**Assignment No 4**

NAME:-Tejas Bhosale

CLASS:-BE A

BATCH:-A3

GR NO:-17U593 ROLL NO:-421073

**Title**: Create a VM using Docker and write your own docker file and publish it. Assignment has 2 parts:

1. Install and use docker
2. Create your own image and share it or publish it. (Image containing base : ubuntu and software installed: Apache web server)

**Theory**:

**What is Docker:**

Docker is a set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, [libraries a](https://en.wikipedia.org/wiki/Library_(computing))nd configuration files; they can communicate with each other through well-defined channels. All containers are run by a single operating system kernel and therefore use fewer resources than virtual machines.

**What is the use of Docker:**

# Fast, consistent delivery of your applications

Docker streamlines the development lifecycle by allowing developers to work in standardized environments using local containers which provide your applications and services. Containers are great for continuous integration and continuous delivery (CI/CD) workflows.

Consider the following example scenario:

* Your developers write code locally and share their work with their colleagues using Docker containers.
* They use Docker to push their applications into a test environment and execute automated and manual tests.
* When developers find bugs, they can fix them in the development environment and redeploy them to the test environment for testing and validation.
* When testing is complete, getting the fix to the customer is as simple as pushing the updated image to the production environment.

# Responsive deployment and scaling

Docker’s container-based platform allows for highly portable workloads. Docker containers can run on a developer’s local laptop, on physical or virtual machines in a data center, on cloud providers, or in a mixture of environments.

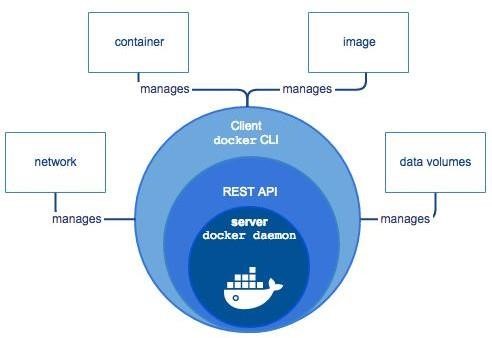
Docker’s portability and lightweight nature also make it easy to dynamically manage workloads, scaling up or tearing down applications and services as business needs dictate, in near real time.

# Running more workloads on the same hardware

Docker is lightweight and fast. It provides a viable, cost-effective alternative to hypervisor-based virtual machines, so you can use more of your compute capacity to achieve your business goals.

Docker is perfect for high density environments and for small and medium deployments where you need to do more with fewer resources.

# Docker Architecture / Docker Engine

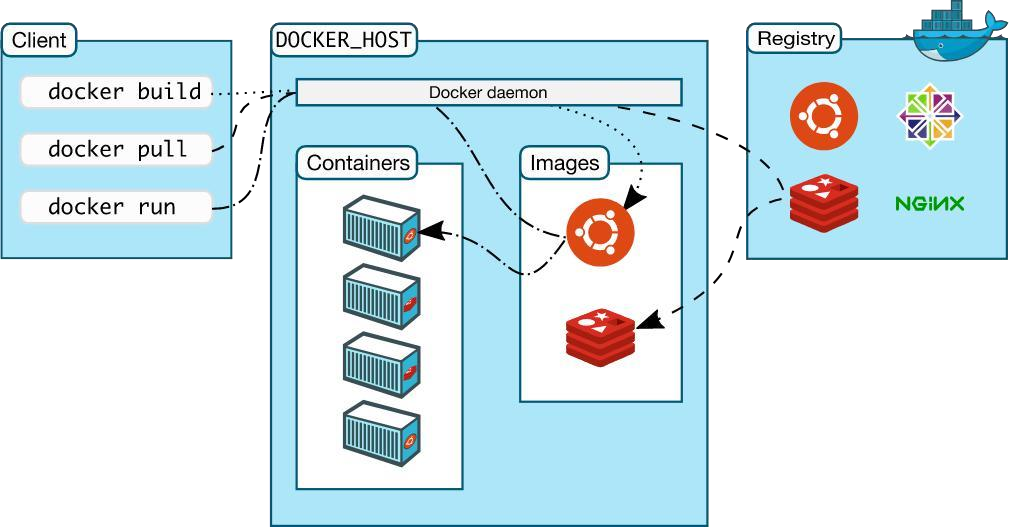


Docker Engine is a client-server application with these major components:

A server which is a type of long-running program called a daemon process (the dockerd command).

A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.

•



A command line interface (CLI) client (the docker command).

# The Docker daemon

The Docker daemon (docker) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

# The Docker client

The Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

# Docker registries

A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.

When you use the docker pull or docker run commands, the required images are pulled from your configured registry. When you use the docker push command, your image is pushed to your configured registry.

# Docker objects

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

# Images

An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a Dockerfile with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

# Containers

A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

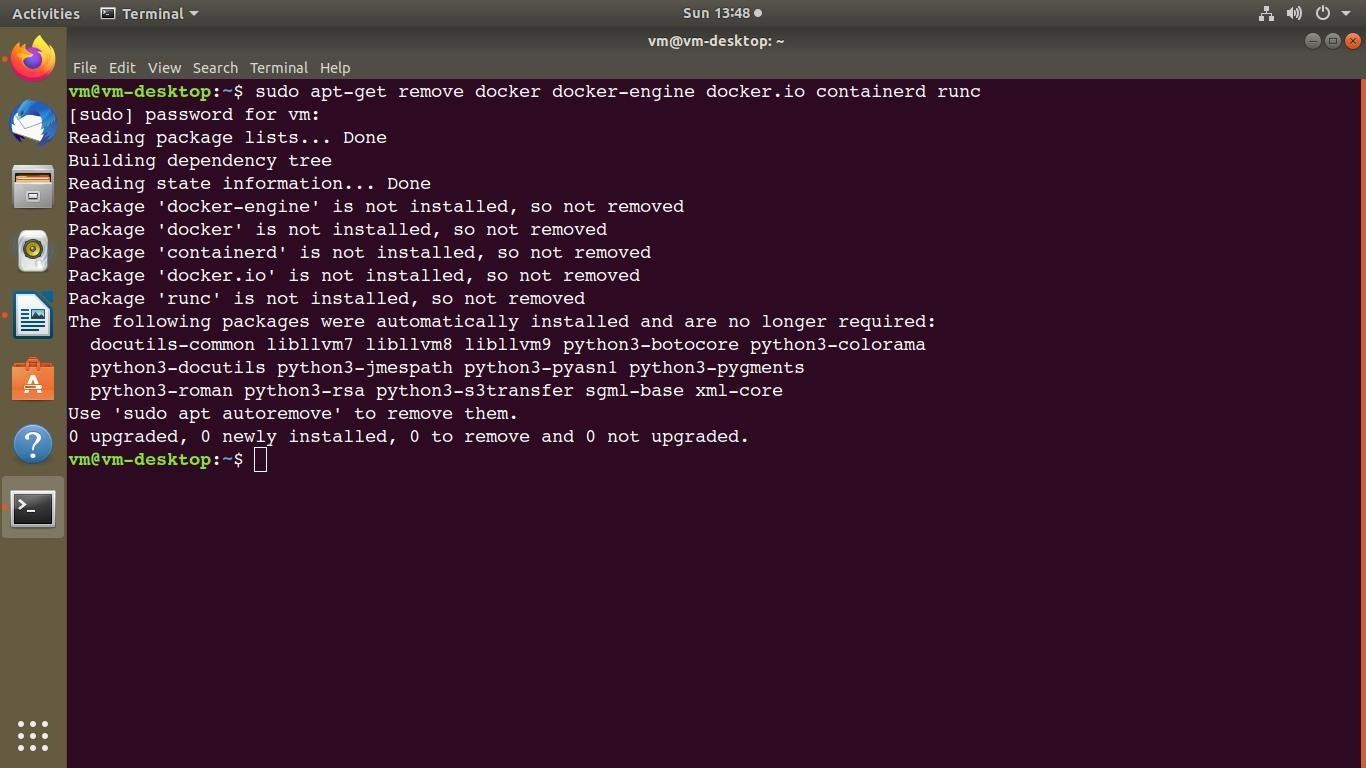
**Steps to install Docker on Ubuntu** You can install docker in 3 ways:

1. Install using the repository (This is the recommended approach)
2. Install from a package
3. Install using the convenience script (most easiest way)

We will use fist method i.e. Install using repository

Step1: Before staring installation remove the old installation if any, by using command:

$ sudo apt-get remove docker docker-engine docker.io containerd runc

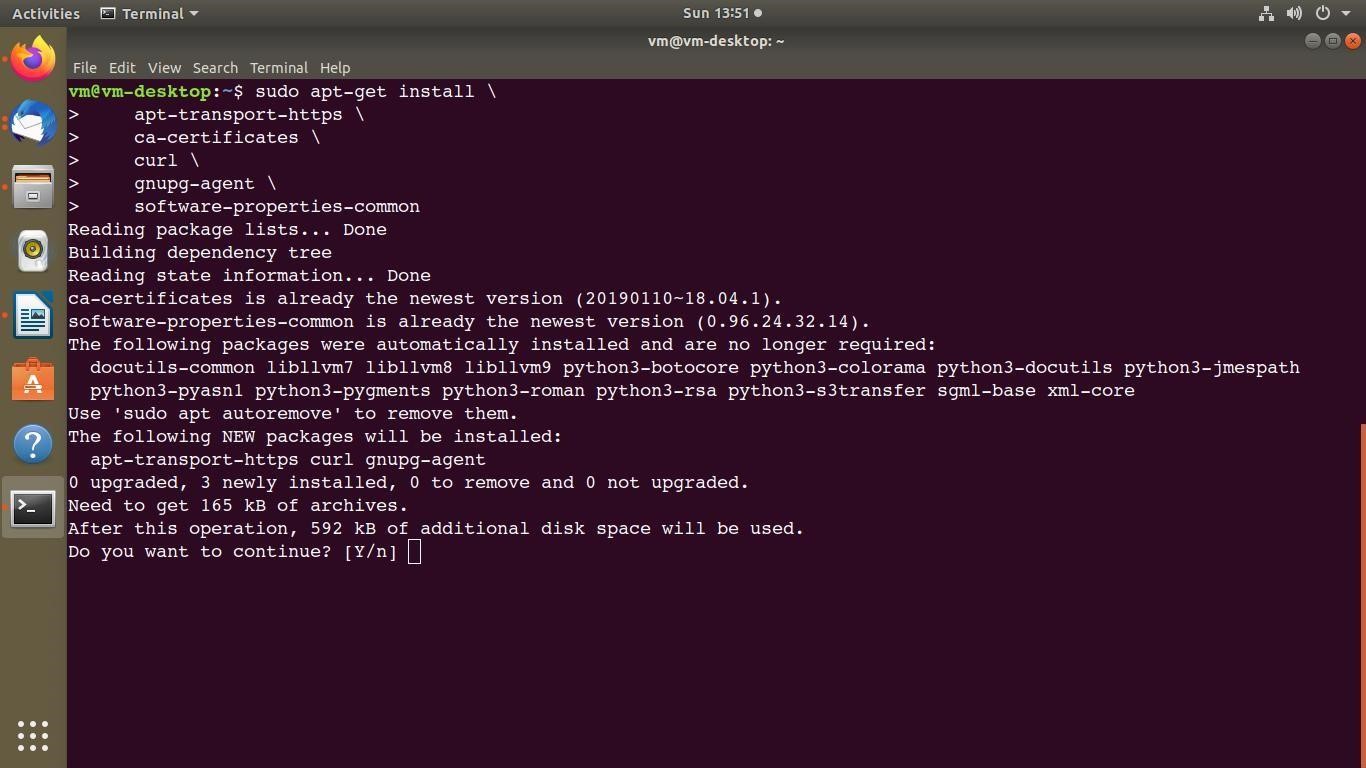


Step 2: update the packages by using command:

sudo apt-get update

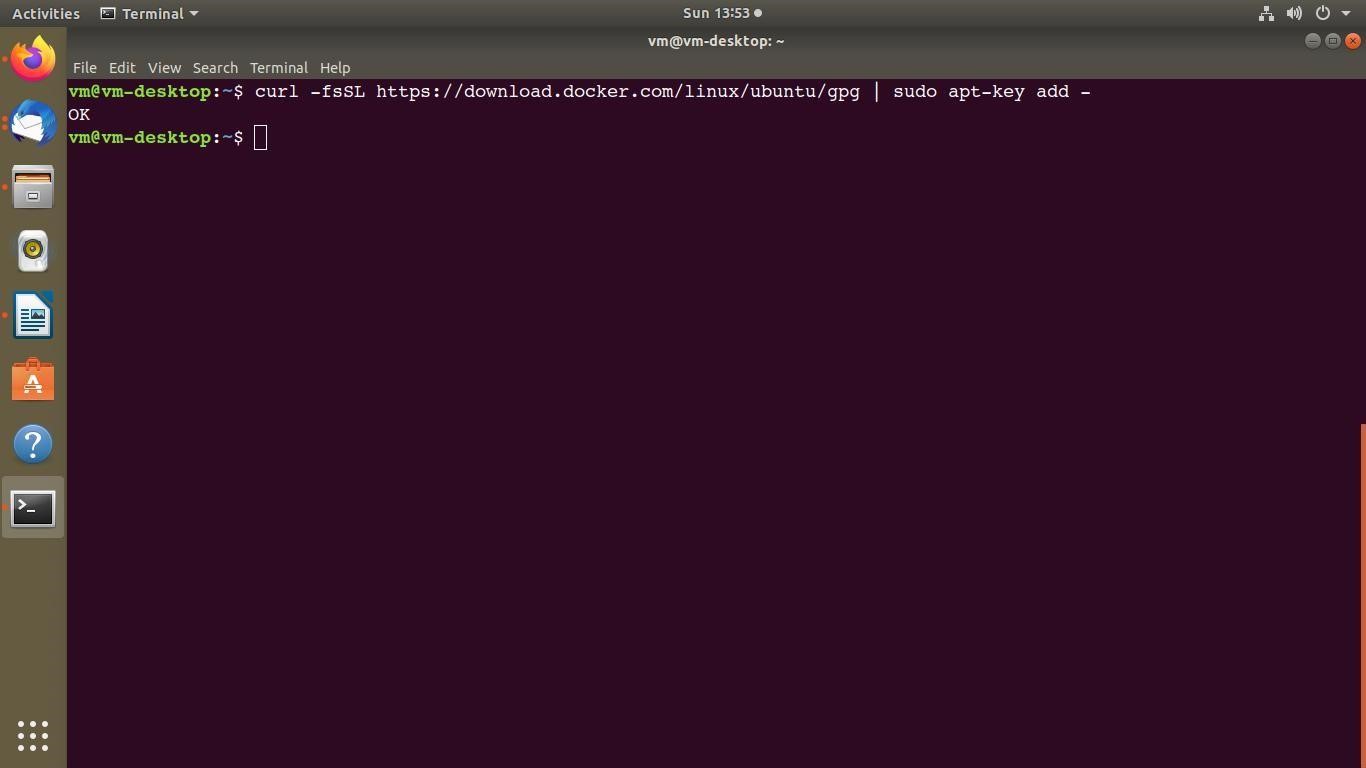
Step 3: install packages to allow apt to use a repository over HTTPS: sudo apt-get install \apt-transport-https \ca-certificates \curl \gnupg-agent \software- properties-

common

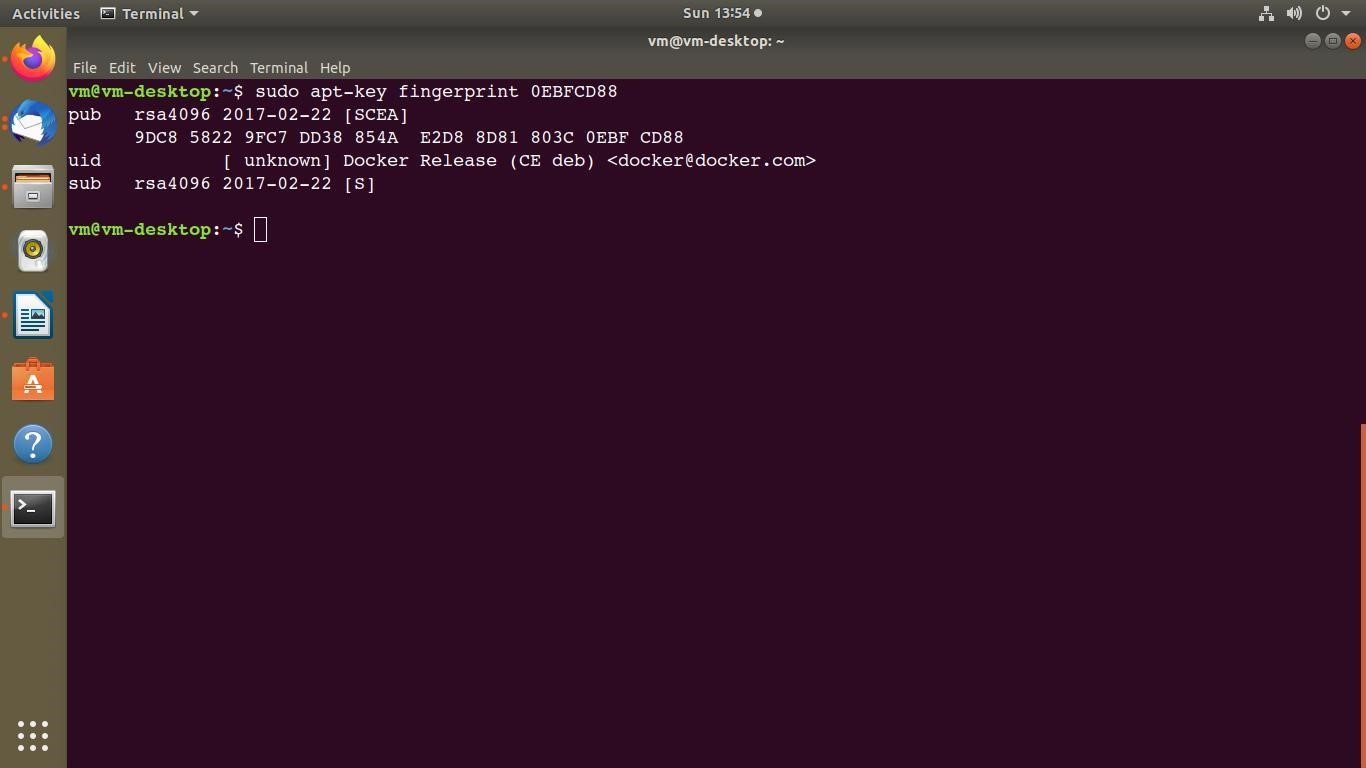


Step 4: Add Docker’s official GPG key:

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

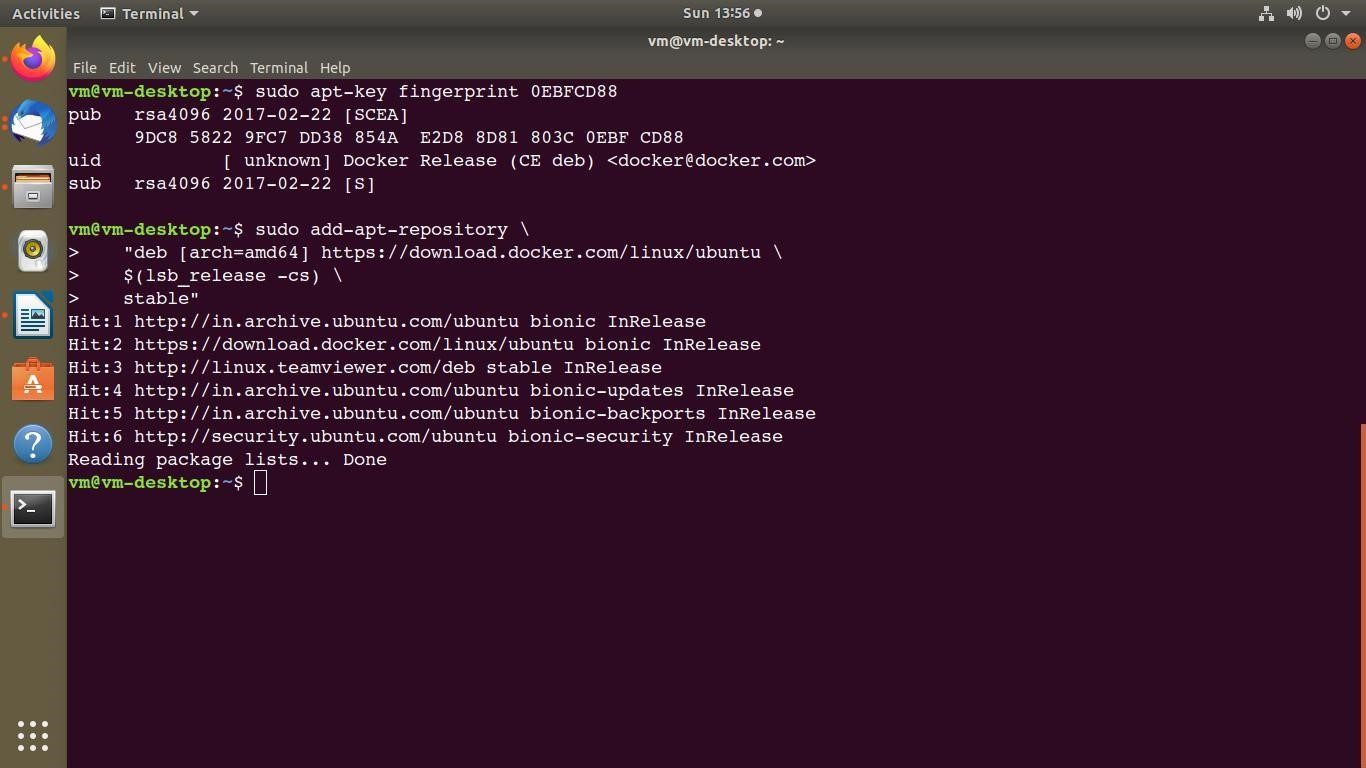


Step 5: Verify that you now have the key with the fingerprint 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88, by searching for the last 8 characters of the fingerprint.



$ sudo apt-key fingerprint 0EBFCD88

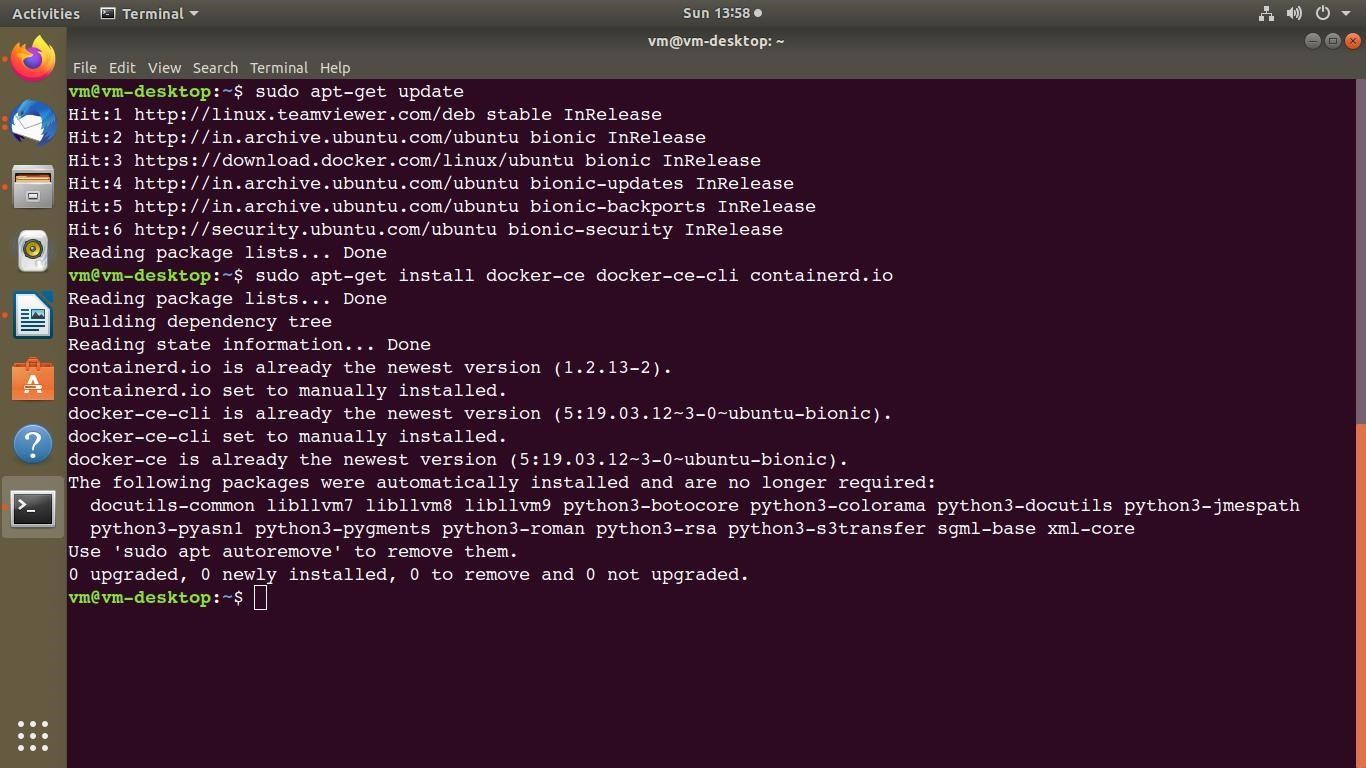
Step 6: Use the following command to set up the **stable** repository. To add the **nightly** or **test** repository, add the word nightly or test (or both) after the word stable in the commands below. $ sudo add-apt-repository \"deb [arch=amd64] https://download.docker.com/linux/ubuntu \ $(lsb\_release -cs) \stable"



Step 7: Update the apt package index, and install the latest version of Docker Engine and containerd, or go to the next step to install a specific version:+

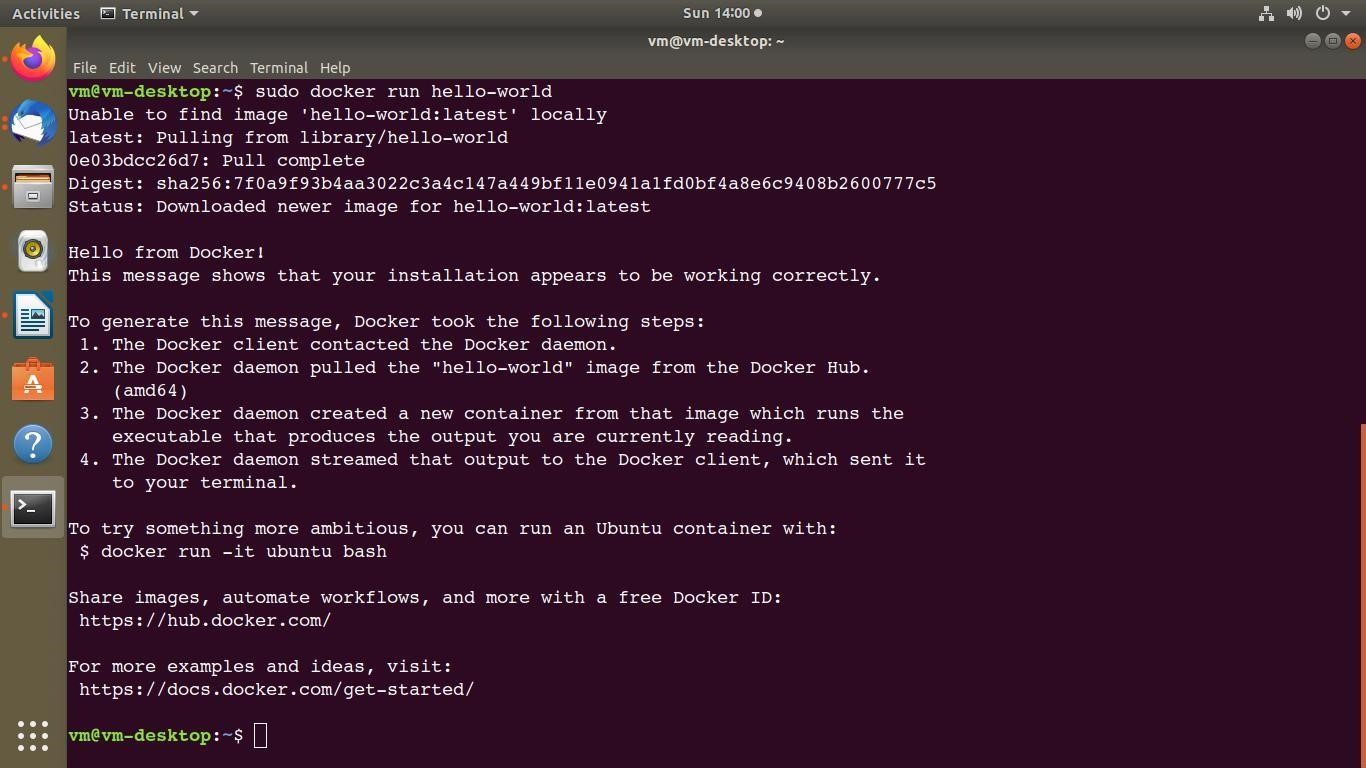
$ sudo apt-get update

$ sudo apt-get install docker-ce docker-ce-cli containerd.io



Step 8: Verify that Docker Engine is installed correctly by running the hello-world image.

$ sudo docker run hello-world



# Uninstall Docker Engine

Step 1: Uninstall the Docker Engine, CLI, and Containerd packages:

$ sudo apt-get purge docker-ce docker-ce-cli containerd.io

Step 2: Images, containers, volumes, or customized configuration files on your host are not automatically removed. To delete all images, containers, and volumes: $ sudo rm -rf /var/lib/docker

# Docker Compose

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application’s services. Then, with a single command, you create and start all the services from your configuration. To learn more about all the features of Compose, see [the list of features.](https://docs.docker.com/compose/)

Compose works in all environments: production, staging, development, testing, as well as CI workflows. You can learn more about each case in [Common Use Cases.](https://docs.docker.com/compose/)

Using Compose is basically a three-step process:

Define your app’s environment with a Dockerfile so it can be reproduced anywhere.

Define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment.

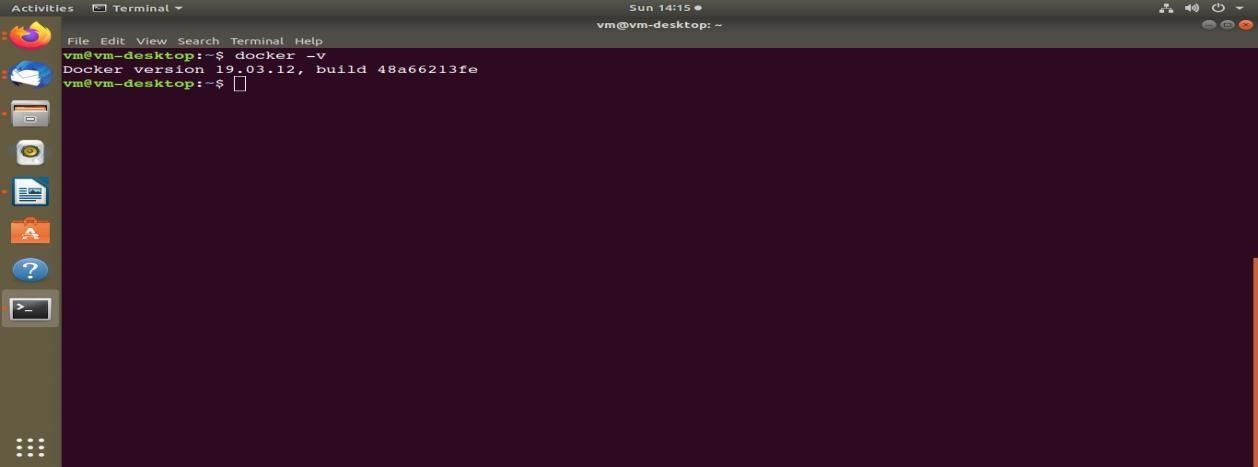
Run docker-compose up and Compose starts and runs your entire app

# Docker Desktop

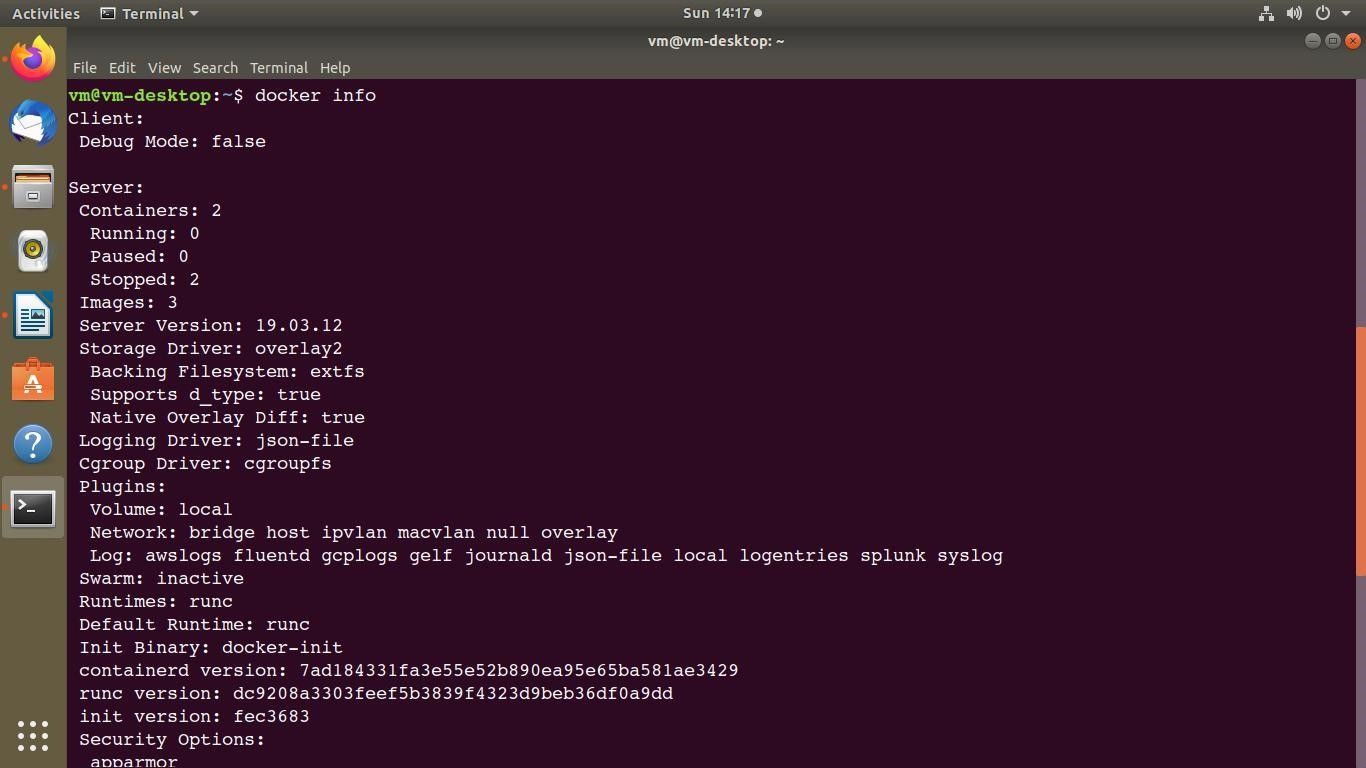
Docker Desktop is an easy-to-install application for your Mac or Windows environment that enables you to build and share containerized applications and microservices. Docker Desktop includ[es Docker Engine,](https://docs.docker.com/engine/) Docker CLI client, [Docker Compose,](https://docs.docker.com/compose/) [Notary,](https://docs.docker.com/notary/getting_started/) [Kubernetes,](https://github.com/kubernetes/kubernetes/) and [Credential](https://github.com/docker/docker-credential-helpers/) Helper.

# Docker Command Line (commands)

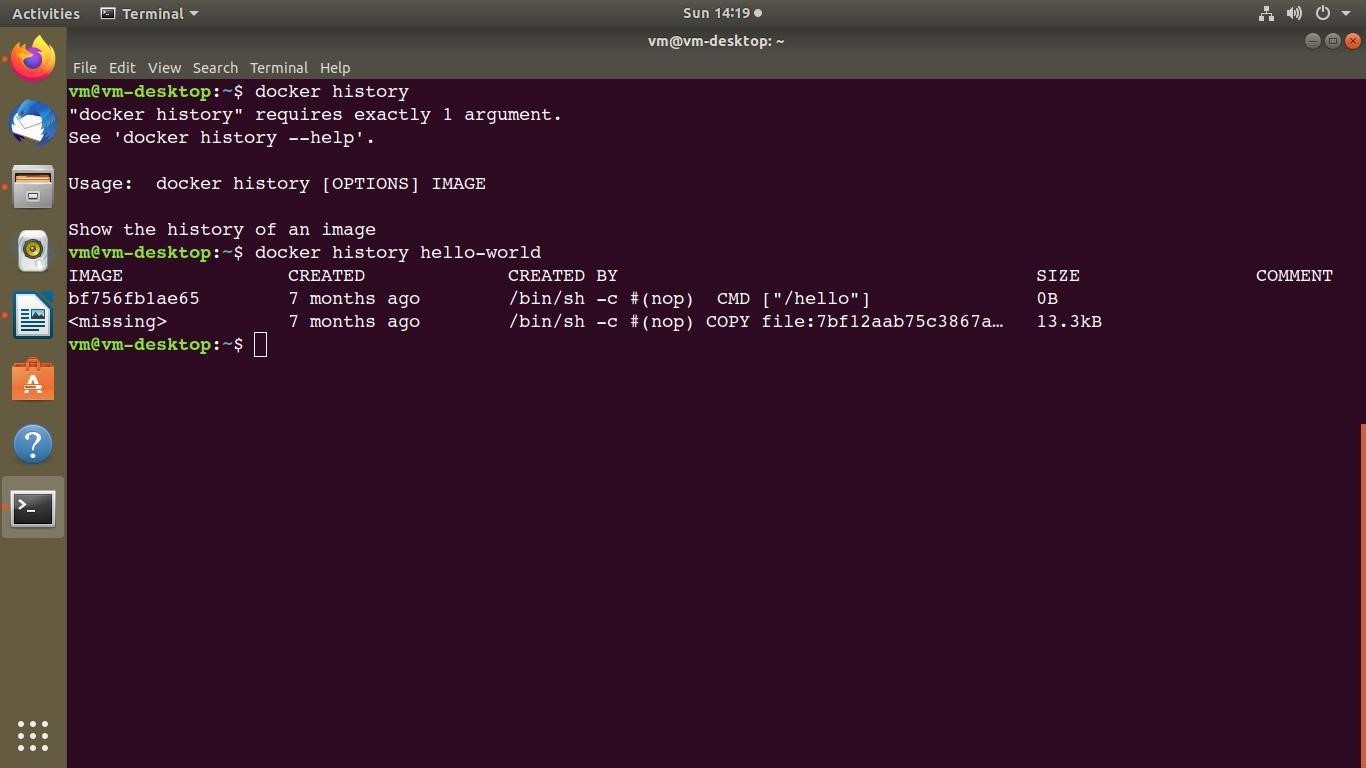
Commands1) docker -v



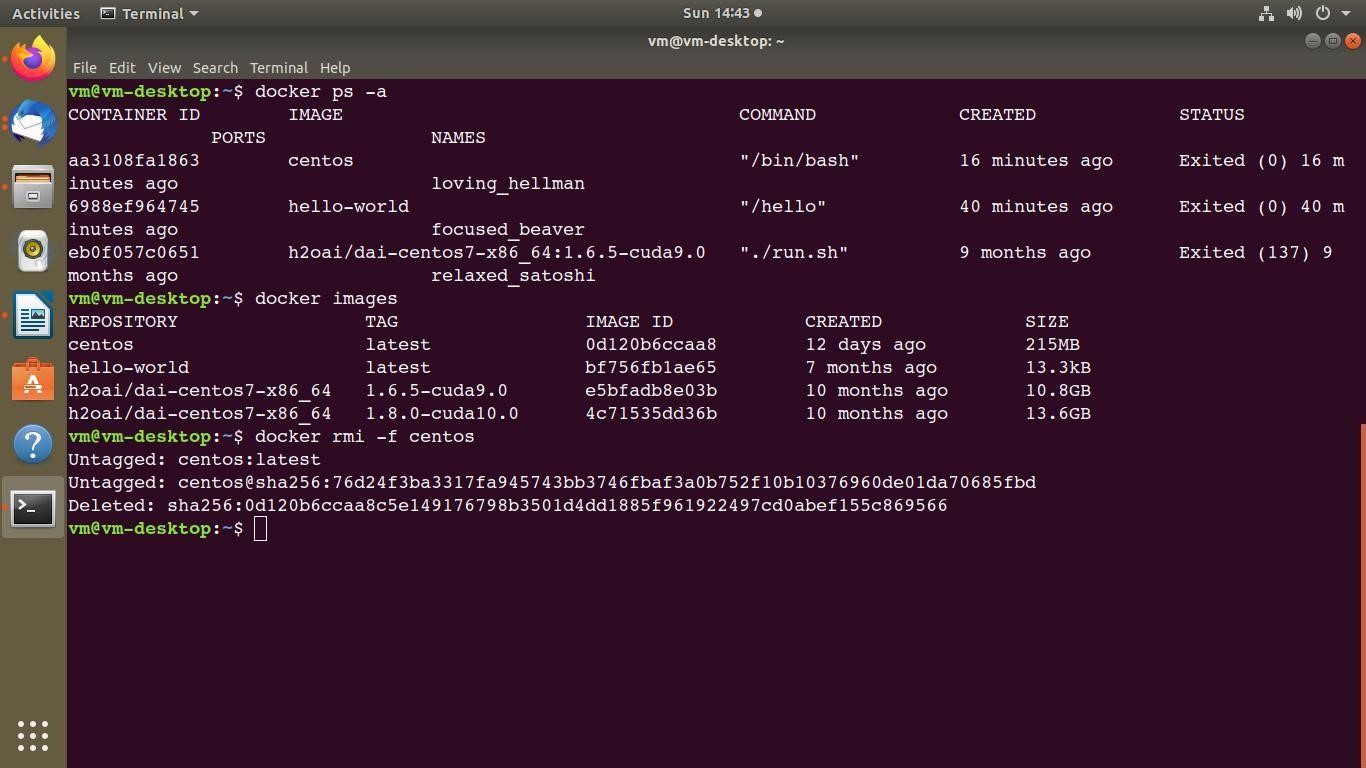
1. docker info - Display system-wide information docker info [OPTIONS]



1. docker history - Show the history of an image docker history [OPTIONS] IMAGE



1. docker image -Manage images
2. docker ps - List containers docker ps [OPTIONS]
3. docker images - List images docker images [OPTIONS] [REPOSITORY[:TAG]]
4. docker rmi -Remove one or more imagesdocker rmi [OPTIONS] IMAGE [IMAGE...]

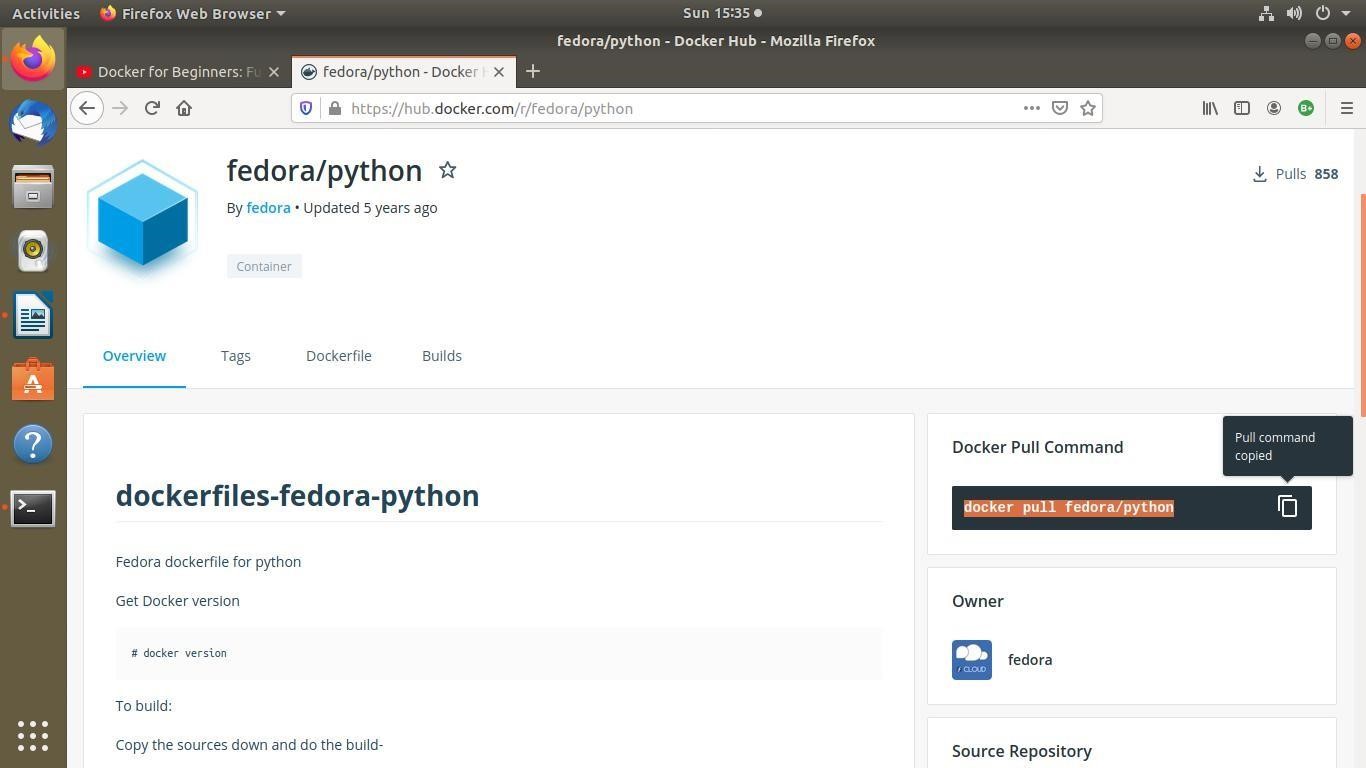


Install Tensorflow image or copy image or any other image you want form the Docker hub.

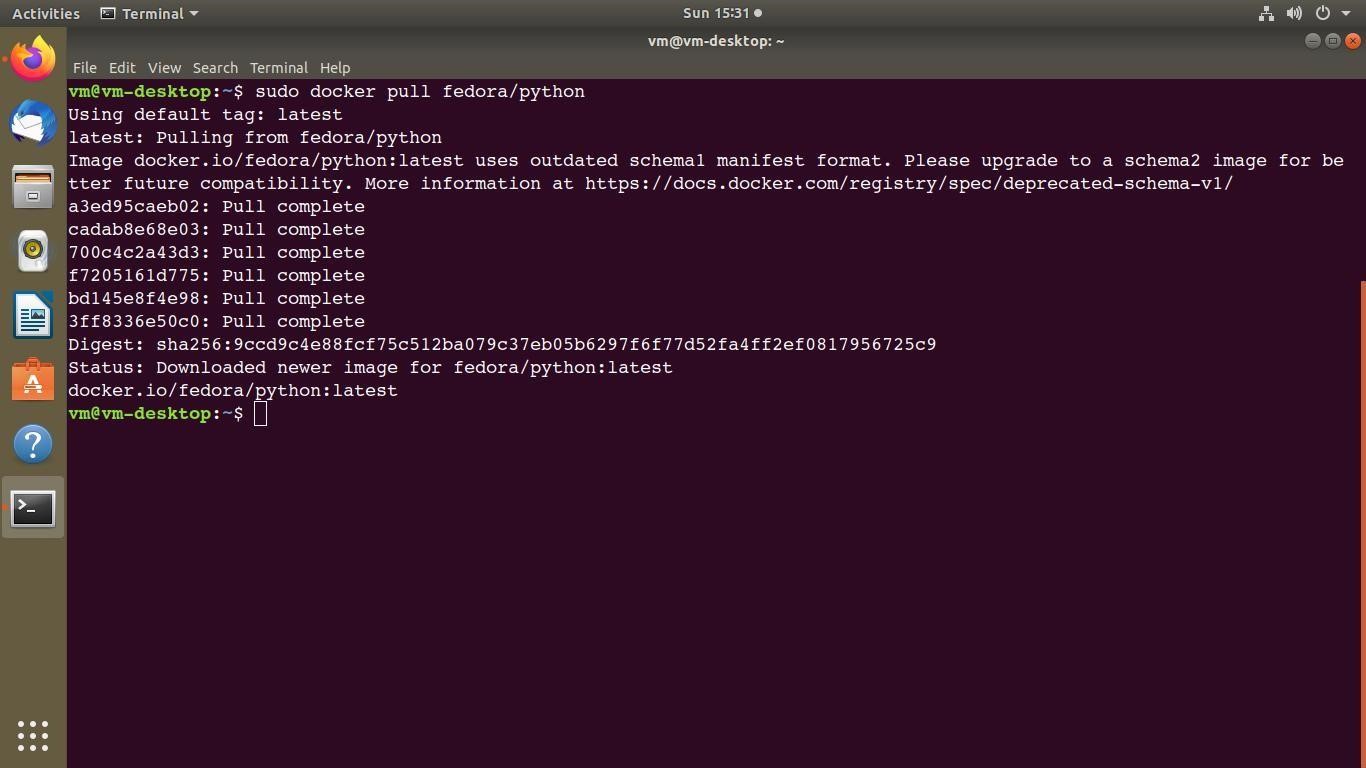
Here, I am pulling plain fedora image from docker.

Steps

1. login to docker hub account.
2. search “image name”in search box (I have searched for fedora/python image)
3. open the image link
4. copy the commnad



1. run the copied command in terminal docker pull fedora/python

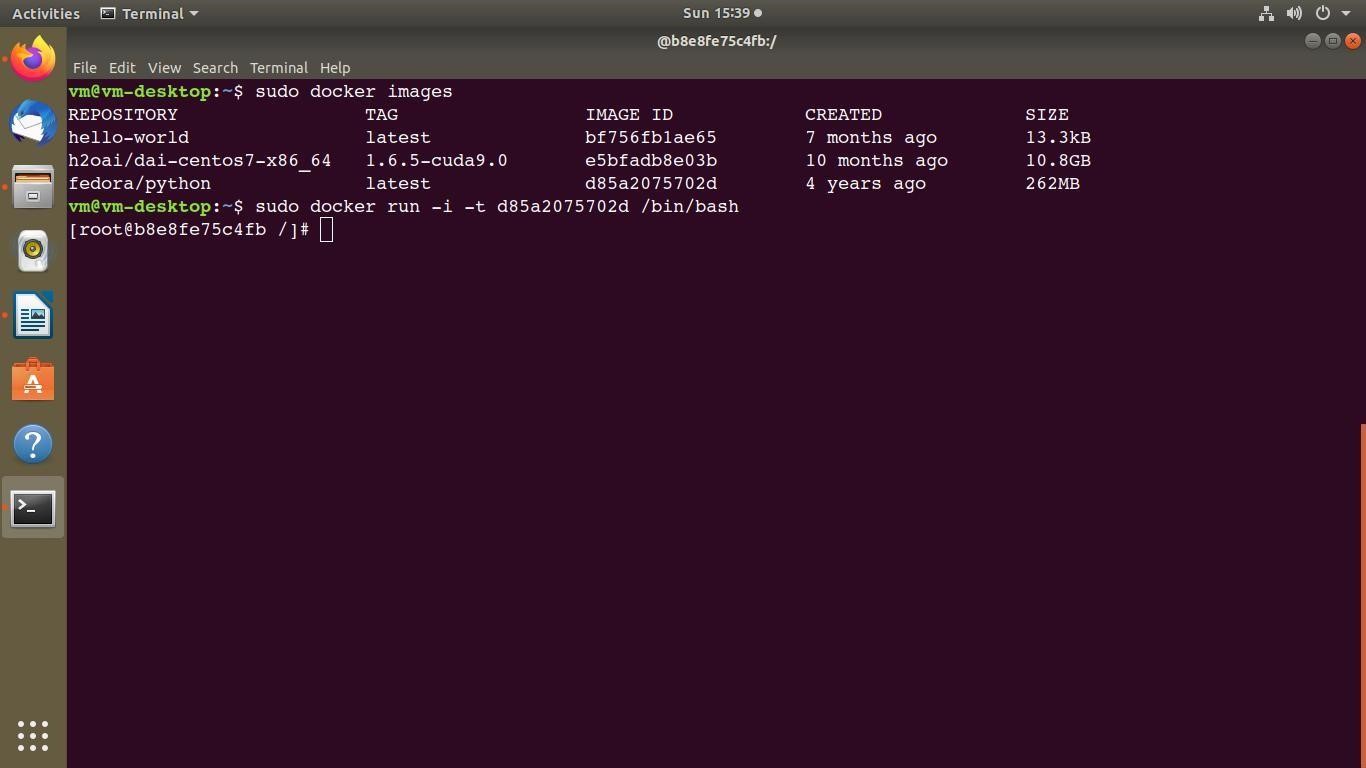


1. list the images by using command: sudo docker images
2. run the currently downloaded image using command:

sudo docker run -i -t d85a2075702d /bin/bash

-i => for input or interactive

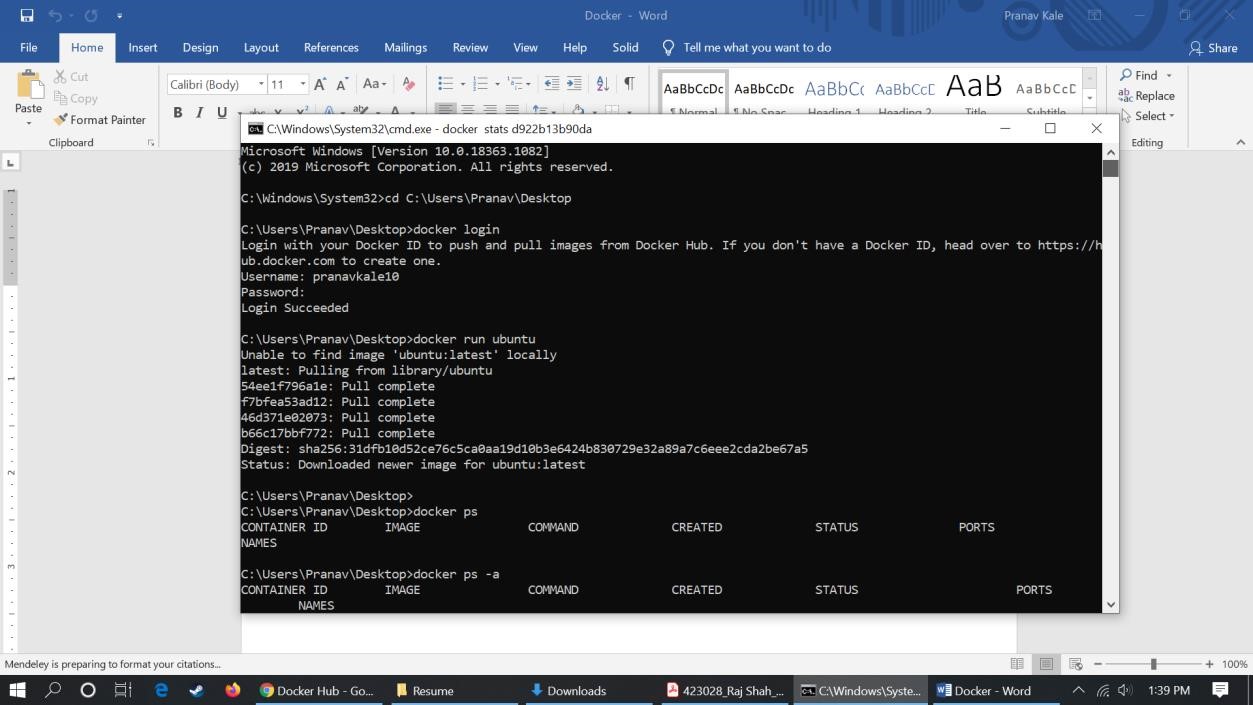
-t => terminal



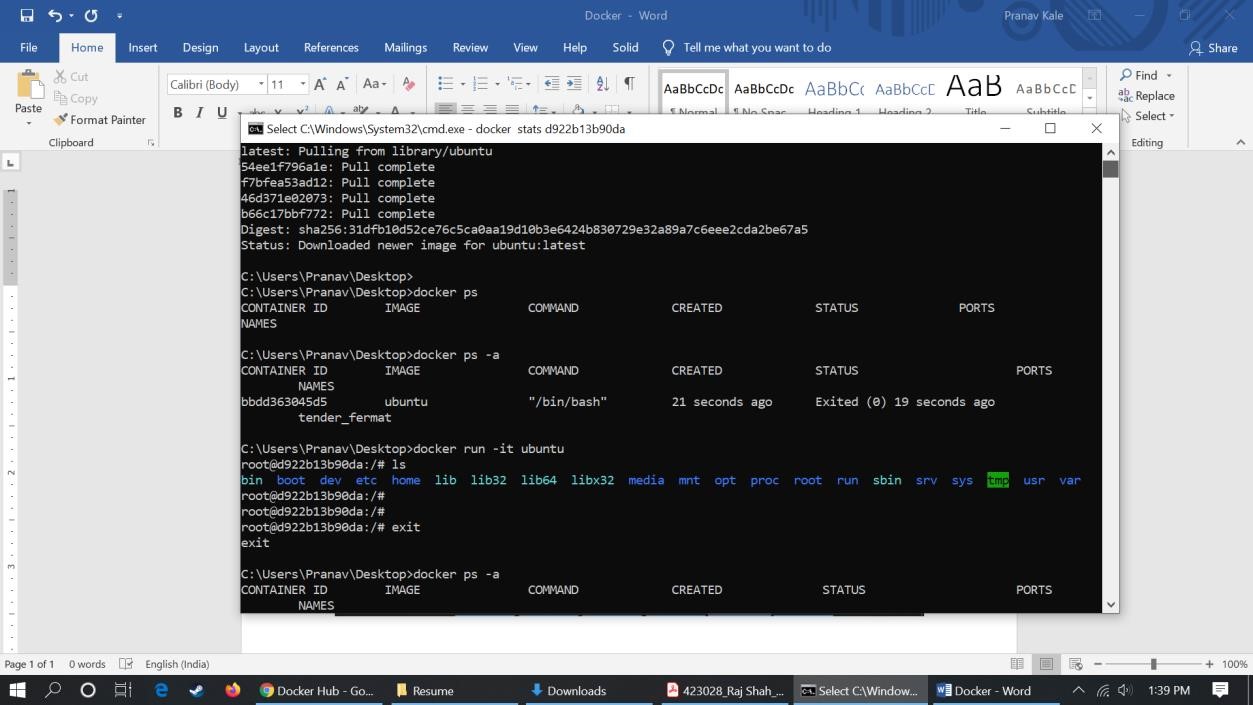
**Docker Container Commands**

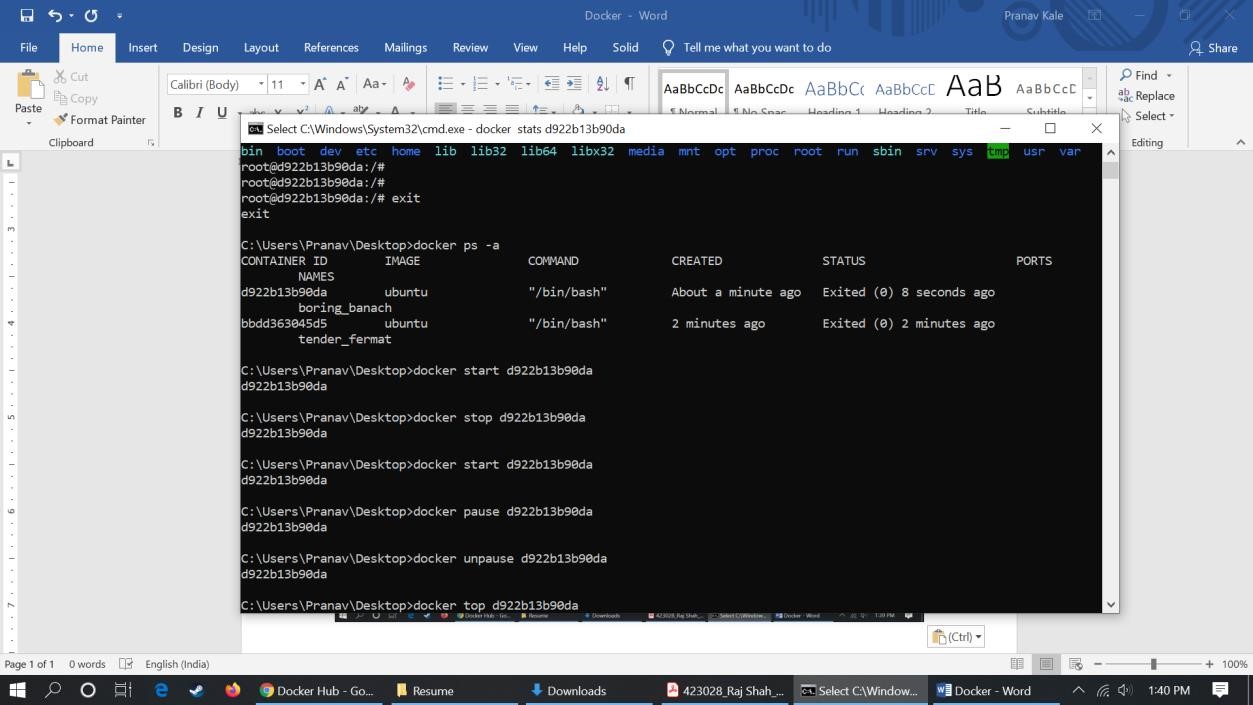
|  |  |
| --- | --- |
| **Command** | **Description** |
| [docker container attach](https://docs.docker.com/engine/reference/commandline/container_attach/) | Attach local standard input, output, and error streams to a running container |
| [docker container commit](https://docs.docker.com/engine/reference/commandline/container_commit/) | Create a new image from a container’s changes |
| [docker container cp](https://docs.docker.com/engine/reference/commandline/container_cp/) | Copy files/folders between a container and the local filesystem |
| [docker container create](https://docs.docker.com/engine/reference/commandline/container_create/) | Create a new container |
| [docker container diff](https://docs.docker.com/engine/reference/commandline/container_diff/) | Inspect changes to files or directories on a container’s filesystem |
| [docker container exec](https://docs.docker.com/engine/reference/commandline/container_exec/) | Run a command in a running container |
| [docker container export](https://docs.docker.com/engine/reference/commandline/container_export/) | Export a container’s filesystem as a tar archive |
| [docker container inspect](https://docs.docker.com/engine/reference/commandline/container_inspect/) | Display detailed information on one or more containers |
| [docker container kill](https://docs.docker.com/engine/reference/commandline/container_kill/) | Kill one or more running containers |
| [docker container logs](https://docs.docker.com/engine/reference/commandline/container_logs/) | Fetch the logs of a container |
| [docker container ls](https://docs.docker.com/engine/reference/commandline/container_ls/) | List containers |
| **Command** | **Description** |
| [docker container pause](https://docs.docker.com/engine/reference/commandline/container_pause/) | Pause all processes within one or more containers |
| [docker container port](https://docs.docker.com/engine/reference/commandline/container_port/) | List port mappings or a specific mapping for the container |
| [docker container prune](https://docs.docker.com/engine/reference/commandline/container_prune/) | Remove all stopped containers |
| [docker container rename](https://docs.docker.com/engine/reference/commandline/container_rename/) | Rename a container |
| [docker container restart](https://docs.docker.com/engine/reference/commandline/container_restart/) | Restart one or more containers |
| [docker container rm](https://docs.docker.com/engine/reference/commandline/container_rm/) | Remove one or more containers |
| [docker container run](https://docs.docker.com/engine/reference/commandline/container_run/) | Run a command in a new container |
| [docker container start](https://docs.docker.com/engine/reference/commandline/container_start/) | Start one or more stopped containers |
| [docker container stats](https://docs.docker.com/engine/reference/commandline/container_stats/) | Display a live stream of container(s) resource usage statistics |
| [docker container stop](https://docs.docker.com/engine/reference/commandline/container_stop/) | Stop one or more running containers |
| [docker container top](https://docs.docker.com/engine/reference/commandline/container_top/) | Display the running processes of a container |
| [docker container unpause](https://docs.docker.com/engine/reference/commandline/container_unpause/) | Unpause all processes within one or more containers |
| [docker container update](https://docs.docker.com/engine/reference/commandline/container_update/) | Update configuration of one or more containers |
| [docker container wait](https://docs.docker.com/engine/reference/commandline/container_wait/) | Block until one or more containers stop, then print their exit codes |

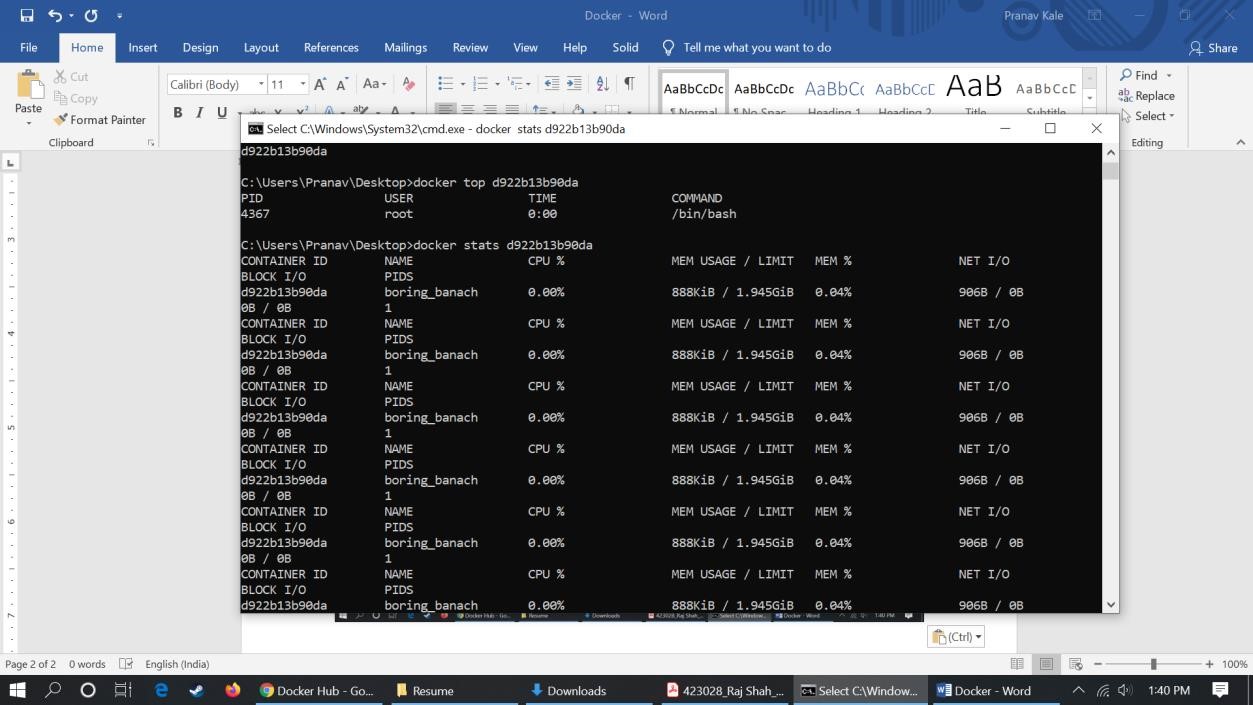
**Screenshots:**



\







# Docker Hub

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

* [Repositories:](https://docs.docker.com/docker-hub/repos/) Push and pull container images.
* [Teams & Organizations:](https://docs.docker.com/docker-hub/orgs/) Manage access to private repositories of container images.
* [Official Images:](https://docs.docker.com/docker-hub/official_images/) Pull and use high-quality container images provided by Docker.
* [Publisher Images:](https://docs.docker.com/docker-hub/publish/customer_faq/) Pull and use high- quality container images provided by external vendors.
* [Builds:](https://docs.docker.com/docker-hub/builds/) Automatically build container images from GitHub and Bitbucket and push them to Docker Hub.
* [Webhooks:](https://docs.docker.com/docker-hub/webhooks/) Trigger actions after a successful push to a repository to integrate Docker Hub with other services.

**Step 1:**

Sign up for Docker Hub

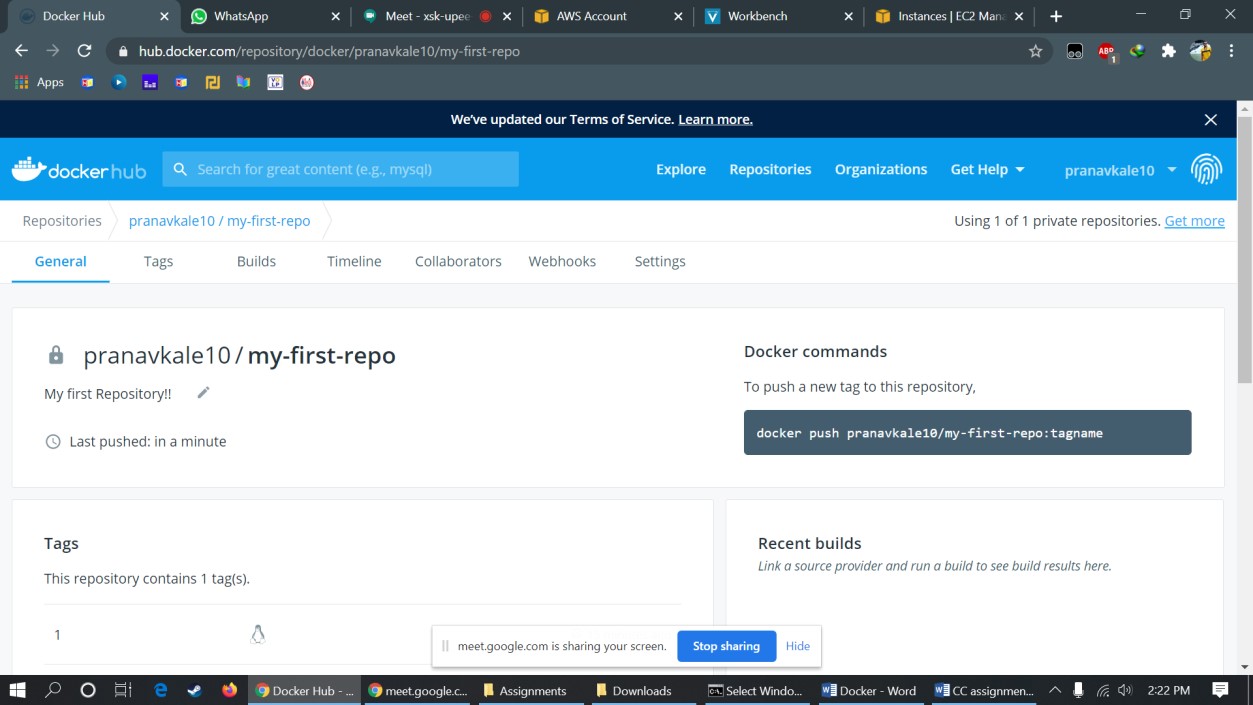
Start by [creating an account.](https://hub.docker.com/signup)On Sign-in Verify with your mail Id.

# Step 2)

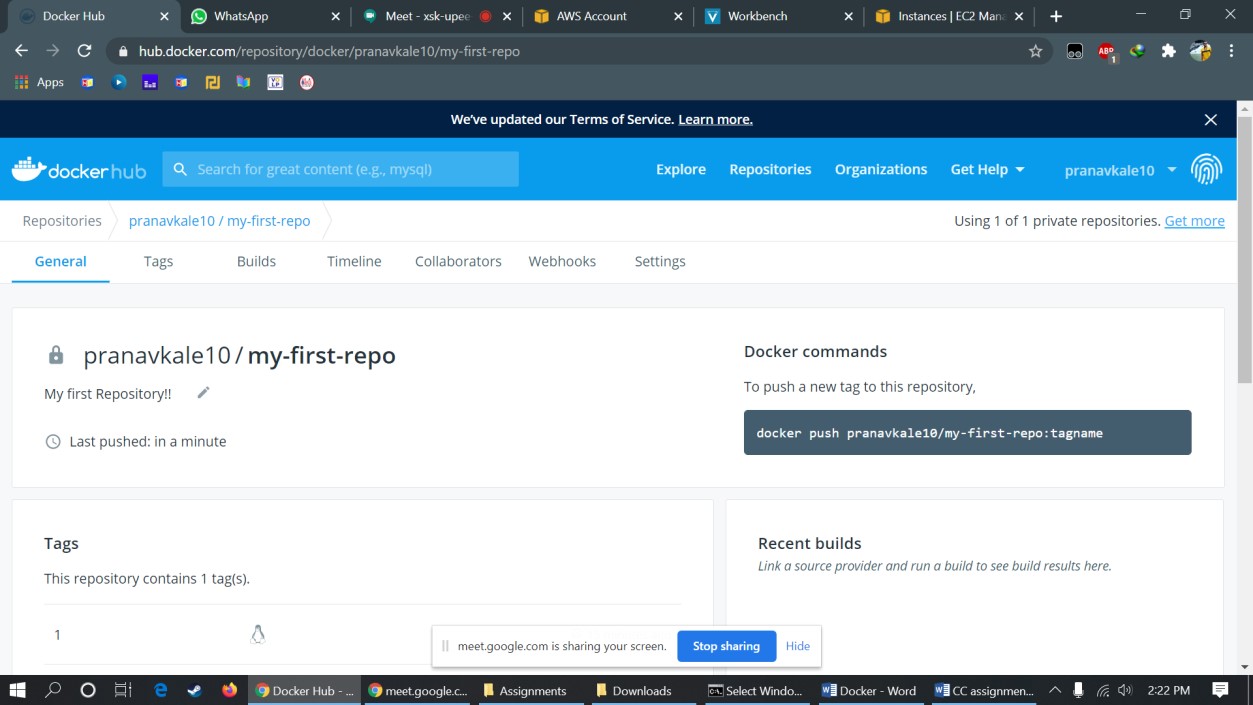
To create a repo:

1. Sign in to [Docker Hub.](https://hub.docker.com/)
2. Click on Create a Repository on the Docker Hub welcome page

3.Name it <your-username>/my-first-repo as shown below.



4.You’ve created your first repo. You should see:



# Step 3: Download and install Docker Deskt[op](https://docs.docker.com/docker-hub/)

We’ll need to download Docker Desktop to build and push a container image to Docker Hub.

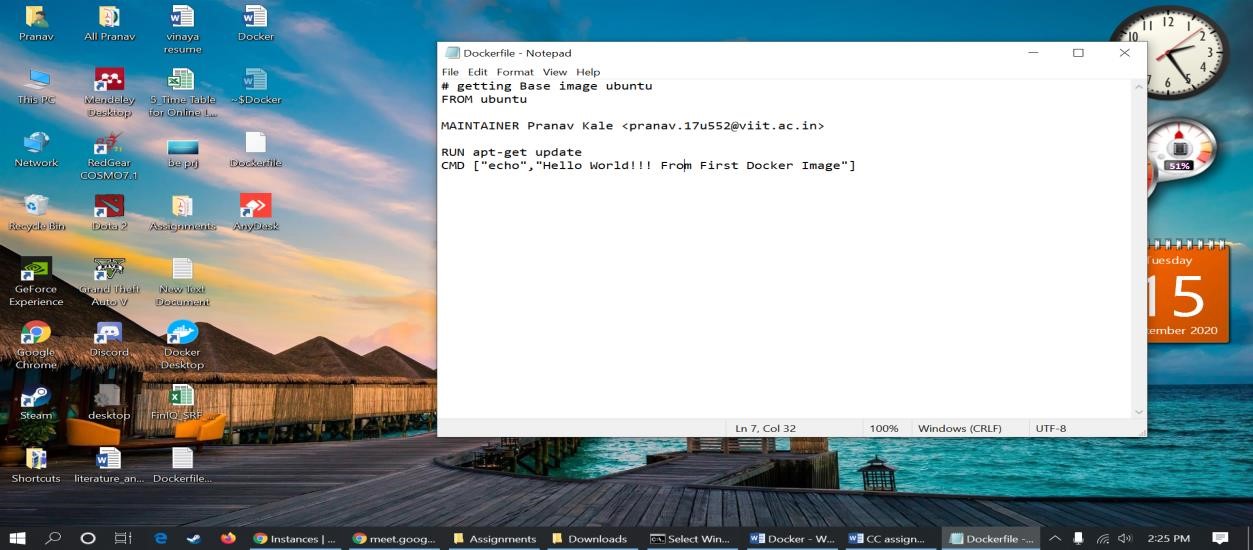
Download and insta[ll Docker Desktop.](https://docker.com/get-started) If on Linux, download [Docker Engine - Community.](https://hub.docker.com/search?type=edition&offering=community) Open the terminal and sign in to Docker Hub on your computer by running docker login. Write own docker file and push it.

**Step 4: Build and push a container image to Docker Hub from your computer**

**1 : Create a file named Dockerfile without any extension**

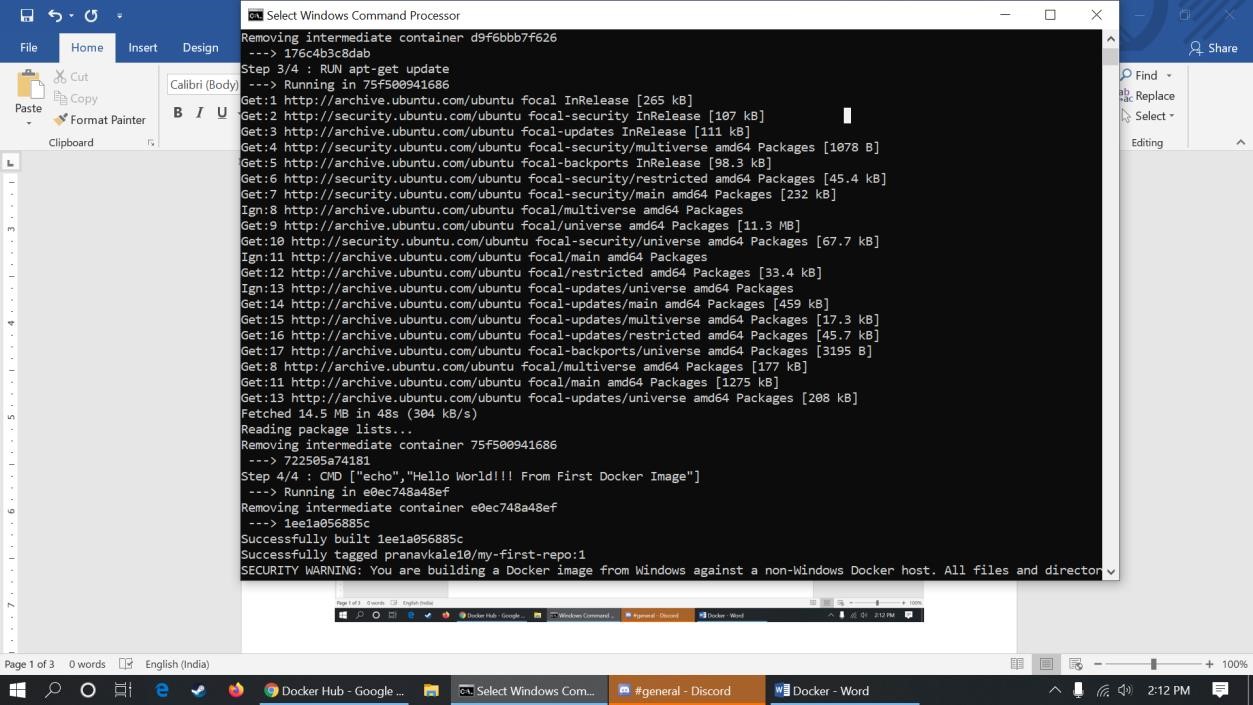
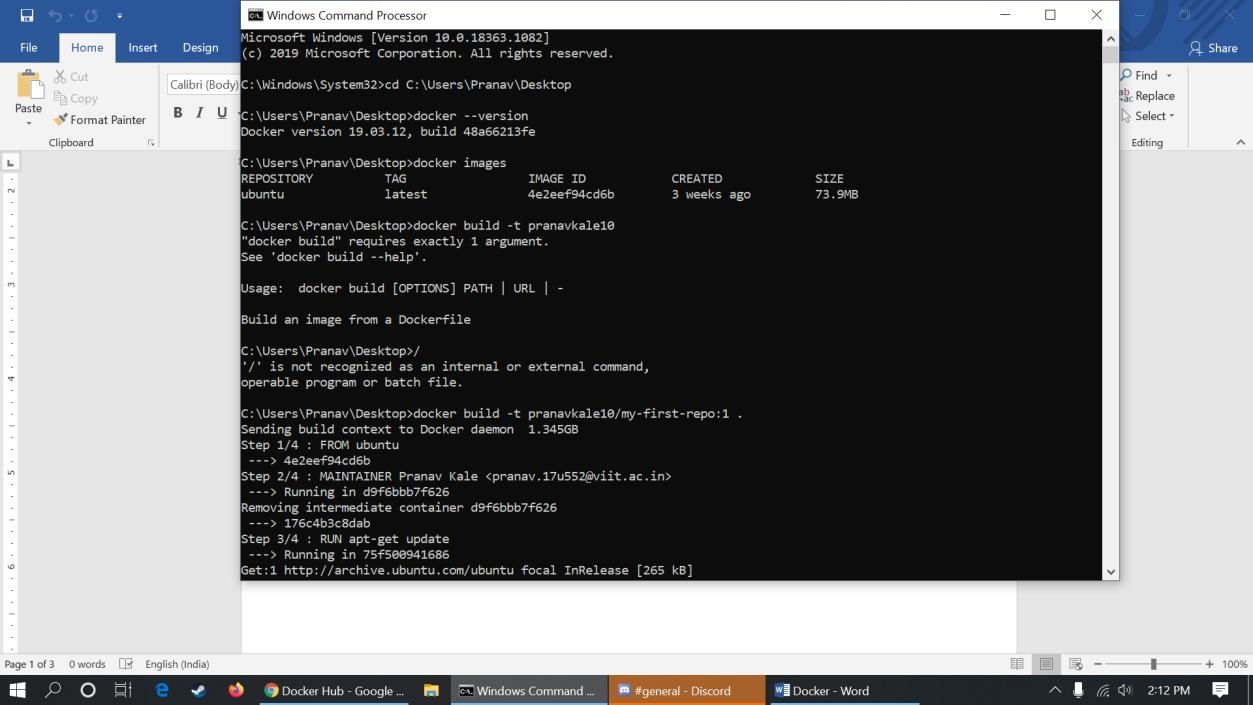
Start by creating a [Dockerfile t](https://docs.docker.com/engine/reference/builder/)o specify your application as shown below

# : Add instructions in Dockerfile



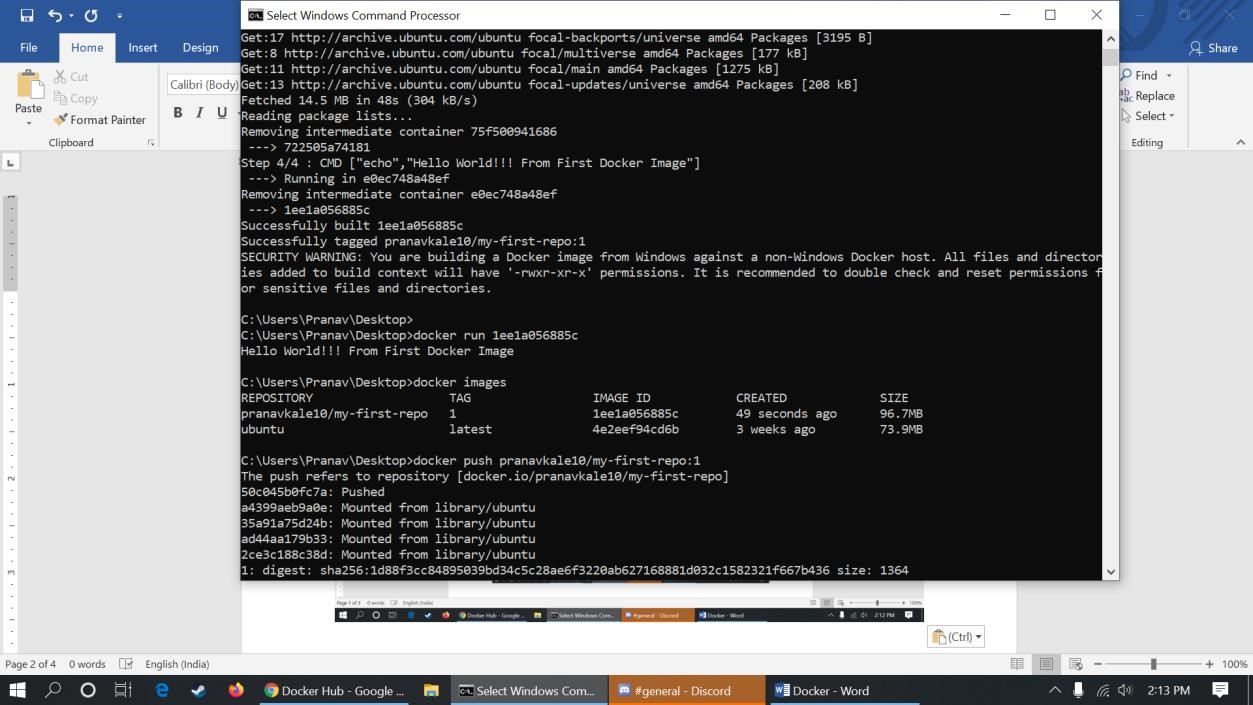
# 3: Build dockerfile to create image

Run docker build -t <your\_username>/image:1 . to build your Docker image.



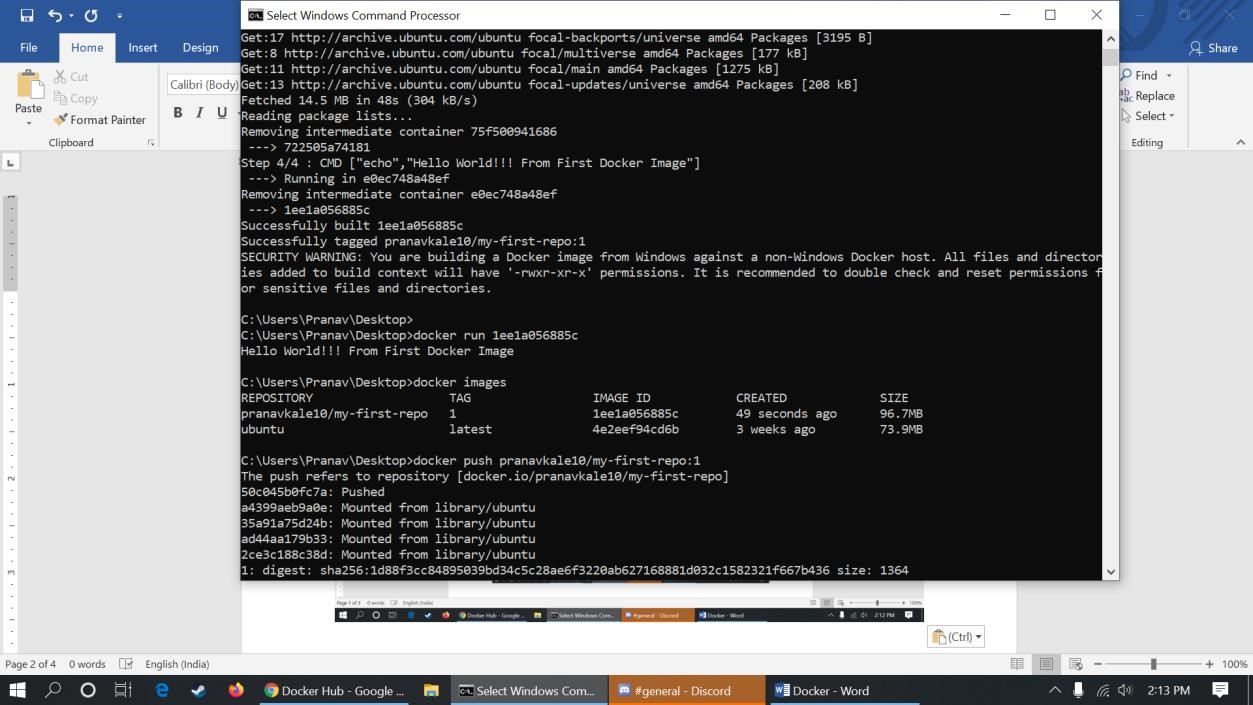
# : Run image to create container

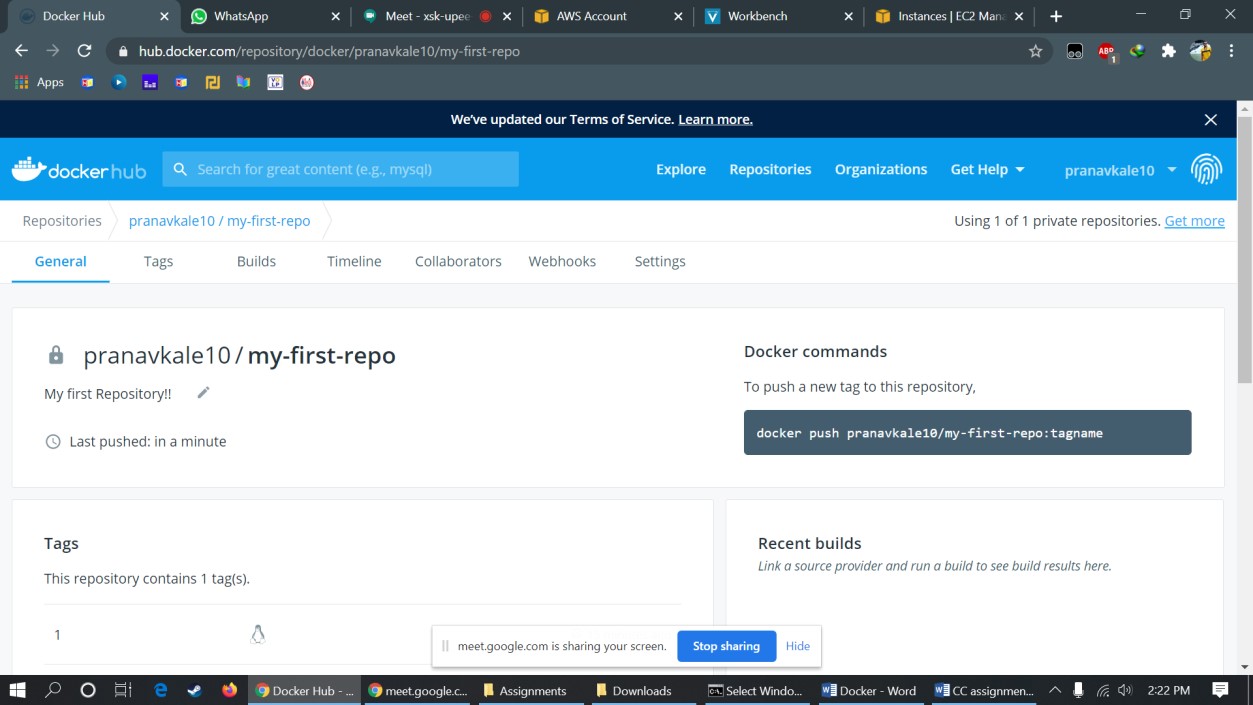
Run docker run <your\_username>/image:1 to test your Docker image locally.



# 5: Push to Docker Hub

Run docker push <your\_username>/image:1 to push your Docker image to Docker Hub.





**Conclusion:**

We have successfully implemented Docker Assignment.