

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

By
Sauvik Maiti

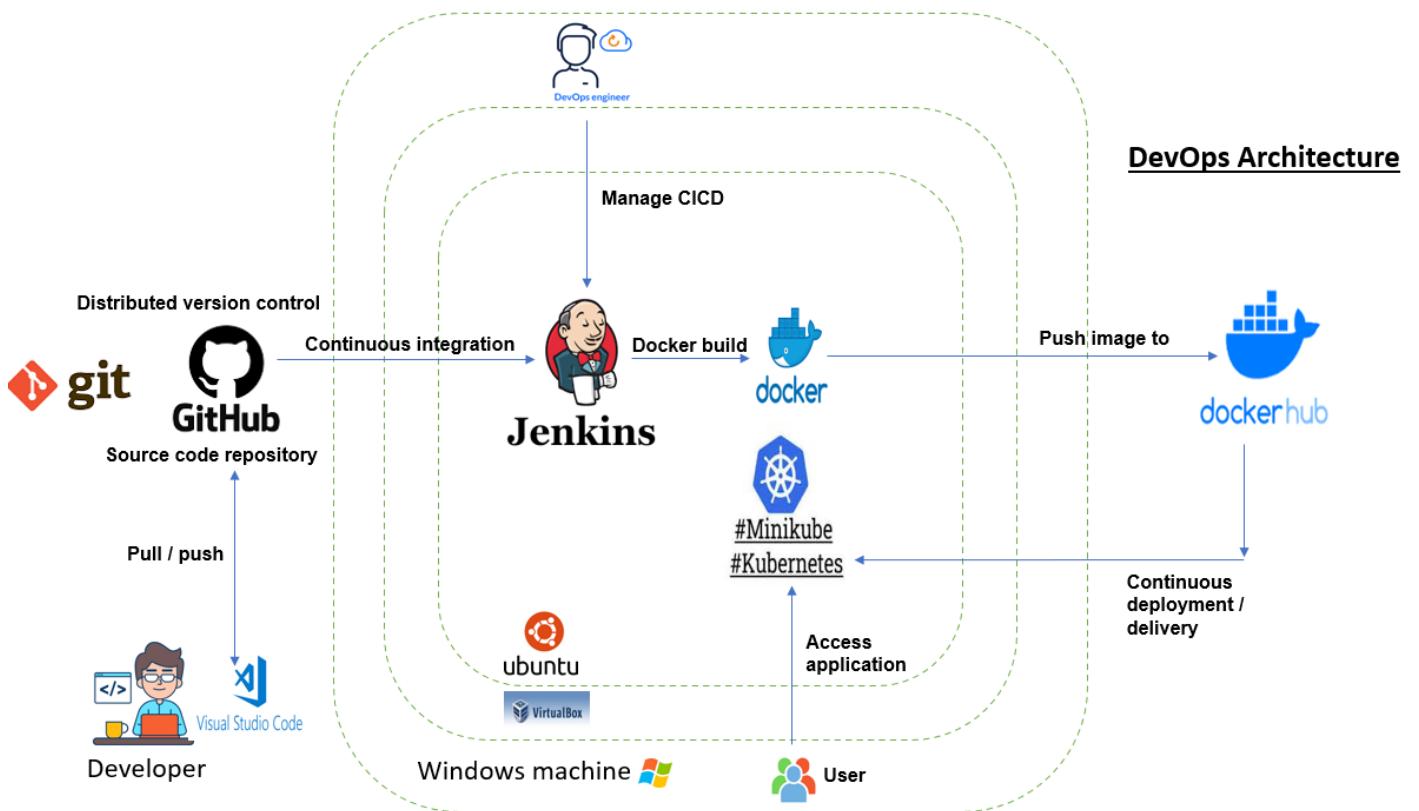
Table of Contents

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

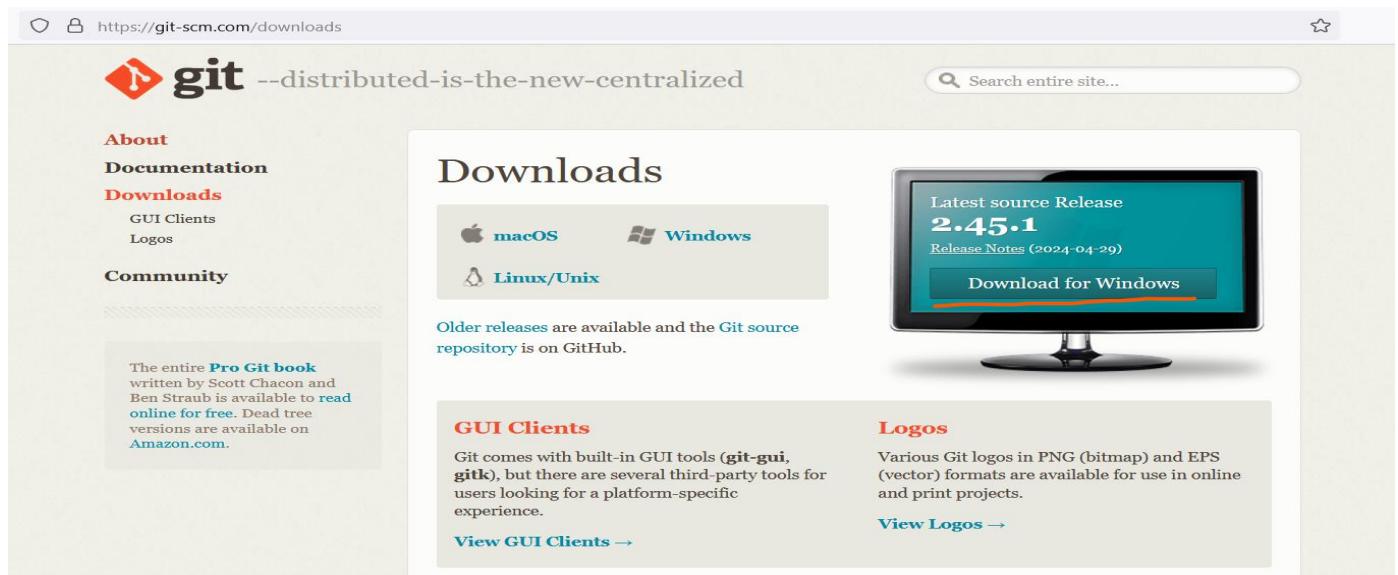


1. Local machine / host OS = Windows; Oracle VirtualBox is installed on Windows; Ubuntu as guest OS is installed on Oracle VirtualBox.
Local machine (Host Windows) --> Oracle Virtual Box --> Ubuntu
2. Jenkins, docker and minikube (Kubernetes) are installed on Ubuntu guest OS.
3. GitHub and DockerHub 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

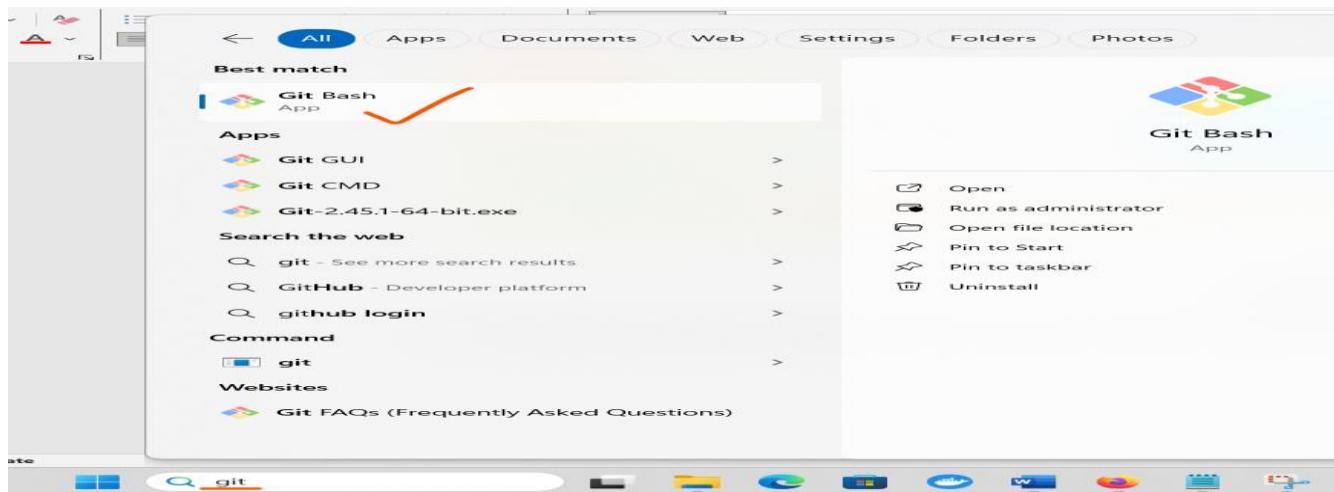
Download Git from the below URL in your local machine.

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

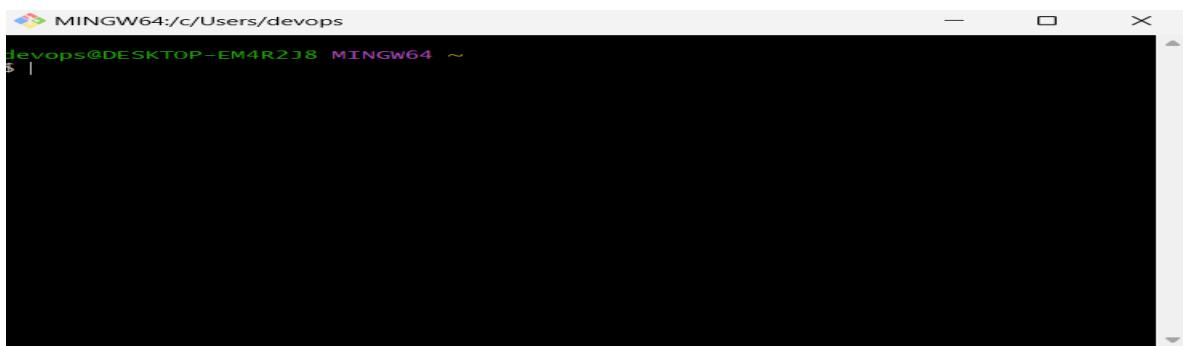


The screenshot shows the official Git website at https://git-scm.com/downloads. The main navigation menu includes About, Documentation, Downloads (selected), and Community. The Downloads section features a large "Downloads" heading with three main categories: macOS, Windows, and Linux/Unix. Below these, a note states: "Older releases are available and the Git source repository is on GitHub." To the right, there's a section for "Logos" with a link to "View Logos →". A prominent feature is a monitor displaying the latest source release "2.45.1" with a "Download for Windows" button.

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.



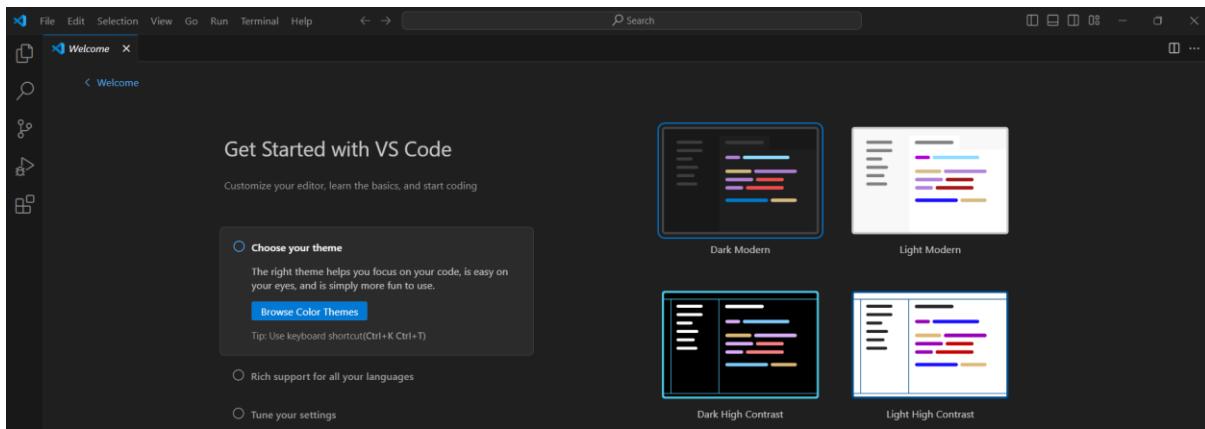
Install & configure Visual Studio Code on local machine

Download and install Visual Studio Code from the below link:

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

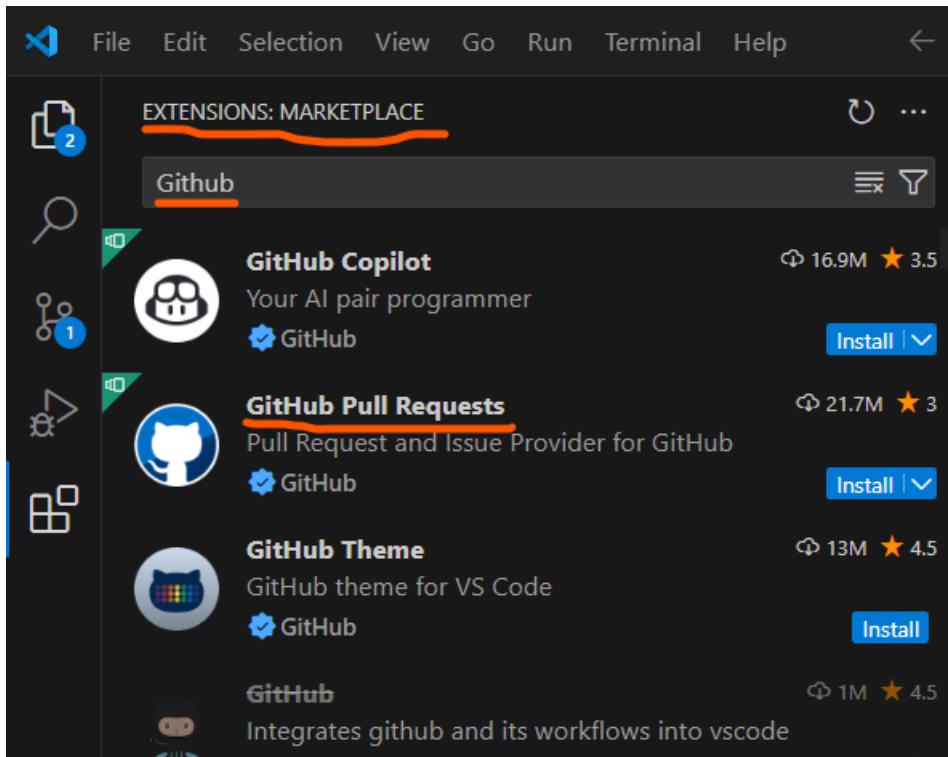
A screenshot of the official Visual Studio Code download page. At the top, there is a navigation bar with links for "Docs", "Updates", "Blog", "API", "Extensions", "FAQ", and "Learn". To the right of the navigation bar is a search bar labeled "Search Docs" and a blue "Download" button. Below the navigation bar, a message states "Version 1.89 is now available! Read about the new features and fixes from April." The main content area features the title "Visual Studio Code" and a sub-headline "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 operating system logo (Windows logo, Tux logo, and Apple logo) to its left. Under each button, there is a table of download links for different architectures (x64, Arm64, etc.) and file formats (.deb, .rpm, .tar.gz, Snap, CLI).

A detailed view of the Visual Studio Code download page for different operating systems. It shows icons for Windows, Linux (Tux), and Mac (Apple). For Windows, there are links for User Installer, System Installer (.zip), and CLI, with options for x64 and Arm64 architectures. For Linux, there are links for .deb, .rpm, .tar.gz, and Snap, with options for x64, Arm32, and Arm64 architectures. For Mac, there are links for .zip, .Intel chip, .Apple silicon, and Universal, with options for Intel chip and Apple silicon architectures. The CLI section also includes links for x64, Arm32, and Arm64 architectures.



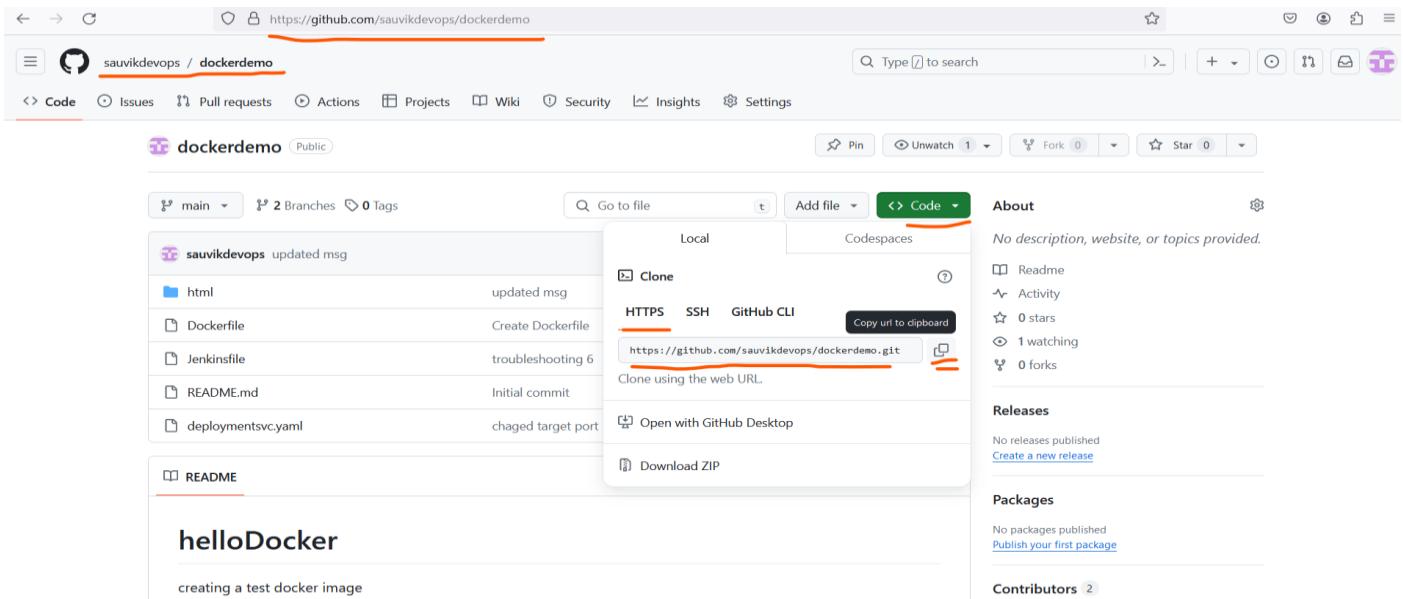
Clone a repository locally

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



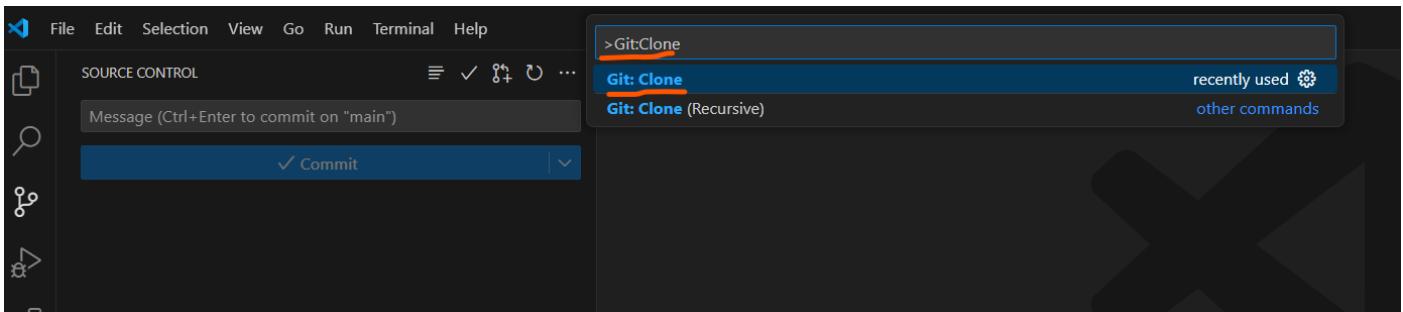
Select GitHub Pull Requests and click on install.

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:



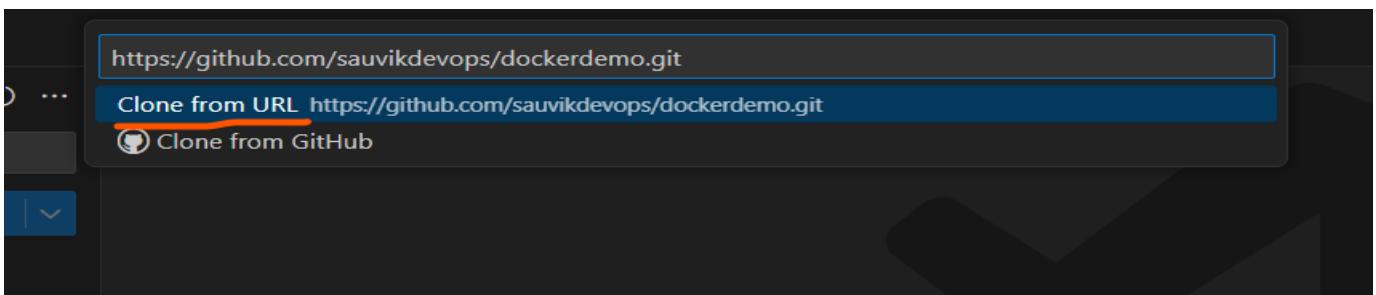
The screenshot shows a GitHub repository page for 'sauvikdevops / dockerdemo'. The URL 'https://github.com/sauvikdevops/dockerdemo' is highlighted in red at the top of the browser. In the 'Code' dropdown menu on the right, the 'HTTPS' tab is selected, and the URL 'https://github.com/sauvikdevops/dockerdemo.git' is highlighted in red. A 'Copy url to clipboard' button is visible next to it.

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



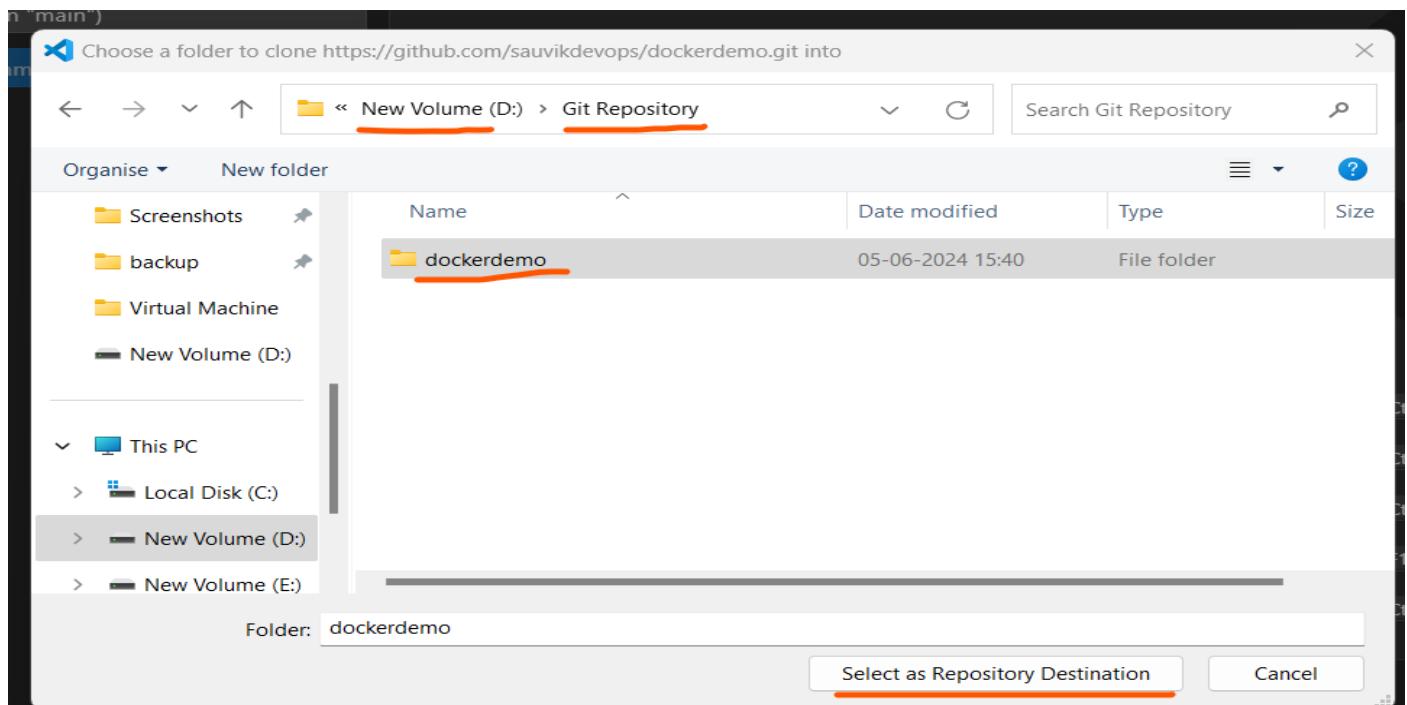
The screenshot shows the Visual Studio Code interface with the command palette open. The search bar contains the text '>Git:Clone'. Below it, the 'Git: Clone' command is highlighted in blue, indicating it is selected. Other options like 'Git: Clone (Recursive)' are also listed.

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

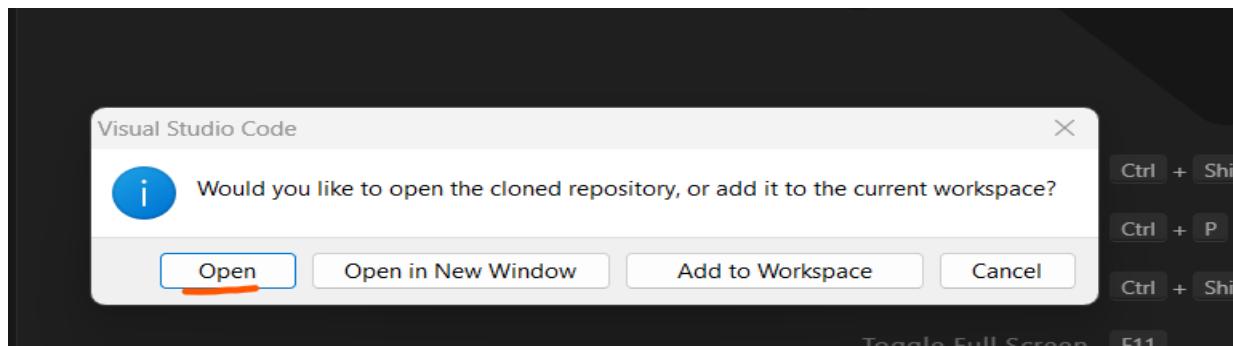


The screenshot shows the Visual Studio Code interface with the command palette open. The search bar contains the URL 'https://github.com/sauvikdevops/dockerdemo.git'. Below it, the 'Clone from URL https://github.com/sauvikdevops/dockerdemo.git' option is highlighted in blue, indicating it is selected. Other options like 'Clone from GitHub' are also listed.

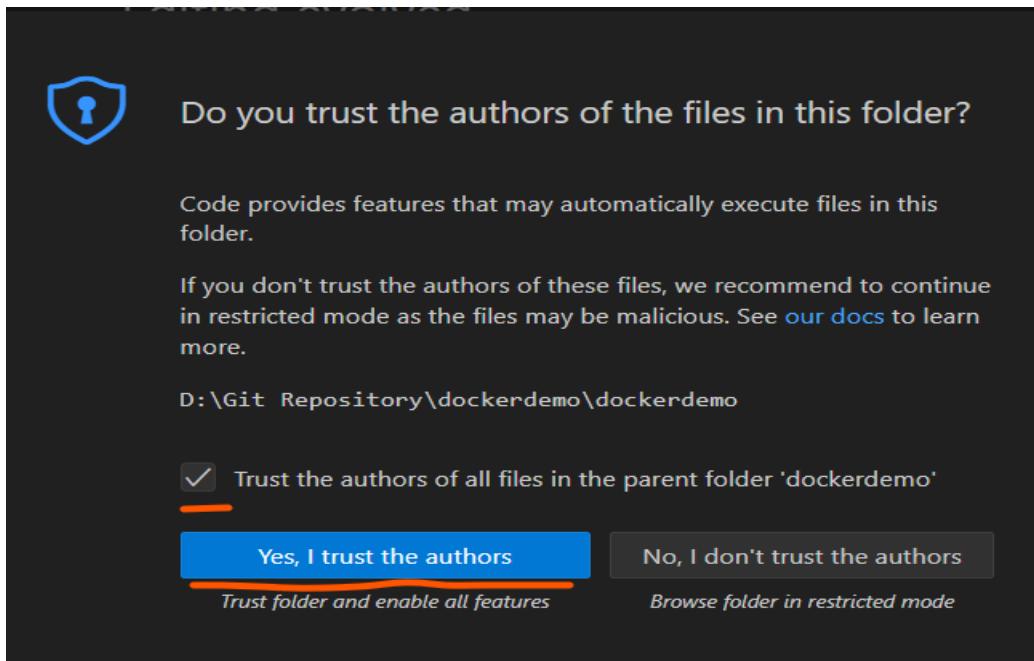
It will now ask for Repository Destination i.e. the folder in your local computer where the Github repo would be cloned. For example I have created a folder called “deckerdemo” (usually we keep it as our project name) inside D:\Git Repository folder. You can keep it anywhere you would like.



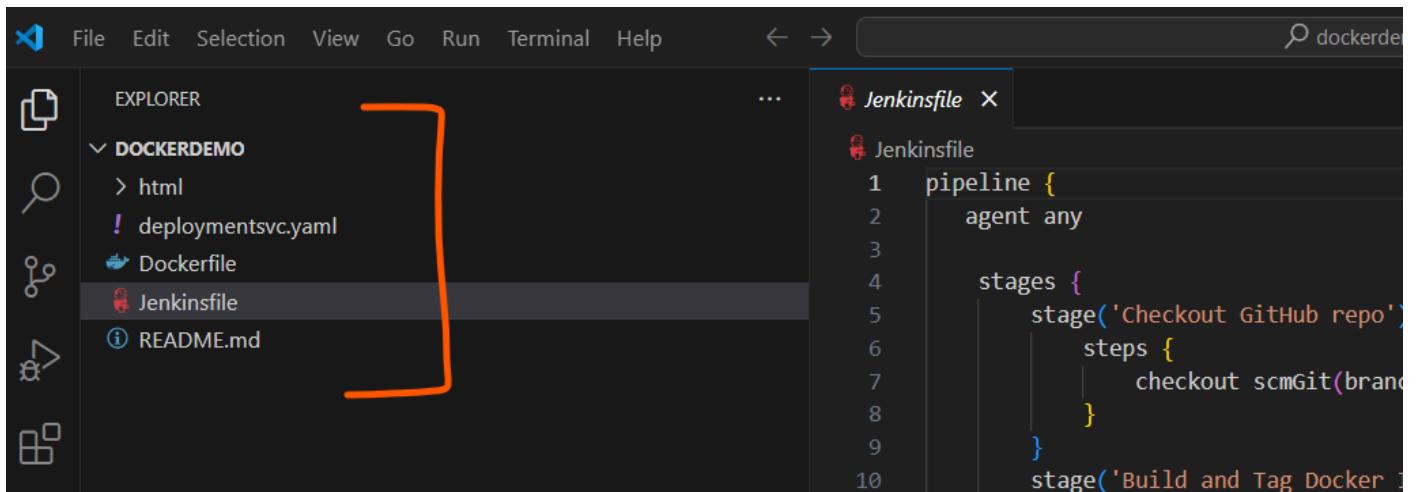
Click on “Open” if it asks. You can also add it into current VsCode workspace as well.



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

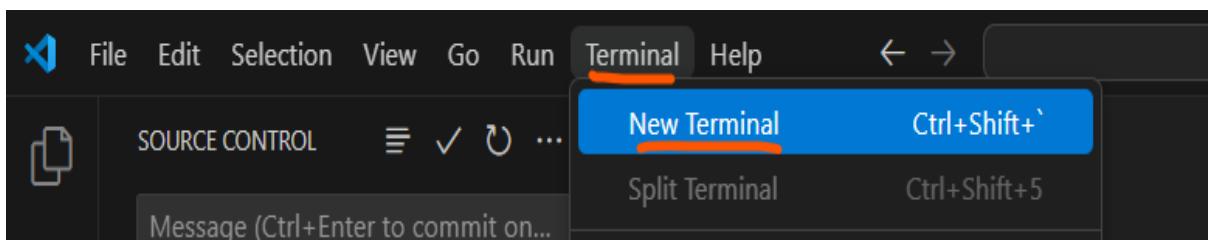


Now, you should be able to see the cloned repo in local.



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"
```

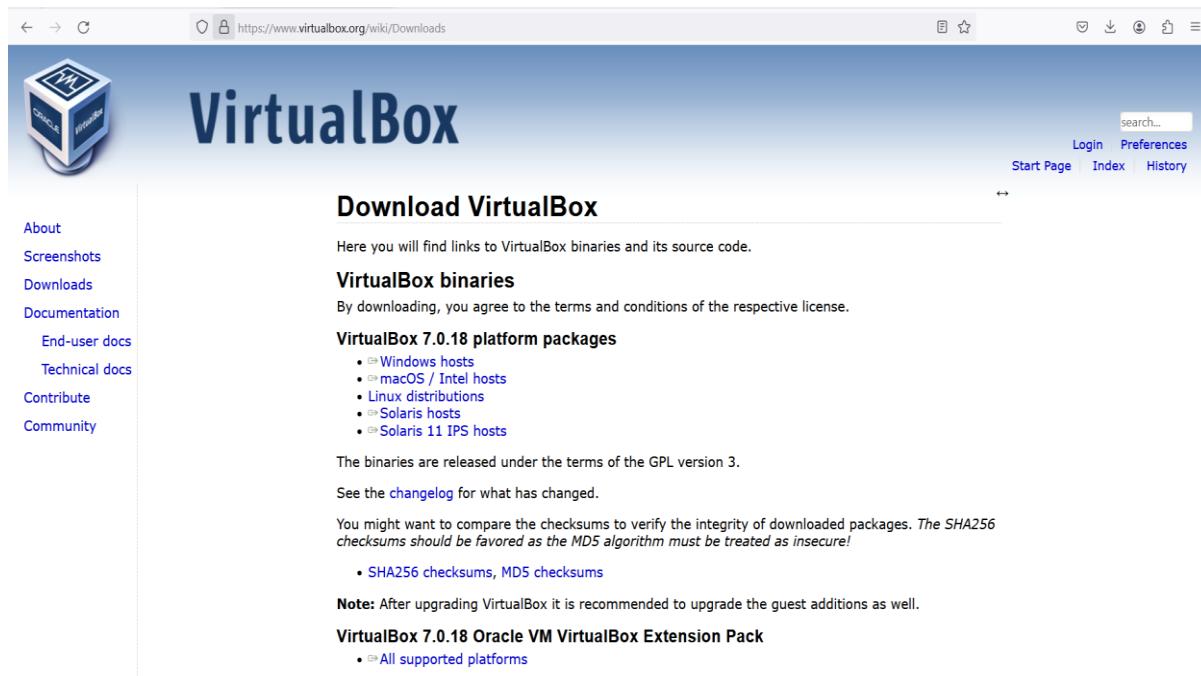
Note: Replace my email with your.

```
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> [REDACTED]
```

Install VirtualBox on local machine

Download Oracle VirtualBox from the below URL

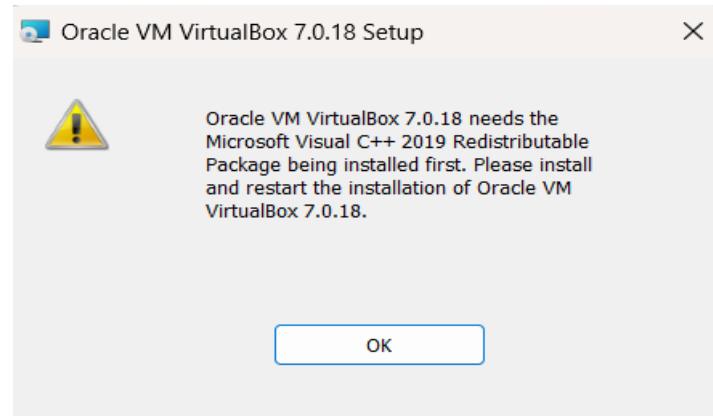
<https://www.virtualbox.org/wiki/Downloads>



The screenshot shows a web browser window with the URL <https://www.virtualbox.org/wiki/Downloads> in the address bar. The page title is "VirtualBox". On the left, there is a sidebar with links: About, Screenshots, Downloads, Documentation, End-user docs, Technical docs, Contribute, and Community. The main content area is titled "Download VirtualBox". It says "Here you will find links to VirtualBox binaries and its source code." Below this is a section for "VirtualBox binaries" with a note: "By downloading, you agree to the terms and conditions of the respective license." A list of "VirtualBox 7.0.18 platform packages" is provided, including Windows hosts, macOS / Intel hosts, Linux distributions, Solaris hosts, and Solaris 11 IPS hosts. A note states: "The binaries are released under the terms of the GPL version 3. See the [changelog](#) for what has changed." Another note says: "You might want to compare the checksums to verify the integrity of downloaded packages. The SHA256 checksums should be favored as the MD5 algorithm must be treated as insecure!" A note at the bottom says: "Note: After upgrading VirtualBox it is recommended to upgrade the guest additions as well." At the very bottom, there is a link for "VirtualBox 7.0.18 Oracle VM VirtualBox Extension Pack" which includes "All supported platforms".

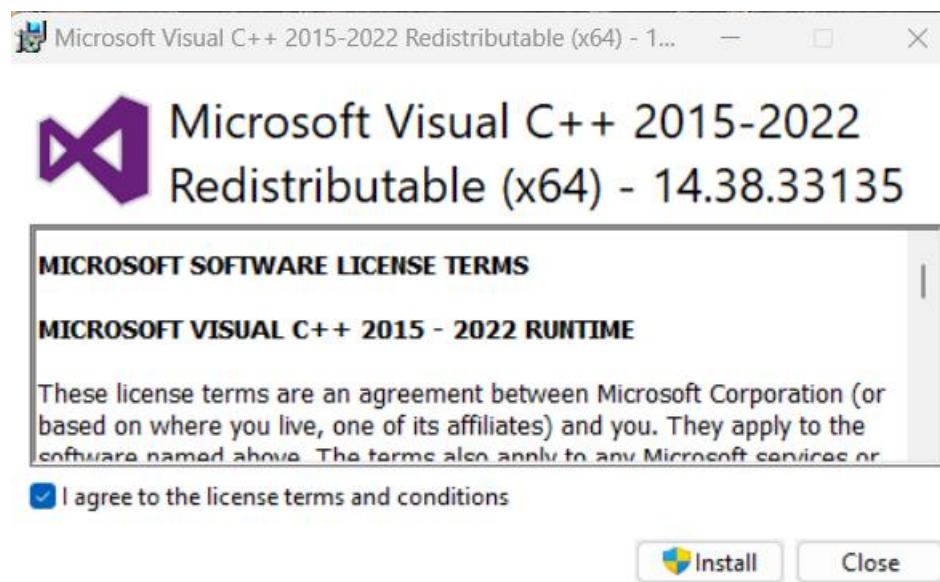
Install Oracle VM VirtualBox pre-requisites

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

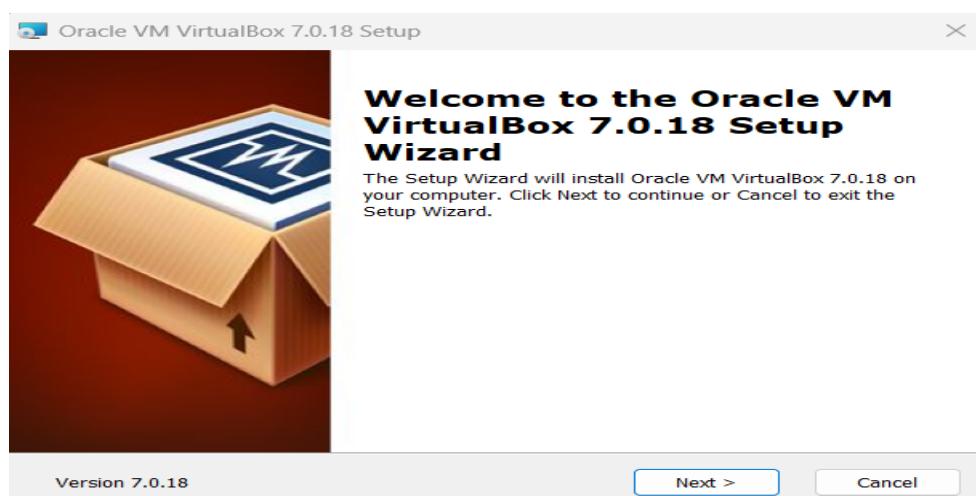


Download and Install Oracle VirtualBox from this link:

<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>

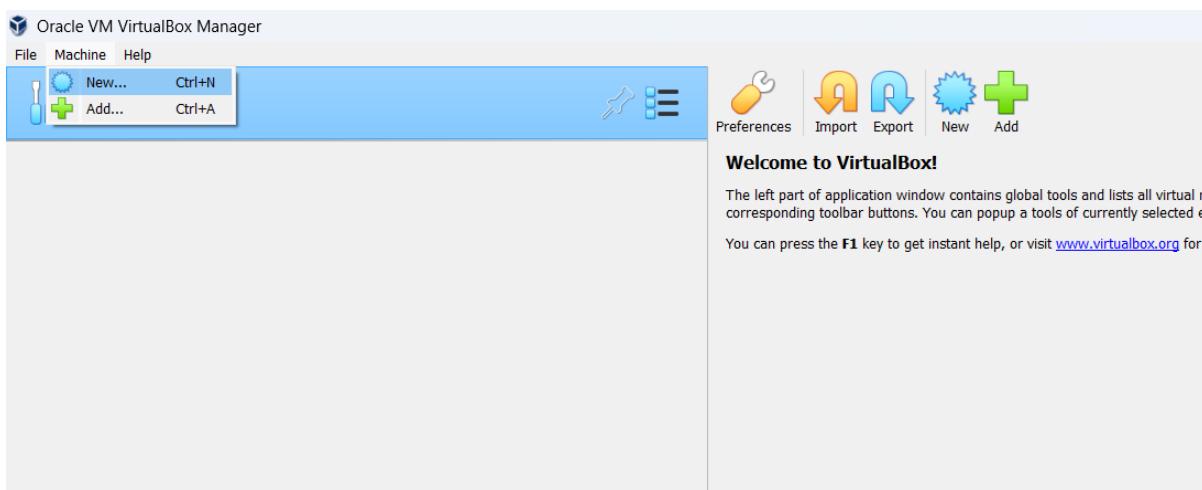
The screenshot shows the Ubuntu Server download page at <https://ubuntu.com/download/server>. The main heading is "Get Ubuntu Server". Below it, there are three tabs: "Manual installation" (selected), "Instant VMs", and "Automated provisioning". Under the "Manual installation" tab, the "Ubuntu 24.04 LTS" section is highlighted. It features a large orange crown icon. To the right, a green button says "Download 24.04 LTS" (2.7GB). Below the button are links for "Alternative downloads" and "Alternative architectures". At the bottom of the section are links for "What's new", "System requirements", and "How to install".

Click on Download to proceed.

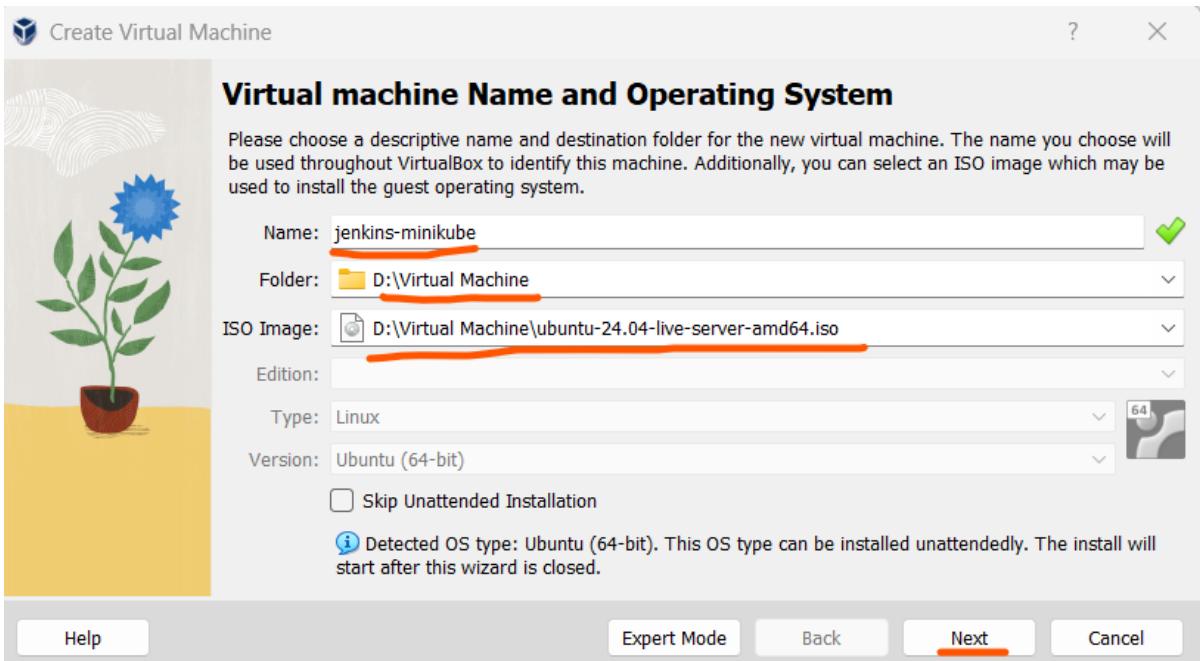
Note: For your lab, you do not need to download this Ubuntu ISO on every machine separately. Just download once and then copy to all the machines.

Create VM inside Oracle VirtualBox

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

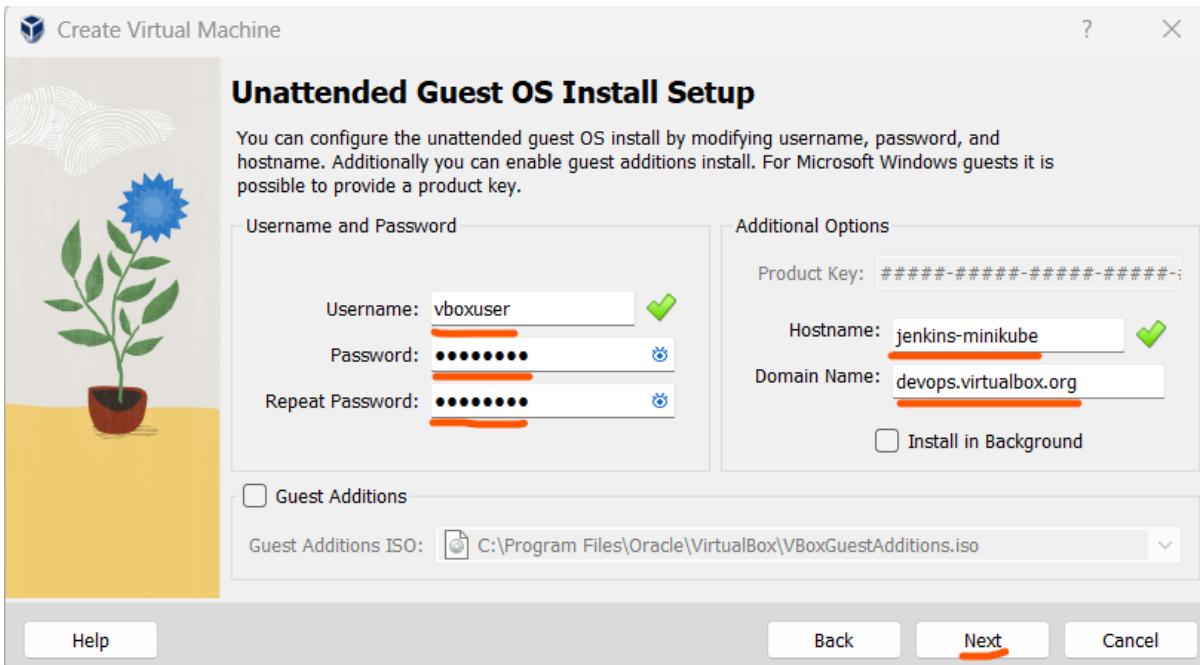


Put machine name of choice. I have given – jenkins-minikube and the folder location in your local machine where the Virtual machine including the disks would be created and location of Ubuntu ISO image which you have downloaded.

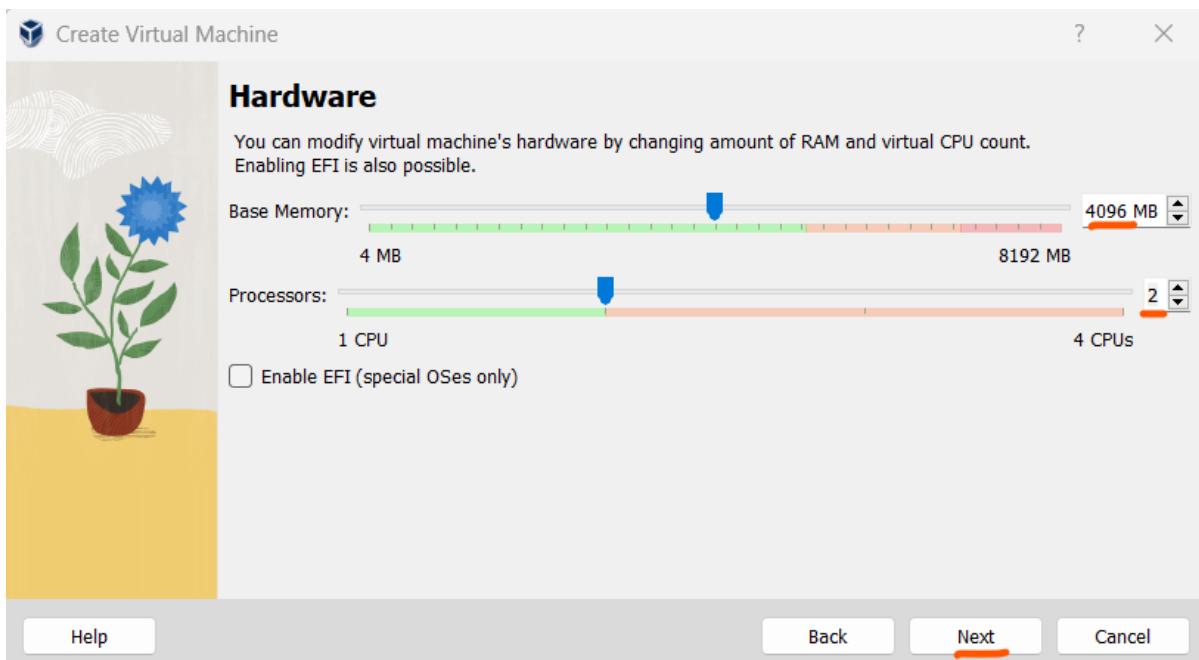


Click Next to proceed.

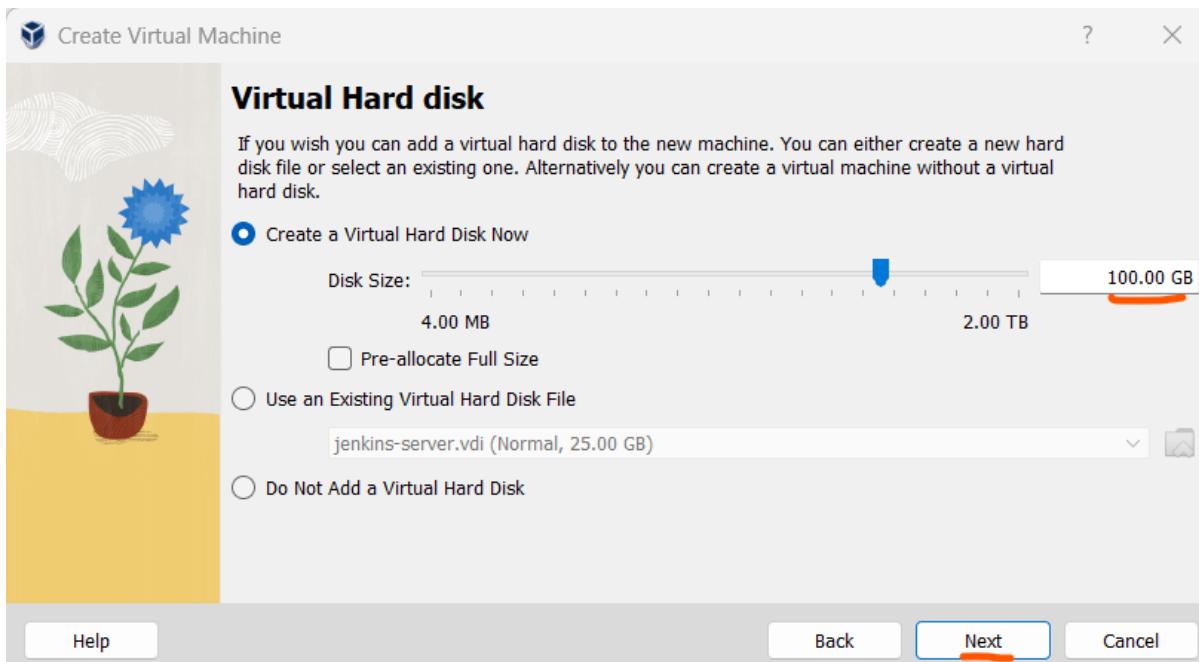
In the next screen, put Ubuntu username of your choice followed by password and confirmation of this new password. Also, put a hostname of this ubuntu guest OS and domain name. For this example, I have given domain name as “devops-virtualbox.org”.



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

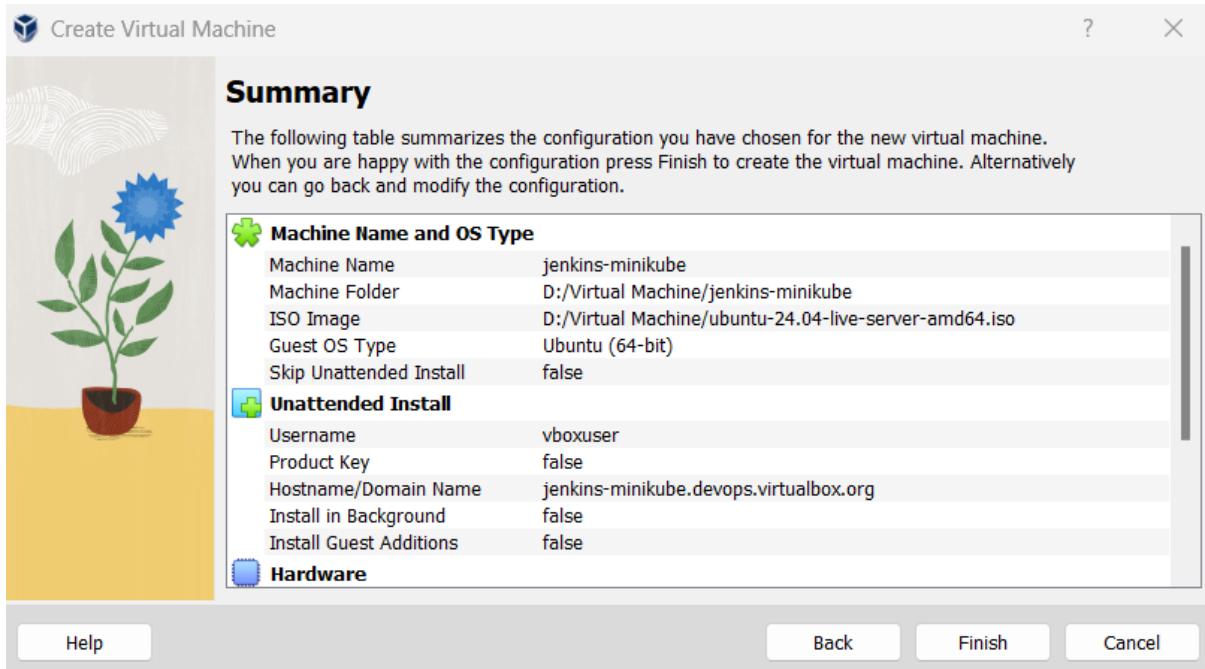


Click Next to proceed.



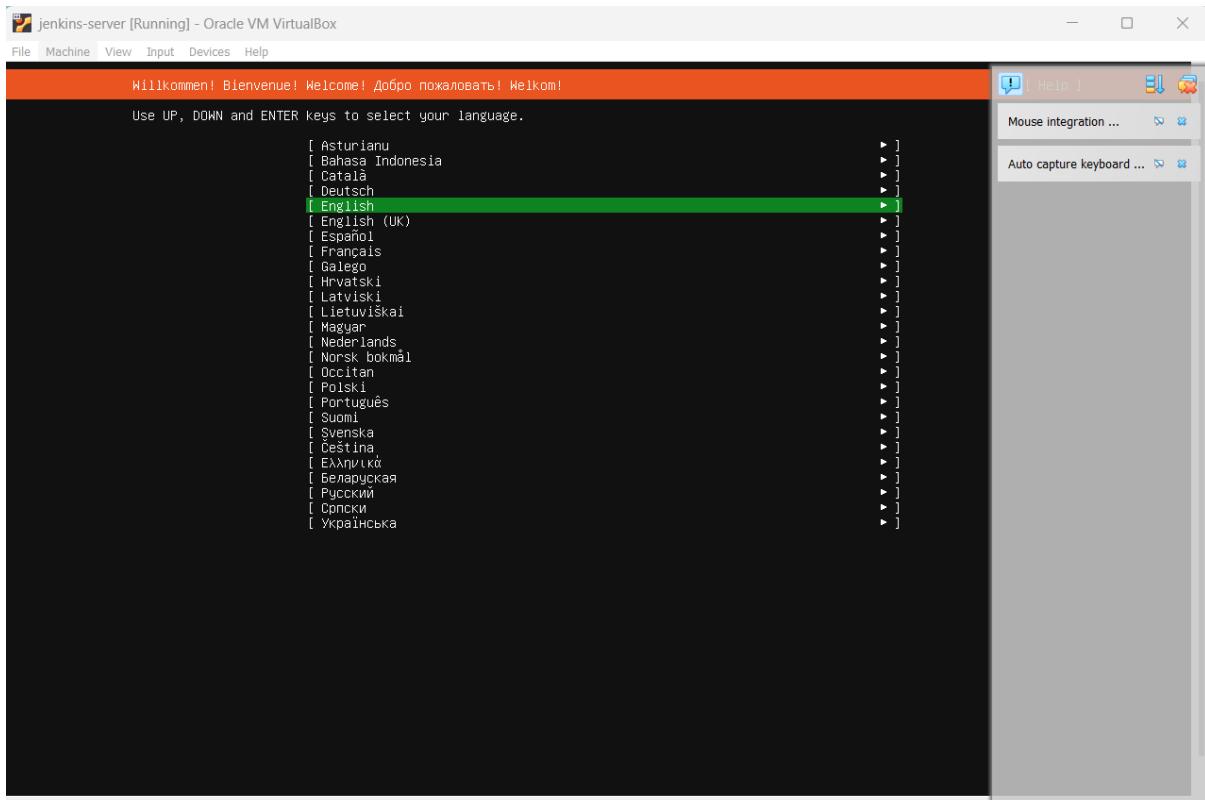
Set the virtual hard disk for this Ubuntu guest OS. I would recommend to have atleast 100 GB.

Click Next to proceed.

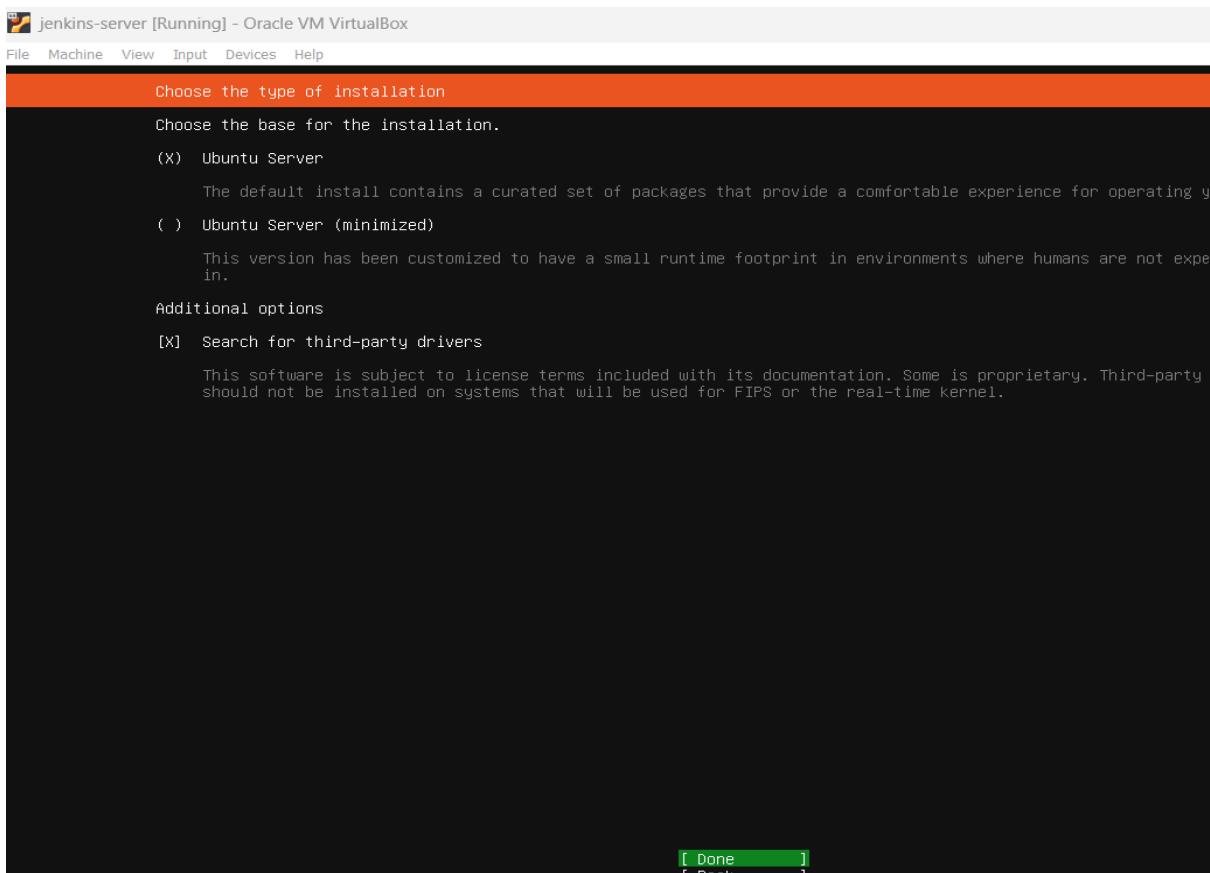
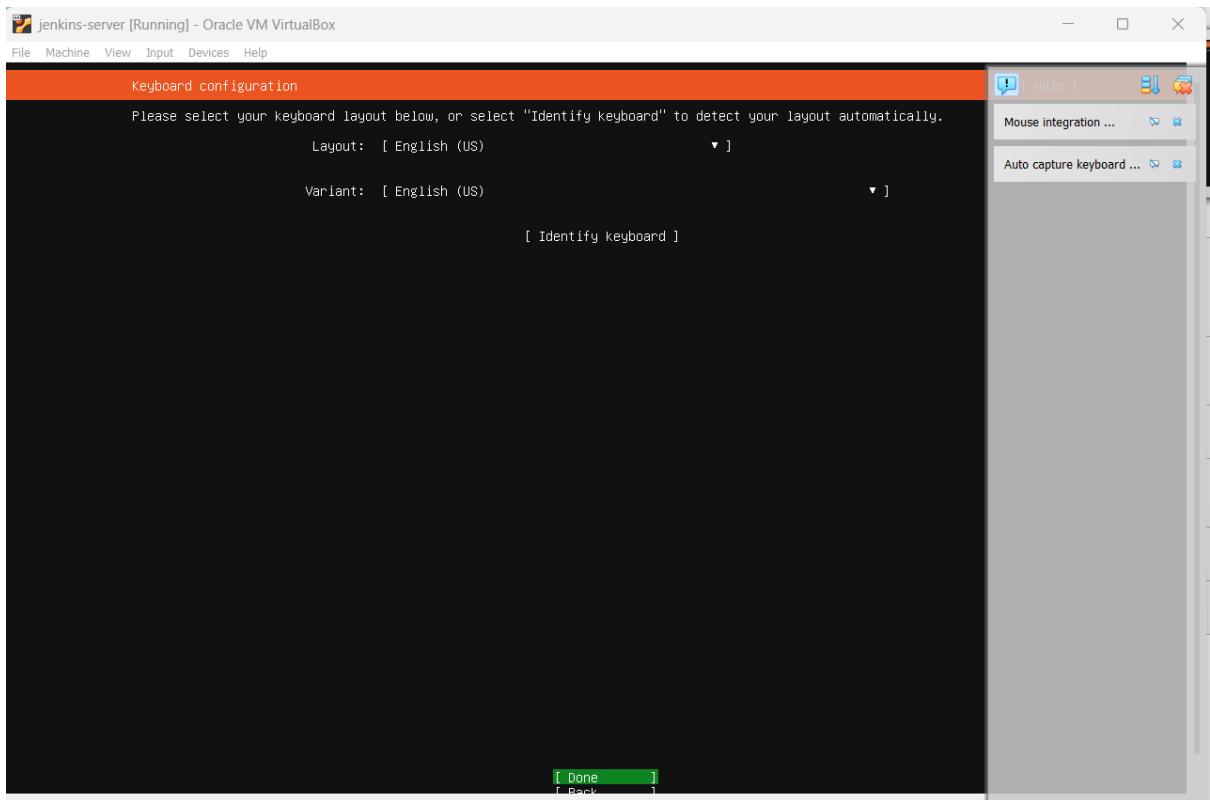


In the Summary page, just verify the settings you have set and click on Finish. It would then ask you for few preferences.

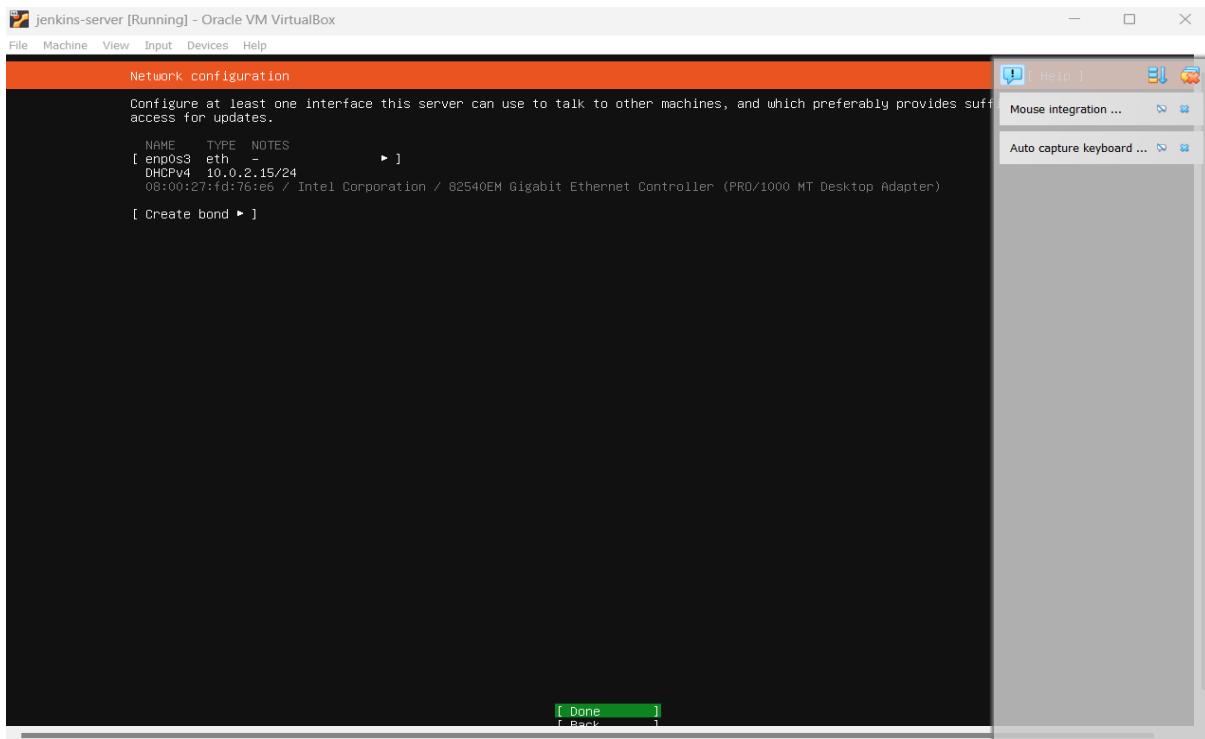
Select (or keep as it is if already selected) English as language and press enter to go to next screen.



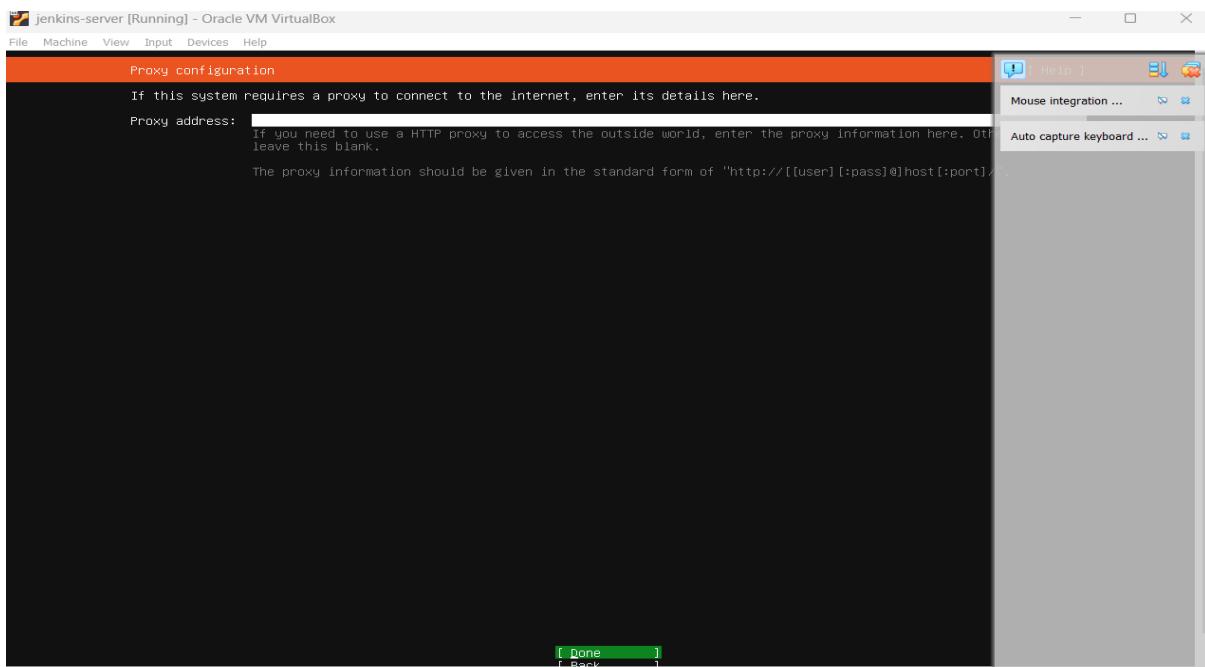
Leave the default keyboard layout and variant as it is and press enter to go to next.



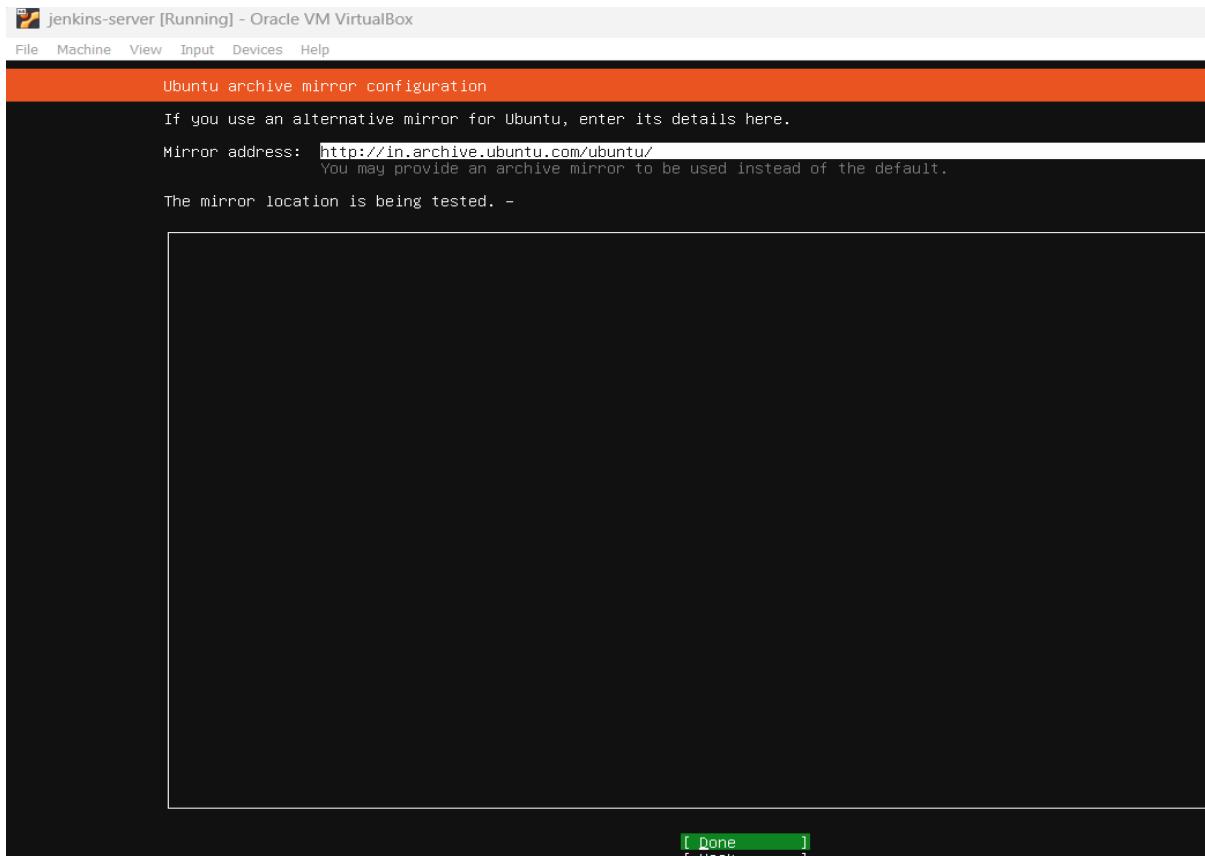
Select (or leave it if already selected) “Ubuntu Server” and press enter on “Done” to proceed.



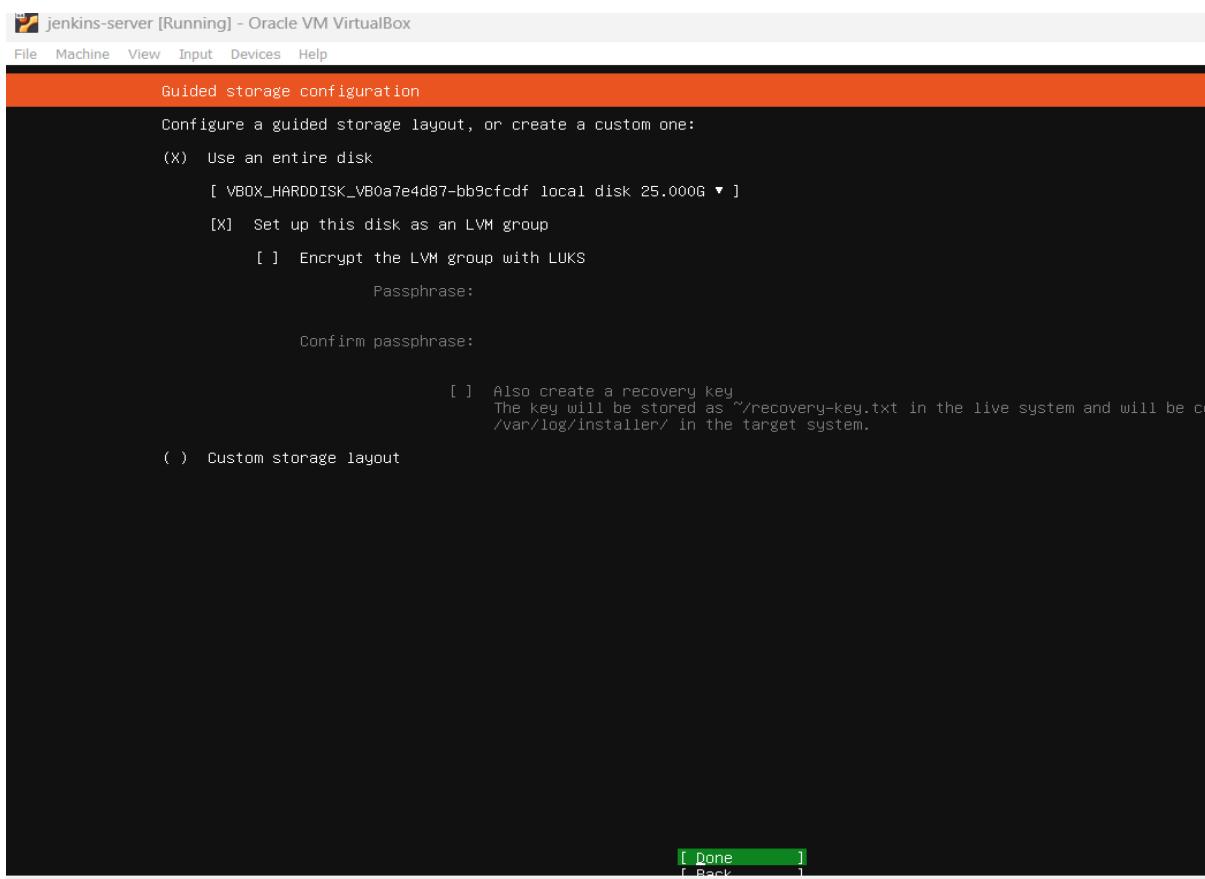
Leave the default network configuration and press enter on “Done” to proceed.



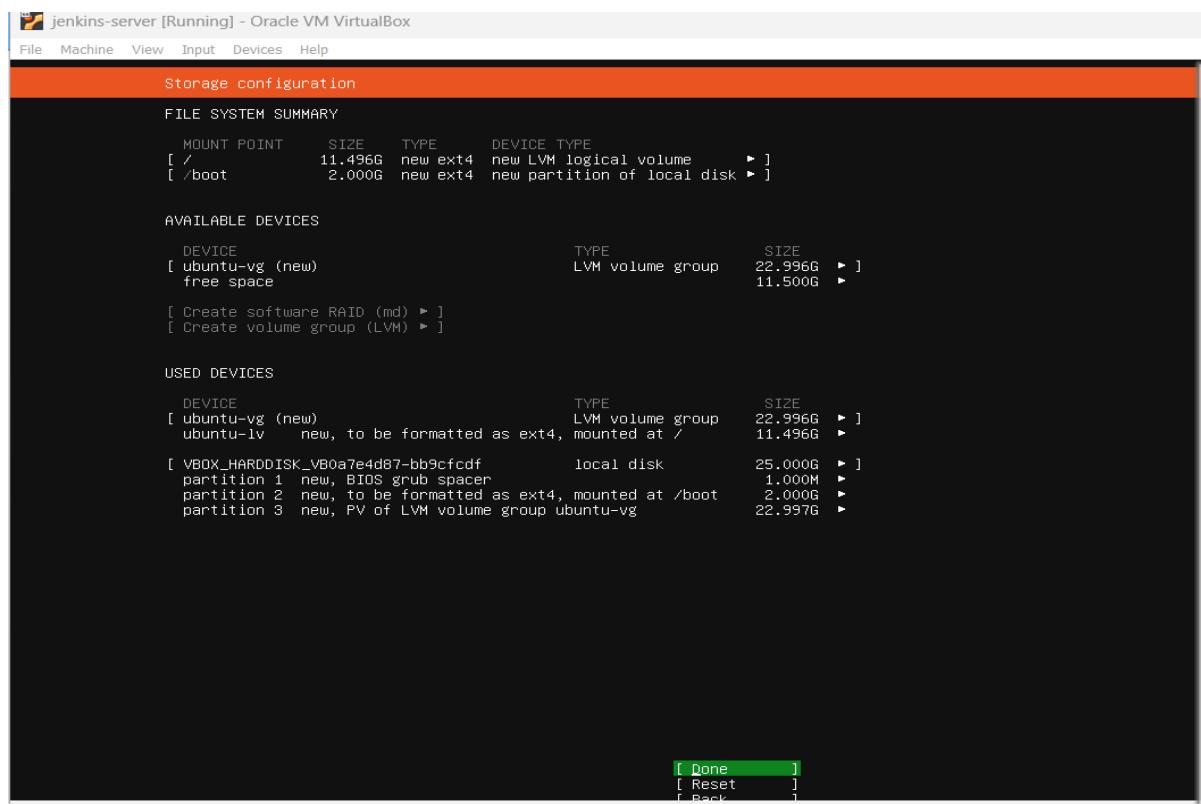
Leave the default Proxy configuration and press enter on “Done” to proceed.



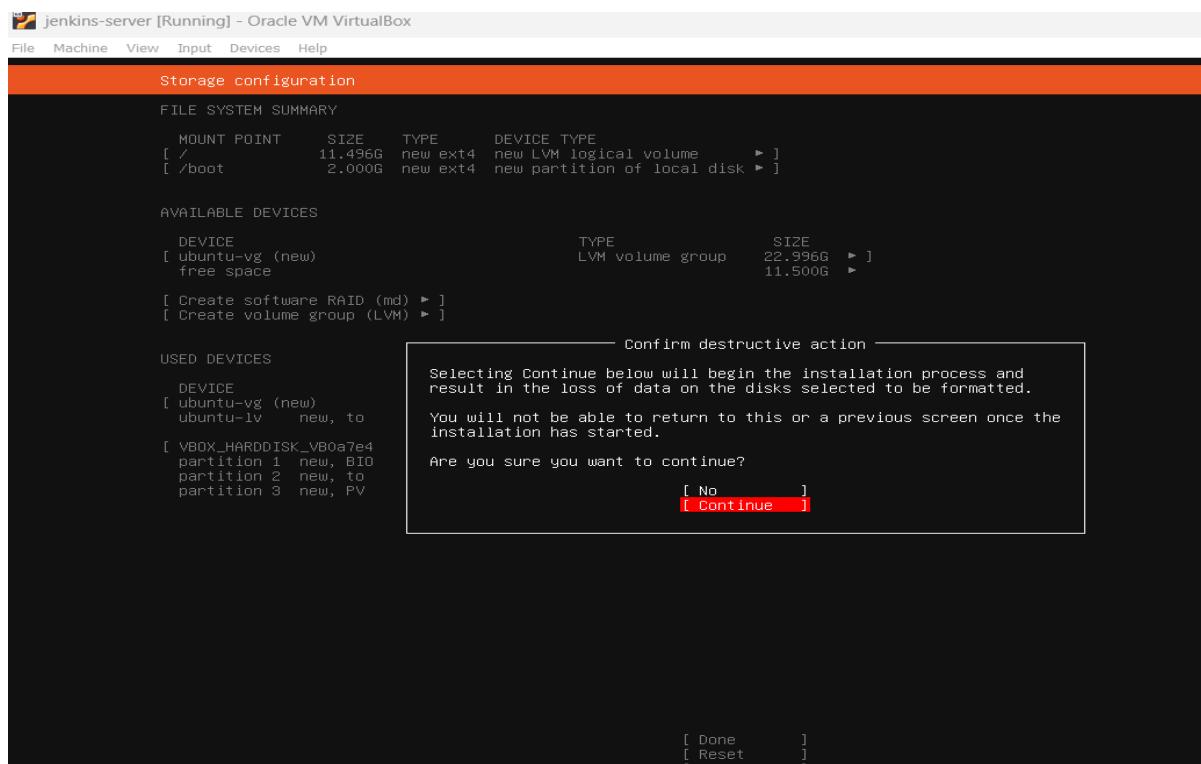
Leave the default Ubuntu archive mirror configuration and press enter on “Done” to proceed.



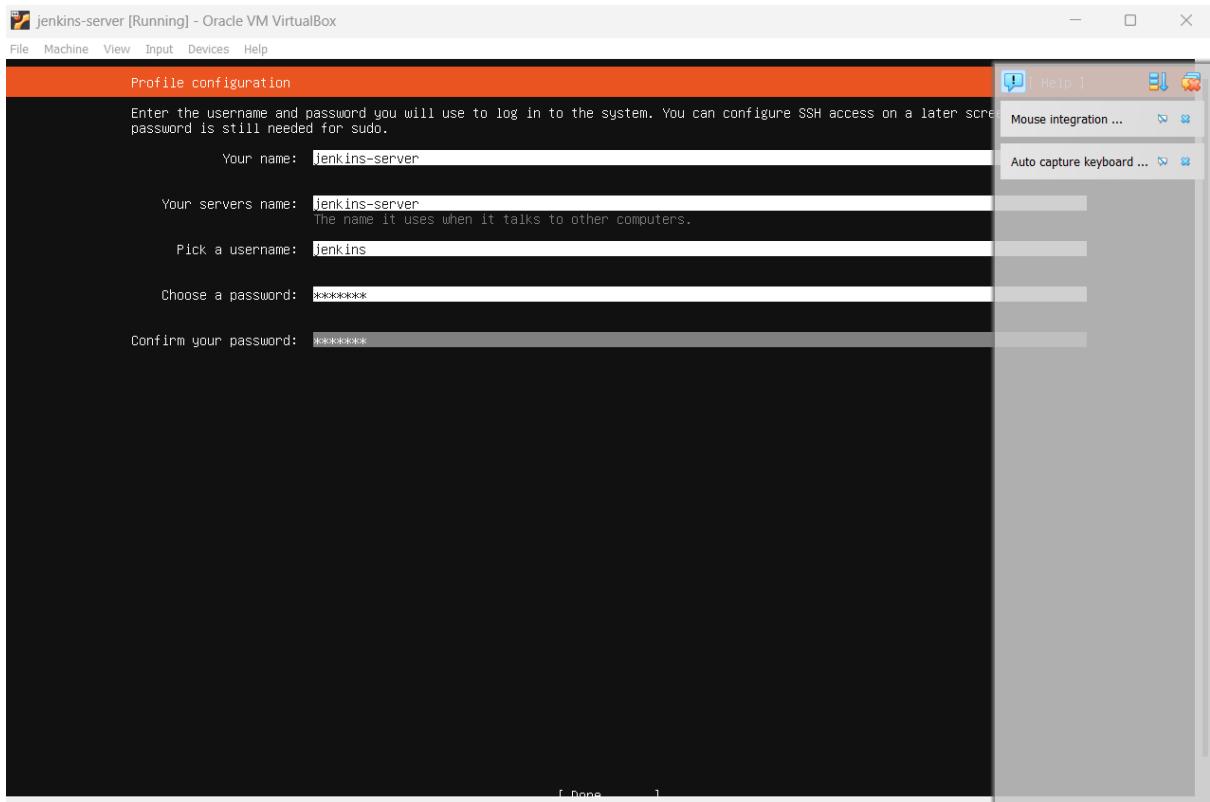
Leave the default storage configuration and press enter on “Done” to proceed.



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

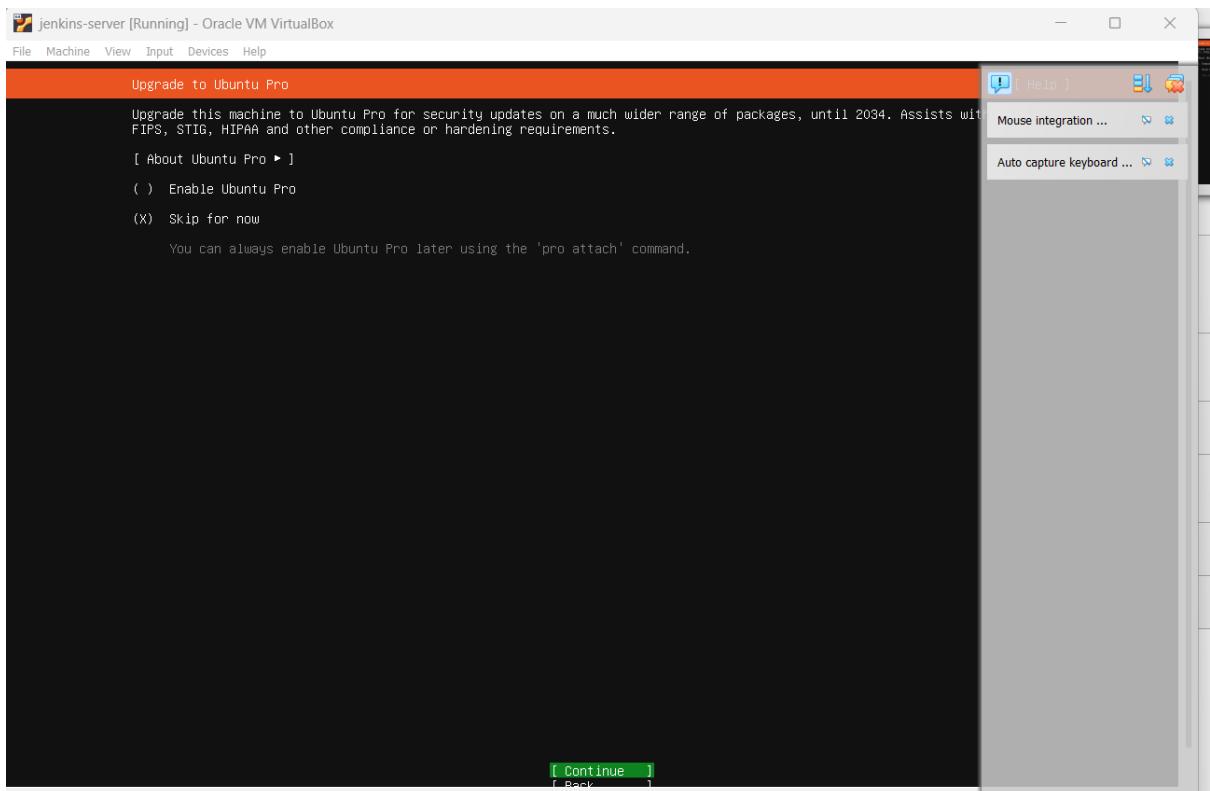


Click on Continue.

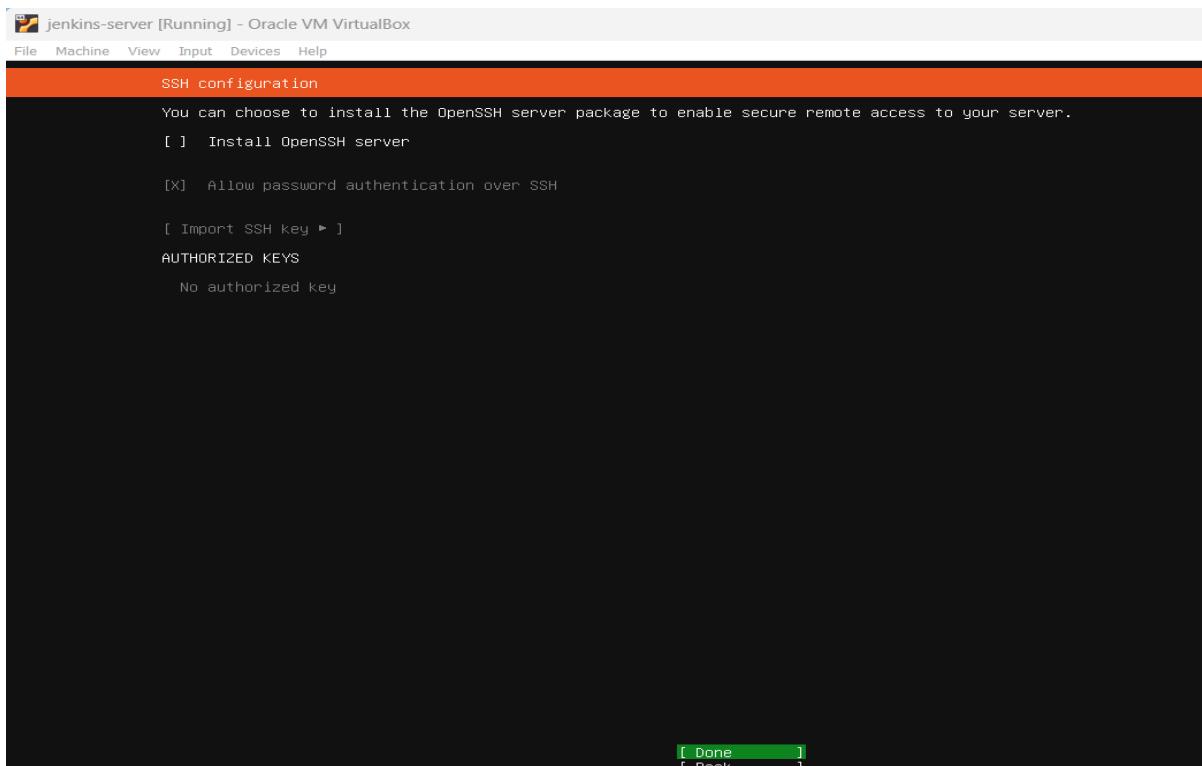


Put your name (any name), your server's name (I have put jenkins-minikube-server), give a username (I have put devops) and password.

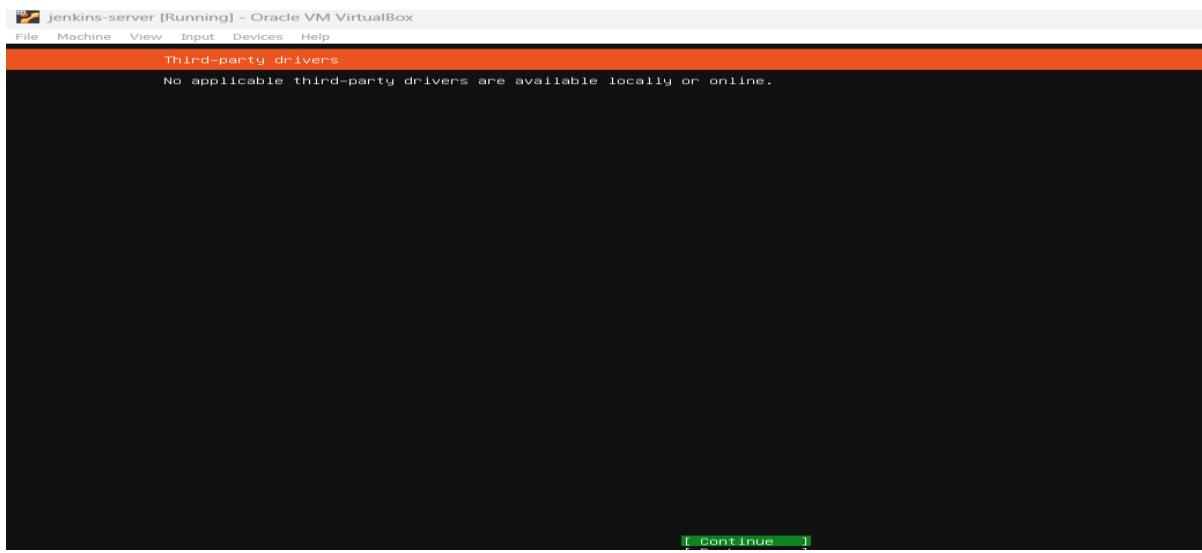
Press enter on “Done” to proceed.



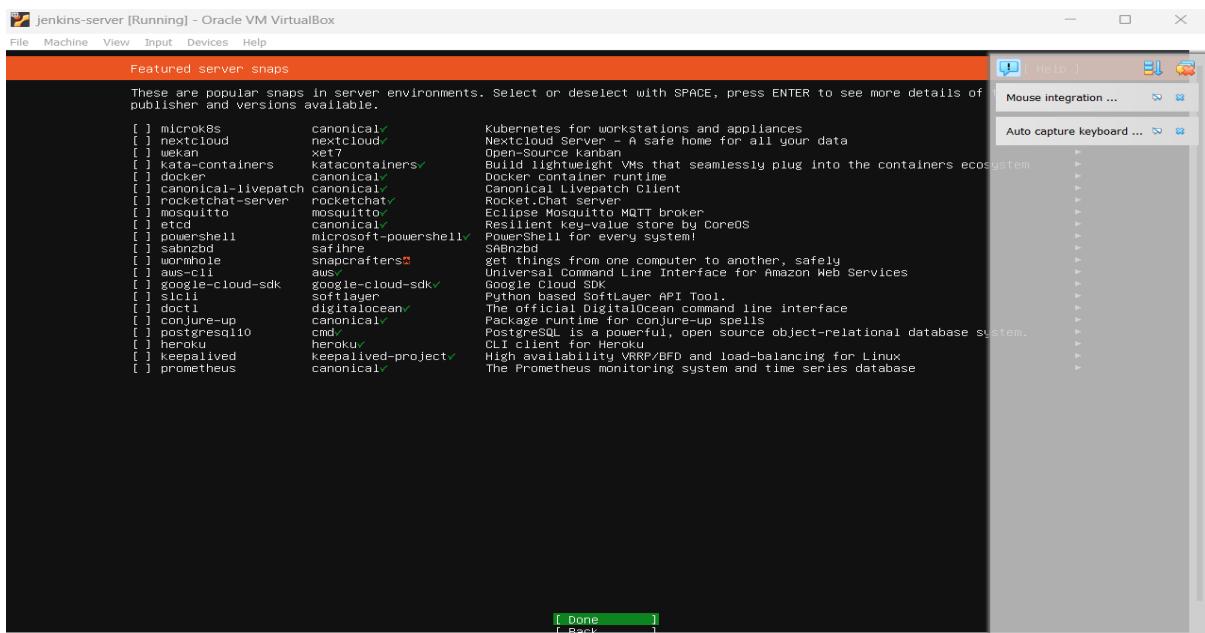
Select skip for now and press enter on “Continue” to proceed.



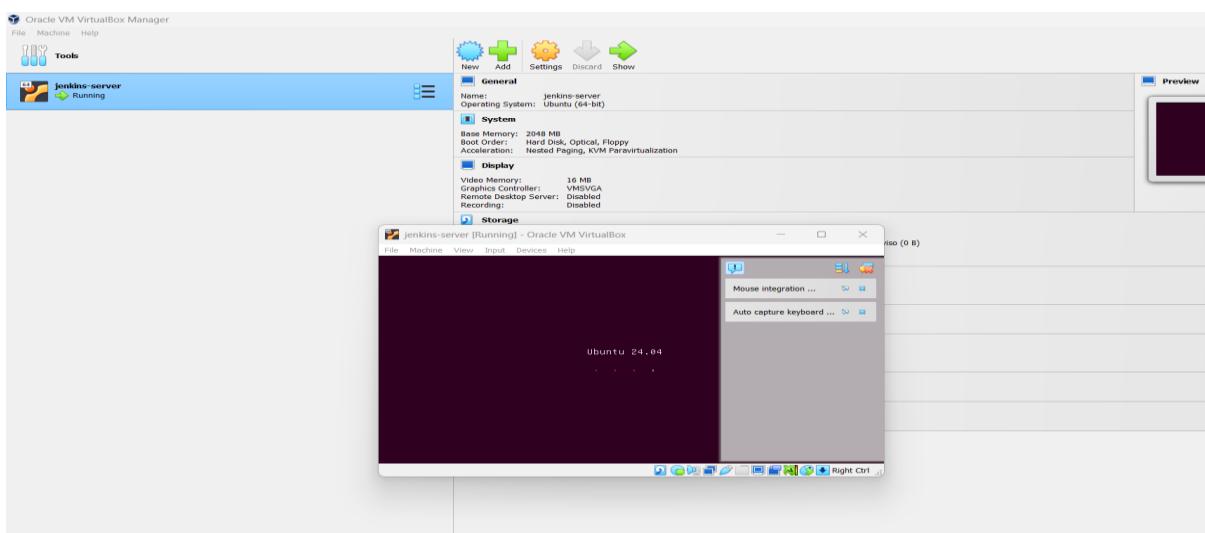
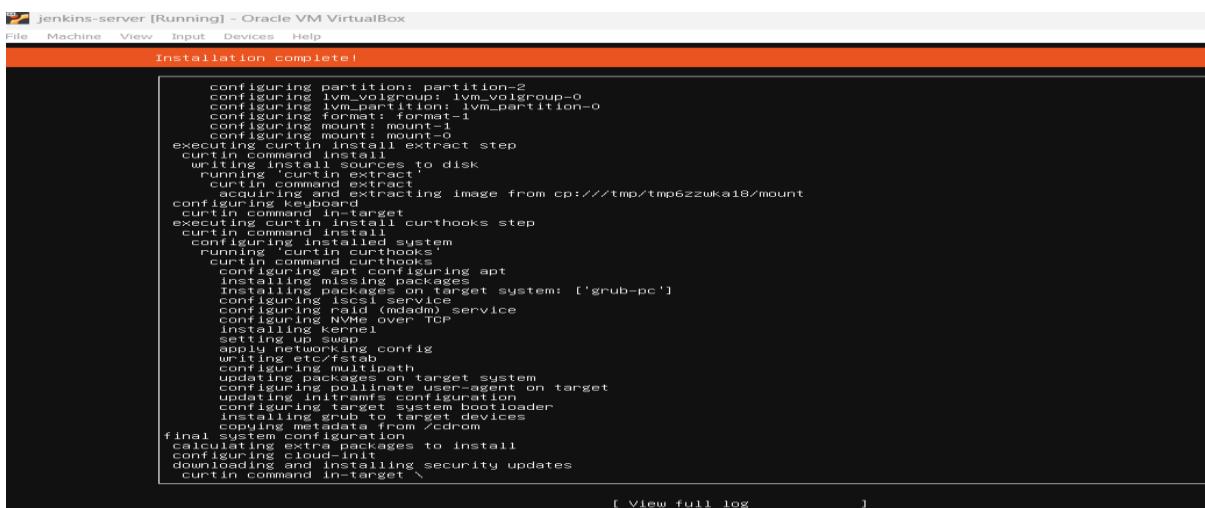
Leave the default option as black and press enter on “Done” to proceed.



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

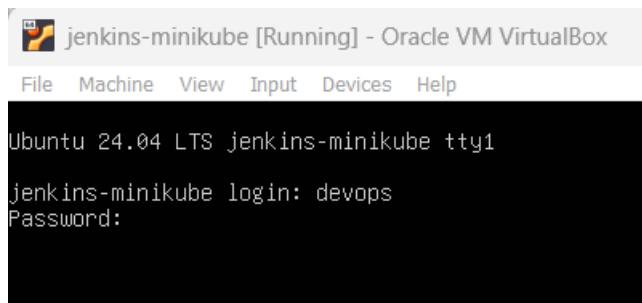


Leave all fields blank and press enter on “Done” to proceed.

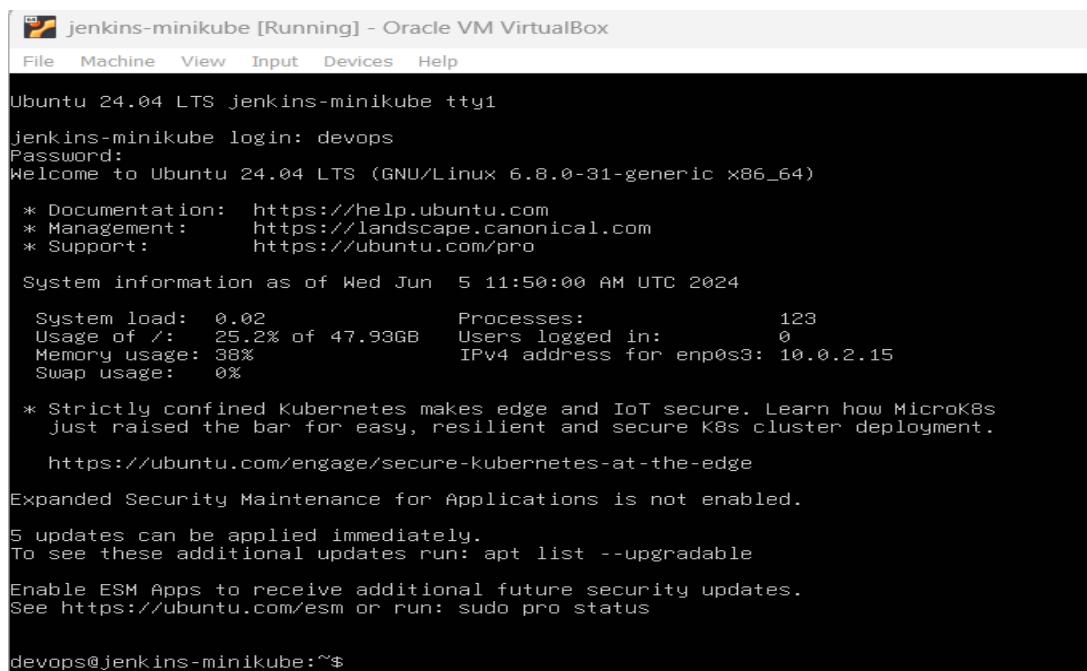


When you see installation complete, reboot the server.

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%
 
 * 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.

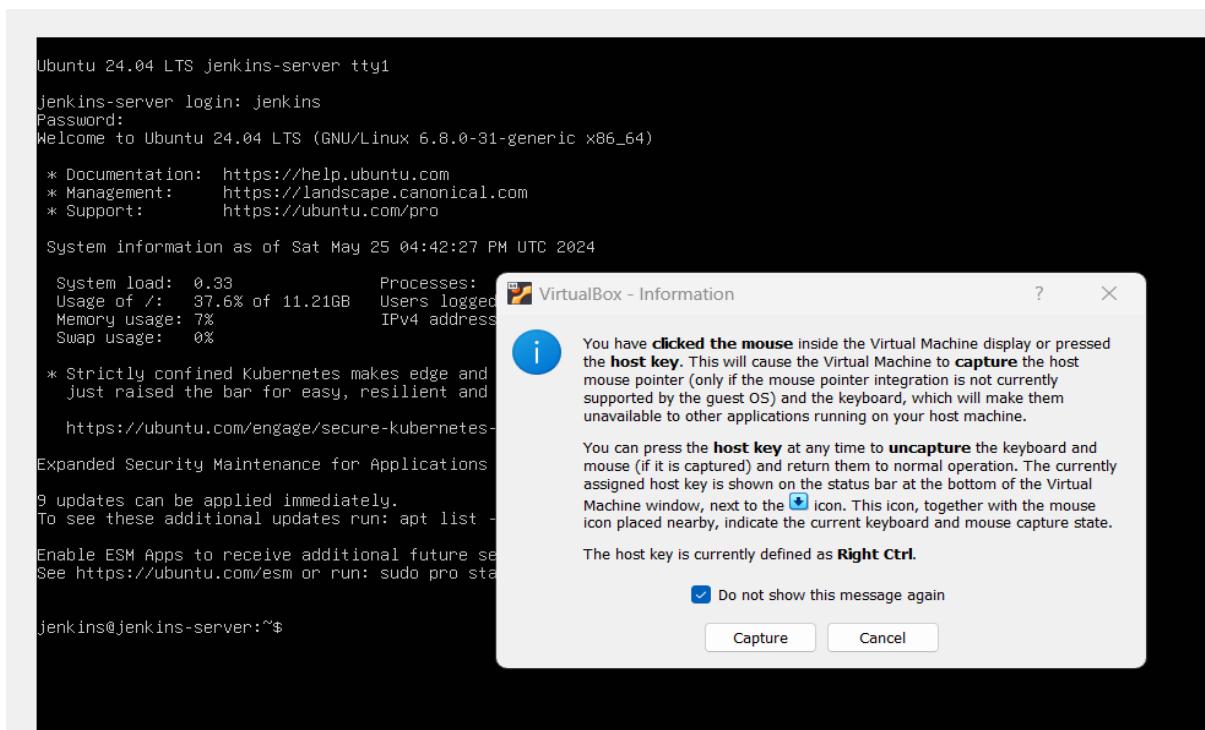
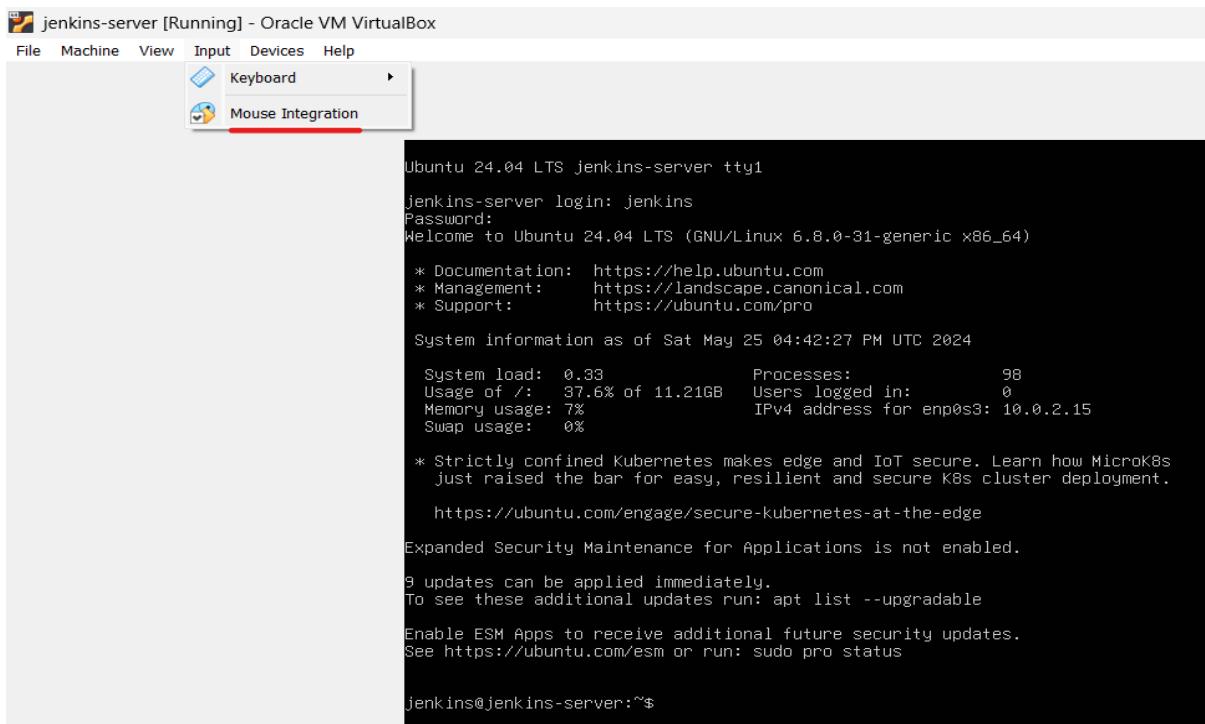
5 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

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.



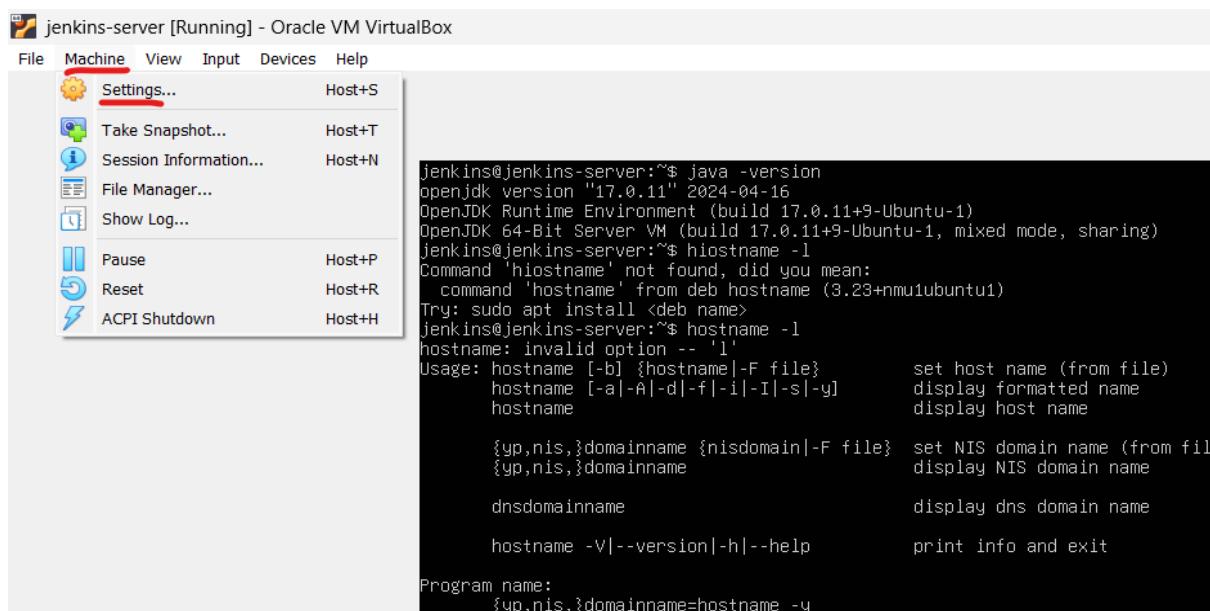
To come out the mouse cursor from guest (Ubuntu) back to the host (Windows), press CTRL from right hand side of keyboard.

Enable copy/paste from host to guest OS in VirtualBox

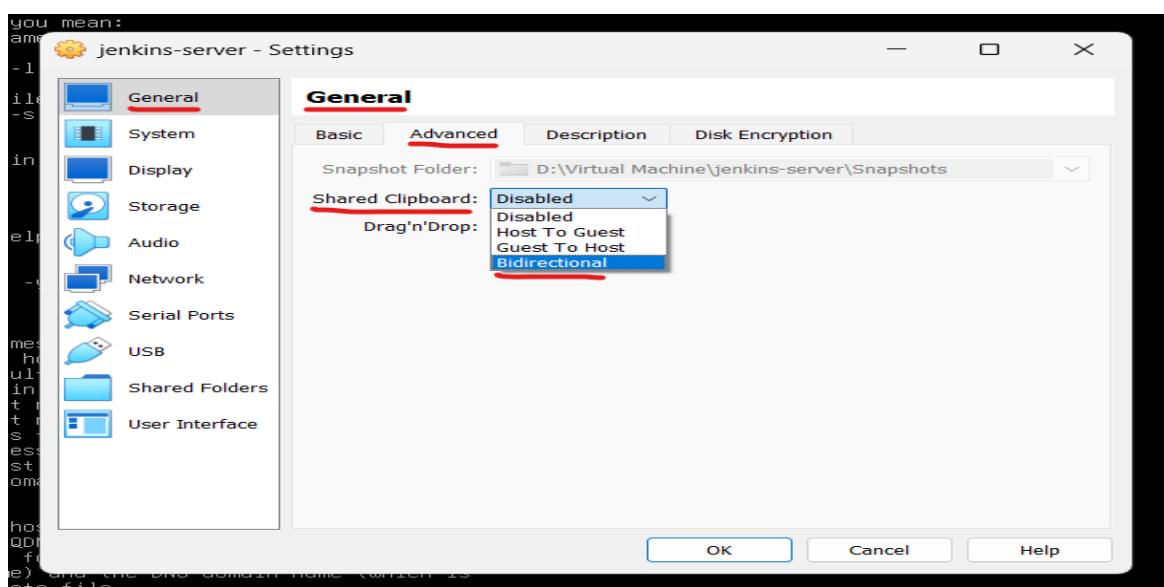
Occasionally, there is a requirement to transfer files or text from our host machine to the Virtual Machine (VM) running inside VirtualBox. This can be accomplished effortlessly by following the steps provided below:

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.

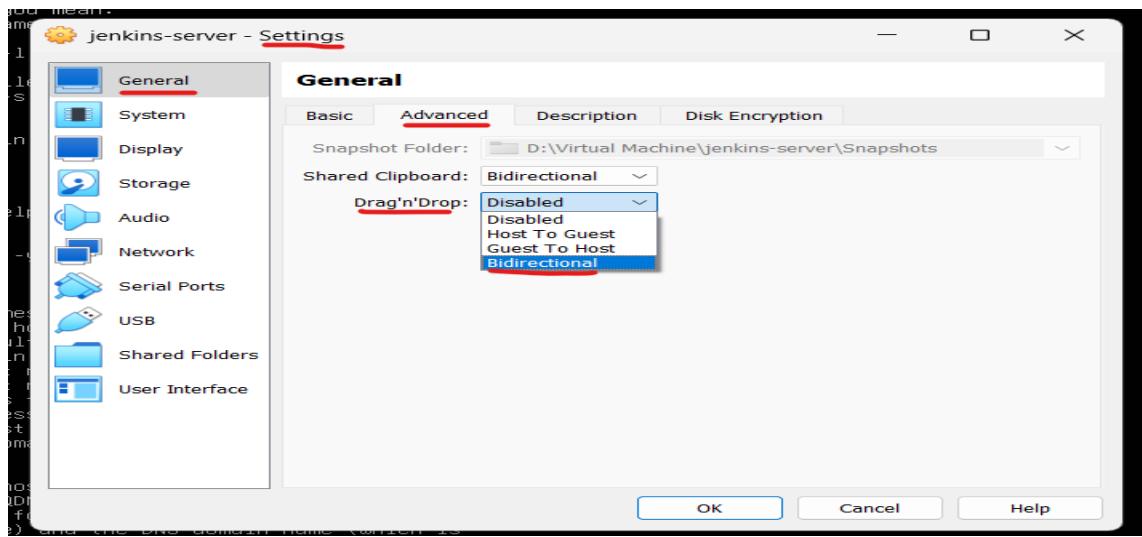


The screenshot shows the Oracle VM VirtualBox Manager interface. A context menu is open over the Jenkins-server [Running] VM, with the 'Settings...' option highlighted. The main window displays a terminal session on the Jenkins-server. The terminal output shows the Java version (openjdk version "17.0.11" 2024-04-16), the OpenJDK Runtime Environment (build 17.0.11+9-Ubuntu-1), and the OpenJDK 64-Bit Server VM (build 17.0.11+9-Ubuntu-1, mixed mode, sharing). It also shows the results of a 'hostname' command and its usage information, including options for setting the host name, displaying formatted or host names, and setting NIS domain names. The terminal prompt is Jenkins@jenkins-server:~\$.



Enable Drag and Drop (between host and guest OS)

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



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

From the Ubuntu Server command prompt,

Execute the below commands:

```
sudo apt-get update
```

```
sudo apt-get install virtualbox-guest-x11
```

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

```
sudo VBoxClient --clipboard
```

This should enable clipboard sharing.

```
Jenkins@jenkins-server:~$ sudo apt-get update
[sudo] password for Jenkins:
Hit:1 http://security.ubuntu.com/ubuntu noble-security InRelease
Hit:2 http://in.archive.ubuntu.com/ubuntu noble InRelease
Get:3 http://in.archive.ubuntu.com/ubuntu noble-updates InRelease [89.7 kB]
Hit:4 http://in.archive.ubuntu.com/ubuntu noble-backports InRelease
Get:5 http://in.archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [77.1 kB]
Get:6 http://in.archive.ubuntu.com/ubuntu noble-updates/universe amd64 Packages [35.6 kB]
Fetched 202 kB in 3s (65.8 kB/s)
Reading package lists... Done
Jenkins@jenkins-server:~$ sudo apt-get install virtualbox-guest-x11
Reading package lists... Done
Building dependency tree... Done
Building state information... 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 libisl123 libmpc3 libnotify-bin
  libnotify0 libwayland-server0 libxcv0 libxfont2 virtualbox-guest-utils x11-xkb-utils x11-xserver-utils xcvt xfntes-base xfntes-encodings xfntes-utils
Suggested packages:
  cpp-doc gcc-13-locales cpp-13-doc notification-daemon nckle cairo-5c xorg-docs-core xfs | xserver xfntes-100dpi | xfntes-75dpi xfntes-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 libisl123 libmpc3 libnotify-bin
  libnotify0 libwayland-server0 libxcv0 libxfont2 virtualbox-guest-utils virtualbox-guest-x11 x11-xkb-utils x11-xserver-utils xcvt xfntes-base
xfntes-encodings xfntes-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:

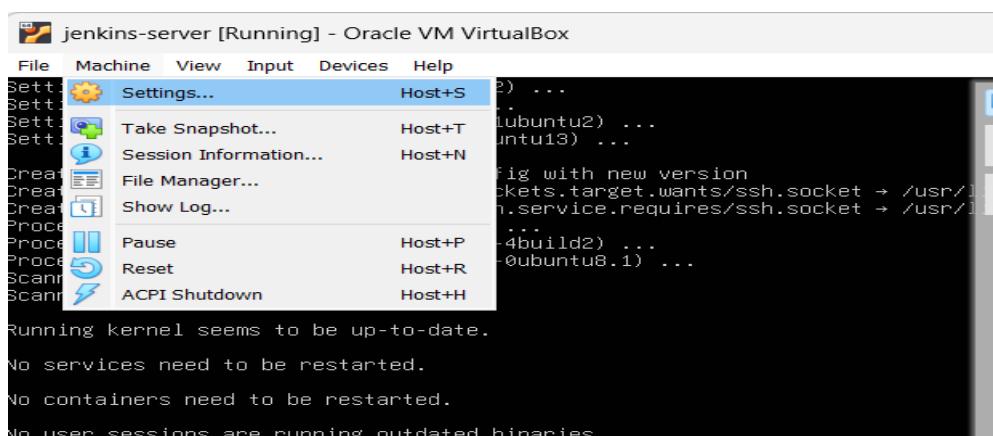
```
sudo systemctl start ssh
```

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

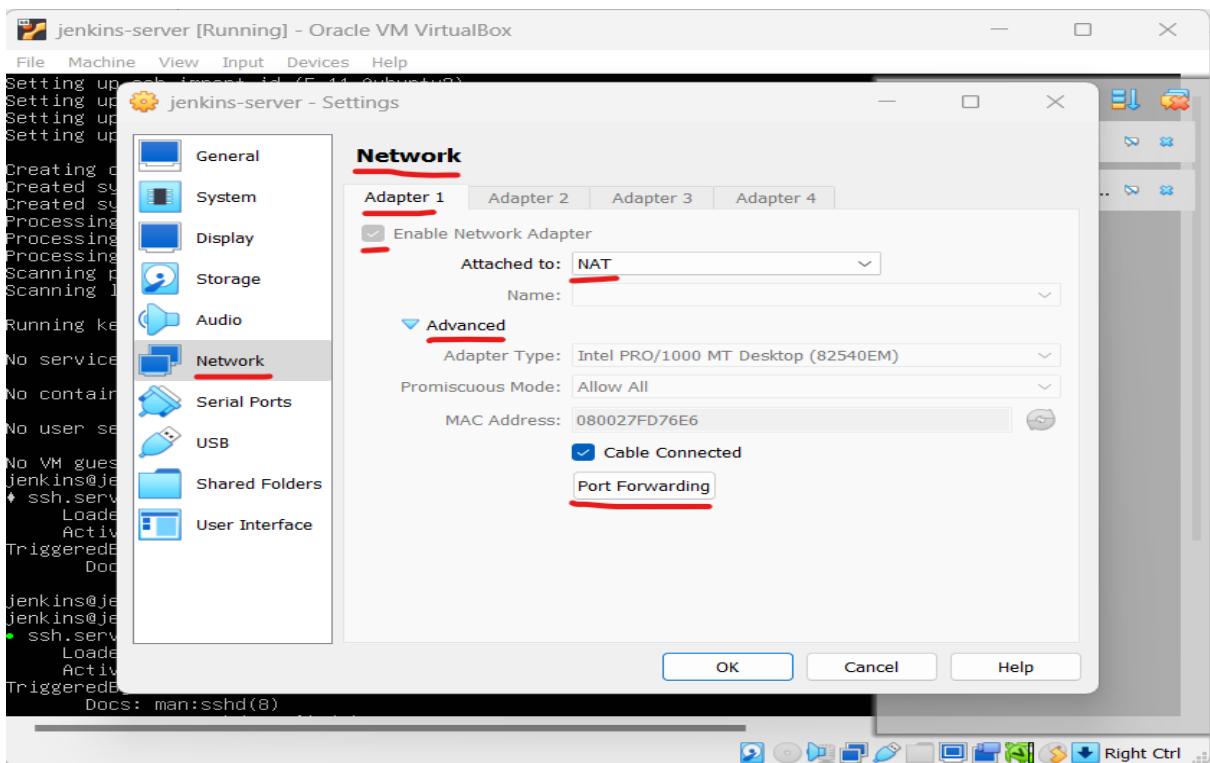
```
sudo systemctl status ssh
```

Enable port-forwarding

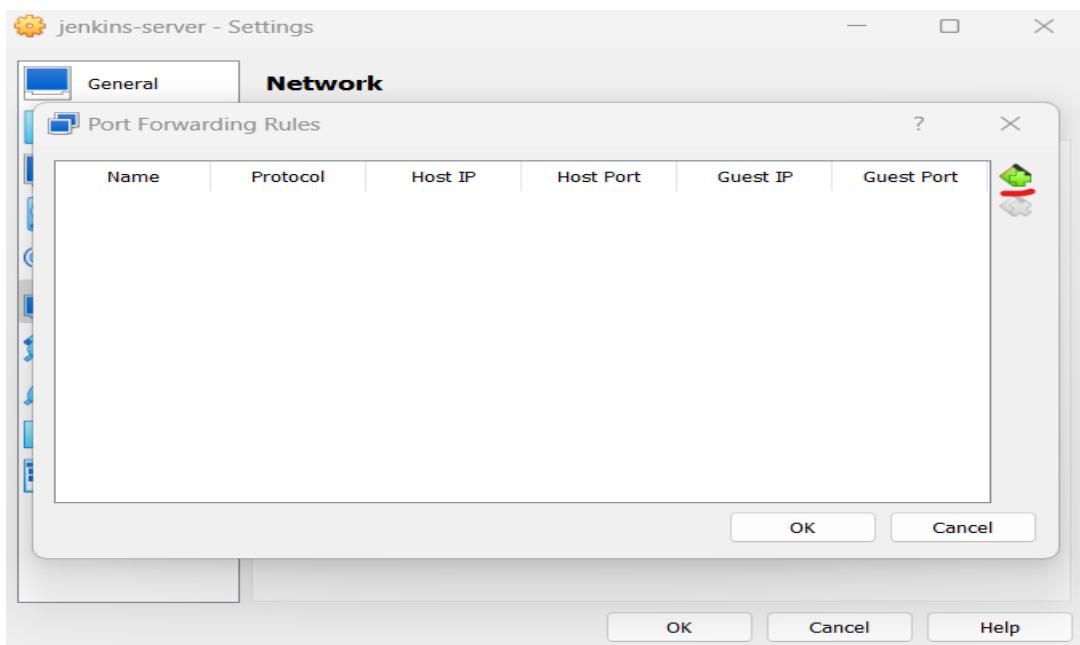
Open settings from guest OS → Machine → Settings...



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

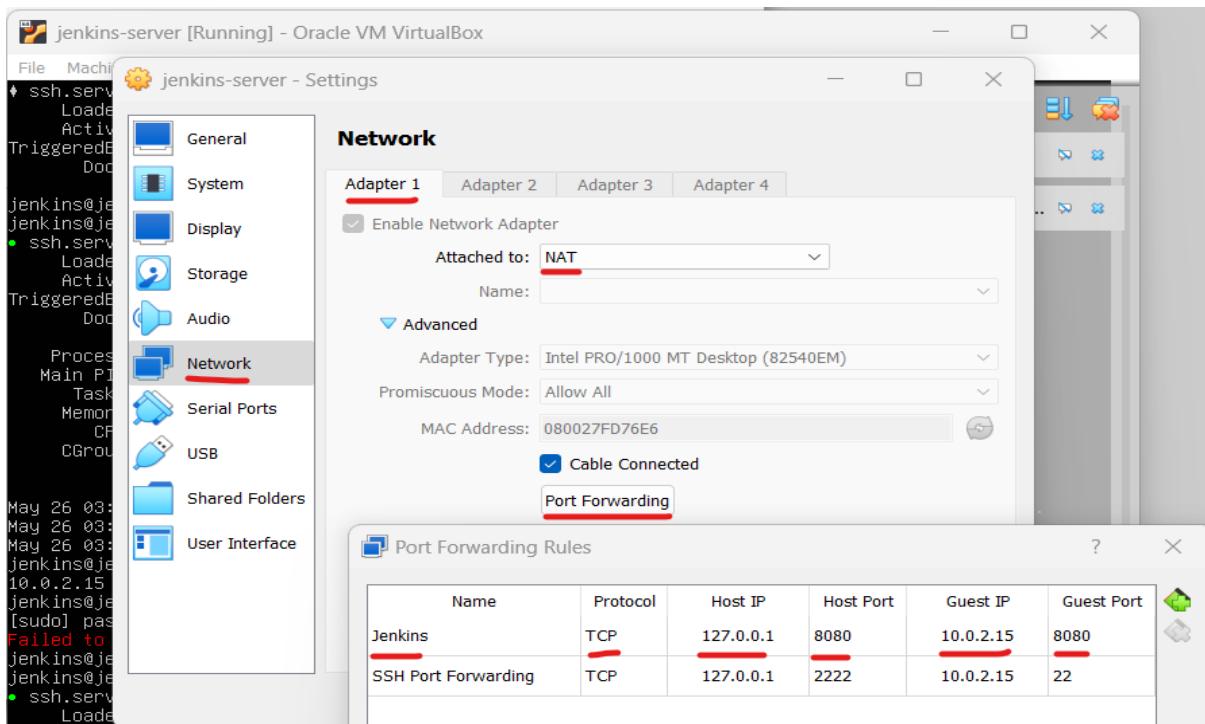


Click on Add sign as shown below:

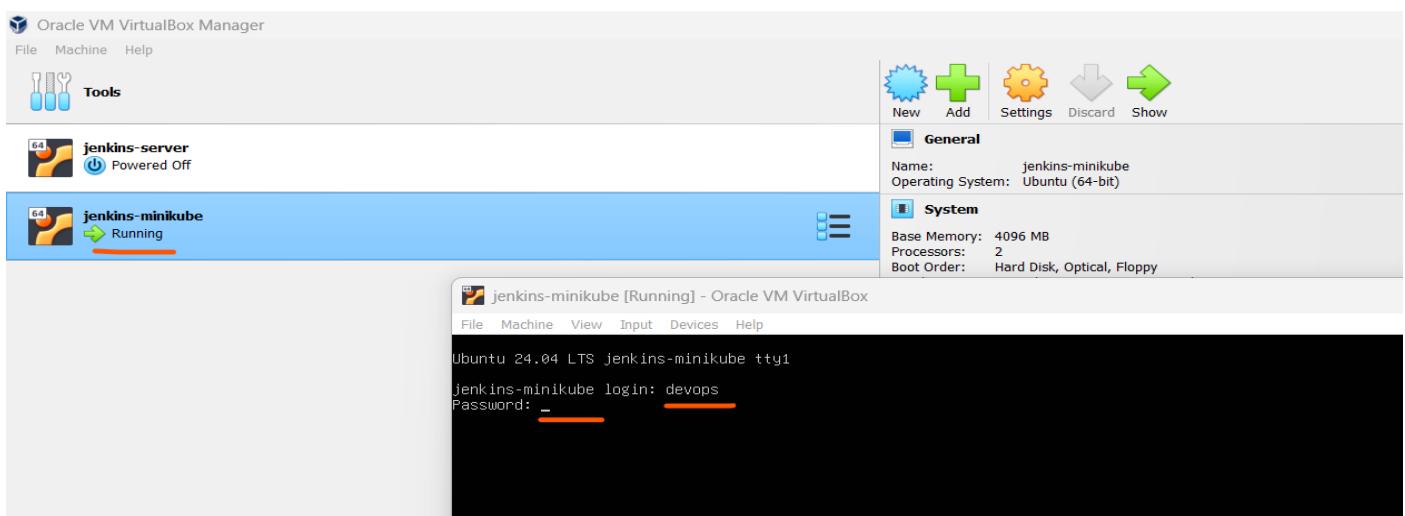


The real problem here is that the "Host IP" parameter in your VirtualBox's port forwarding rule only tells it to listen on 127.0.0.1 (loopback address). This means it will only accept connections made to 127.0.0.1

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

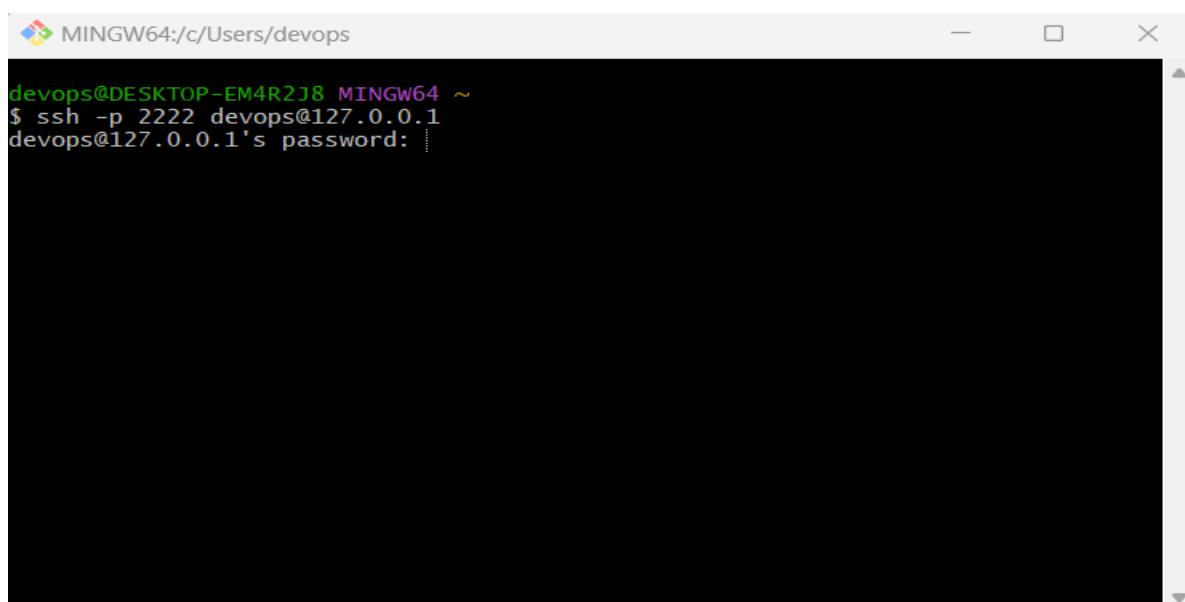
you might need to **restart ssh service** on VirtualBox for the changes to take effect.

```
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
```



```
MINGW64:/c/Users/devops
devops@DESKTOP-EM4R2J8 MINGW64 ~
$ ssh -p 2222 devops@127.0.0.1
devops@127.0.0.1's password: |
```

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:
  atk-update-icon-cache hicolor-icon-theme humanity-icon-theme java-common libasound2-data libasound2-dbg libatk-bridge2.0-0t64 libatk-wrapper-java libatk-wrapper-java-jni libatk1.0-0t64 libatspi2.0-0t64
  libavahi-client3 libavahi-common-data libavahi-common3 libcairo-gobject2 libcairo2 libcurl2-64 libdatriel libdconf1 libdrm-amdgpu libdrm-intel libdrm-nouveau libgbm-radeon1 libgail18t64
  libgd-pixbuf-2.0-0 libgd-pixbuf2.0-0-bin libgd-pixbuf2.0-common libgif7 libgl1 libgl1-amber-dri libgl1-mesa-dri libglapi-mesa libglvnd libglx-mesa0 libglx0 libgraphite2-3 libgtk2.0-0t64 libgtk2.0-0-bin
  libgtkt1.0 libgtkt1.0-0t64 libgtkt1.0-0t64 libgtkt1.0-xcb1 libgxaw1 libxcb-dri3-0 libxcb-dri3-0 libxcb-dri3-0 libxcb-dri3-0 libxcb-dri3-0 libxcb-dri3-0 libxcb-present0 libxcb-randr0 libxcb-render0 libxcb-shape0 libxcb-shm0 libxcb-sync1 libxcb-xfixes0
  libxcompositelib libxcursor1 libxdamage1 libxfixes3 libxft2 libxinerama1 libxbpfile1 libxmu0 libxrandr2 libxrender1 libxshmfence1 libxtst6 libxv1 libxf86dg1 libxf86vm1 mesa-vu kan-drivers
  openjdk-17-jre-headless session-migration ubuntu-mono x11-common x11-utils
Suggested packages:
  default-base libalsa-util libasound2-plugins cups-common gvfs libcurl2-utils pscd librsvg2-bin libnss-mdns fonts-ipafont-gothic fonts-ipafont-mincho fonts-wqy-microhei | fonts-wqy-zenhei fonts-indic
  x11-utils
Recommended packages:
  libui
The following NEW packages will be installed:
  atk-update-icon-cache hicolor-icon-theme humanity-icon-theme java-common libasound2-data libasound2-dbg libatk-bridge2.0-0t64 libatk-wrapper-java libatk-wrapper-java-jni libatk1.0-0t64 libatspi2.0-0t64
  libavahi-client3 libavahi-common-data libavahi-common3 libcairo-gobject2 libcairo2 libcurl2-64 libdatriel libdconf1 libdrm-amdgpu libdrm-intel libdrm-nouveau libgbm-radeon1 libgail18t64
  libgd-pixbuf-2.0-0 libgd-pixbuf2.0-0-bin libgd-pixbuf2.0-common libgif7 libgl1 libgl1-amber-dri libgl1-mesa-dri libglapi-mesa libglvnd libglx-mesa0 libglx0 libgraphite2-3 libgtk2.0-0t64 libgtk2.0-0-bin
  libgtkt1.0 libgtkt1.0-0t64 libgtkt1.0-0t64 libgtkt1.0-xcb1 libgxaw1 libxcb-dri3-0 libxcb-dri3-0 libxcb-dri3-0 libxcb-dri3-0 libxcb-dri3-0 libxcb-present0 libxcb-randr0 libxcb-render0 libxcb-shape0 libxcb-shm0 libxcb-sync1 libxcb-xfixes0
  libxcompositelib libxcursor1 libxdamage1 libxfixes3 libxft2 libxinerama1 libxbpfile1 libxmu0 libxrandr2 libxrender1 libxshmfence1 libxtst6 libxv1 libxf86dg1 libxf86vm1 mesa-vu kan-drivers
openjdk-17-jre-headless session-migration ubuntu-mono x11-common x11-utils
0 upgraded, 104 newly installed, 0 to remove and 3 not upgraded.
Need to get 501 MB of additional disk space.
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 libxau-dev libxcb1-dev libxdmp-dev libxt-dev openjdk-17-jdk-headless x11proto-dev xorg-sgml-doctools xtrans-dev
Suggested packages:
  libice-dev libx11-doc libxcb-doc libxt-doc openjdk-17-demo openjdk-17-source visualvm
The following NEW packages will be installed:
  libice-dev libpthread-stubs0-dev libsm-dev libxau-dev libxcb1-dev libxdmp-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.
Executing: /usr/lib/systemd/systemd-sysv-install enable jenkins
root@jenkins-server:~# sudo systemctl start jenkins
root@jenkins-server:~# curl -s http://127.0.0.1:8080
  % Total    % Received % Xferd  Average Speed   Time    Time     Time Current
                                 Dload  Upload   Total   Spent    Left  Speed
*   [100%]   100  100  0  0 --:--:-- --:--:-- --:--:-- 0
jenkins.service Jenkins Continuous Integration Server
   Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: enabled)
     Active: active (running) since Sun 2024-05-26 05:14:31 UTC; 6min ago
       Main PID: 143 (java)
          Tasks: 39 (limit: 2276)
         Memory: 447.9M (peak: 463.1M)
            CPU: 1m 4.396s
       CGroup: /system.slice/jenkins.service
               └─ 143 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

May 26 05:13:44 jenkins-server jenkins[200]: [fc9eef1d014b7eb006f9d3aa0d6c9]
May 26 05:13:44 jenkins-server jenkins[200]: May also found file: /lib/jenkins/secrets/initialAdminPassword
May 26 05:13:44 jenkins-server jenkins[200]: ****
May 26 05:13:44 jenkins-server jenkins[200]: [fc9eef1d014b7eb006f9d3aa0d6c9] Jenkins_InitialReactorRunner$1@onAttained: Completed initialization
May 26 05:13:44 jenkins-server jenkins[200]: [fc9eef1d014b7eb006f9d3aa0d6c9] hudson.lifecycle.Lifecycle$OnReady: Jenkins is fully up and running
May 26 05:13:44 jenkins-server jenkins[200]: [fc9eef1d014b7eb006f9d3aa0d6c9] Jenkins_InitialReactorRunner$1@onAttained: Completed initialization
May 26 05:13:44 jenkins-server jenkins[200]: [fc9eef1d014b7eb006f9d3aa0d6c9] Started jenkins.service - Jenkins Continuous Integration Server.
May 26 05:13:44 jenkins-server jenkins[200]: [fc9eef1d014b7eb006f9d3aa0d6c9] Jenkins_InitialReactorRunner$1@onAttained: Completed initialization
May 26 05:13:44 jenkins-server jenkins[200]: [fc9eef1d014b7eb006f9d3aa0d6c9] h.m.downloadableLoad@load: obtained the update data file for hudson.tasks.Maven.MavenInstaller
May 26 05:13:44 jenkins-server jenkins[200]: [fc9eef1d014b7eb006f9d3aa0d6c9] hudson.util.RetriggerStart: Performed the action check updates server successfully at the attempt #1
[09:13:45] [INFO] [jenkins] [85%]
```

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

`sudo ufw status`

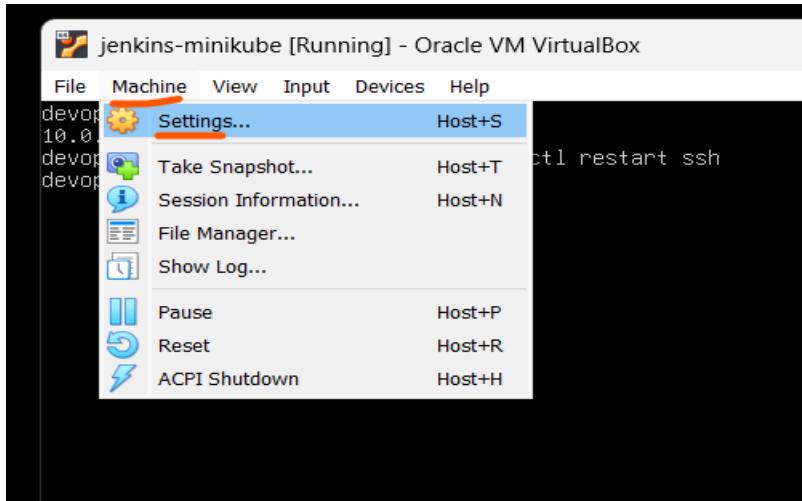
`sudo ufw disable`

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

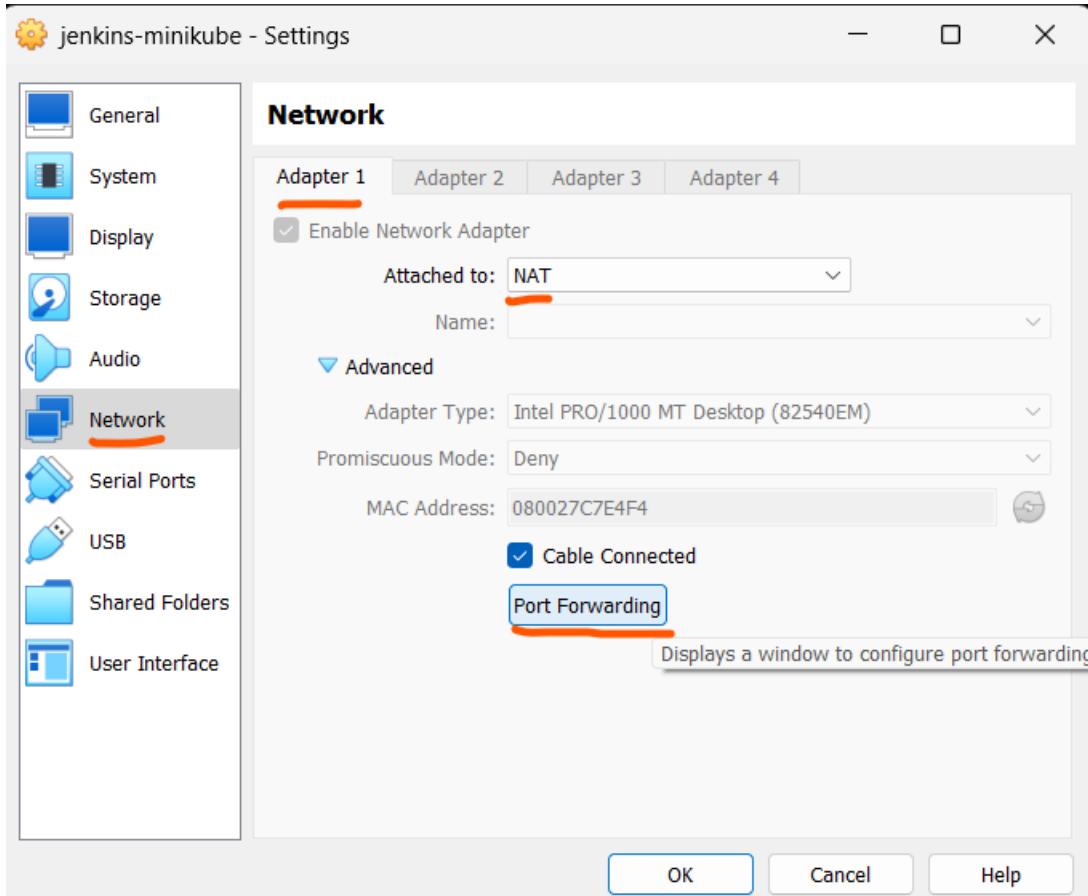
Set port forwarding for Jenkins (guest / Ubuntu to host / Windows)

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

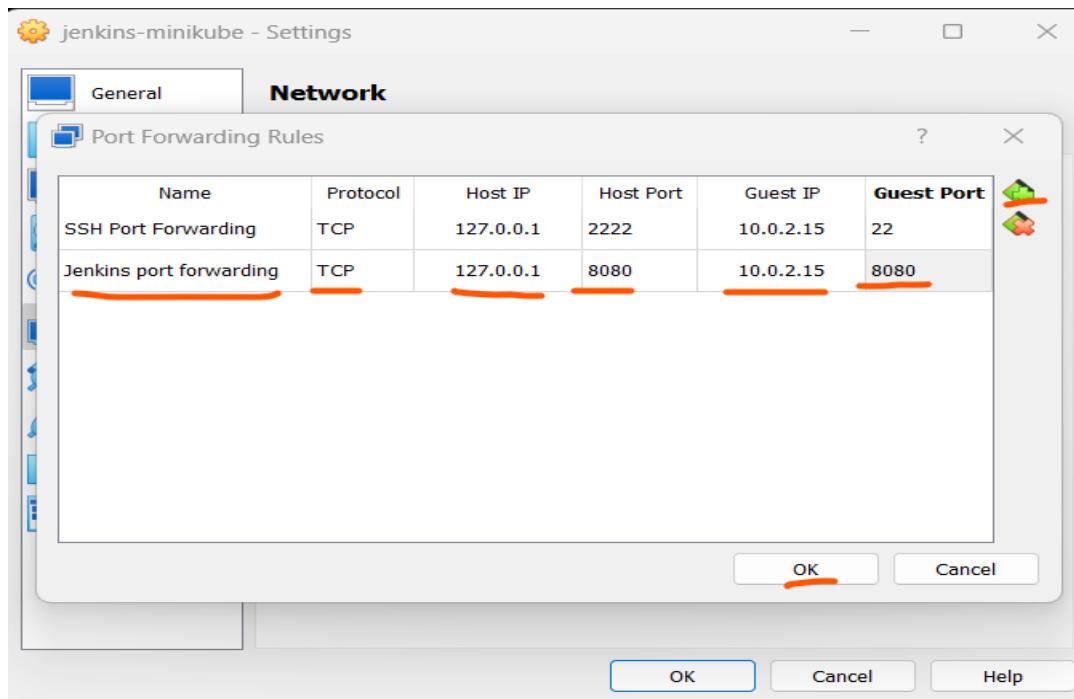


Click on network → Adapter 1 → NAT → click on Port Forwarding



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

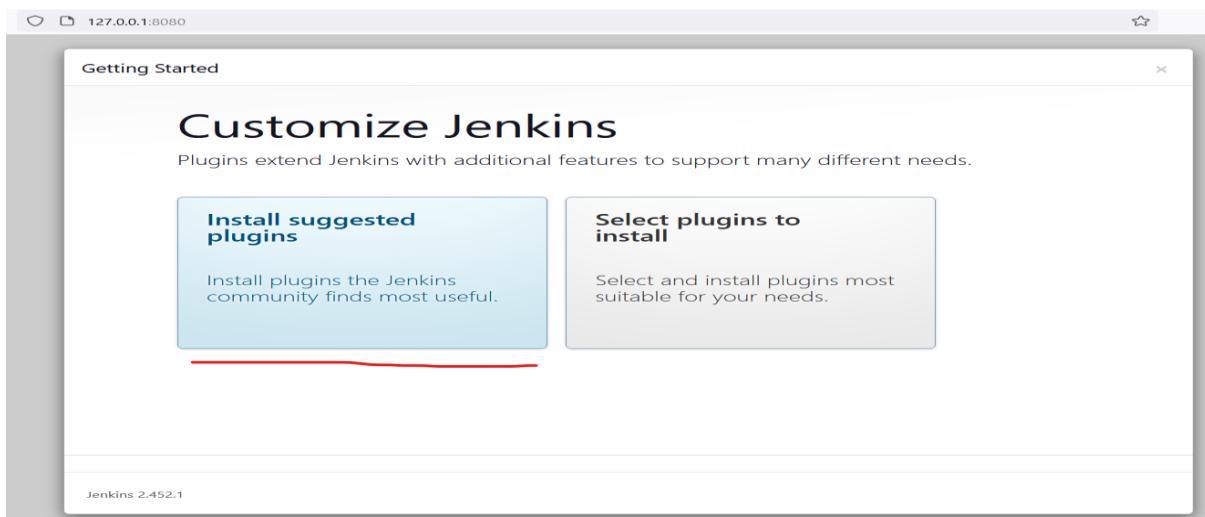
Initial Jenkins configuration

First time installation – default of set plugins.

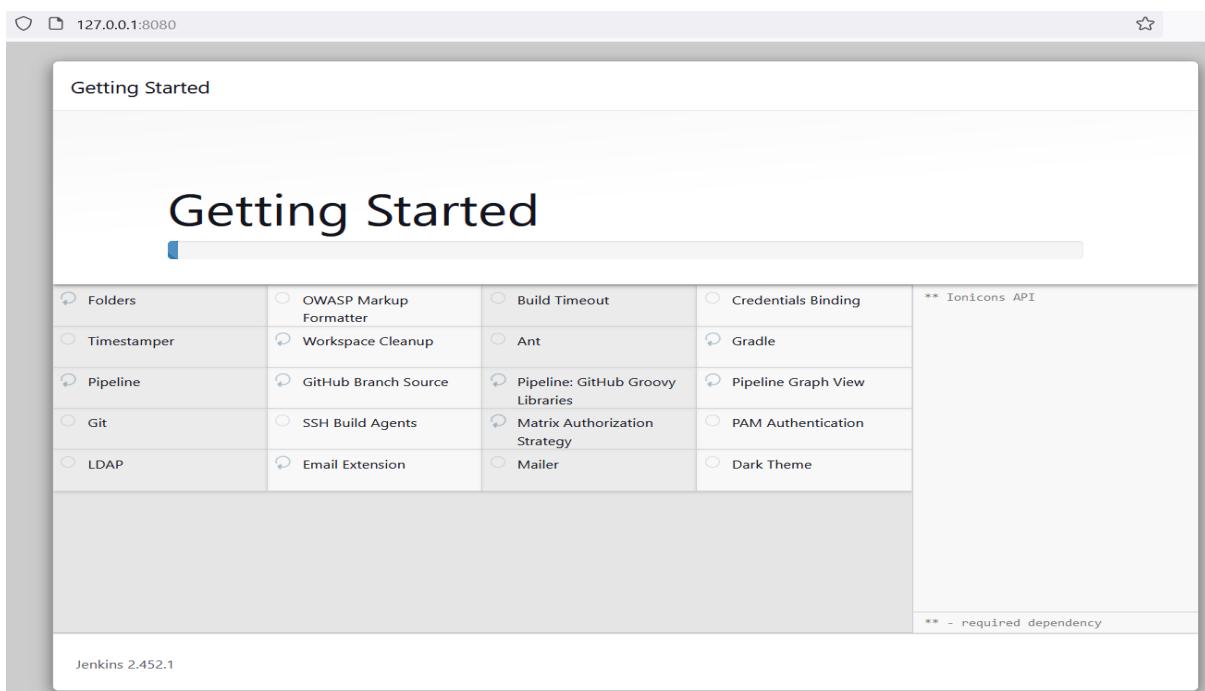
Plugins extend Jenkins with additional features to support many different needs.

For first time, you will get these options.

Please select – “Install suggested plugins”



You will 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

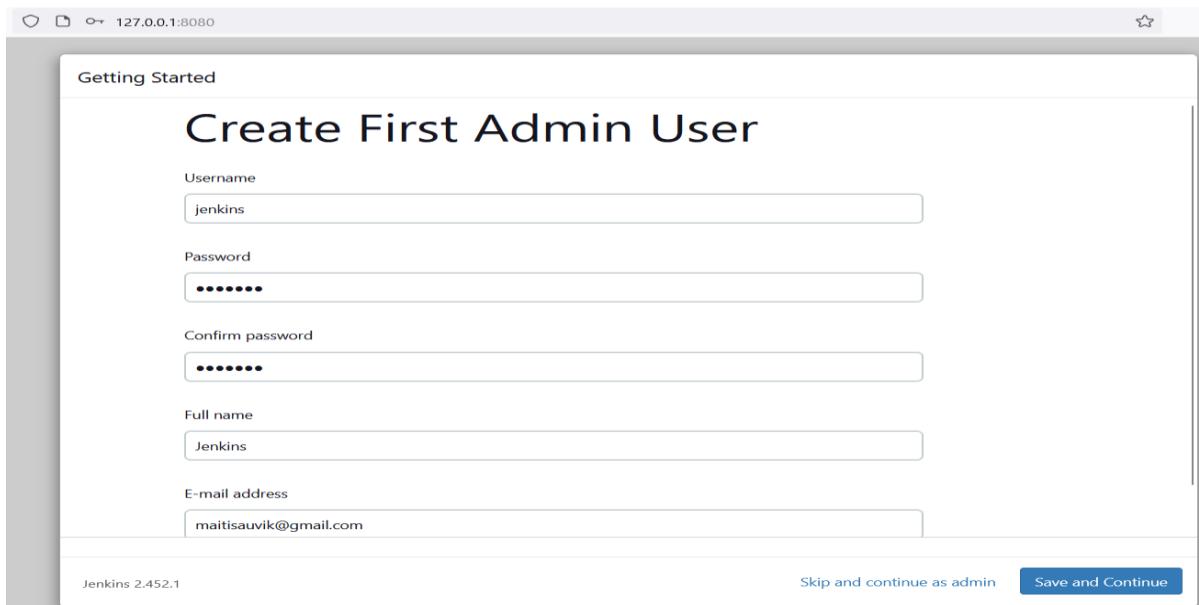
Password

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.

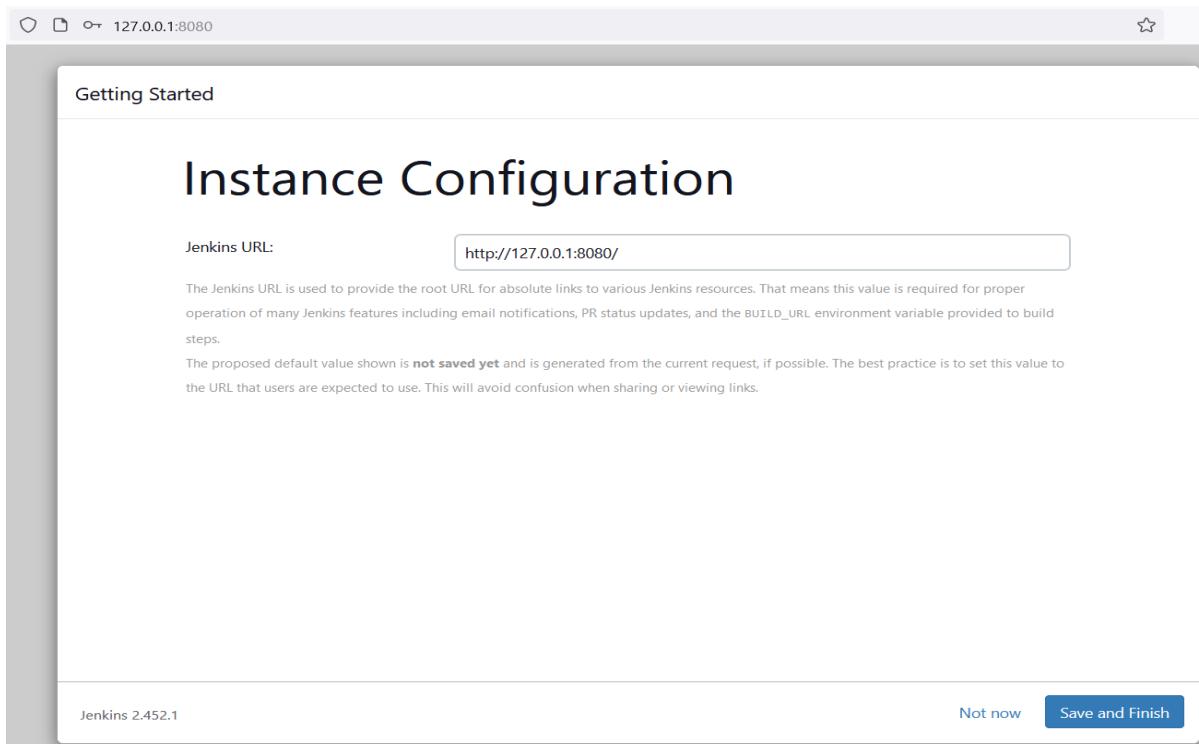
Getting Started

Instance Configuration

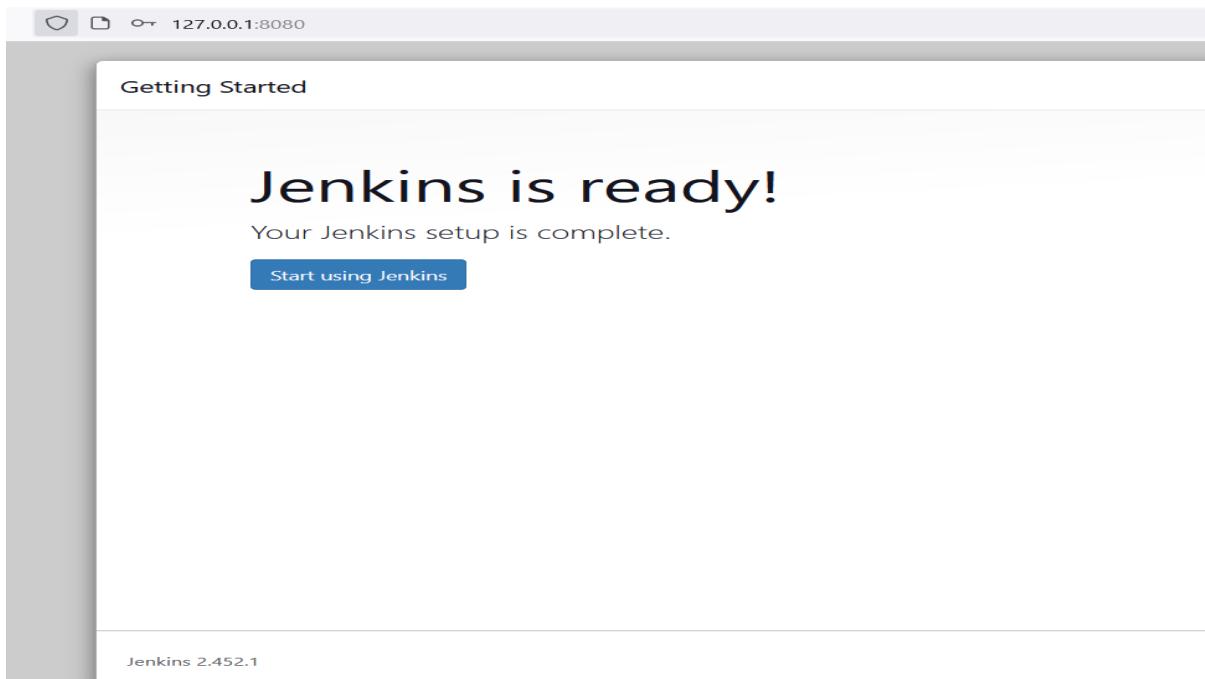
Jenkins URL:

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 Dashboard. The top navigation bar includes links for 'Dashboard', 'Build History', 'Manage Jenkins', and 'My Views'. The main content area features a 'Welcome to Jenkins!' message with a subtext about setting up distributed builds. It includes a 'Create a job' button and a 'Set up a distributed build' section with links for 'Set up an agent', 'Configure a cloud', and 'Learn more about distributed builds'. On the left, there are sections for 'Build Queue' (empty) and 'Build Executor Status' (2 idle). The bottom right corner shows links for 'REST API' and 'Jenkins 2.452.1'.

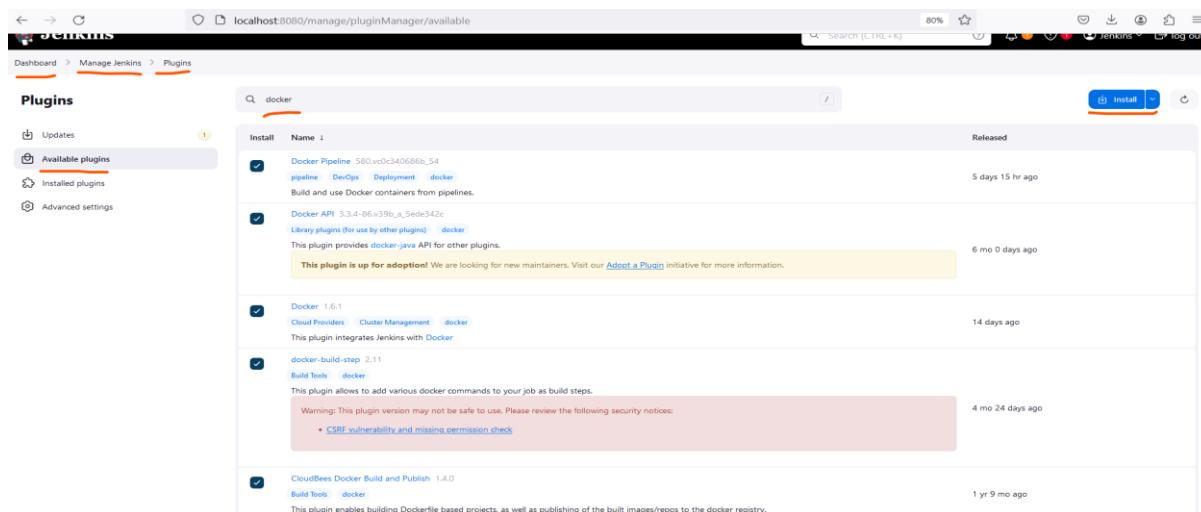
Jenkins plugins Installation

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

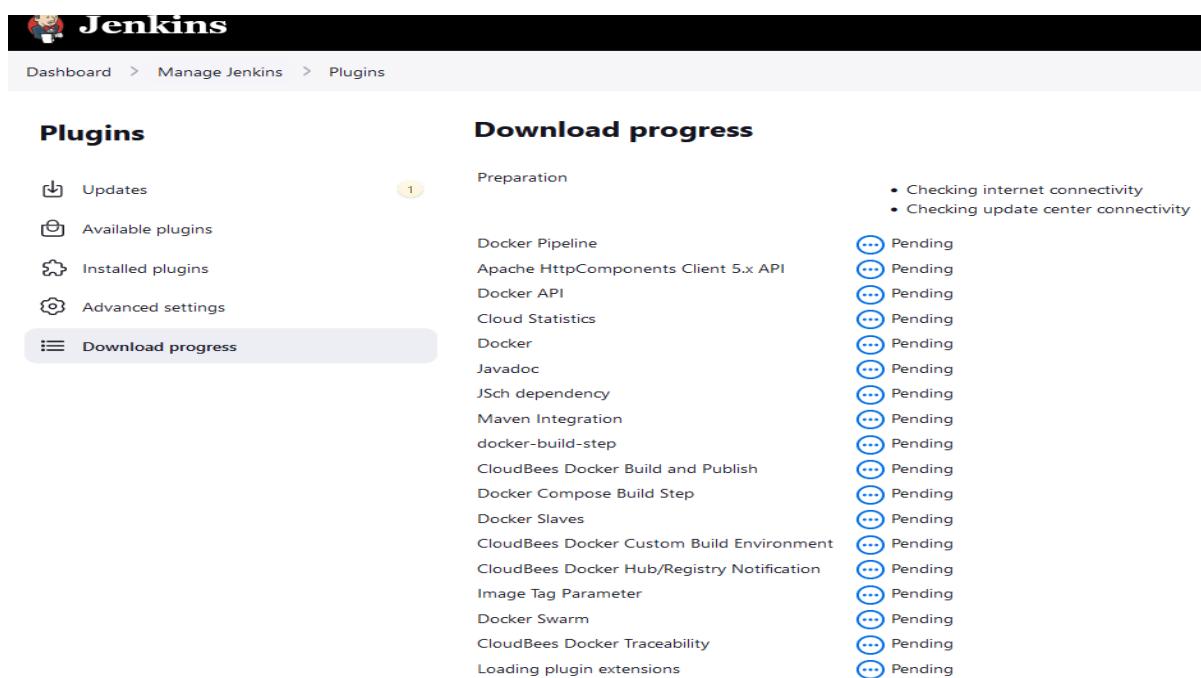
Dashboard → Manage Jenkins → Plugins → Available plugins

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 table listing several Jenkins plugins. The columns in the table are 'Install', 'Name', and 'Released'. The 'Install' column has checkboxes next to each plugin. The 'Name' column lists the plugin names and their versions. The 'Released' column shows the date when each plugin was released. One plugin, 'Docker Pipeline', has a yellow box drawn around its 'Install' button, indicating it is the target for installation.

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



The screenshot shows the Jenkins Plugin Manager interface with the 'Available plugins' tab selected. On the left, there is a sidebar with navigation links: 'Updates', 'Available plugins' (which is highlighted), 'Installed plugins', and 'Advanced settings'. On the right, there is a 'Download progress' section. This section has two main parts: 'Preparation' and a list of plugins. The 'Preparation' part lists tasks: 'Checking internet connectivity' and 'Checking update center connectivity'. The main list contains a large number of Jenkins plugins, many of which are currently in a 'Pending' state, indicated by blue circular icons with three dots. The plugins listed include 'Docker Pipeline', 'Apache HttpComponents Client 5.x API', 'Docker API', 'Cloud Statistics', 'Docker', 'Javadoc', 'JSch dependency', 'Maven Integration', 'docker-build-step', 'CloudBees Docker Build and Publish', 'Docker Compose Build Step', 'Docker Slaves', 'CloudBees Docker Custom Build Environment', 'CloudBees Docker Hub/Registry Notification', 'Image Tag Parameter', 'Docker Swarm', 'CloudBees Docker Traceability', and 'Loading plugin extensions'.

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 list of plugins is displayed, each with a checkbox, name, version, and release date. The 'Install' button is highlighted at the top right of the list.

Install	Name	Released
<input checked="" type="checkbox"/>	Kubernetes Client API 6.10.0-240.v57880ce8b_0b_2	4 mo 2 days ago
<input checked="" type="checkbox"/>	kubernetes - Library plugin (for use by other plugins)	5 days 2 hr ago
<input checked="" type="checkbox"/>	Kubernetes Credentials 173.v04e9c17cf07	4 days 5 hr ago
<input checked="" type="checkbox"/>	Kubernetes 4233eb_67a_0e11a_039	9 mo 1 day ago
<input checked="" type="checkbox"/>	Kubernetes CLI 1.7.1	2 mo 28 days ago
<input checked="" type="checkbox"/>	Kubernetes :: Pipeline :: DevOps Steps 1.6	5 yr 4 mo ago
<input type="checkbox"/>	GitLab Credentials - Kubernetes Integration 259.v7c7289bd530	3 mo 5 days ago

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

The screenshot shows the Jenkins 'Download progress' page. It lists the installed Kubernetes plugins: Kubernetes Client API, Kubernetes Credentials, Kubernetes, Kubernetes CLI, Kubernetes Credentials Provider, and Kubernetes :: Pipeline :: DevOps Steps. Each plugin has a green checkmark indicating success. A note at the bottom says 'Loading plugin extensions' with a 'Running' status.

Preparation:

- Checking internet connectivity
- Checking update center connectivity
- Success

Successes:

- Kubernetes Client API
- Kubernetes Credentials
- Kubernetes
- Kubernetes CLI
- Kubernetes Credentials Provider
- Kubernetes :: Pipeline :: DevOps Steps

Running:

- Loading plugin extensions

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/
Hit:6 https://pkg.jenkins.io/debian-stable binary/ InRelease
Fetched 89.7 kB in 5s (19.9 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:~$
```

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 ca-certificates curl software-properties-common
0 upgraded, 0 newly installed, 0 to remove and 10 not upgraded.
Need to get 3,974 B of archives.
After this operation, 35.1 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu/noble/universe amd64 apt-transport-https all 2.7.14build2 [3,974 B]
Fetched 3,974 B in 1s (0.223 B/s)
Setting up apt-transport-https (2.7.14build2) ...
(Reading database ... 98678 files and directories currently installed.)
Preparing to unpack .../apt-transport-https_2.7.14build2_all.deb ...
Unpacking apt-transport-https (2.7.14build2) ...
Setting up apt-transport-https (2.7.14build2) ...
Scanning Linux images...
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:~$ 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:~$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu ${lsb_release -cs} stable"
Repository: 'deb [arch=amd64] https://download.docker.com/linux/ubuntu ${lsb_release -cs} stable'
  Origin: https://download.docker.com/linux/ubuntu
  Label: Docker Repository
  Suite: ${lsb_release -cs}
  Arch: amd64
  Component: stable
  More info: https://download.docker.com/linux/ubuntu
  APT::Architectures::amd64
Press [ENTER] to continue or Ctrl-C to cancel.
Adding deb entry to /etc/apt/sources.list.d/archive_uri-https_download_docker_com_linux_ubuntu-noble.list
Adding disabled deb entry to /etc/apt/sources.list.d/archive_uri-https_download_docker_com_linux_noble.list
Get:1 https://archive.ubuntu.com/ubuntu/noble InRelease
Hit:2 https://pkgs.jenkins.io/debian-stable binary InRelease
Hit:3 http://in.archive.ubuntu.com/ubuntu/noble InRelease
Hit:4 http://security.ubuntu.com/ubuntu/noble-security InRelease
Get:5 http://in.archive.ubuntu.com/ubuntu/noble-updates InRelease [89.7 kB]
Hit:6 http://in.archive.ubuntu.com/ubuntu/noble-updates InRelease
Get:7 https://download.docker.com/linux/ubuntu/noble/stable amd64 Packages [6,952 kB]
Get:8 http://in.archive.ubuntu.com/ubuntu/noble/stable amd64 Packages [77.1 kB]
Get:9 http://in.archive.ubuntu.com/ubuntu/noble-updates/universe amd64 Packages [33.7 kB]
Fetched 258 kB in 5s (50.5 kB/s)
Reading package lists...
w: https://download.docker.com/linux/ubuntu/dists/noble/InRelease: Key is stored in legacy trusted.gpg keyring (/etc/apt/trusted.gpg). see the DEPRECATION section in apt-key(8) for details.
jenkins@jenkins-server:~$ |
```

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
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 state 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 libslirp0 pigz slirp4netns
Suggested packages:
libufs-tools libufs-mount | cgroup-lite
The following NEW packages will be installed:
containerd.io docker-buildx-plugin docker-ce docker-ce-rootless-extras docker-compose-plugin libltdl7 libslirp0 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 1:6.32-1 [30.0 MB]
Get:2 https://in.archive.ubuntu.com/ubuntu/noble/universe amd64 libltdl7 amd64 2.4.7-7build1 [65.6 kB]
Get:3 https://in.archive.ubuntu.com/ubuntu/noble/main amd64 libslirp0 amd64 4.7.0-1ubuntu3 [63.8 kB]
Get:4 https://in.archive.ubuntu.com/ubuntu/noble/main amd64 docker-buildx-plugin amd64 0.14.0-1ubuntu.24.04~noble [34.9 kB]
Get:5 https://download.docker.com/linux/ubuntu/noble/stable amd64 docker-buildx-plugin amd64 1:2.1-1build2 [29.7 MB]
Get:6 https://in.archive.ubuntu.com/ubuntu/noble/universe amd64 slirp4netns amd64 1:2.1-1build2 [34.9 kB]
Get:7 https://download.docker.com/linux/ubuntu/noble/stable amd64 docker-ce-cli amd64 5:26.1.3-1~ubuntu.24.04~noble [14.6 MB]
Get:8 https://download.docker.com/linux/ubuntu/noble/stable amd64 docker-ce amd64 5:26.1.3-1~ubuntu.24.04~noble [25.3 MB]
Get:9 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:10 https://download.docker.com/linux/ubuntu/noble/stable amd64 docker-compose-plugin amd64 2.27.0-1ubuntu.24.04~noble [12.5 MB]
Fetched 122 MB in 10s (12.1 MB/s)
Selecting previously unselected package pigz.
(Reading database ... 98682 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 (1.6.32-1) ...
```

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.



```
jenkins@jenkins-server:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; preset: enabled)
   Active: active (running) since Sun 2024-05-26 09:31:24 UTC; 18min ago
     TriggeredBy: ● docker.socket
      Main PID: 6381 (dockerd)
        Memory: 27.8M (peak: 28.5M)
       CGroup: /system.slice/docker.service
               └─6381 /usr/bin/dockerd -d --containerd=/run/containerd/containerd.sock

May 26 09:31:21 jenkins-server systemd[1]: Starting docker.service - Docker Application Container Engine...
May 26 09:31:21 jenkins-server dockerd[6381]: time="2024-05-26T09:31:21.288547997Z" level=info msg="detected 127.0.0.53 nameserver, assuming systemd-resolved, so using resolv.conf: /run/systemd/resolve/resolv.conf"
May 26 09:31:21 jenkins-server dockerd[6381]: time="2024-05-26T09:31:21.290566024Z" level=info msg="parsed configuration file: /etc/docker/daemon.json"
May 26 09:31:24 jenkins-server dockerd[6381]: time="2024-05-26T09:31:24.126050000Z" level=info msg="Loading containerd snapshotter"
May 26 09:31:24 jenkins-server dockerd[6381]: time="2024-05-26T09:31:24.1342798Z" level=info msg="Docker daemon initialized"
May 26 09:31:24 jenkins-server dockerd[6381]: time="2024-05-26T09:31:24.137326752Z" level=info msg="API Listen on /run/docker.sock"
May 26 09:31:24 jenkins-server systemd[1]: Started docker.service - Docker Application Container Engine.

lines 1-11/21 (END)
```

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.

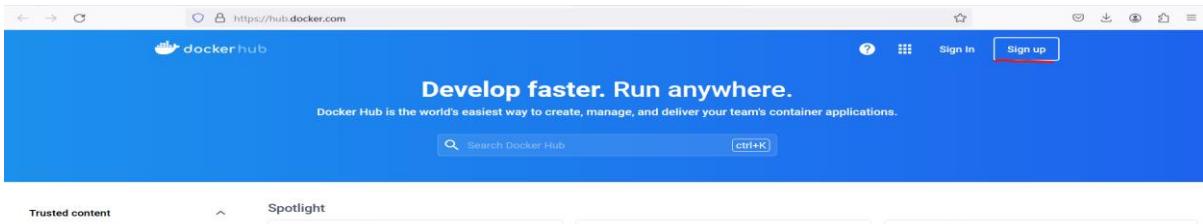
```
sudo systemctl daemon-reload
```

```
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 and the word "docker". Below it is a "Sign in" button. A message reads: "Using Docker for work? We recommend signing in with your work email address." A text input field is shown with the email "sauvik.devops@gmail.com" entered. Below the input field is a large blue "Continue" button. Underneath the "Continue" button is the word "OR". Below "OR" are two options: "Continue with Google" (with the Google logo) and "Continue with GitHub" (with the GitHub logo). At the bottom of the form 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 with the 'Repositories' tab highlighted in red. The main heading is 'Welcome to Docker' followed by 'Download the desktop application'. Below this, there are three cards: 'Create a Repository', 'Docker Hub Basics', and 'Language-Specific Guides'. At the bottom, 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. The 'Namespace' dropdown is set to 'sauvikdevops' and the 'Repository Name' field contains 'learning'. A note below says 'This repository has been created for learning purpose'. Under 'Visibility', the 'Private' option is selected, indicated by a red underline. The 'Create' button is at the bottom right. On the right side, 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 to replace `tagname` with the desired image repository tag.

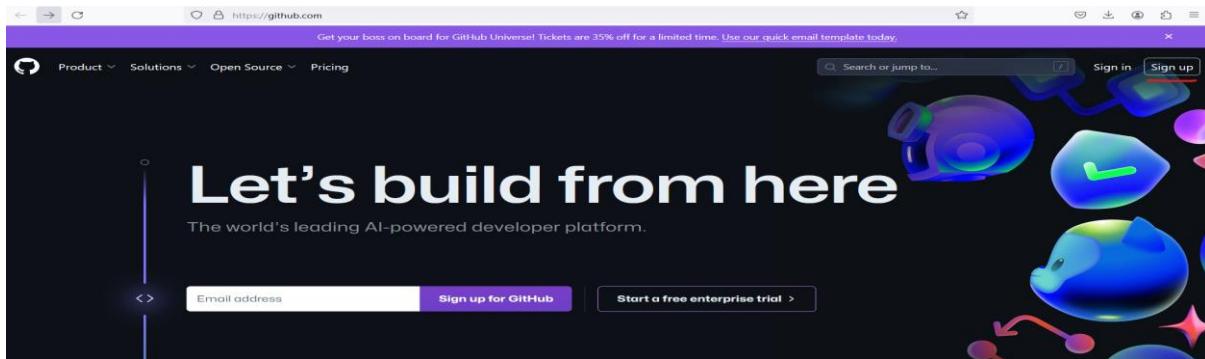
Your container registry would be created like below

The screenshot shows the DockerHub interface for a private repository named 'sauvikdevops/learning'. The top navigation bar includes 'Explore', 'Repositories', 'Organizations', a search bar, and a 'Using 1 of 1 private repositories. Get more' button. The repository details show it was created less than a minute ago and is intended for learning purposes. A 'Docker commands' section provides a command to push a new tag: 'docker push sauvikdevops/learning:tagname'. Below this are sections for 'Tags' (empty) and 'Automated Builds' (disabled). A 'Repository overview' section is also visible.

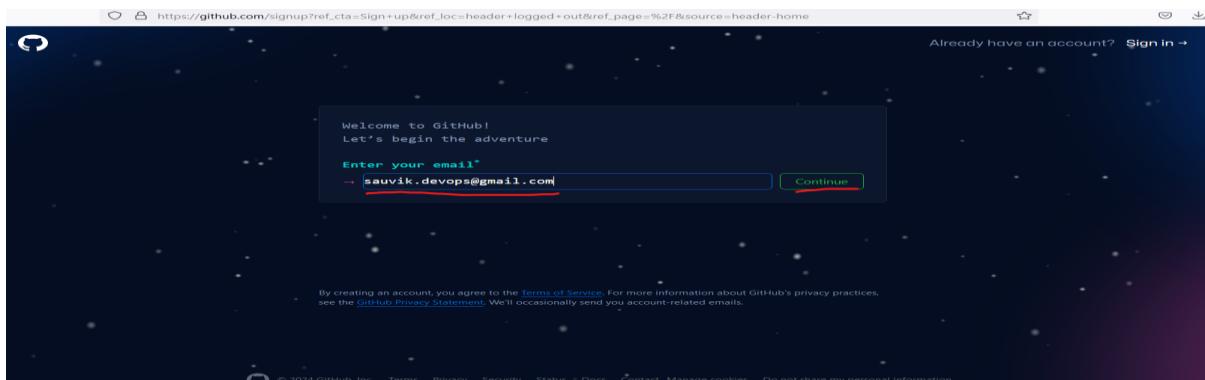
Create / manage Git Hub account

[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.

Welcome to GitHub!
Let's begin the adventure

Enter your email*

✓ sauvik.devops@gmail.com

Create a password*

✓ ••••••••••••••

Enter a username*

→ sauvikdevops

Continue

sauvikdevops is available.

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.

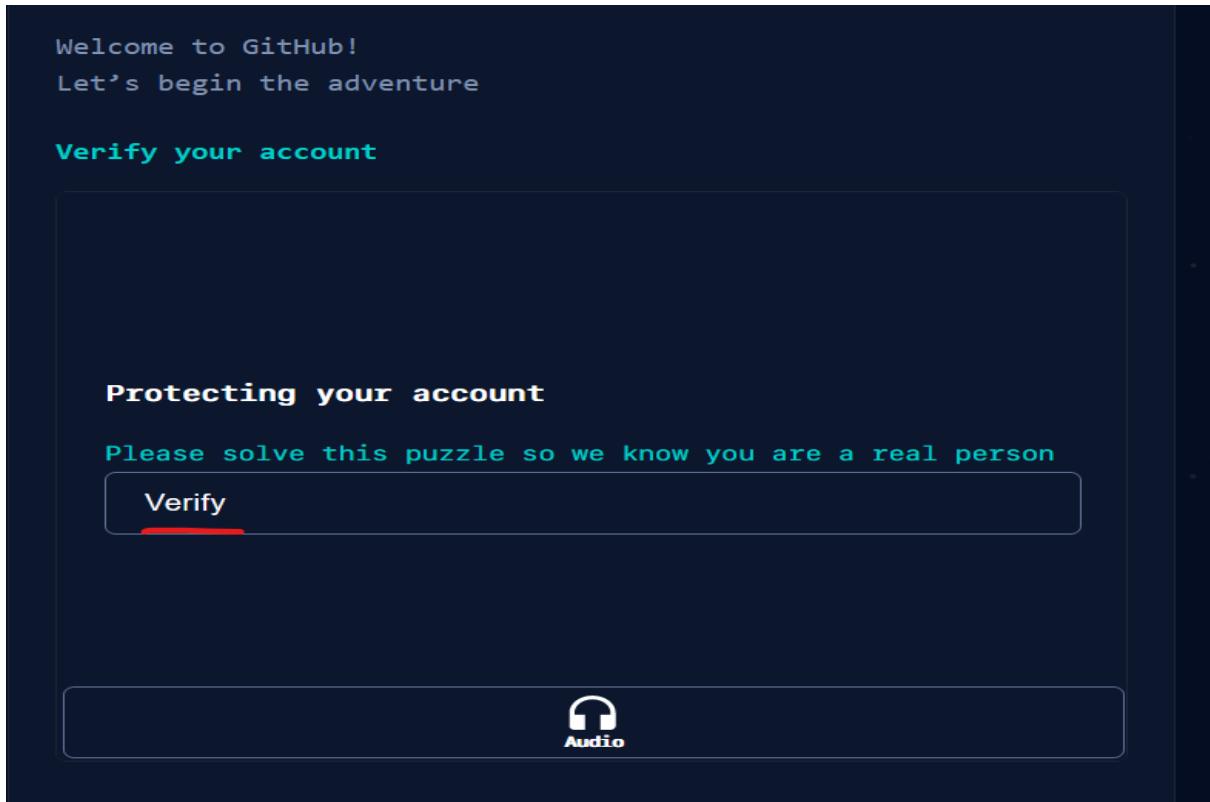
Welcome to GitHub!
Let's begin the adventure

Verify your account

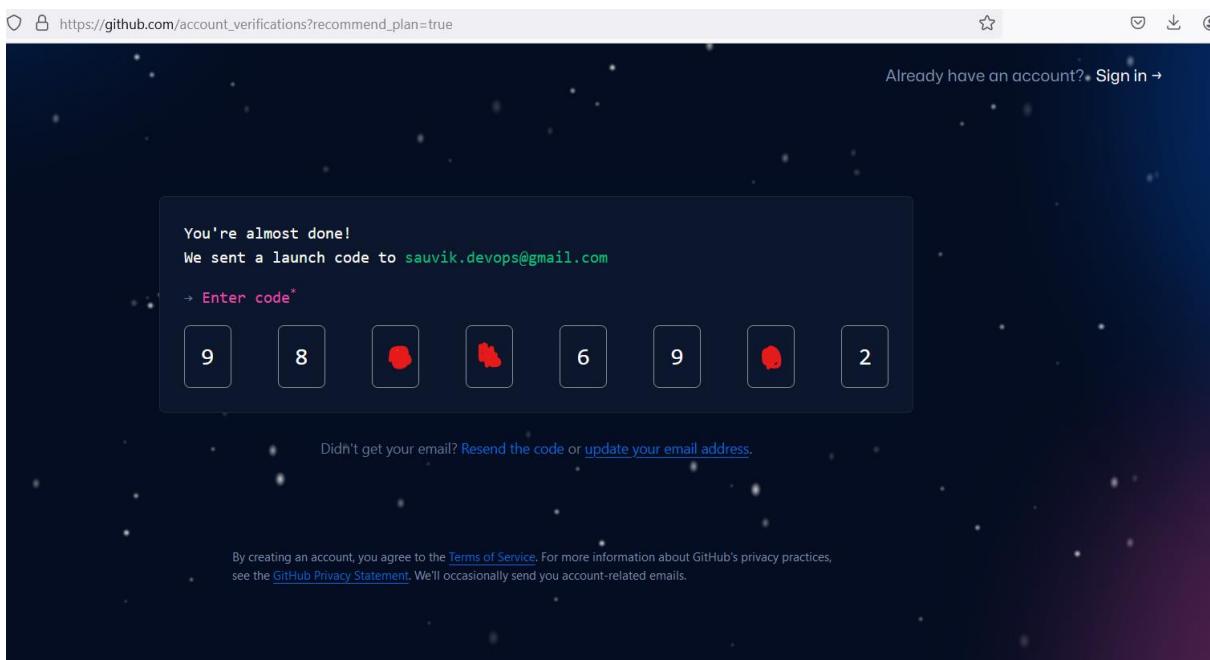


By creating an account, you agree to the [Terms of Service](#). For more information about GitHub's privacy practices, see the [GitHub Privacy Statement](#). We'll occasionally send you account-related emails.

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.

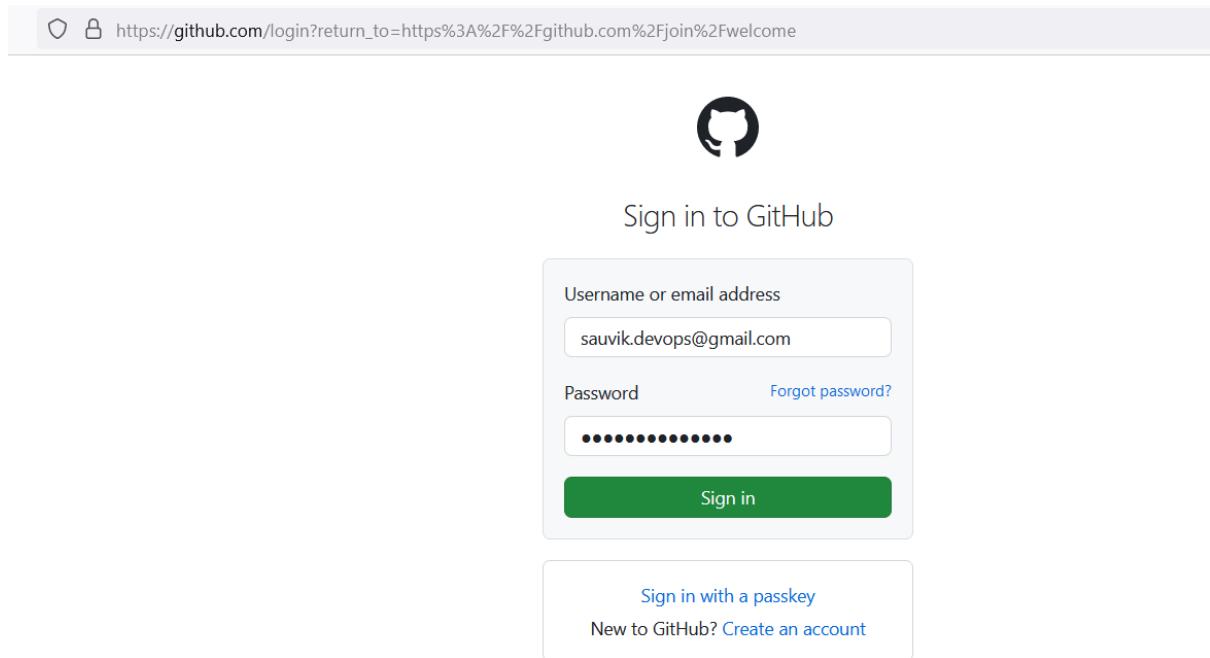


You are all set now.

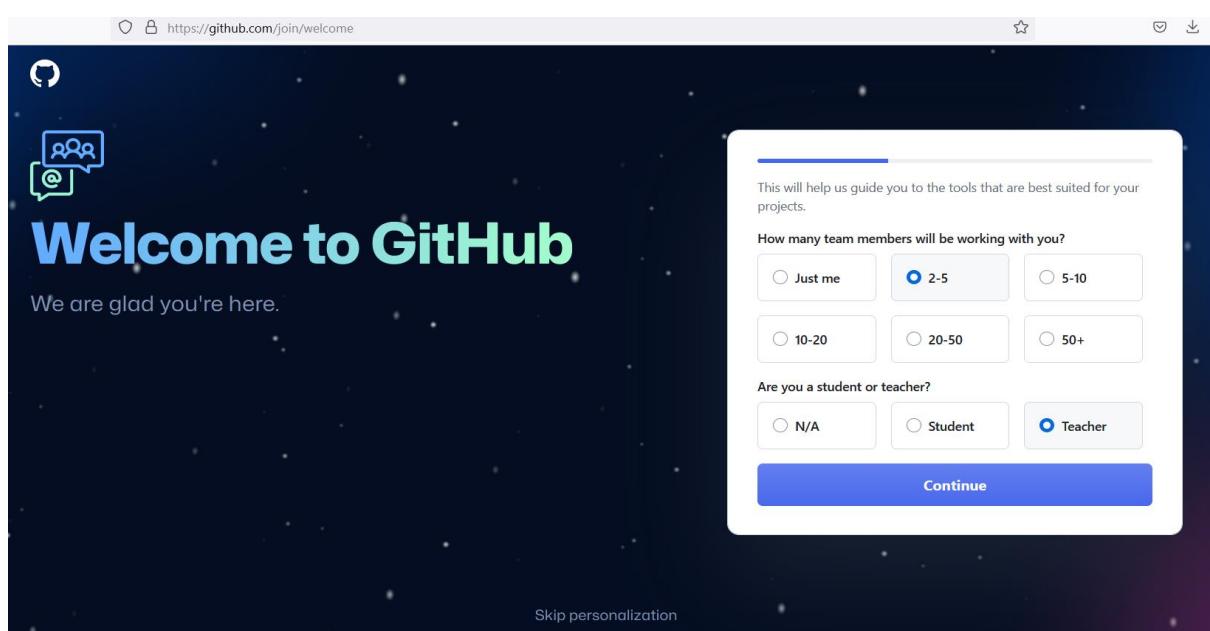
Login to GitHub account

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
- 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 CI/CD minutes/month
Free for public repositories
- 500MB of Packages storage
Free for public repositories
- 120 core-hours of Codespaces compute
- 15GB of Codespaces storage
- Community support

Get additional teacher benefits

GitHub Team

- Protect your branches
Ensure that collaborators on your repository cannot make irreversible changes to branches.
- Draft pull requests
- Required reviewers
- 3,000 CI/CD minutes/month
Free for public repositories
- 2GB of Packages storage
Free for public repositories
- Web-based support

GitHub Teacher Toolbox

- Free access to the industry's best developer tools
Hundreds of offers, including DigitalOcean, Microsoft Azure, Heroku, MongoDB, Datadog, Twilio, and Stripe.

GitHub Classroom

- Automate your course
Create assignments in your dashboard, grade work automatically, and help students when they get stuck.

GitHub Campus Advisors

- Join a community for teachers
GitHub Campus Advisor Git and GitHub, and champion the use of free developer tools at their schools.

Continue for free

Apply for your GitHub teacher benefits

Skip personalization

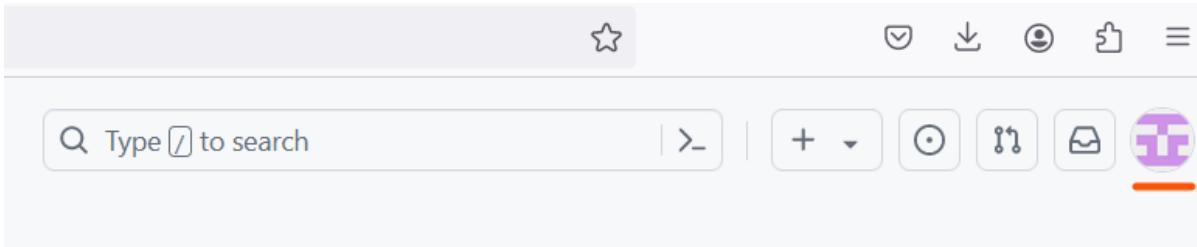
Your GitHub account setup is done and you will see the blow screen.

The screenshot shows the GitHub Home page. On the left, there's a sidebar with 'Create your first project' and 'Recent activity'. The main area has a 'Home' header and a 'Updates to your homepage feed' section. Below that is a 'Start a new repository' form where 'sauvikdevops' is entered. The 'Private' option is selected. A 'Create' button is visible. To the right of the repository form is a 'Introduce yourself with a profile README' section. At the bottom of the page are several promotional banners: 'GitHub Galaxy', 'UNIVERSE24', and 'GitHub Universe'.

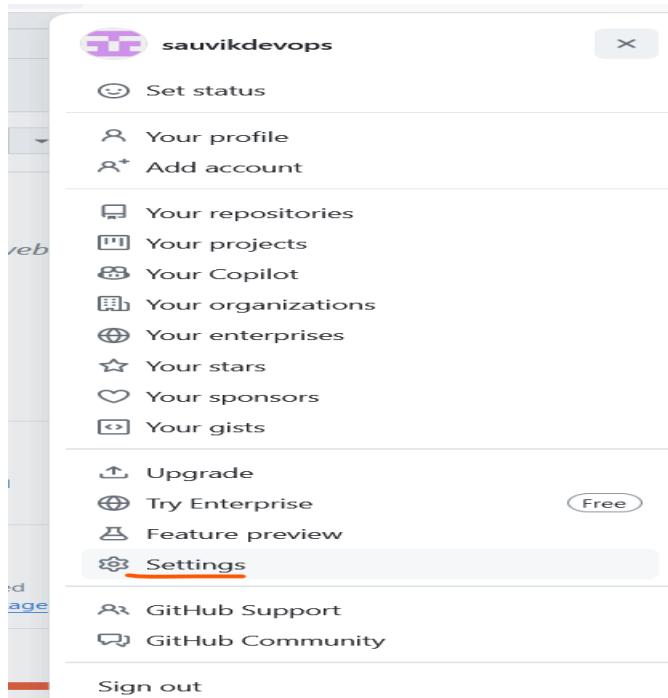
Generate Personal Access Token in GitHub

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

The screenshot shows the GitHub Public profile settings page. The URL in the address bar is <https://github.com/settings/profile>. The page title is 'sauvikdevops (sauvikdevops) Your personal account'. On the left, there's a sidebar with various settings categories:

- Public profile** (selected)
- Account**
- Appearance**
- Accessibility**
- Notifications**
- Access**
 - Billing and plans**
 - Emails**
 - Password and authentication**
 - Sessions**
 - SSH and GPG keys**
 - Organizations**
 - Enterprises**
 - Moderation**
 - Code, planning, and automation**
 - Repositories**
 - Codespaces**
 - Packages**
 - Copilot**
 - Pages**
 - Saved replies**
 - Security**
 - Code security and analysis**
 - Integrations**
 - Applications**
 - Scheduled reminders**
 - Archives**
 - Security log**
 - Sponsorship log**
 - <> Developer settings** (highlighted with a red underline)

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'. Under 'GitHub Apps', the 'Personal access tokens' option is selected and highlighted with a red underline. A sub-menu below it shows 'Tokens (classic)' also underlined in red. To the right, there is a 'GitHub Apps' section with a brief description and a 'Beta' badge.

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

The screenshot shows the 'Personal access tokens (classic)' page at <https://github.com/settings/tokens>. The left sidebar has 'Personal access tokens' selected. On the right, a dropdown menu for 'Generate new token' is open, with 'Generate new token (classic)' highlighted with a red underline. The page includes a search bar and standard GitHub footer links.

Login if asked –

The screenshot shows the 'Confirm access' page at <https://github.com/settings/tokens/new>. It features a GitHub logo, a 'Signed in as @sauvikdevops' message, a password input field with masked text, a 'Forgot password?' link, and a large green 'Confirm' button. A tip at the bottom explains sudo mode. The page includes a standard GitHub footer.

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)

<input checked="" type="checkbox"/> copilot	Full control of GitHub Copilot settings and seat assignments
<input type="checkbox"/> manage_billing:copilot	View and edit Copilot Business seat assignments
<input checked="" type="checkbox"/> project	Full control of projects
<input type="checkbox"/> read:project	Read access of projects
<input checked="" type="checkbox"/> admin:gpg_key	Full control of public user GPG keys
<input checked="" type="checkbox"/> write:gpg_key	Write public user GPG keys
<input checked="" type="checkbox"/> read:gpg_key	Read public user GPG keys
<input checked="" type="checkbox"/> admin:ssh_signing_key	Full control of public user SSH signing keys
<input checked="" type="checkbox"/> write:ssh_signing_key	Write public user SSH signing keys
<input checked="" type="checkbox"/> read:ssh_signing_key	Read public user SSH signing keys

Generate token [Cancel](#)

Your Personal access token is created successfully

Settings / Developer Settings

Some of the scopes you've selected are included in other scopes. Only the minimum set of necessary scopes has been saved.

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](#).

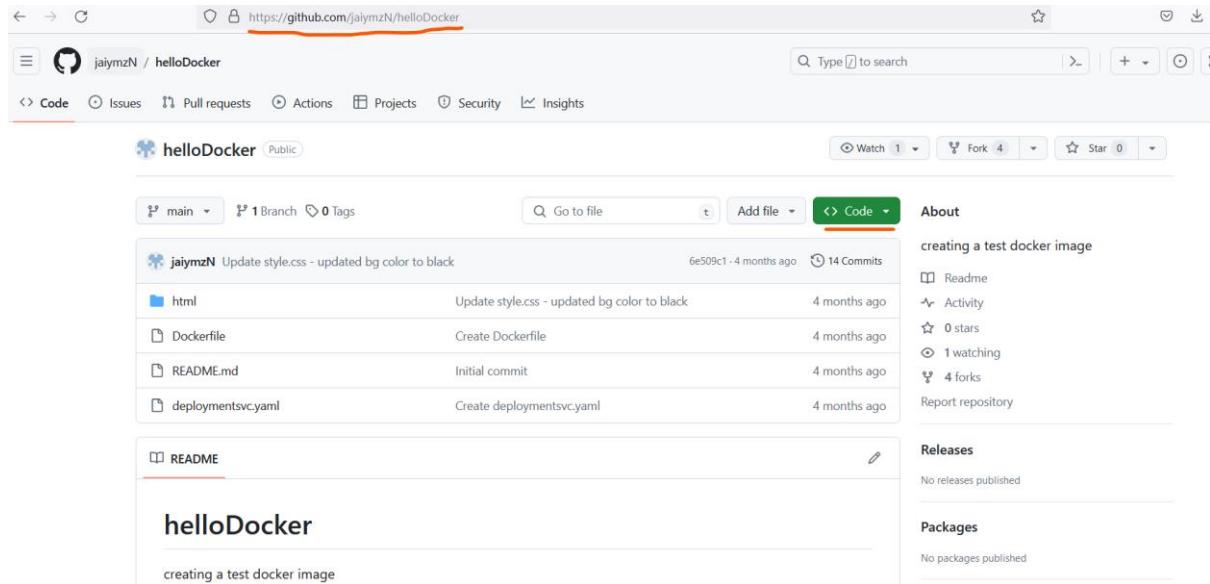
Note: Make sure to copy your personal access token now. You won't be able to see it again!

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).

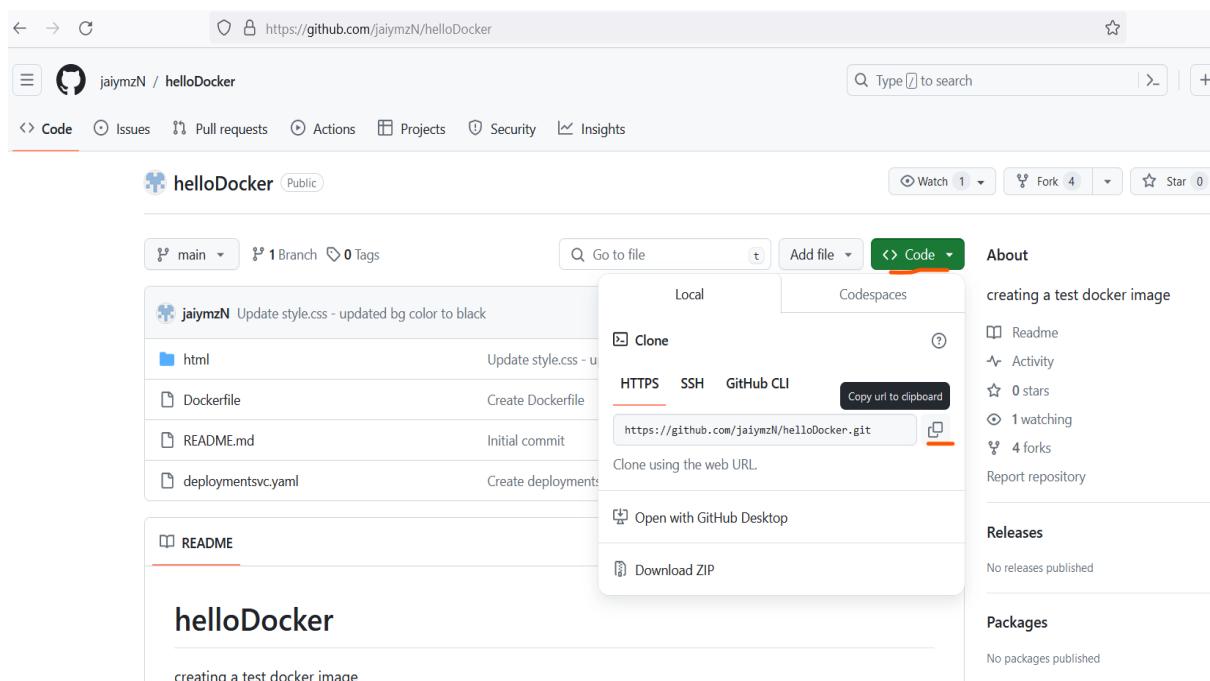
For example, to clone the below public repo (we would not need any credential for this), open this URL in browser:

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



The screenshot shows a GitHub repository page for 'helloDocker'. The URL in the address bar is <https://github.com/jaiymzN/helloDocker>. The repository is public and has 14 commits. The 'Code' button is highlighted with a red box. To its right, there is a copy icon (a square with a curved arrow) and a 'Copy' label.

Click on “Code” followed by copy sign → 



The screenshot shows the same GitHub repository page for 'helloDocker'. The 'Code' button is again highlighted with a red box. A dropdown menu is open under the 'Code' button, specifically the 'Clone' section. It shows three options: 'HTTPS', 'SSH', and 'GitHub CLI'. The 'HTTPS' option is selected, and its URL, <https://github.com/jaiymzN/helloDocker.git>, is highlighted with a red box and has a 'Copy url to clipboard' button next to it. Below the URL, there is a 'Clone using the web URL.' link and two download links: 'Open with GitHub Desktop' and 'Download ZIP'.

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 sidebar with 'Create your first project' and two buttons: 'Create repository' (green) and 'Import repository' (blue, with a red underline). Below these are sections for 'Recent activity' and a note about providing links to actions across GitHub. The main area is titled 'Home' and features a section for 'Updates to your homepage feed' with a note about combining the Following and For you feeds. A 'Learn more' link and a code editor interface ('Start writing code') are also visible.

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' form. The URL field contains 'https://github.com/jaiymzN/helloDocker'. The 'Your new repository details' section shows the owner as 'sauvikdevops' and the repository name as 'dockerdemo'. The 'Public' radio button is selected. The 'Begin import' button is at the bottom right.

5. Click on “Begin Import”

A screenshot of a web browser showing a GitHub import progress page. The URL in the address bar is <https://github.com/sauvikdevops/dockerdemo/import>. The page title is "Preparing your new repository". A sub-header says "There is no need to keep this window open. We'll email you when the import is done." Below the header is a large circular progress bar. Underneath it, the text "Your import will begin shortly..." is displayed.

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

A screenshot of a web browser showing a GitHub import confirmation page. The URL in the address bar is <https://github.com/sauvikdevops/dockerdemo/import>. The page title is "Preparing your new repository". A sub-header says "There is no need to keep this window open. We'll email you when the import is done." Below the header is a green checkmark icon. Underneath it, the text "Importing complete! Your new repository [sauvikdevops/dockerdemo](https://github.com/sauvikdevops/dockerdemo) is ready." is displayed.

Click on the newly created repo link (for example [sauvikdevops/dockerdemo](https://github.com/sauvikdevops/dockerdemo) here) to go to there.

A screenshot of a web browser showing a GitHub repository page for "dockerdemo". The URL in the address bar is <https://github.com/sauvikdevops/dockerdemo>. The repository name "dockerdemo" is underlined in red. The "Code" dropdown menu is open, showing options for "Local" and "Codespaces". The "Clone" section is highlighted, showing "HTTPS", "SSH", and "GitHub CLI" options. The "HTTPS" URL <https://github.com/sauvikdevops/dockerdemo.git> is also underlined in red. Other options include "Copy url to clipboard", "Clone using the web URL", "Open with GitHub Desktop", and "Download ZIP". To the right of the code dropdown, there is an "About" section with a "No description, website, or topics provided." message, and sections for "Releases", "Packages", and "Settings".

Note the repo URL which we would need later for CI/CD 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: ~  
root@docker-minikube-server:~# curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64  
sudo install minikube-linux-amd64 /usr/local/bin/minikube  
minikube version  
  % 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
```

```
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
```

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

```
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/>

Welcome to Jenkins!

This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.

Start building your software project

Set up a distributed build

REST API Jenkins 2.452.1

Configuring Docker Hub authentication

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

Welcome to Jenkins!

This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.

Start building your software project

Click on Credentials ->

Manage Jenkins

System Configuration

- System**: Configure global settings and paths.
- Tools**: Configure tools, their locations and automatic installers.
- Plugins**: Add, remove, disable or enable plugins that can extend the functionality of Jenkins.
- Appearance**: Configure the look and feel of Jenkins.

Security

- Credentials**: Configure credentials (highlighted with a red border).
- Credential Providers**: Configure the credential providers and types.
- Users**: Create/delete/modify users that can log in.

Click on "(global) ->

The screenshot shows the Jenkins 'Credentials' management interface. At the top, there's a navigation bar with links to 'Dashboard', 'Manage Jenkins', and 'Credentials'. A search bar is also present. Below the navigation, the title 'Credentials' is displayed. A table header row includes columns for 'T', 'P', 'Store ↓', 'Domain', 'ID', and 'Name'. Under the 'Stores scoped to Jenkins' section, a sub-table shows a single entry for 'System' under the 'Domains' column, with '(global)' highlighted in red. At the bottom left, there are icon size options: S, M, and L.

Credentials

T	P	Store ↓	Domain	ID	Name
Stores scoped to Jenkins					
P Store ↓ Domains					
	System		(global)		
Icon:	S	M	L		

Click on Add Credentials

The screenshot shows the 'Global credentials (unrestricted)' page. The URL is 'localhost:8080/manage/credentials/store/system/domain/_/'. The page title is 'Global credentials (unrestricted)'. There is a blue button labeled '+ Add Credentials' with a red underline. Below it, a message says 'Credentials that should be available irrespective of domain specification to requirements matching.' A table header row includes columns for 'ID', 'Name', 'Kind', and 'Description'. A note at the bottom states 'This credential domain is empty. How about adding some credentials?' At the bottom left, there are icon size options: S, M, and L.

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

New credentials

Kind

Secret text

Scope ?

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

Secret

.....

ID ?

docker_hub

Description ?

Docker Hub authentication

Create

Configure Kubernetes authentication for Jenkins

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

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

Go to “manage Jenkins” -> plugins

Manage Jenkins

System Configuration

Tools

Clouds

Plugins

Build Queue

No builds in the queue.

Build Executor Status

1 idle
2 idle

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 highlighted. It contains a 'File' input field with 'Browse...' and 'No file selected.' text, and a 'URL' input field with a placeholder 'URL'. A 'Deploy' button is at the bottom.

The screenshot shows the Jenkins 'Advanced settings' page under the 'Plugins' section. The 'Deploy Plugin' section is highlighted. A 'File' input field has 'Browse...' and 'No file selected.' text. An 'Open' button is highlighted in red. A 'File Upload' dialog is open, showing a file selection tree. The 'kubernetes-cd.hpi' file is selected in the 'Virtual Machine' folder. The 'File name:' field shows 'kubernetes-cd.hpi' and the 'Open' button is also highlighted in red.

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 the Manage Jenkins menu. The left sidebar has links for Updates, Available plugins, Installed plugins, Advanced settings, and Download progress, with 'Download progress' currently selected. The main content area is titled 'Download progress' and shows a table of plugin installations:

Preparation	
SSH server	Success
Azure Commons	Installing
Authentication Tokens API	Pending
Docker Commons	Pending
Preparation	Pending
kubernetes-cd	Pending

Below the table 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: ~
[jenkins@jenkins-server ~]$ sudo systemctl restart 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: 2276)
        Memory: 347.0M (peak: 348.5M)
         CPU: 1min 37.153s
        CGroup: /system.slice/jenkins.service
               └─4146 /usr/bin/java -Djava.awt.headless=true -jar /var/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$1#onAttained: Configuration for all jobs updated
May 27 10:48:33 jenkins-server jenkins[4146]: 2024-05-27 10:48:33.430+0000 [id=42]      INFO    hudson.util.Retriger$Start: Attempt #1 to do the action check updates server
May 27 10:48:33 jenkins-server jenkins[4146]: 2024-05-27 10:48:33.431+0000 [id=43]      INFO    jenkins.InitReactorRunner$1#onAttained: Completed initialization
May 27 10:48:34 jenkins-server jenkins[4146]: 2024-05-27 10:48:34.126+0000 [id=24]      INFO    hudson.lifecycle.Lifecycle$LifecycleContainer: Jenkins is fully up and running
May 27 10:48:34 jenkins-server systemd[1]: Started Jenkins.service - Jenkins Continuous Integration Server.
May 27 10:49:13 jenkins-server jenkins[4146]: 2024-05-27 10:49:13.139+0000 [id=47]      INFO    h.m.DownloadService$Downloadable#load: Obtained the updated data file for hudson.tasks.Maven.MavenInstaller
May 27 10:49:14 jenkins-server jenkins[4146]: 2024-05-27 10:49:14.422+0000 [id=48]      INFO    h.m.DownloadService$Downloadable#load: Obtained the updated data file for hudson.tasks.Ant.AntInstaller
May 27 10:49:15 jenkins-server jenkins[4146]: 2024-05-27 10:49:15.175+0000 [id=49]      INFO    h.m.DownloadService$Downloadable#load: Obtained the updated data file for hudson.tools.JDKInstaller
May 27 10:49:23 jenkins-server jenkins[4146]: 2024-05-27 10:49:23.209+0000 [id=47]      INFO    hudson.util.Retriger$Start: Performed the action check updates server successfully at the attempt #1
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)'. The 'Scope' dropdown is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'ID' field contains 'k8_auth'. The 'Description' field contains 'Kubernetes Authentication'. Below the fields, there are two options: 'Enter directly' (selected) and 'From a file on the Jenkins master'. A blue '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

From a file on the Jenkins master

From a file on the Kubernetes master node

`apiVersion: v1
clusters:
- cluster:
 certificate-authority-data:
LS0tLS1CRUJdTlBRVlUSUZJQ0FURS0tLS0tClk1JSURCakNDQWU2Z0F3SUJBZ0lCQVRBTkJna3Foa2lHOXcwQkFrC0ZBREFWTvJn0VRWURWUVFERXdwdGFXNXAKYTNWaVpVtIjNQjRyRfRjME1EVXgNVUvTURnME5sb1hEV0wTURVek1ERTJNRGcwImxvd0ZURVRNokVHOTFVRQoBeE1LYldsWFxdDFZbVZEUVRDOqFTSxdeUvIKs29aSWh2Y05BUUVCOlEBRGdnRVBBREND0Fv...`

Create

Click on - Create

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

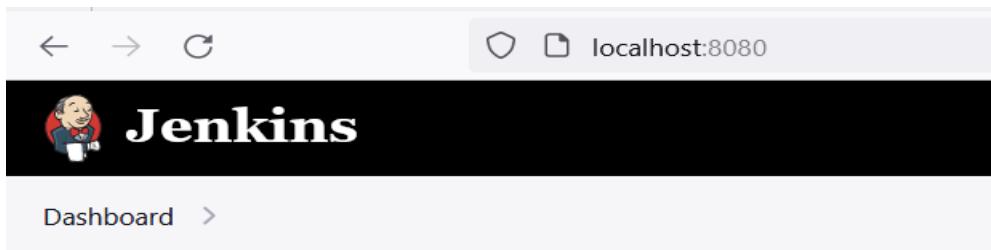
Global credentials (unrestricted)

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

Create CICD pipeline in Jenkins

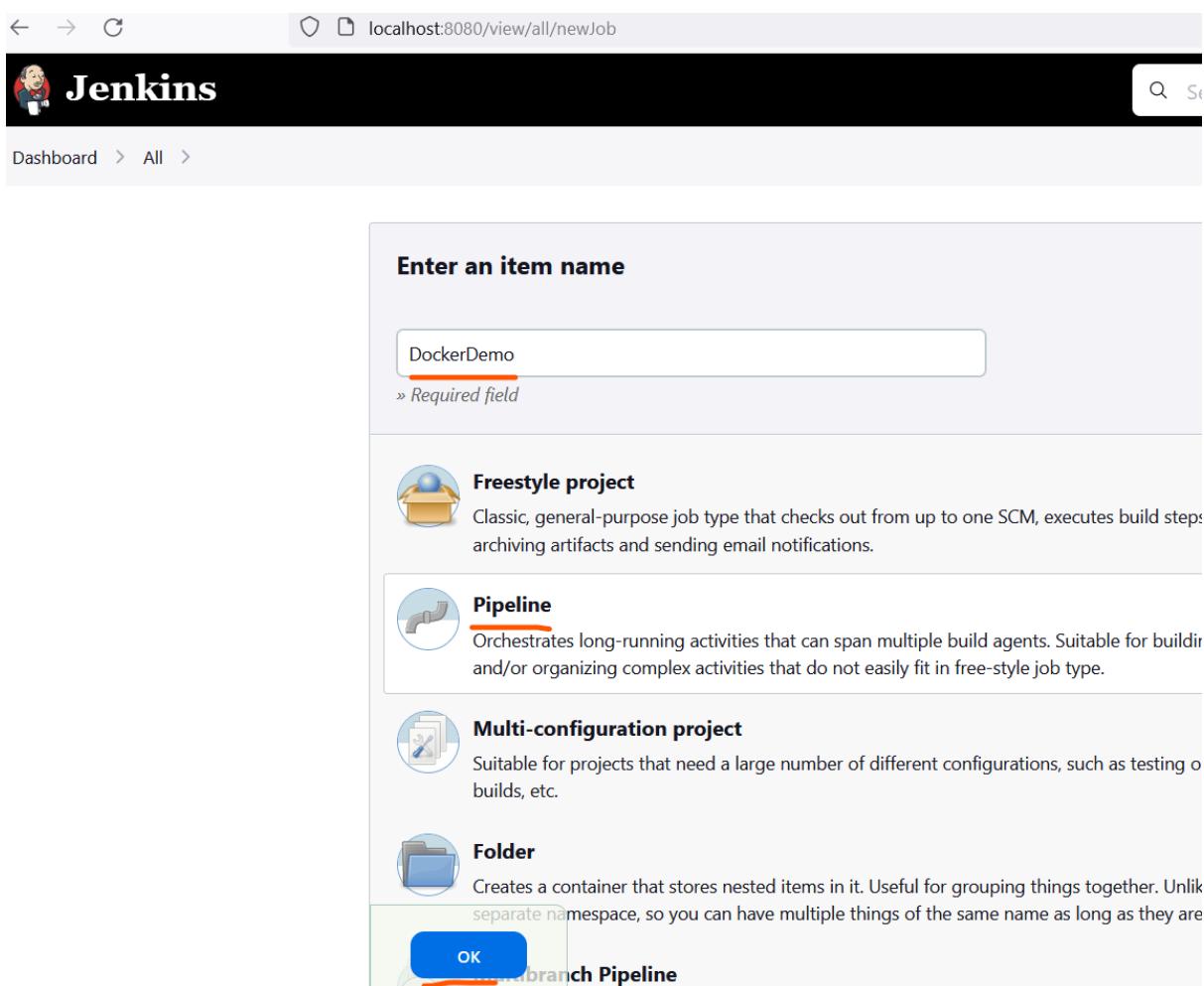
Go to your Jenkins URL / console and configure the following

Create a new item



The screenshot shows the Jenkins dashboard at localhost:8080. At the top, there's a navigation bar with back, forward, and refresh buttons, and a URL field showing 'localhost:8080'. Below the header is the Jenkins logo and the word 'Jenkins' in large white letters. A 'Dashboard' link is visible. On the left, a sidebar contains a 'New Item' button (which is underlined), 'Build History', 'Manage Jenkins', and 'My Views'.

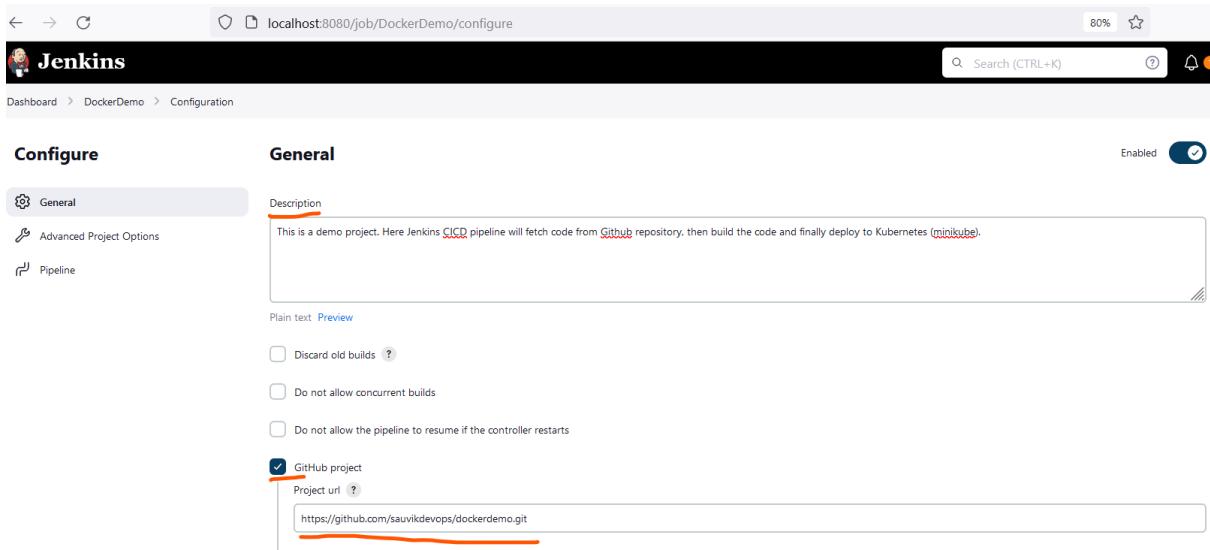
Select the Pipeline Option



The screenshot shows the 'New Job' creation page at localhost:8080/view/all/newJob. The title bar says 'localhost:8080/view/all/newJob'. The main area has a heading 'Enter an item name' with a text input field containing 'DockerDemo' (which is underlined) and a note '» Required field'. Below this, there are four project types listed: 'Freestyle project' (with a box icon), 'Pipeline' (with a pipe icon), 'Multi-configuration project' (with a gear icon), and 'Folder' (with a folder icon). Each item has a brief description. At the bottom right of the form, there are 'OK' and 'Cancel Pipeline' buttons.

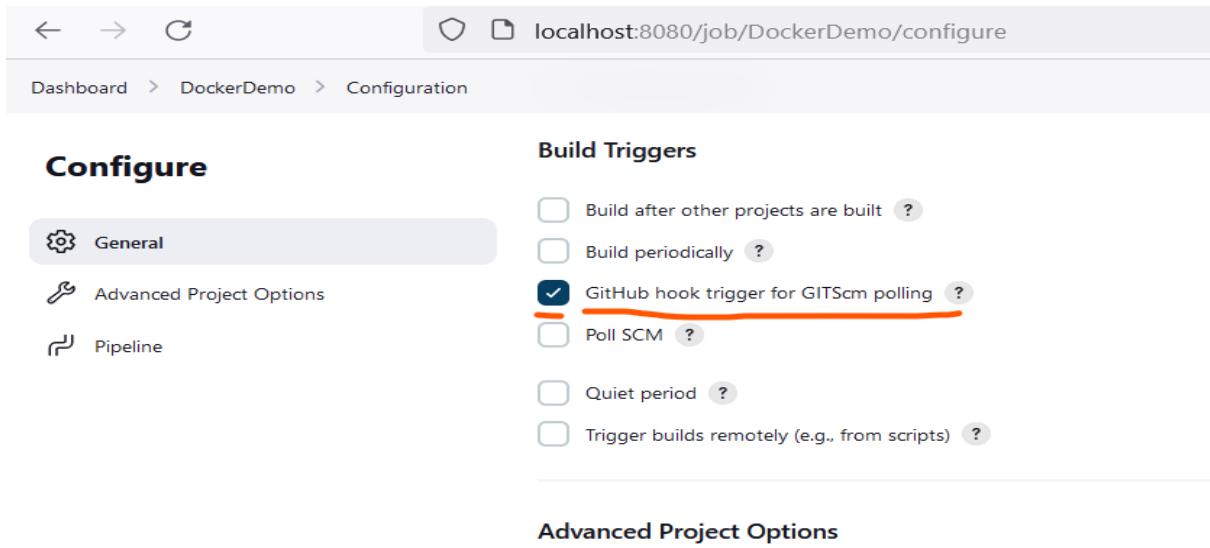
Enter a Description (optional)

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 CI/CD pipeline will fetch code from GitHub repository, then build the code and finally deploy to Kubernetes (minikube)'. Below the description, there are several configuration options: 'Discard old builds', 'Do not allow concurrent builds', 'Do not allow the pipeline to resume if the controller restarts', and 'GitHub project'. The 'GitHub project' option is selected, and its 'Project url' is set to 'https://github.com/sauvikdevops/dockerdemo.git'. A red box highlights the 'Project url' input field.

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 trigger options like 'Build after other projects are built', 'Build periodically', 'Poll SCM', 'Quiet period', and 'Trigger builds remotely' are also listed but not selected. A red box highlights the 'GitHub hook trigger for GITscm polling' checkbox.

Go to the pipeline section and fill the following to add CI/CD 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)

Pipeline

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://github.com/sauvikdevops/dockerdemo

Credentials ?

- none -

+ Add ▾

Advanced ▾

This screenshot shows the 'Pipeline' configuration page in Jenkins. Under the 'Definition' section, 'Pipeline script from SCM' is selected. In the 'SCM' section, 'Git' is chosen. The 'Repository URL' field contains 'https://github.com/sauvikdevops/dockerdemo'. The 'Credentials' dropdown is set to '- none -'. There are 'Add' and 'Advanced' buttons at the bottom of this section.

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

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

Branches to build ?

Branch Specifier (blank for 'any') ?

*/main

Add Branch

Repository browser ?

(Auto)

Additional Behaviours

Add ▾

Script Path ?

Jenkinsfile

Lightweight checkout ?

Pipeline Syntax

Save

Apply

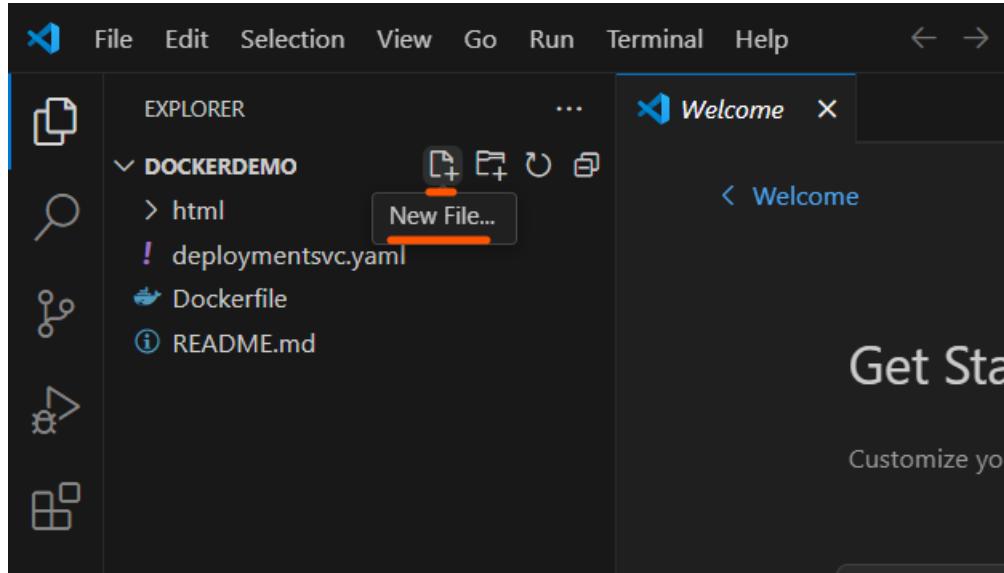
This screenshot continues the Jenkins Pipeline configuration. It shows the 'Branches to build' section with 'Branch Specifier' set to '*/main'. Below it is an 'Add Branch' button. The 'Repository browser' is set to '(Auto)'. Under 'Additional Behaviours', there is an 'Add' dropdown. The 'Script Path' is set to 'Jenkinsfile'. A checked checkbox for 'Lightweight checkout' is also present. At the bottom 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'
    }
}

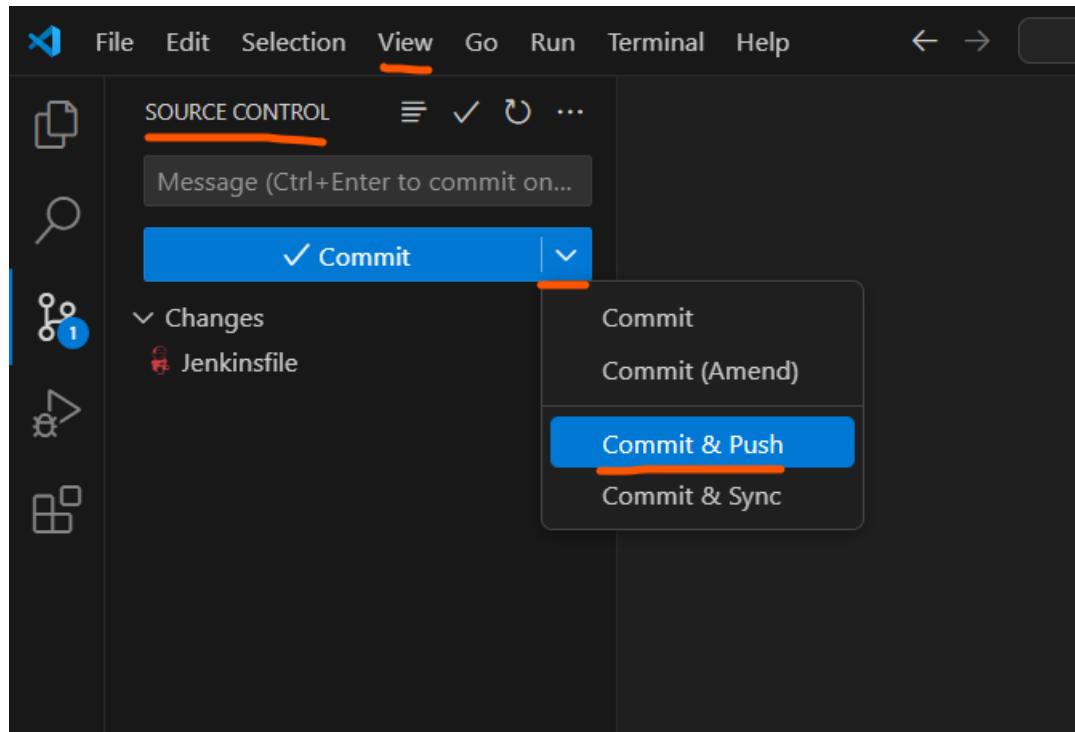
stage('Deploy deployment and service file') {
    steps {
        script {
            kubernetesDeploy configs: 'deploymentsvc.yaml', kubeconfigId: 'k8_auth'
        }
    }
}

```

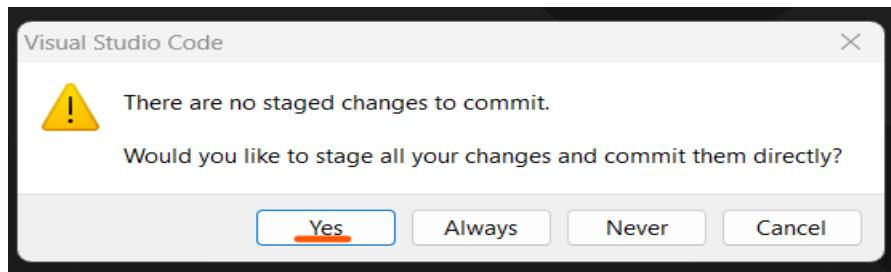
After that save the changes using Ctrl+ S or File menu → Save

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

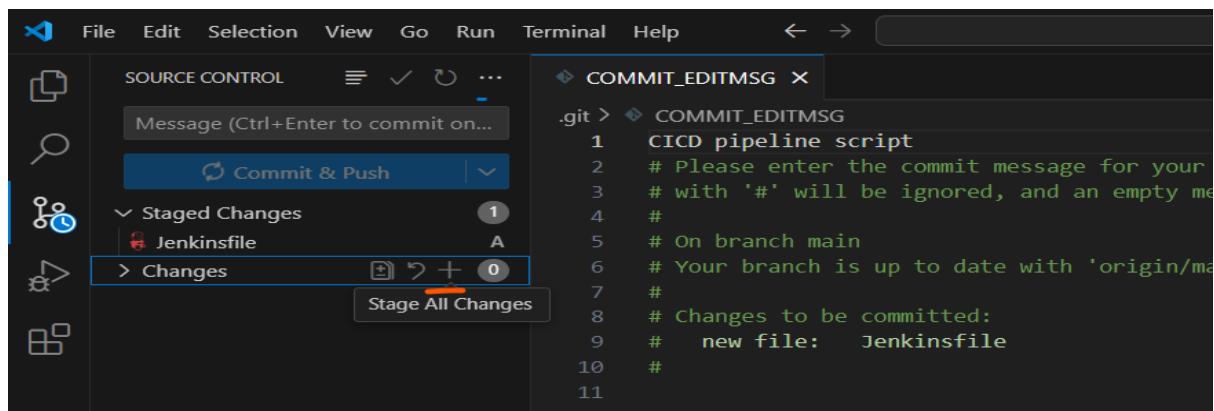
Now, click on the downward pointing arrow at right side of Commit and click on Commit & Push.



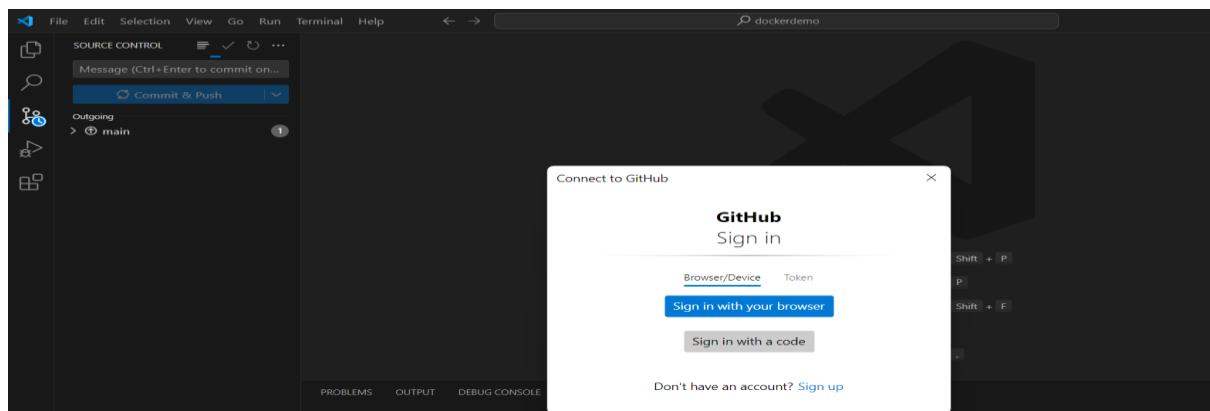
Click on Yes to continue



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.



Don't have an account? [Sign up](#)

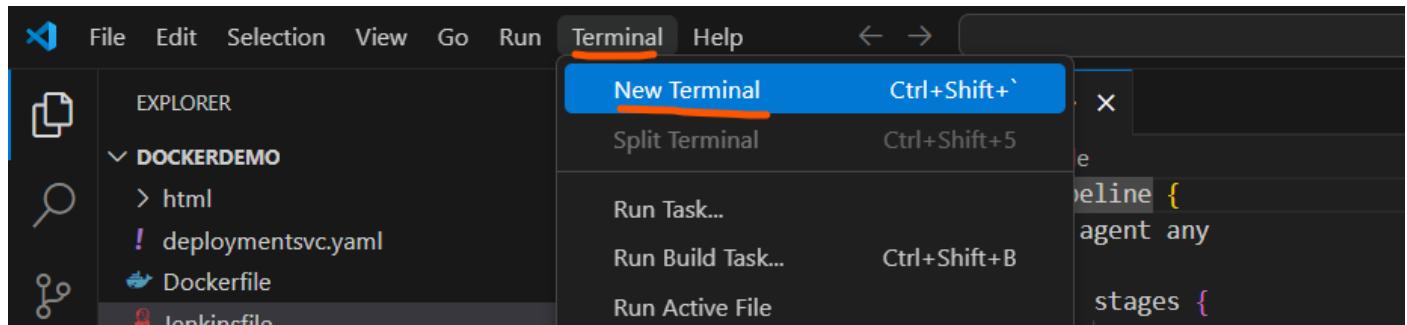
Now you can check GitHub URL to see that the changes reflected.

The screenshot shows a GitHub repository page for 'sauvikdevops / dockerdemo'. The repository is public and contains 15 commits. The commits are as follows:

- html: Update style.css - updated bg color to black (4 months ago)
- Dockerfile: Create Dockerfile (4 months ago)
- Jenkinsfile: CICD pipeline script (1 hour ago)
- README.md: Initial commit (4 months ago)
- deploymentsvc.yaml: Create deploymentsvc.yaml (4 months ago)

The repository has 2 branches and 0 tags. On the right side, there is an 'About' section with no description, website, or topics provided. It also shows activity, 0 stars, 1 watching, and 0 forks. There are sections for 'Releases' (no releases published), 'Packages' (no packages published), and 'Contributors' (2 contributors).

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

```
git add .
```

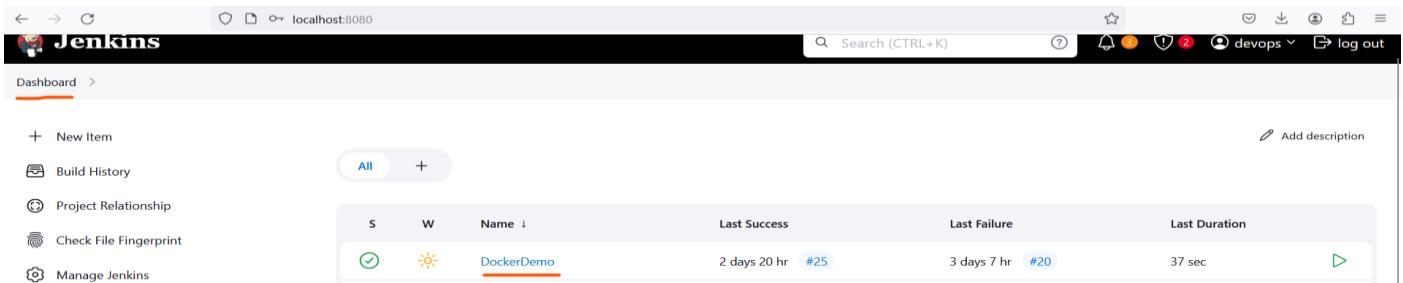
```
git commit -m "your commit message"
```

```
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. 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 'Pipeline' icon. Below it, a table lists a single pipeline job:

S	W	Name ↓	Last Success	Last Failure	Last Duration
		DockerDemo	2 days 20 hr #25	3 days 7 hr #20	37 sec

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

Snippet Generator will come up

Pipeline script to check out the GitHub repository

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

The screenshot shows the Jenkins Pipeline Syntax Snippet Generator page. The URL is `localhost:8080/job/DockerDemo/pipeline-syntax/`. The page has a sidebar with links like Dashboard, Back, Snippet Generator (which is selected), 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, a "checkout" step is configured under "SCM". The "Repository URL" field contains `https://github.com/sauvikdevops/dockerdemo.git`. The "Credentials" dropdown is 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 for a job named 'DockerDemo'. The 'Branches to build' section has a branch specifier of '*/*main'. Under 'Additional Behaviours', 'Include in polling?' and 'Include in changelog?' are checked. The 'Generate Pipeline Script' button is visible, and the generated code is:

```
checkout scmGit(branches: [[name: '*/*main']], extensions: [], userRemoteConfigs: [[url: 'https://github.com/sauvikdevops/dockerdemo.git']])
```

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
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']])
```

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

This screenshot shows the Jenkins Pipeline Syntax Snippet Generator interface. The URL is `localhost:8080/job/DockerDemo/pipeline-syntax/`. The left sidebar has links for Dashboard, DockerDemo, Pipeline Syntax, Snippet Generator (which is selected), 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' is shown with the code `withCredentials: Bind credentials to variables`. Below it, another 'withCredentials' step is listed with the note "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". There is a 'Bindings' section with an 'Add' dropdown menu. At the bottom is a 'Generate Pipeline Script' button and a preview box containing the Jenkinsfile code:

```
checkout scmGit(branches: [[name: '/main']], extensions: [], userRemoteConfigs: [[url: 'https://github.com/sauvikdevops/dockerdemo.git']])
```

From the dropdown, select - Secret text

This screenshot shows the same Jenkins Pipeline Syntax Snippet Generator interface as the previous one. The 'Secret text' option is highlighted in the dropdown menu under the 'withCredentials' step. The preview box at the bottom shows the Jenkinsfile code:

```
checkout scmGit(branches: [[name: '/main']], extensions: [], userRemoteConfigs: [[url: 'https://github.com/sauvikdevops/dockerdemo.git']])
```

Now, put the variable = docker_hub

For credential, click on + Add

Select Jenkins

This screenshot shows the Jenkins Credentials Provider interface. It lists a single item: 'Docker hub authentication'. Below it, the Jenkins Credentials Provider section shows two Jenkins items: 'Jenkins' and 'Jenkins'.

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

+ Add

Add

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.

The screenshot shows the Jenkins Pipeline Snippet Generator interface. Under the 'Steps' section, a 'Sample Step' is selected as 'kubernetesDeploy: Deploy to Kubernetes'. This step has a 'Kubeconfig' dropdown set to 'k8_auth (Kubernetes Authentication)'. Other options like 'Deprecated Kubeconfig Settings' and 'Config Files' are present. A checkbox for 'Enable Variable Substitution in Config' is checked. At the bottom, there's a 'Verify Configuration' button and a 'Generate Pipeline Script' button. The generated pipeline script is displayed below, starting with 'kubernetesDeploy config: "", kubeConfig: [path: ""]'.

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')
```

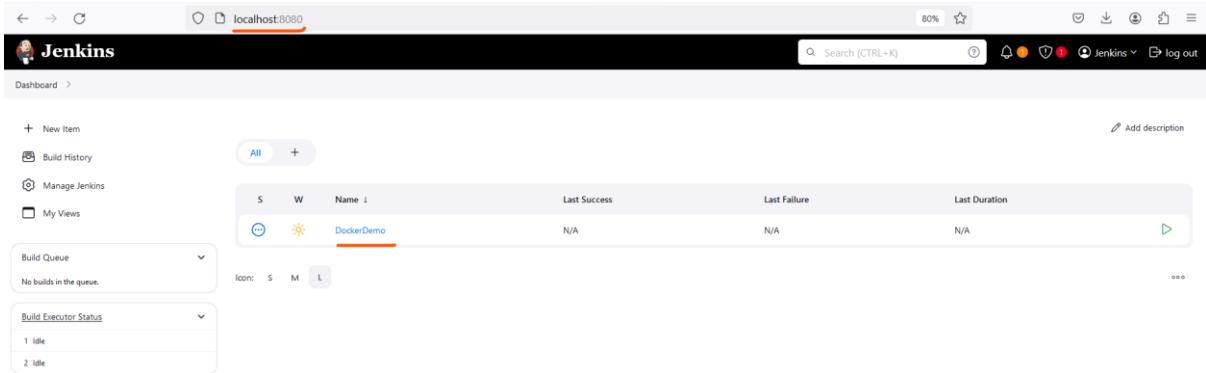
Remove the unnecessary sections so that your code will look like this:

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

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

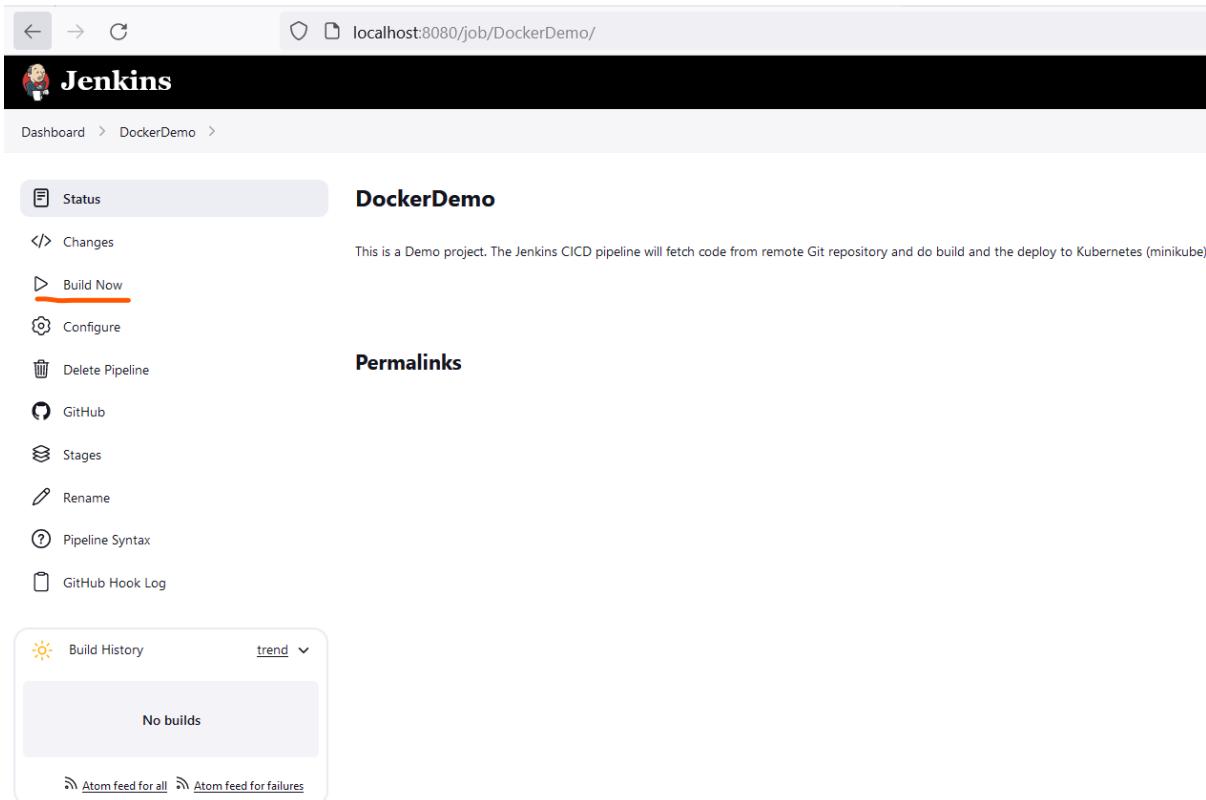
Run Jenkins Pipeline

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. The 'DockerDemo' project is listed in the main table under the 'All' category. The table columns are S (Status), W (Work), Name (DockerDemo), Last Success (N/A), Last Failure (N/A), and Last Duration (N/A). On the left sidebar, there are links for 'New Item', 'Build History', 'Manage Jenkins', and 'My Views'. Below the sidebar are sections for 'Build Queue' (No builds in the queue) and 'Build Executor Status' (1 idle, 2 idle).

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



The screenshot shows the Jenkins job page for 'DockerDemo' at localhost:8080/job/DockerDemo/. The top navigation bar shows 'Dashboard > DockerDemo >'. The left sidebar has options: Status (highlighted), Changes, Build Now (highlighted), Configure, Delete Pipeline, GitHub, Stages, Rename, Pipeline Syntax, and GitHub Hook Log. The main content area is titled 'DockerDemo' and contains the message: '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. The 'Build History' section shows 'No builds' and includes links for 'Atom feed for all' and 'Atom feed for failures'.

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

A screenshot of a web browser showing the Jenkins interface. The address bar shows 'localhost:8080/job/DockerDemo'. The page title is 'Jenkins'. Below it, the breadcrumb navigation shows 'Dashboard > DockerDemo >'. The main content area has a header 'DockerDemo' with a 'Status' button. 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'. A callout box highlights the 'Build History' section, which shows a single build entry: '#1 May 27, 2024, 4:05 PM'. At the bottom of the history section, there are links for 'Atom feed for all' and 'Atom feed for failures'. The status of the build is indicated by a blue progress bar.

DockerDemo

</> Changes

This is a Demo project. The Jenkin

▷ Build Now

⚙ Configure

🗑 Delete Pipeline

Permalinks

GitHub

☰ Stages

✍ Rename

(?) Pipeline Syntax

📋 GitHub Hook Log

A callout box highlights the 'Build History' section of the Jenkins interface. It shows a single build entry: '#1 May 27, 2024, 4:05 PM'. The status of the build is indicated by a blue progress bar. At the bottom of the history section, there are links for 'Atom feed for all' and 'Atom feed for failures'.

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 a cloud icon, 'Build History', a dropdown menu set to 'trend', a search bar with 'Filter...', and a URL 'localhost:8080/job/DockerDemo/'. Below the header, the Jenkins logo is displayed. The main content area has a dark header with the project name 'DockerDemo'. On the left, a sidebar menu is open, showing options like 'Status' (selected), 'Changes' (highlighted in red), 'Console Output' (selected), 'Edit Build Information', 'Delete build #1', 'Timings', 'Git Build Data', 'Pipeline Overview', 'Pipeline Console', 'Restart from Stage', 'Replay', 'Pipeline Steps', and 'Workspaces'. The 'Console Output' tab is currently active. The main content area displays the Jenkins pipeline configuration. A section titled 'Permalinks' 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
- Last completed build (#1), 2 min 1 sec ago

At the bottom of the page, there are links for 'Atom feed for all' and 'Atom feed for failures'.

You can see CI/CD 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] 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 fetch --tags --force --progress -- https://github.com/sauvikdevops/dockerdemo +refs/heads/*:refs/remotes/origin/*
> git rev-parse refs/remotes/origin/main^{commit} # timeout=10
Checking out Revision 71aa617a34a0ff9cfe7887b582277a5d28947818a (refs/remotes/origin/main)
> git config core.sparsecheckout # timeout=10
> git checkout -f 71aa617a34a0ff9cfe7887b582277a5d28947818a # timeout=10
Commit message: "updated msg"
> git rev-list --no-walk 782e27117ed4247eebe84ee2e17692d53658cd1d # timeout=10
[Pipeline]
[Pipeline] // stage
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] {
[Pipeline] checkout

```

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: ~
devops@jenkins-minikube:~$ minikube dashboard --url
* Verifying dashboard health ...
* Launching proxy ...
* Verifying proxy health ...
http://127.0.0.1:4317/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/#/workloads?namespace=default>

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

Name	Images	Labels	Pods	Created
helldocker-deployment	jamesndubuis/helldocker	-	1 / 1	9.hours.ago

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
```

Page 96 of 98

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

```
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:

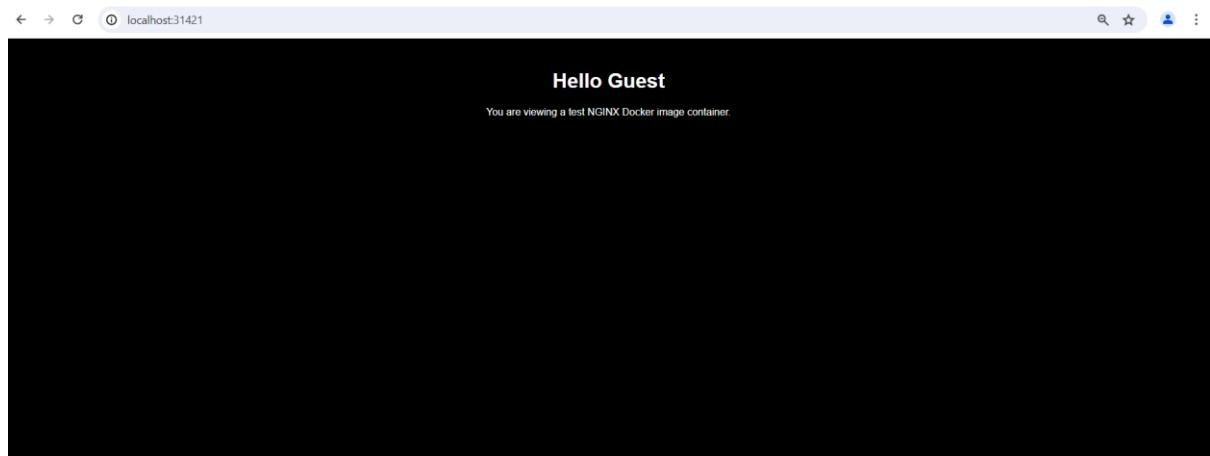
```
sudo lsof -i -P -n | grep LISTEN
```

```
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.