to build' section, 'Branch Specifier (blank for 'any')' is set to `*/main`. Below it is an 'Add Branch' button. Under the 'Repository browser' section, '(Auto)' is selected. Under 'Additional Behaviours', there is an 'Add' dropdown. Under the 'Script Path' section, 'Jenkinsfile' is specified. Below it is a checked checkbox for 'Lightweight checkout'. At the bottom, there are 'Save' and 'Apply' buttons.

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

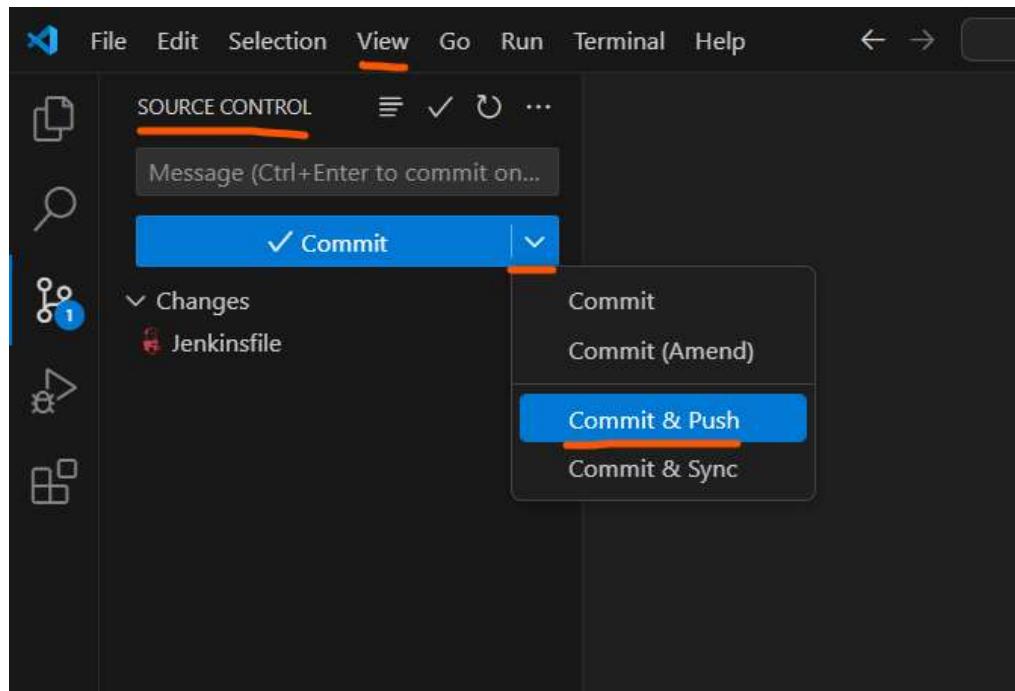
```
- pipeline {
-   agent any
-
-   stages {
-     stage('Checkout GitHub repo') {
-       steps {
-         checkout scmGit(branches: [[name: '*/main']], extensions: [], userRemoteConfigs: [[url:
'https://github.com/sauvikdevops/dockerdemo']])
-       }
-     }
-     stage('Build and Tag Docker Image') {
-       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'
        }
    }
}
```

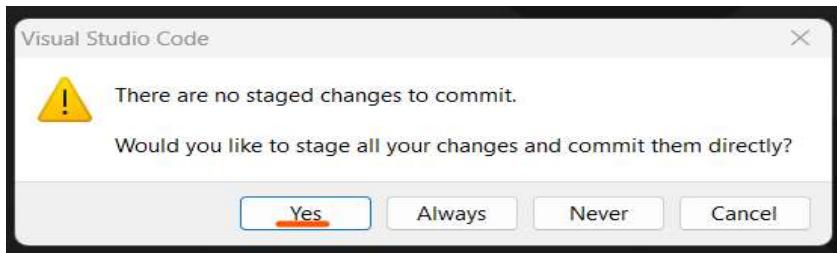
After that save the changes using **Ctrl+ S** or **File menu → Save**

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

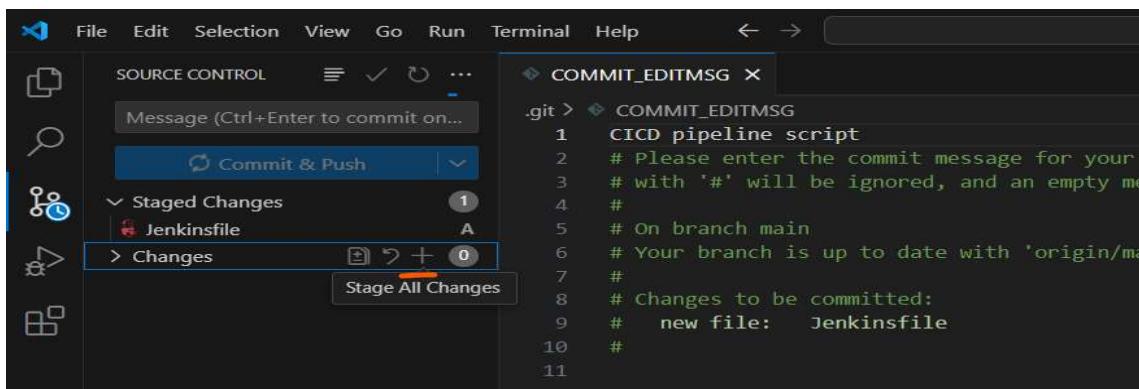
Now, click on the downward pointing arrow at right side of Commit and click on **Commit & Push**.



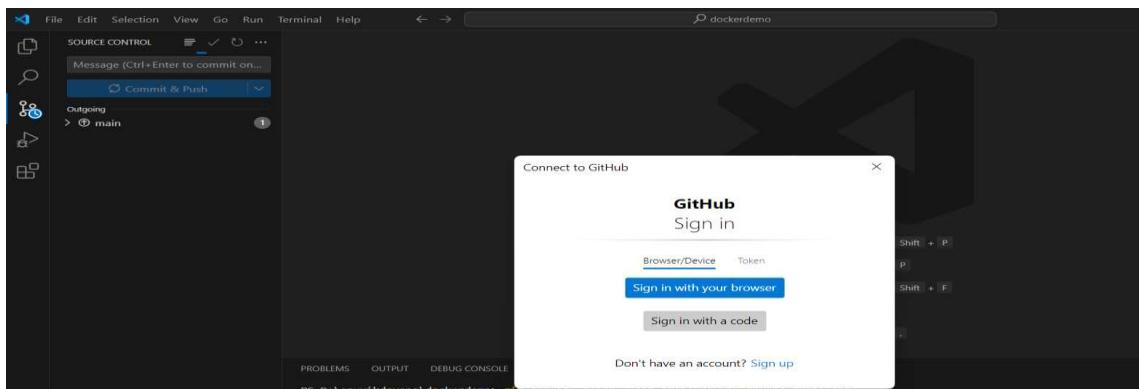
Click on Yes to continue



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



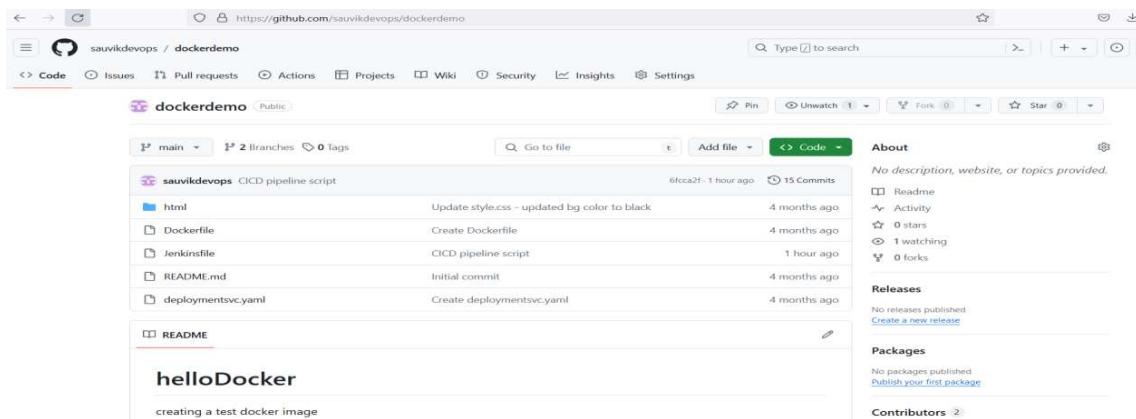
Please enter the commit message for your changes and close it to proceed.



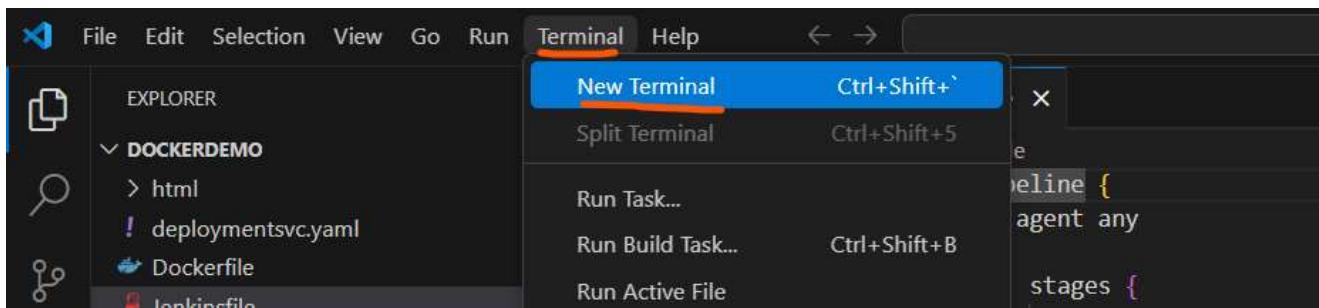
Click on "Token" and put the previously created token.



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



Note: alternatively, when you are done 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

Page 84 of 98

Jenkins CI/CD with GitHub, DockerHub and Kubernetes by Sauvik Maiti

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.



The screenshot shows the Jenkins dashboard at `localhost:8080`. The 'Dashboard' link is highlighted in orange. On the left sidebar, there are links for 'New Item', 'Build History', 'Project Relationship', 'Check File Fingerprint', and 'Manage Jenkins'. In the center, there is a search bar with 'Search (CTRL+K)' and a '+' button. Below it is a table with columns: S, W, Name, Last Success, Last Failure, and Last Duration. The 'DockerDemo' job is listed with a green checkmark icon, a yellow sun icon, and the name 'DockerDemo'. The last success was 2 days 20 hr ago, build #25. The last failure was 3 days 7 hr ago, build #20. The last duration was 37 sec. A green right-pointing arrow icon is at the bottom right of the table.

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

The screenshot shows the Jenkins interface for the 'DockerDemo' job. On the left, a sidebar menu includes 'Status' (selected), 'Changes', 'Build Now', 'Configure', 'Delete Pipeline', 'GitHub', 'Stages', 'Rename', 'Pipeline Syntax' (underlined in red), and 'GitHub Hook Log'. The main content area displays a green checkmark icon and the text 'DockerDemo'. Below it, a note says 'This is a demo project which will fetch code from Github'. A 'Permalinks' section lists several build history items. At the bottom, there's a 'Build History' card showing build #25 from June 2, 2024, at 7:10 PM.

Snippet Generator will come up

The screenshot shows the Jenkins 'Pipeline Syntax' snippet generator. The left sidebar lists options like 'Snippet Generator' (selected), 'Declarative Directive Generator', 'Declarative Online Documentation', 'Steps Reference', 'Global Variables Reference', 'Online Documentation', 'Examples Reference', and 'IntelliJ IDEA GDSDL'. The main content area has an 'Overview' section with a note about the Snippet Generator. Below it is a 'Steps' section showing a sample step: 'approveReceivedEvent: Updates an Approve event in Elasticsearch'. A 'Generate Pipeline Script' button is at the bottom.

Pipeline script to check out the GitHub repository

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

The screenshot shows the Jenkins Pipeline Syntax Snippet Generator interface. The URL is `localhost:8080/job/DockerDemo/pipeline-syntax/`. The navigation bar includes 'Dashboard', 'DockerDemo', and 'Pipeline Syntax'. A sidebar on the left lists links: 'Snippet Generator' (highlighted), 'Declarative Directive Generator', 'Declarative Online Documentation', 'Steps Reference', 'Global Variables Reference', 'Online Documentation', 'Examples Reference', and 'IntelliJ IDEA GDSL'. The main content area is titled 'Overview' and contains a 'Steps' section. A 'Sample Step' is shown with the code: `checkout: Check out from version control`. Below it, under 'SCM', is a 'Git' configuration panel. The 'Repositories' section shows a single repository with the 'Repository URL' set to `https://github.com/sauvikdevops/dockerdemo.git` and 'Credentials' set to '- none -'. There is also a '+ Add' button and an 'Advanced' dropdown.

Select SCM = Git

Repository URL = URL of your GitHub repo

Scroll down and change Branch specifier = */main

Click on – “Generate Pipeline Script”

The screenshot shows the Jenkins Pipeline Syntax configuration page. The URL is `localhost:8080/job/DockerDemo/pipeline-syntax/`. The pipeline configuration is as follows:

```

pipeline {
    agent any
    stages {
        stage('Checkout GitHub repo') {
            steps {
                checkout scmGit(branches: [[name: '/main']], extensions: [], userRemoteConfigs: [[url: 'https://github.com/sauvikdevops/dockerdemo.git']]) // Line highlighted with an orange box
            }
        }
    }
}

```

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:

The Jenkinsfile in VS Code looks like this:

```

Jenkinsfile
1 pipeline {
2     agent any
3
4     stages {
5         stage('Checkout GitHub repo') {
6             steps {
7                 checkout scmGit(branches: [[name: '/main']], extensions: [], userRemoteConfigs: [[url: 'https://github.com/sauvikdevops/dockerdemo.git']]) // Line highlighted with an orange box
8             }
9         }
10    }
11 }

```

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

The screenshot shows the Jenkins Pipeline Syntax Snippet Generator interface. The left sidebar has links for Snippet Generator, Declarative Directive Generator, Declarative Online Documentation, Steps Reference, Global Variables Reference, Online Documentation, Examples Reference, and IntelliJ IDEA GDSL. The main area is titled 'Overview' and contains a 'Steps' section. A 'Sample Step' box shows the code 'withCredentials: Bind credentials to variables'. Below it is a 'withCredentials' dropdown menu with options like Certificate, Docker client certificate, Git Username and Password, Kubeconfig Content, Microsoft Azure Service Principal, SSH User Private Key, Secret ZIP file, Secret file, Secret text (which is highlighted), Username and password (conjoined), and Username and password (separated). A 'Generate Pipeline Script' button is at the bottom.

From the dropdown, select - Secret text

This screenshot is similar to the previous one but focuses on the 'Secret text' option in the dropdown menu. The 'Secret text' option is highlighted with a red box. The rest of the interface is identical to the first screenshot.

Now, put the variable = docker_hub

For credential, click on + Add

Select Jenkins

The screenshot shows the Jenkins Credentials configuration page. It lists a 'Docker hub authentication' item under 'Credentials'. Below it is a 'Jenkins Credentials Provider' section with a 'Jenkins' provider selected. There is also another 'Jenkins' entry in the list.

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

Jenkins Credentials Provider: Jenkins

Add Credentials

Domain: Global credentials (unrestricted)

Kind: Username with password

Scope: Global (Jenkins, nodes, items, all child items, etc)

Username: Your docker hub user name

Treat username as secret:

Password:

ID:

Description:

Cancel Add

Sample Step

withCredentials: Bind credentials to variables

withCredentials ?

Secret values are masked on a best-effort basis to prevent *accidental* disclosure. Multiline secrets, such as the contents of a SSH private key file, are not masked.

Bindings

Secret text: Variable: docker_hub

Credentials: Docker hub authentication

Generate Pipeline Script

```
withCredentials([string(credentialsId: 'docker_hub', variable: 'docker_hub')]) {
    // some block
}
```

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.

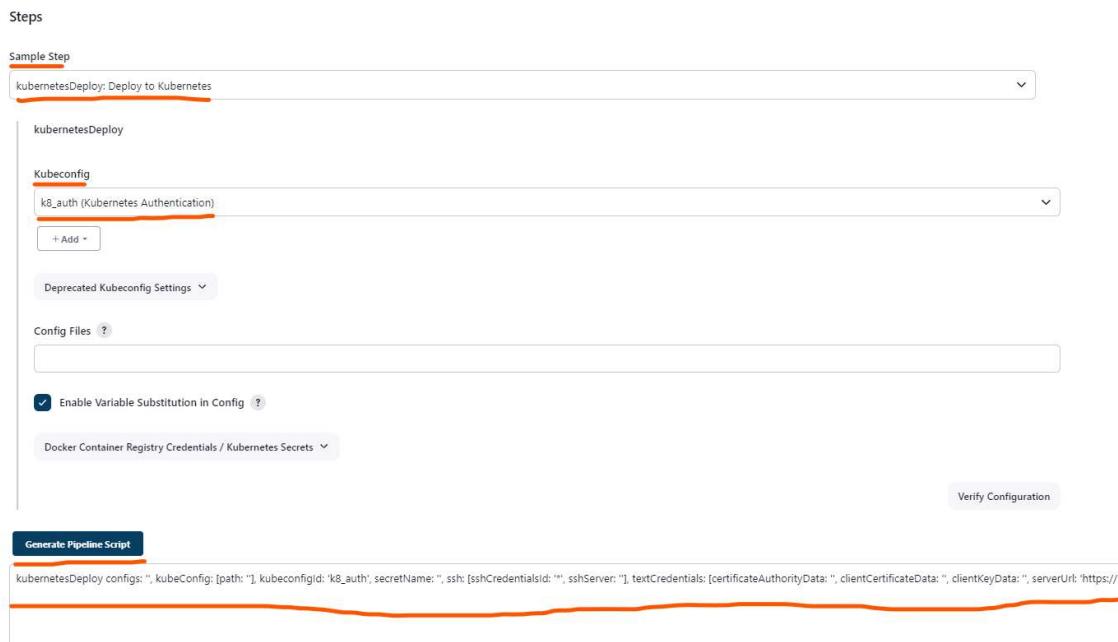
Pipeline script to allow authentication for Kubernetes deployment:

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 config: '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”

The screenshot shows the Jenkins dashboard at localhost:8080. On the left sidebar, there are links for 'New Item', 'Build History', 'Manage Jenkins', and 'My Views'. Below these are sections for 'Build Queue' (empty) and 'Build Executor Status' (1 idle, 2 idle). In the center, a table lists projects: 'DockerDemo' is highlighted with an orange border. At the bottom right of the dashboard is a 'Add description' button.

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

The screenshot shows the Jenkins job page for 'DockerDemo' at localhost:8080/job/DockerDemo/. The left sidebar has options: 'Status' (selected), 'Changes', 'Build Now' (highlighted with an orange border), 'Configure', 'Delete Pipeline', 'GitHub', 'Stages', 'Rename', 'Pipeline Syntax', and 'GitHub Hook Log'. The main content area is titled 'DockerDemo' and contains a description: 'This is a Demo project. The Jenkins CI/CD pipeline will fetch code from remote Git repository and do build and the deploy to Kubernetes (minikube)'. Below this is a 'Permalinks' section. At the bottom left, there is a 'Build History' card with the message 'No builds'.

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

The screenshot shows a browser window with the address bar displaying 'localhost:8080/job/DockerDemo'. The main content area is titled 'DockerDemo' and features a Jenkins logo. Below the title, there's a status bar with 'Status' highlighted. To the right, there's a brief description: 'This is a Demo project. The Jenkins'. On the left, there's a sidebar with various options: 'Changes', 'Build Now', 'Configure', 'Delete Pipeline', 'GitHub', 'Stages', 'Rename', 'Pipeline Syntax', and 'GitHub Hook Log'. On the right, there's a section titled 'Permalinks' and a 'Build History' card showing one build entry: '#1 May 27, 2024, 4:05 PM'.

Dashboard > DockerDemo >

Status

DockerDemo

Changes

This is a Demo project. The Jenkins

Build Now

Configure

Delete Pipeline

Permalinks

GitHub

Stages

Rename

Pipeline Syntax

GitHub Hook Log



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

The screenshot shows the Jenkins interface for the 'DockerDemo' project. At the top, there's a navigation bar with 'Build History' and a dropdown set to 'trend'. Below it is a search bar with 'Filter...' and a date range from 'May 27, 2024, 4:05 PM'. A link to 'Atom feed for all' is also present. The main content area has a back arrow, forward arrow, and a refresh icon. The URL 'localhost:8080/job/DockerDemo/' is shown in the address bar. The page title is 'Jenkins' with a logo. Below the title, the breadcrumb navigation shows 'Dashboard > DockerDemo >'. The main content area has tabs for 'Status' (selected) and 'DockerDemo'. Under 'Status', there are links for 'Changes' (with a dropdown menu showing 'Console Output' selected), 'Build Now', 'Edit Build Information', 'Delete build #1', 'Timings', 'Git Build Data', 'Pipeline Overview', 'Pipeline Console', 'Restart from Stage', 'Replay', 'Pipeline Steps', and 'Workspaces'. A 'Permalinks' section lists recent builds: 'Last build (#1), 2 min 1 sec ago', 'Last failed build (#1), 2 min 1 sec ago', 'Last unsuccessful build (#1), 2 min 1 sec ago', and 'Last completed build (#1), 2 min 1 sec ago'. At the bottom, there are links for 'Atom feed for all' and 'Atom feed for failures'.

You can see CICD pipeline build and deploy log here.

```

Started by user devops
Obtained Jenkinsfile from git https://github.com/sauvikdevops/dockerdemo
[Pipeline] start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/DockerDemo
[Pipeline] [
[Pipeline] stage
[Pipeline] {
[Pipeline] node
[Pipeline] {
[Pipeline] stage
[Pipeline] {
[Pipeline] checkout
Selected Git installation does not exist. Using Default
The recommended git tool is: NONE
No credentials specified
> git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/DockerDemo/.git # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/sauvikdevops/dockerdemo # timeout=10
Fetching upstream changes from https://github.com/sauvikdevops/dockerdemo
> git --version # timeout=10
> git --version # git version 2.43.0
> git fetch --tags --force --progress -- https://github.com/sauvikdevops/dockerdemo +refs/heads/*:refs/remotes/origin/* # timeout=10
> git rev-parse refs/remotes/origin/main^{commit} # timeout=10
Checking out Revision 7a34efcfe7a878582277ad38947818a (refs/remotes/origin/main)
> git config core.sparsecheckout # timeout=10
> git checkout -f 7a34efcfe7a878582277ad38947818a # timeout=10
git: checked out '7a34efcfe7a878582277ad38947818a' # timeout=10
Commit message: "updated msg"
> git rev-list --no-walk 7a34efcfe7a878582277ad38947818a # timeout=10
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] }
[Pipeline] stage
[Pipeline] {
[Pipeline] [
[Pipeline] checkout
[Pipeline] {

```

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
```

```

devops@jenkins-minikube:~$ minikube dashboard --url
* Verifying dashboard health ...
* Launching proxy ...
* Verifying proxy health ...
http://127.0.0.1:43317/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/

```

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:

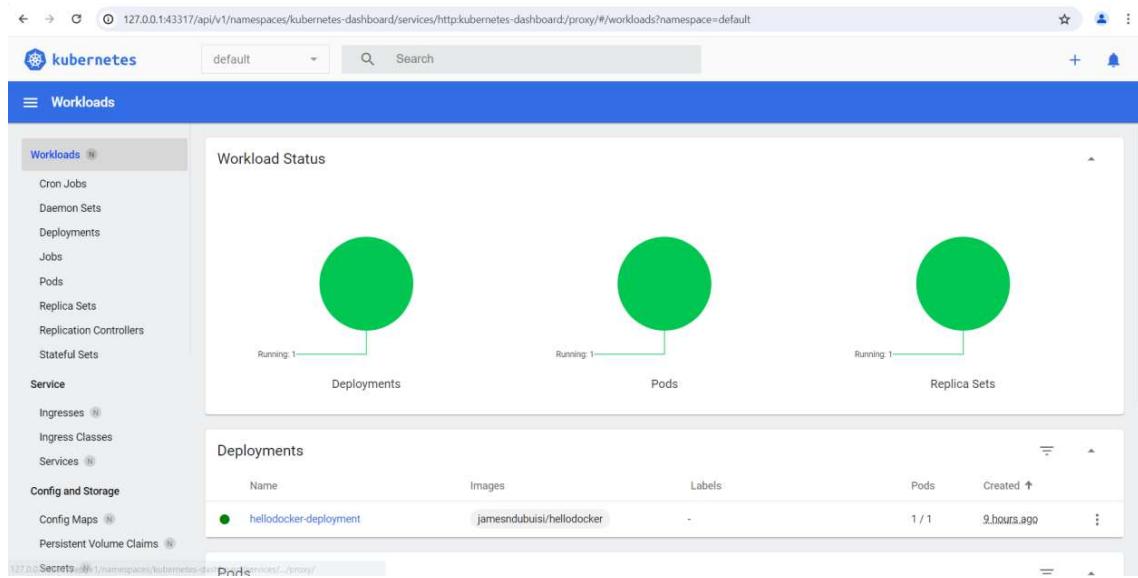
In another terminal window (Gitbash) from local machine, run this command to connect to Ubuntu server:

```
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
helldocker-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/helldocker-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 | helldocker-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 helldocker-service

```
devops@jenkins-minikube:~$ minikube service --url helldocker-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/helldocker-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:

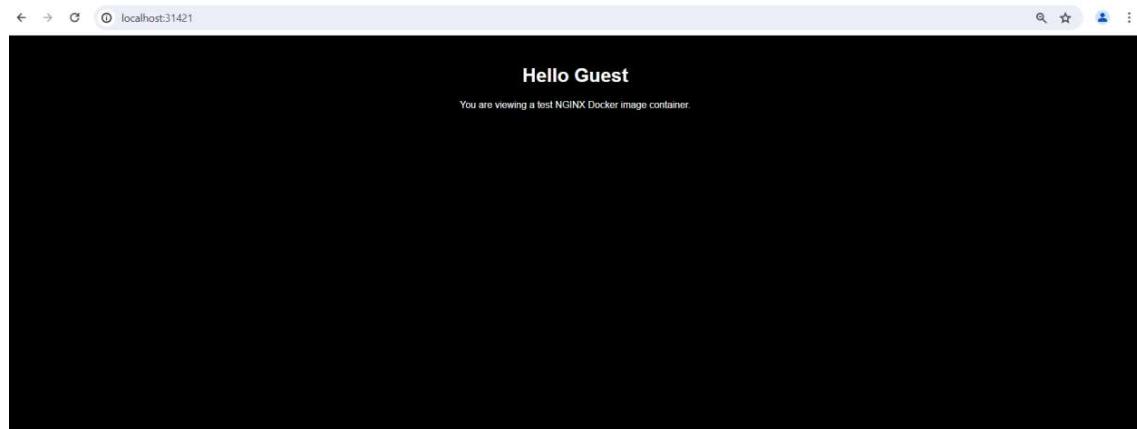
```
sudo lsof -i -P -n | grep LISTEN
```

```
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.