DOCKER

By Chaitanya Raj Nandan

Contents

[1 Before Containers 3](#_Toc172585420)

[1.1 What is A Container? 3](#_Toc172585421)

[1.2 Containers Vs Virtual Machines 3](#_Toc172585422)

[1.3 Advantages of Containerization 4](#_Toc172585423)

[2 Introduction 5](#_Toc172585424)

[2.1 Why Use Docker 5](#_Toc172585425)

[2.2 Docker Terminologies / Docker Architecture 5](#_Toc172585426)

[3 Dockerfiles 7](#_Toc172585427)

[4 Install Docker in Linux (Ubuntu) 9](#_Toc172585428)

[5 Basic Docker Commands 12](#_Toc172585429)

[5.1 Docker PS 12](#_Toc172585430)

[5.2 Docker Pull 12](#_Toc172585431)

[5.3 Docker Images 14](#_Toc172585432)

[5.4 Docker Run 14](#_Toc172585433)

[5.5 Docker PS 16](#_Toc172585434)

[5.6 Docker Stop 17](#_Toc172585435)

[5.7 Docker RM (Remove Container) 17](#_Toc172585436)

[5.8 Docker RMI (Remove Image) 17](#_Toc172585437)

[5.9 Docker Build 18](#_Toc172585438)

[5.10 Docker Push 18](#_Toc172585439)

[5.11 Docker Exec 18](#_Toc172585440)

[5.12 Docker Port 18](#_Toc172585441)

[5.13 Docker stats 18](#_Toc172585442)

[5.14 Docker Inspect 18](#_Toc172585443)

[6 Three WAYS TO USE Docker 19](#_Toc172585444)

[6.1 Run Docker containers 19](#_Toc172585445)

[Create and Open NGINIX website by running NGINIX Container on docker with the Help of AWS EC2 (ubuntu AMI) virtual machine. 19](#_Toc172585446)

[6.1.1 Launch an EC2 Instance: 19](#_Toc172585447)

[6.1.2 Connect to the EC2 Instance 19](#_Toc172585448)

[6.1.3 Install Docker on EC2 Instance 19](#_Toc172585449)

[6.1.4 Verify Docker Installation 20](#_Toc172585450)

[6.1.5 Run a Simple Docker Container 20](#_Toc172585451)

[6.1.6 Access Nginx in the Browser 21](#_Toc172585452)

[6.1.7 Cleanup (Optional) 21](#_Toc172585453)

[6.2 Develop and build your application 22](#_Toc172585454)

[Building a Python Web App 22](#_Toc172585455)

[7 Uninstall & Remove DOCKER 23](#_Toc172585456)

[8 ERROR AND ITS SOLUTION 25](#_Toc172585457)

[8.1 Unauthorized: incorrect username or password 25](#_Toc172585458)

# Before Containers

Let us dial back the clock. Back when computers were a novelty, and companies wanted to run their applications, they would have had to buy a server for doing so. However, this came with a caveat: you could only run one application per server. If you wanted to run multiple applications, you would have to buy as many servers. As costs would skyrocket, this was a problem, for both the company and the environment on a larger scale

IBM fixed this issue by introducing the concept of *virtual machines* into the game. With them, we could run multiple applications on the same server. You may have heard of / done dual booting on your PC; like installing windows on your Mac device or Ubuntu on Windows. These are virtual machines. However, they had a problem as well. They needed their own operating system, which tended to take up a lot of memory and storage on your hard disk. This made them slow and not so efficient.

In came containers.

## What is A Container?

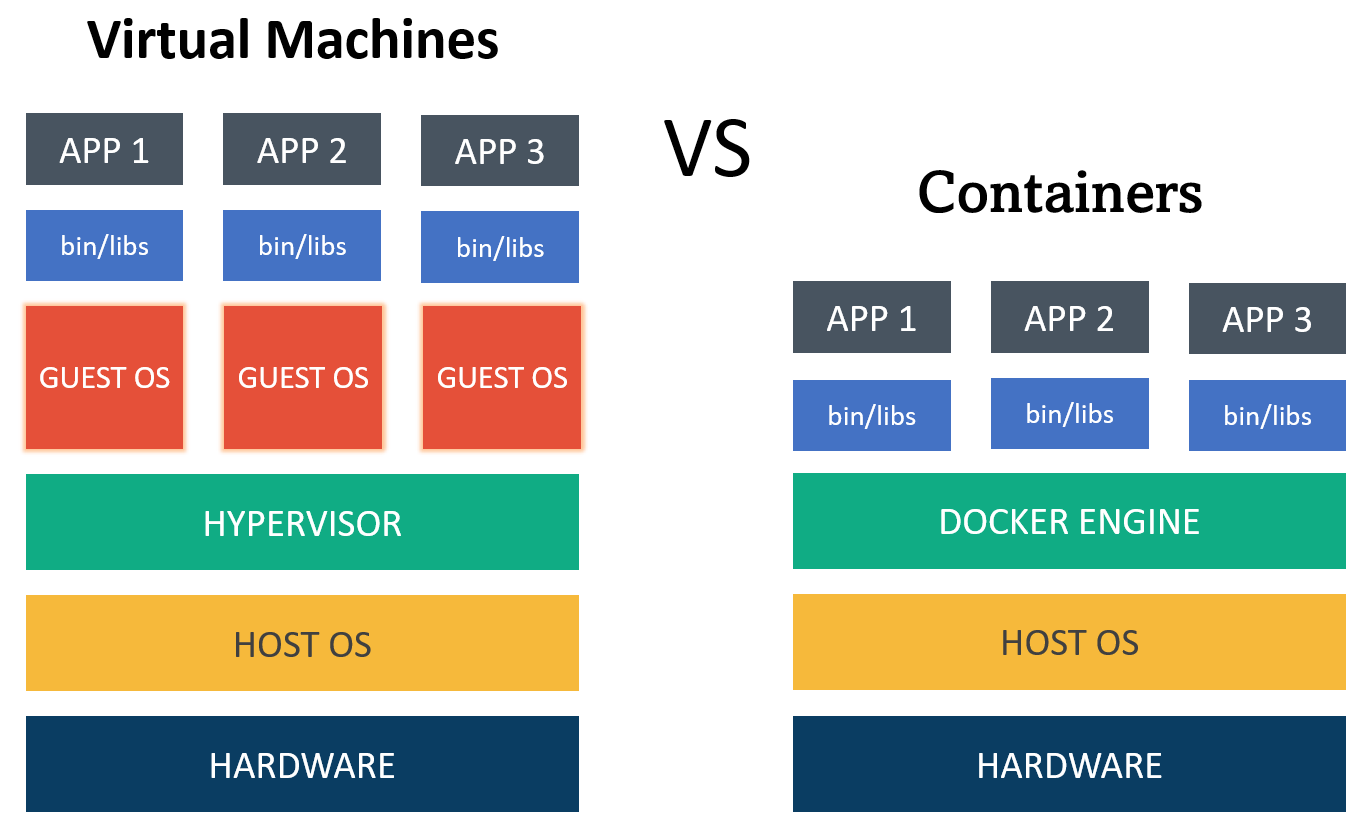
Imagine you are moving to US from India. Would it make sense to send your belongings one by one, or would you rather put them all in one giant box and ship the box directly? Obviously, the latter, right?

That big box is, in simple terms, a container.

Let’s say you built a website that works well on your system but your friend runs into some problems with it when they try using it on their computer. To avoid such hassles, you can use a container, which would ship the entire website along with its dependencies, such as the web database, front end, back end, source code, etc. This would ensure that the website runs smoothly.

In computing terms, containerization is an efficient method for running, deploying and scaling applications.

## Containers Vs Virtual Machines



Any application which runs on a virtual machine will require a guest OS to run, which would require a hypervisor. A hypervisor manages the virtual machines and is used to create multiple machines on the host operating system. As you can see from the picture, every OS would require a dedicated amount of space in the hardware, which is *virtually* divided.

In the case of containers, you need have only one operating system, and on top of that, you have a container engine, on which you run applications. They use the idea of isolating your application from the main operating system.

To sum it up succinctly, virtual machines use multiple operating systems to run multiple applications whereas containers use only the host operating system to run applications via the container engine.

In reality, however, containers run on top of virtual machines. This is because virtual machines allow faster access to hardware resources and hypervisors provide better security compared to containers.

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Containers** | **Virtual Machines** |
| OS | Share host’s Kernel | Has its own Kernel |
| Resource Usage | Lightweight, efficient | Heavier, more resource usage |
| Startup Time | Quick start | Slower start |
| Isolation | Process-level separation | Full OS isolation |
| Portability | Highly portable | Compatibility concerns |
| Resource Overhead | Minimal overhead | Higher overhead |
| Isolation Level | Lighter isolation | Stronger isolation |

## Advantages of Containerization

* Increased Portability
* Easier Scalability
* Easy and Fast Deployments
* Better Productivity
* Improved Security
* Consistent test environment for development and QA.
* Cross-platform packages called images.
* Isolation and encapsulation of application dependencies.
* Ability to scale efficiently, easily, and in real time.
* Enhances efficiency via easy reuse of images.

# Introduction

Docker is a platform for developing, shipping, and running applications in containers. Containers are lightweight, portable, and self-sufficient units that can run applications and their dependencies in isolated environments. Docker provides a consistent and reproducible environment across different machines, making it easier to manage and deploy applications.

## Why Use Docker

* Helps transport code faster.
* Makes scaling, deploying, and identifying issues in the application easier.
* Saves up space as you do not need to install the whole application locally.

## Docker Terminologies / Docker Architecture

**Docker Engine:** The core of Docker is the Docker Engine, which is responsible for building, running, and managing containers. It consists of a server (daemon) and a REST API that allows you to interact with the daemon. It is the heart of the Docker architecture, which does the crucial work of building, running, and distributing the containers. It also manages the Docker images and the containers.

* **Server** – which runs the daemon
* **Rest API** – deals with the interaction of applications with their server
* **Client** – which is nothing but the command line interface (CLI)

**Docker Images:** Images are lightweight, standalone, and executable packages that include everything needed to run a piece of software, including the code, runtime, libraries, and system tools. Images are used to create containers.

**Docker Containers:** Containers are instances of Docker images that run in isolated environments. They encapsulate an application and its dependencies, ensuring consistency across different environments. Containers are portable and can run on any system that supports Docker.

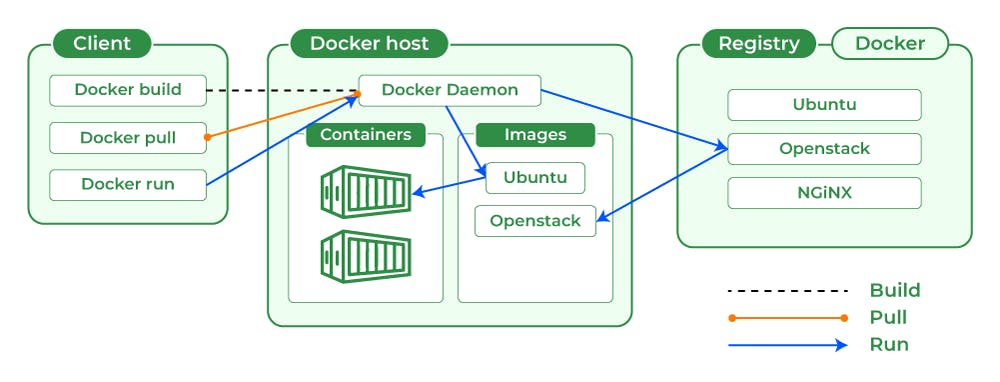
**Docker Runtime:** It allows us to start and stop containers. There are two types:

* Low-level runtime (runc)
* High-level runtime (containerd)

**Docker Client:** It is what we use to interact with Docker. You can think of it like the user interface for Docker. Whenever you type in a command, the client sends these to the daemon.

**Docker CLI:** It allows users to issue commands to the Docker Daemon. Docker uses a client – server architecture.

**Docker Registries**: Docker Registries store Docker images. The default public registry is Docker Hub, but you can also use private registries or other public registries. Docker Hub is a cloud-based repository where Docker users and partners create, test, store, and distribute container images. It is the default registry used by Docker, making it easy to find and share container images.



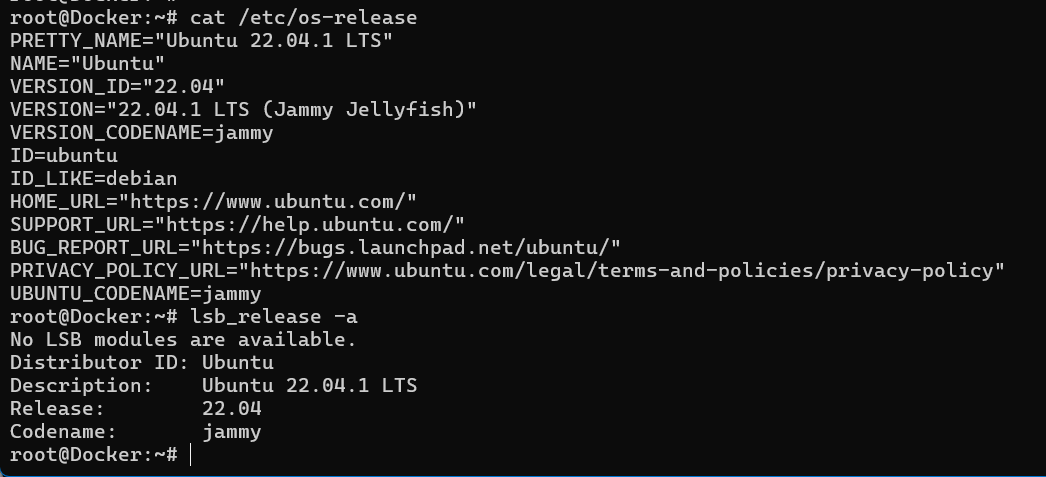
# Install Docker in Linux (Ubuntu)

To install Docker using package managers on Ubuntu, you can use the following steps:

1. Check ubuntu version.

**lsb\_release -a**

**cat /etc/os-release**

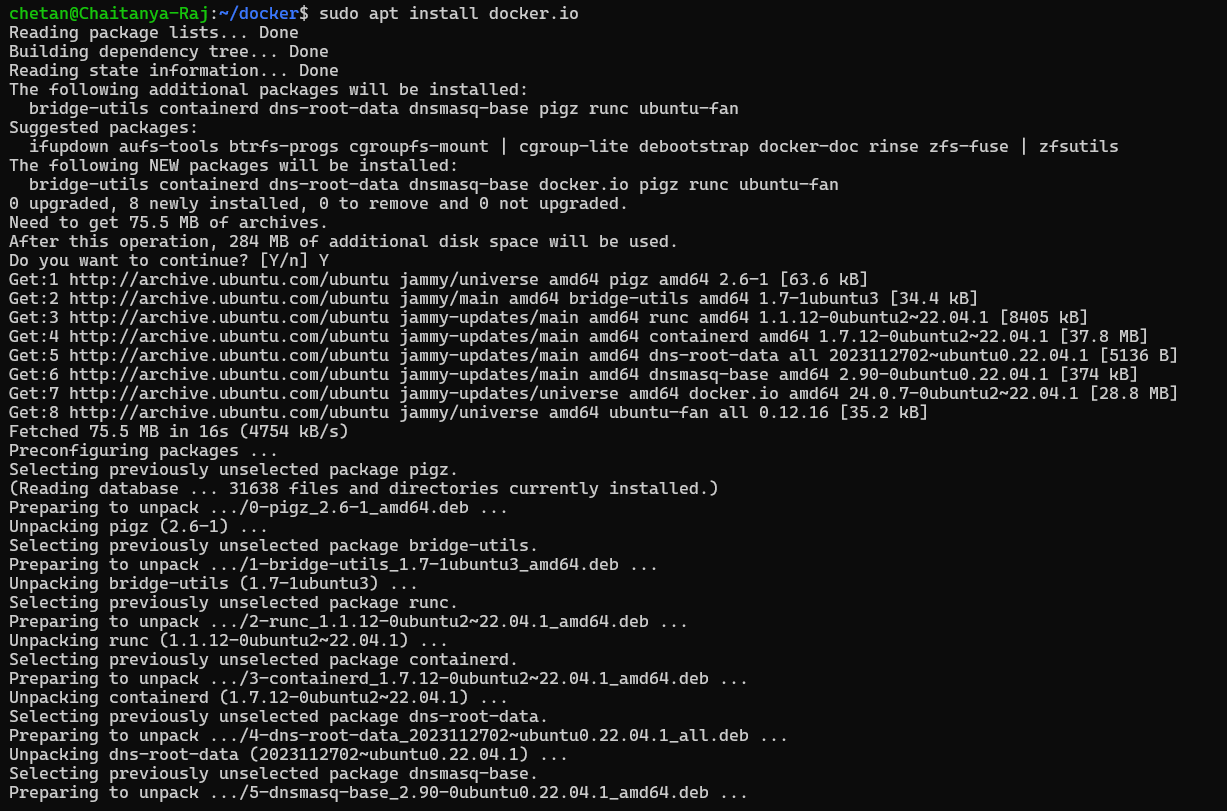


1. Update the package cache

**sudo apt update**

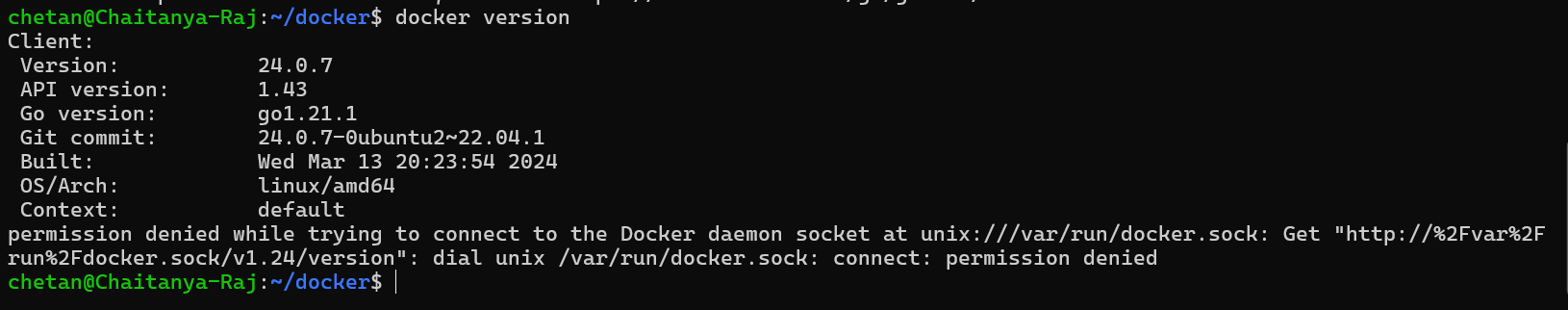
1. Install the Docker package

**sudo apt install docker.io**

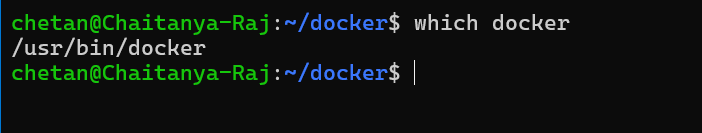


1. Verify the installation

**docker version**

****

**which docker**

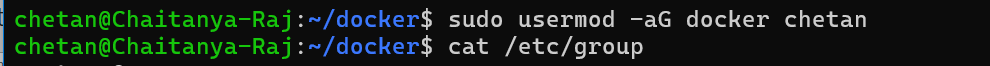


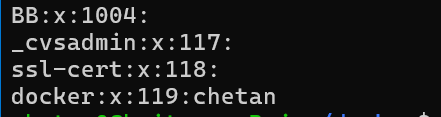
This command should output the version of Docker that is installed on your system.

1. Add the current user to the docker group\

**sudo usermod -aG docker** chetan

This will allow you to run Docker commands without having to use sudo.





1. Log out and log back in

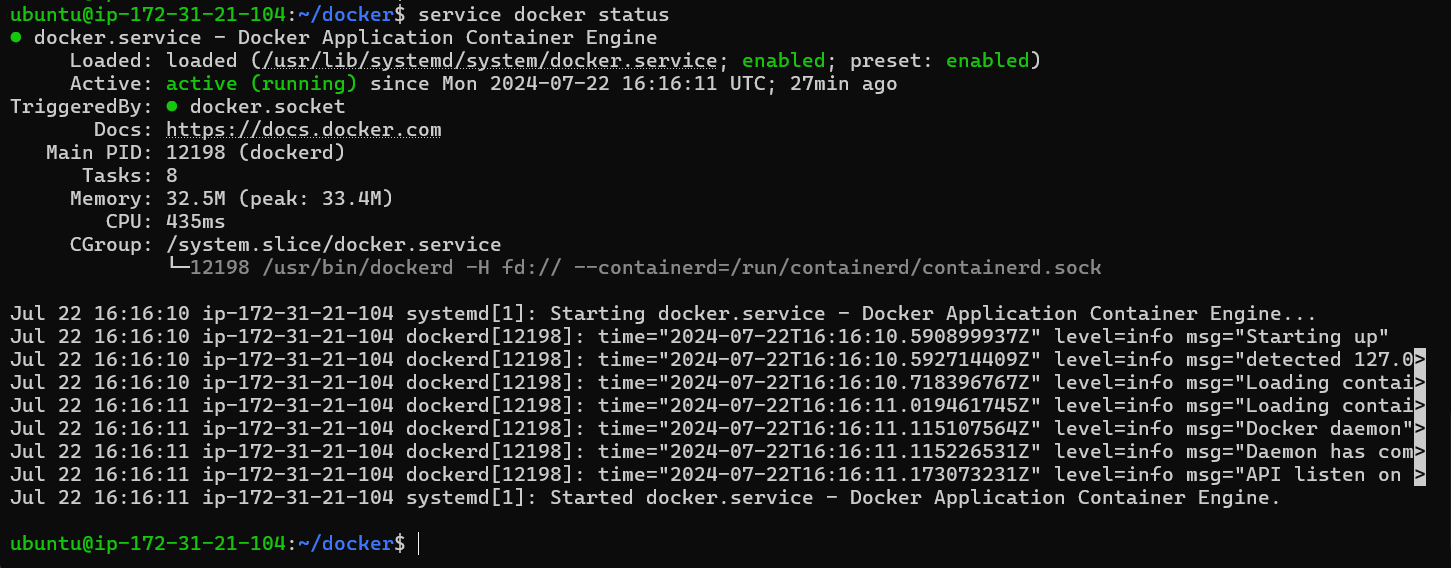
**sudo reboot**

This is necessary for the changes to take effect.

Once you have completed these steps, you will be able to use Docker on your Ubuntu system.

1. Check docker status whether it is running or stopped.

**service docker status**



If it is stopped then run below command to start docker: (Use sudo in beginning of command if ask root permission)

**service docker start**

# Basic Docker Commands

Now, let us look at some common Docker commands and their usage:

## Docker PS

**Command: docker ps [**options**]**

**Options:**

**-a, --all**: Show all containers (default shows just running).

**-q, --quiet**: Only display container IDs.

**--no-trunc**: Don't truncate output.

**-n, --last n**: Show n last created containers (includes all states, default shows just running).

**--latest, -l**: Show the latest created container (includes all states, default shows just running).

**--size, -s**: Display total file sizes.

**--format "TEMPLATE"**: Pretty-print containers using a Go template.

**--filter FILTER**: Filter output based on conditions provided (e.g., docker ps --filter "status=running").

**--format**: Pretty-print containers using a Go template.

**--format "TEMPLATE"**: Format the output using the given Go template.

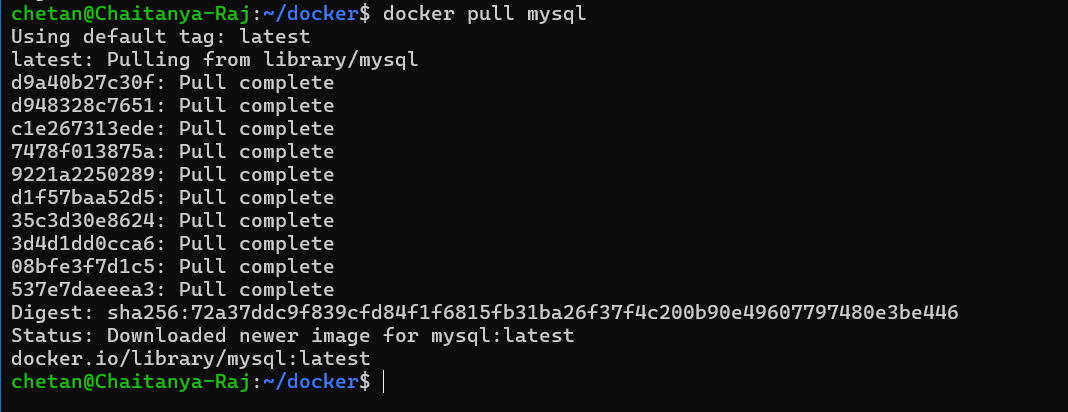
**Usage:** Lists the currently running containers.



## Docker Pull

**Command:**  **docker pull** image\_name

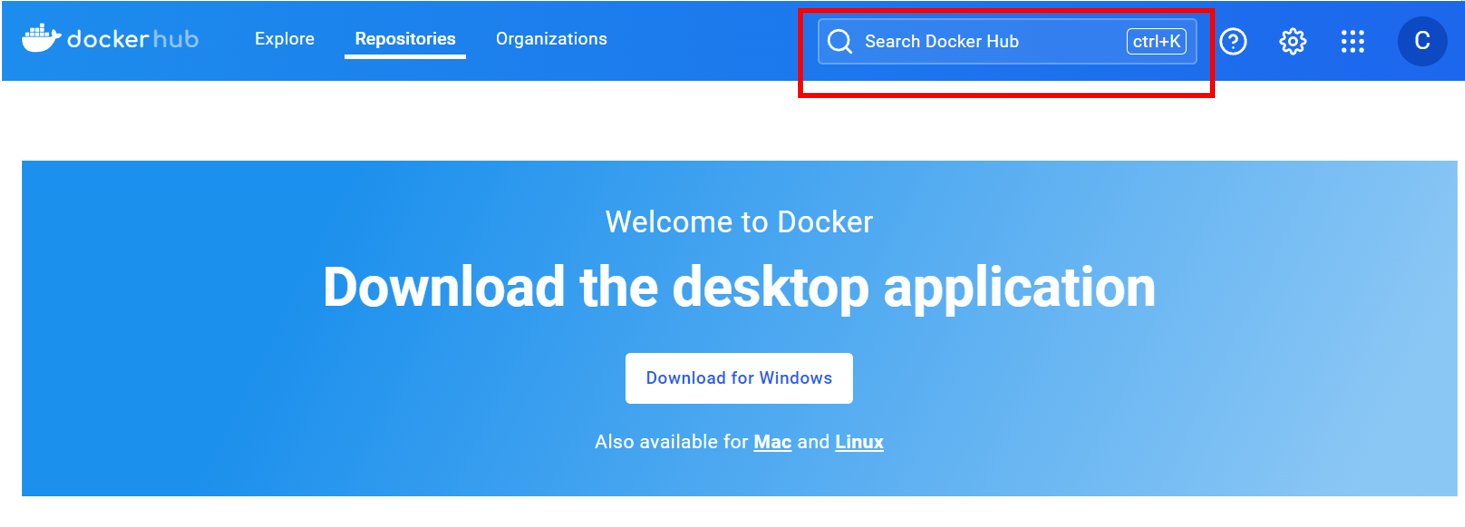
**Usage:** Downloads a Docker image from a registry.

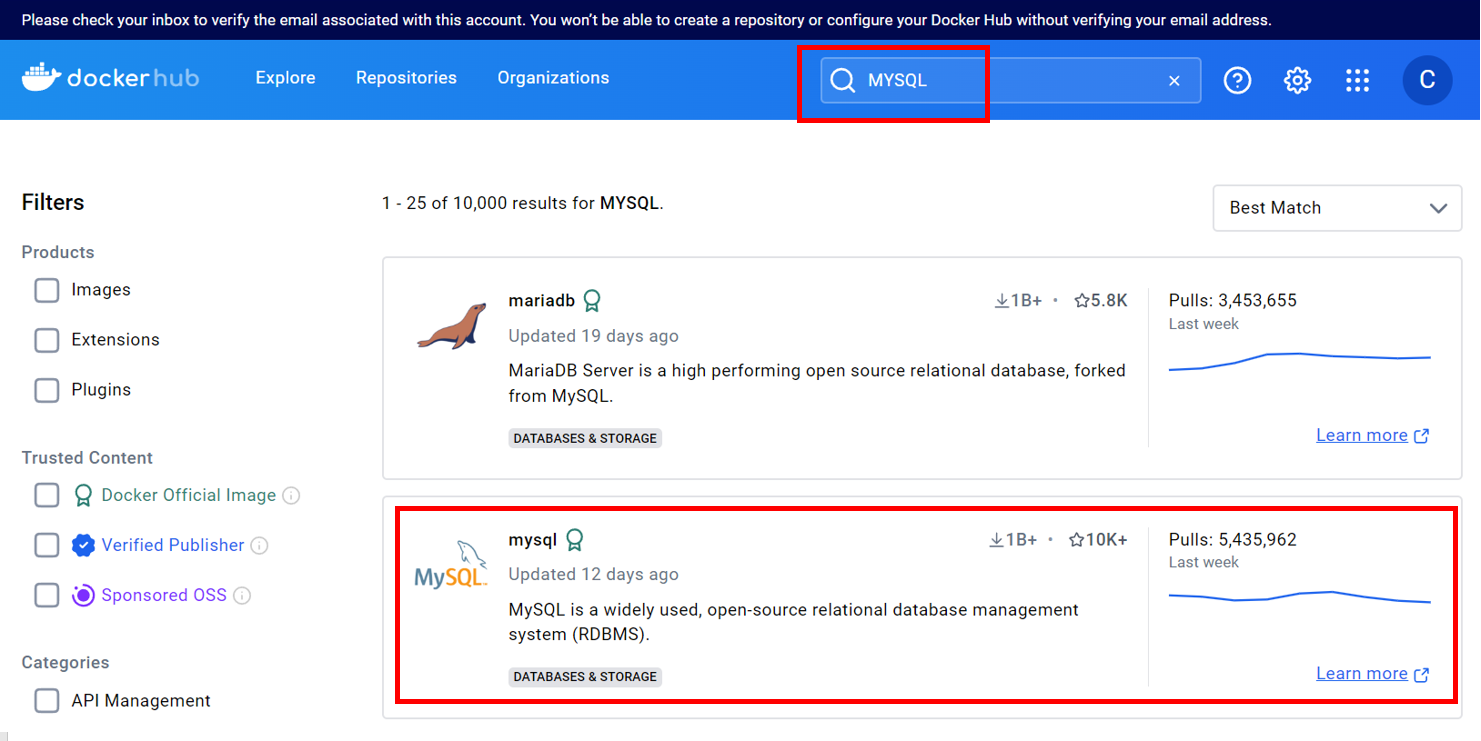


**Note!!!!**

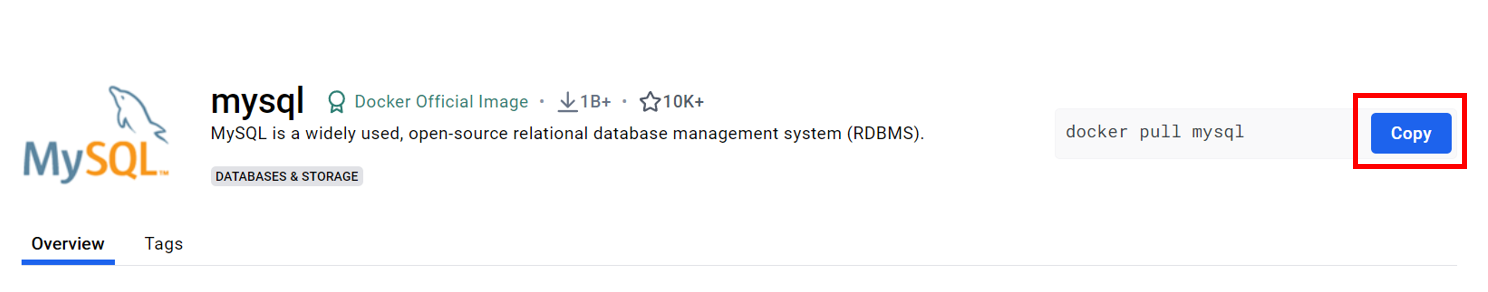
In command “**docker pull** image\_name” image name can be taken from the DOCKER HUB GUI.

Log in to docker HUB GUI and SEARCH for the application package you want to pull from DOCKER HUB GUI.





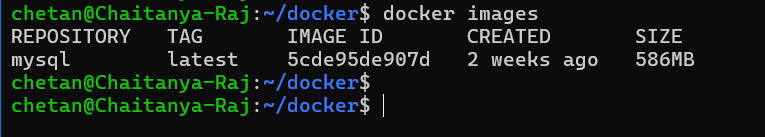
Copy the pull command in the page given and paste in the linux terminal.



## Docker Images

**Command:** **docker images [**options**]**

**Usage:** Lists all available Docker images on the local machine.



## Docker Run

**Command:** **docker run [**options**]** image\_name:tag

**Usage:** Creates and runs a container based on a specified image.

**Note:** Different Applications have different docker run options like -i, -d, -p, -t, etc.

1. **Interactive Mode (-i, --interactive)**:

-i, --interactive: Keep STDIN open even if not attached.

**docker run -i -t ubuntu bash**

Starts a new container interactively with a bash shell.

1. **TTY Mode (-t, --tty)**:

**-t, --tty:** Allocate a pseudo-TTY.

**docker run -it ubuntu**

Allocates a pseudo-TTY (terminal) and connects it to the container's STDIN, allowing you to interact with the container's shell.

1. **Detached Mode (-d, --detach)**:

**-d, --detach:** Run container in background and print container ID.

docker run -d nginx

Runs the container in the background (detached mode), printing the container ID.

1. **Name Container (--name)**:

--name: Assign a name to the container.

docker run --name my-container nginx

Assigns the name my-container to the container running the nginx image.

1. **Publish Ports (-p, --publish)**:

-p, --publish: Publish a container's port(s) to the host.

docker run -p 8080:80 nginx

Publishes port 80 from the container to port 8080 on the host.

1. **Mount Volumes (-v, --volume)**:

-v, --volume: Bind mount a volume.

docker run -v /host/path:/container/path nginx

Mounts a host directory (/host/path) into the container at /container/path.

1. **Environment Variables (-e, --env)**:

-e, --env: Set environment variables.

docker run -e MYSQL\_ROOT\_PASSWORD=password mysql

Sets the environment variable MYSQL\_ROOT\_PASSWORD to password for the mysql container.

**FOR EXAMPLE:**

chetan@Chaitanya-Raj:~/docker$ **docker run -e MYSQL\_ROOT\_PASSWORD=Infy@1028517 mysql**

2024-07-21 15:59:27+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 9.0.0-1.el9 started.

2024-07-21 15:59:27+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'

2024-07-21 15:59:27+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 9.0.0-1.el9 started.

2024-07-21 15:59:27+00:00 [Note] [Entrypoint]: Initializing database files

2024-07-21T15:59:27.712820Z 0 [System] [MY-015017] [Server] MySQL Server Initialization - start.

2024-07-21T15:59:27.714128Z 0 [System] [MY-013169] [Server] /usr/sbin/mysqld (mysqld 9.0.0) initializing of server in progress as process 81

2024-07-21T15:59:27.722706Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.

2024-07-21T15:59:27.967225Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.

2024-07-21T15:59:28.930371Z 6 [Warning] [MY-010453] [Server] root@localhost is created with an empty password ! Please consider switching off the --initialize-insecure option.

2024-07-21T15:59:30.675576Z 0 [System] [MY-015018] [Server] MySQL Server Initialization - end.

2024-07-21 15:59:30+00:00 [Note] [Entrypoint]: Database files initialized

2024-07-21 15:59:30+00:00 [Note] [Entrypoint]: Starting temporary server

2024-07-21T15:59:30.797074Z 0 [System] [MY-015015] [Server] MySQL Server - start.

2024-07-21T15:59:31.024473Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 9.0.0) starting as process 133

2024-07-21T15:59:31.035365Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.

2024-07-21T15:59:31.192902Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.

2024-07-21T15:59:31.418660Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem is self signed.

2024-07-21T15:59:31.418725Z 0 [System] [MY-013602] [Server] Channel mysql\_main configured to support TLS. Encrypted connections are now supported for this channel.

2024-07-21T15:59:31.423495Z 0 [Warning] [MY-011810] [Server] Insecure configuration for --pid-file: Location '/var/run/mysqld' in the path is accessible to all OS users. Consider choosing a different directory.

2024-07-21T15:59:31.459220Z 0 [System] [MY-011323] [Server] X Plugin ready for connections. Socket: /var/run/mysqld/mysqlx.sock

2024-07-21T15:59:31.459284Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready for connections. Version: '9.0.0' socket: '/var/run/mysqld/mysqld.sock' port: 0 MySQL Community Server - GPL.

2024-07-21 15:59:31+00:00 [Note] [Entrypoint]: Temporary server started.

'/var/lib/mysql/mysql.sock' -> '/var/run/mysqld/mysqld.sock'

Warning: Unable to load '/usr/share/zoneinfo/iso3166.tab' as time zone. Skipping it.

Warning: Unable to load '/usr/share/zoneinfo/leap-seconds.list' as time zone. Skipping it.

Warning: Unable to load '/usr/share/zoneinfo/leapseconds' as time zone. Skipping it.

Warning: Unable to load '/usr/share/zoneinfo/tzdata.zi' as time zone. Skipping it.

Warning: Unable to load '/usr/share/zoneinfo/zone.tab' as time zone. Skipping it.

Warning: Unable to load '/usr/share/zoneinfo/zone1970.tab' as time zone. Skipping it.

2024-07-21 15:59:32+00:00 [Note] [Entrypoint]: Stopping temporary server

2024-07-21T15:59:32.835818Z 10 [System] [MY-013172] [Server] Received SHUTDOWN from user root. Shutting down mysqld (Version: 9.0.0).

2024-07-21T15:59:33.826132Z 0 [System] [MY-010910] [Server] /usr/sbin/mysqld: Shutdown complete (mysqld 9.0.0) MySQL Community Server - GPL.

2024-07-21T15:59:33.826212Z 0 [System] [MY-015016] [Server] MySQL Server - end.

2024-07-21 15:59:33+00:00 [Note] [Entrypoint]: Temporary server stopped

2024-07-21 15:59:33+00:00 [Note] [Entrypoint]: MySQL init process done. Ready for start up.

2024-07-21T15:59:33.850319Z 0 [System] [MY-015015] [Server] MySQL Server - start.

2024-07-21T15:59:34.027025Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 9.0.0) starting as process 1

2024-07-21T15:59:34.032494Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.

2024-07-21T15:59:34.171669Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.

2024-07-21T15:59:34.303989Z 0 [Warning] [MY-010068] [Server] CA certificate ca.pem is self signed.

2024-07-21T15:59:34.304030Z 0 [System] [MY-013602] [Server] Channel mysql\_main configured to support TLS. Encrypted connections are now supported for this channel.

2024-07-21T15:59:34.306425Z 0 [Warning] [MY-011810] [Server] Insecure configuration for --pid-file: Location '/var/run/mysqld' in the path is accessible to all OS users. Consider choosing a different directory.

2024-07-21T15:59:34.323433Z 0 [System] [MY-011323] [Server] X Plugin ready for connections. Bind-address: '::' port: 33060, socket: /var/run/mysqld/mysqlx.sock

2024-07-21T15:59:34.323459Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready for connections. Version: '9.0.0' socket: '/var/run/mysqld/mysqld.sock' port: 3306 MySQL Community Server - GPL.

2024-07-21T16:05:16.505164Z 0 [System] [MY-013172] [Server] Received SHUTDOWN from user <via user signal>. Shutting down mysqld (Version: 9.0.0).

2024-07-21T16:05:17.764934Z 0 [System] [MY-010910] [Server] /usr/sbin/mysqld: Shutdown complete (mysqld 9.0.0) MySQL Community Server - GPL.

2024-07-21T16:05:17.764966Z 0 [System] [MY-015016] [Server] MySQL Server - end.

chetan@Chaitanya-Raj:~/docker$

chetan@Chaitanya-Raj:~/docker$

1. **Container Resource Limits (--memory, --cpus)**:

--memory: Limit memory usage.

--cpus: Limit CPU usage.

docker run --memory 512m --cpus 0.5 nginx

Limits the memory to 512 MB and CPU usage to 0.5 CPUs for the nginx container.

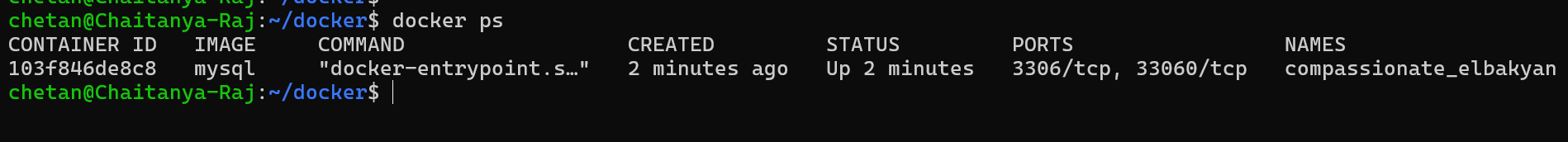
Example:

docker run -d --name my-nginx -p 8080:80 nginx

## Docker PS

**Command: docker ps [**options**]**

**Usage:** Lists the currently running containers.



**Options:**

**-a, --all**: Show all containers (default shows just running).

**-q, --quiet**: Only display container IDs.

**--no-trunc**: Don't truncate output.

**-n, --last n**: (Here n = number): Show “n” last created containers (includes all states, default shows just running).

**--latest, -l**: Show the latest created container (includes all states, default shows just running).

**--size, -s**: Display total file sizes.

**--format "TEMPLATE"**: Pretty-print containers using a Go template.

**--filter FILTER**: Filter output based on conditions provided (e.g., docker ps --filter "status=running").

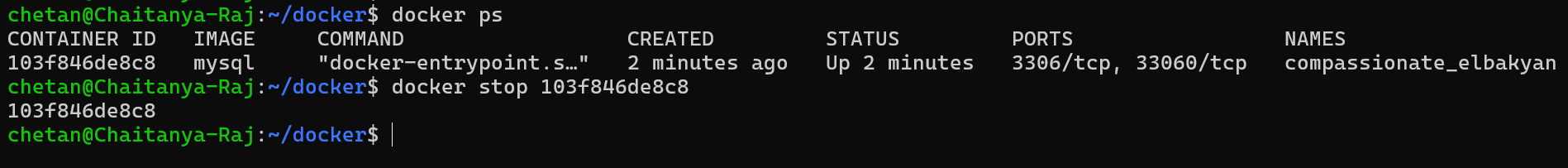
**--format**: Pretty-print containers using a Go template.

**--format "TEMPLATE"**: Format the output using the given Go template.

## Docker Stop

**Command:** **docker stop** container\_id

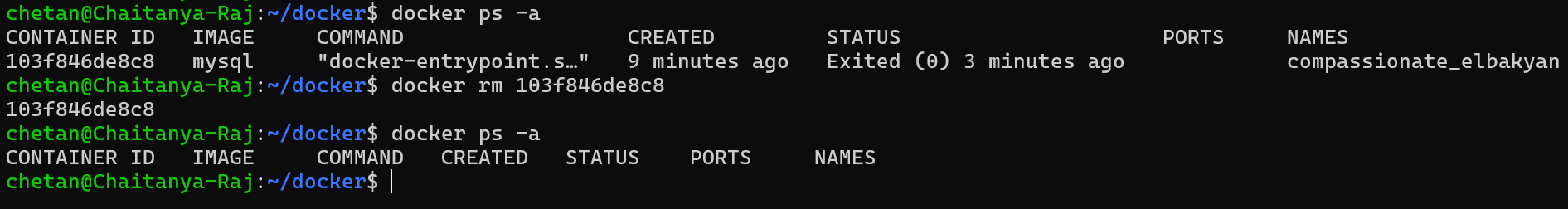
**Usage:** Stops a running container.

****

## Docker RM (Remove Container)

**Command:** **docker rm** container\_id ***//OR//* docker remove** container\_id

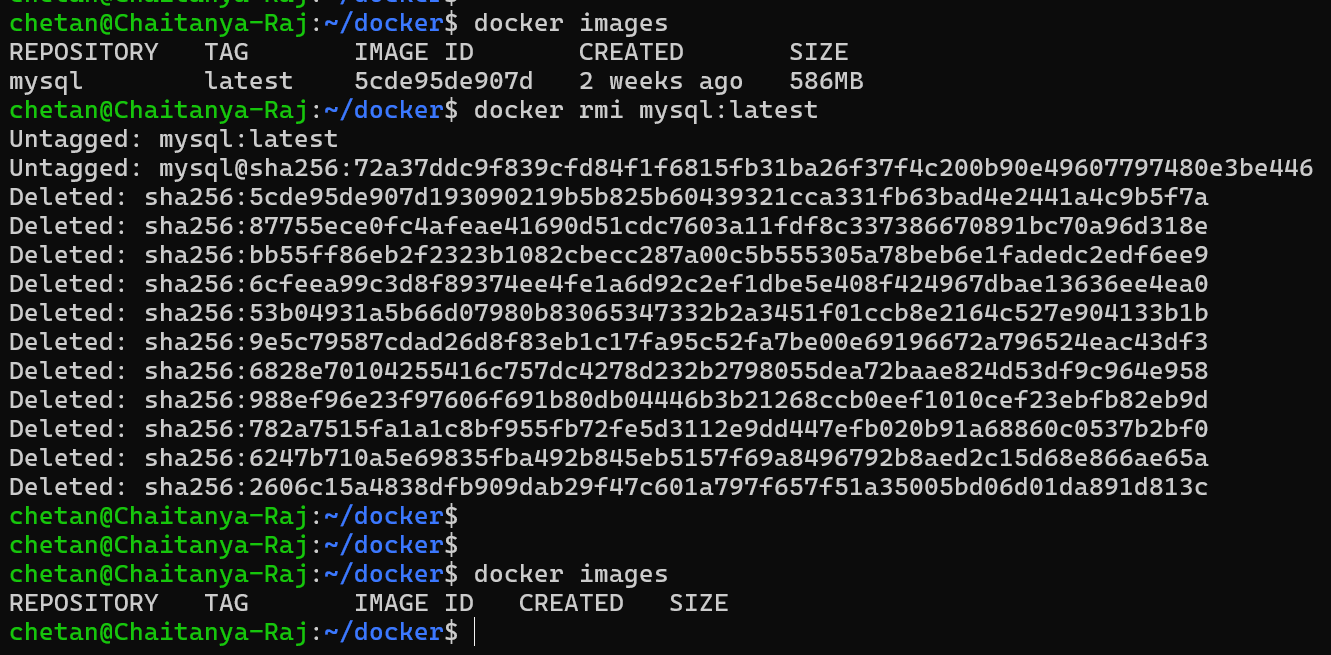
**Usage:** Removes a stopped container.

****

## Docker RMI (Remove Image)

**Command:** **docker rmi** image\_name:tag

**Usage:** Removes a Docker image from the local machine.

****

## Docker Build

**Command:** **docker build -t** image\_name:tag **.**

**Usage:** Builds a Docker image from a Dockerfile in the current directory.

## Docker Push

**Command:** **docker push** image\_name:tag

**Usage:** Uploads a Docker image to a registry.

## Docker Exec

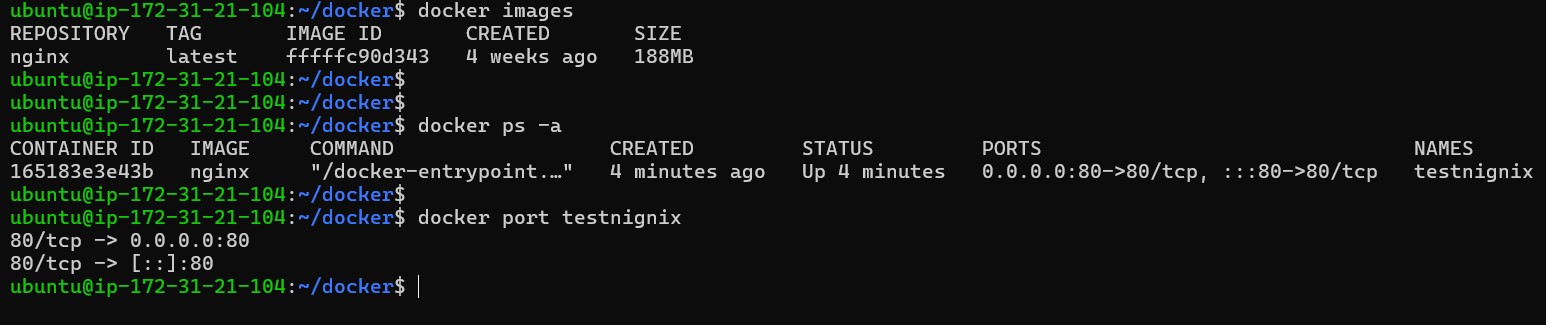
**Command:** **docker exec [**options**]** container\_idcommand

**Usage:** Executes a command inside a running container.

## Docker Port

**Command:** **docker port <**containername**>**

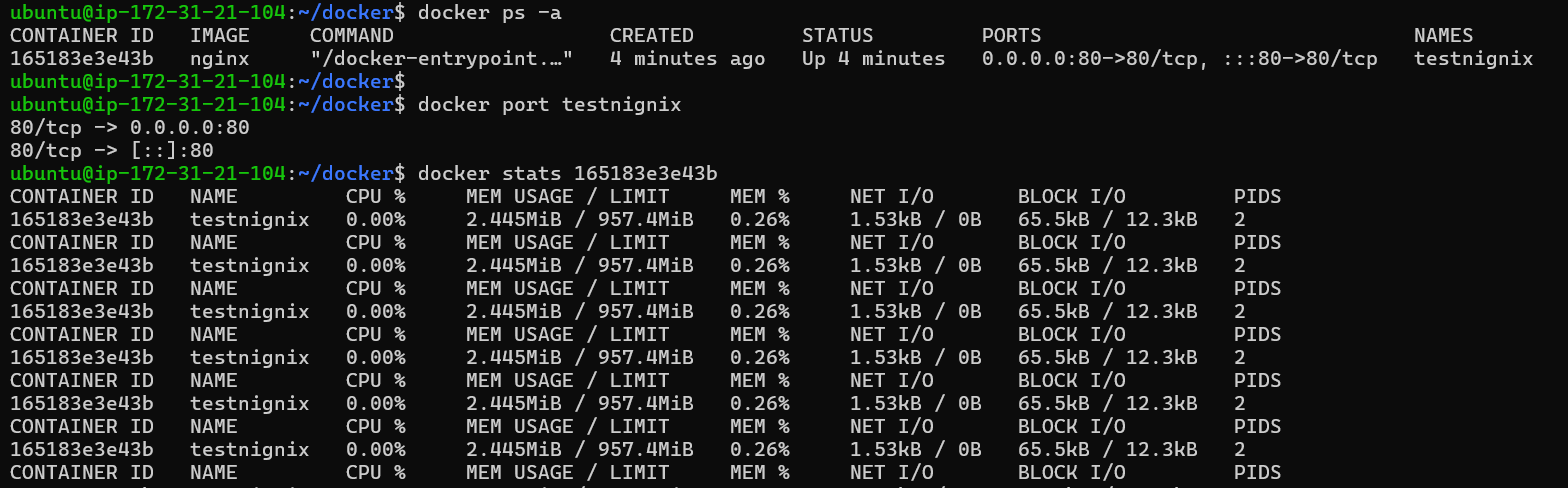
**Usage:** Use the docker port command to list the port mappings for a container.



## Docker stats

**Command:** **docker stats <**container name or id**>**

**Usage:** Use the docker stats command to view resource usage statistics for one or more containers.



## Docker Inspect

**Command:** **docker inspect <**image or container name**>**

**Usage:** Use the docker inspect command to view detailed information about a container or image.

# Three WAYS TO USE Docker

**1: Develop and build your application**: Start by developing your application code using your preferred programming language and tools. Once you have a working application, you will need to create a Dockerfile. A Dockerfile is a text file that contains a set of instructions for building a Docker image. It defines the base image, adds any necessary dependencies, copies your application code into the image, and specifies the commands to run when the container is started.

**2: Build a Docker image:** Use the Docker CLI (Command Line Interface) or a build tool like Docker Compose or a CI/CD (Continuous Integration/Continuous Deployment) system to build a Docker image based on your Dockerfile. The Docker image is a standalone package that includes your application code, runtime environment, and any required dependencies.

**3: Run Docker containers:**Once you have a Docker image, you can create and run Docker containers from it. Containers are instances of Docker images that can be started, stopped, and managed independently. You can run containers locally on your development machine or deploy them to a production environment.

## Run Docker containers

### Create and Open NGINIX website by running NGINIX Container on docker with the Help of AWS EC2 (ubuntu AMI) virtual machine.

Here's a step-by-step guide:

### Launch an EC2 Instance:

1. Log in to the AWS Management Console.
2. Navigate to the EC2 dashboard.
3. Click "Launch Instance" and choose an Amazon Machine Image (AMI). Select an image that suits your needs, such as "Amazon Linux 2" or "Amazon Linux 2 LTS."

### Connect to the EC2 Instance

1. Once the instance is running, connect to it using SSH. Use the key pair you selected during the instance launch.

**ssh -i YourKeyPair.pem ec2-user@your-ec2-instance-ip**

### Install Docker on EC2 Instance

For Amazon Linux 2:

**sudo yum update -y**

**sudo amazon-linux-extras install docker**

**sudo service docker start**

**sudo usermod -aG docker ec2-user**

For Ubuntu:

**sudo apt-get update**

**sudo apt-get install docker.io**

**sudo systemctl start docker**

**sudo systemctl enable docker**

**sudo usermod -aG docker ubuntu**

For CentOS:

**sudo yum update -y**

**sudo yum install docker -y**

**sudo systemctl start docker**

**sudo systemctl enable docker**

**sudo usermod -aG docker centos**

### Verify Docker Installation

**docker --version**

**docker info**

### Run a Simple Docker Container

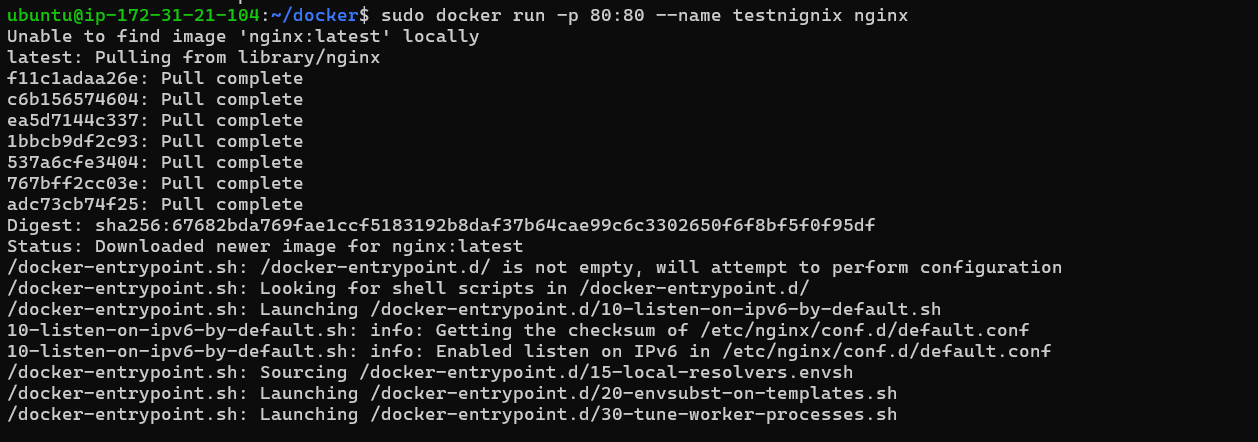
Let us run the official Nginx web server as a simple example:

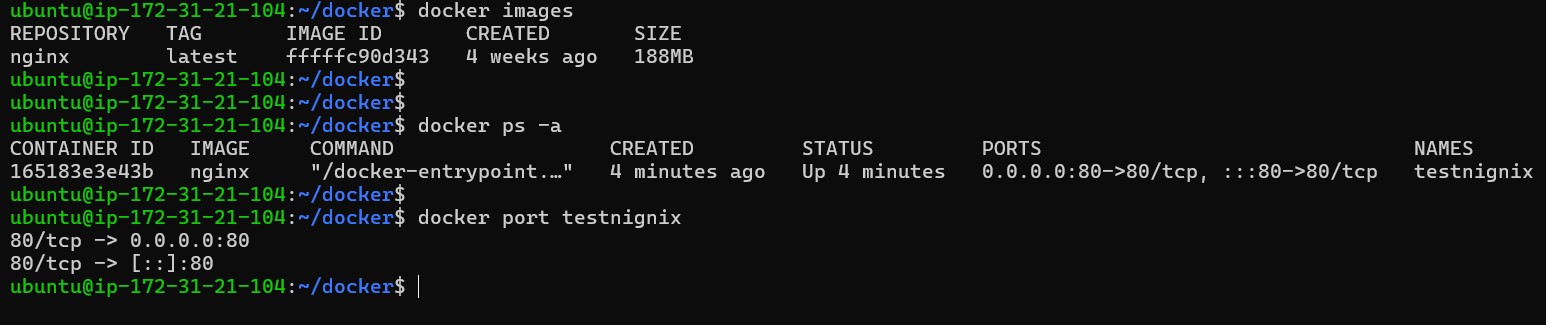
**docker run -d -p 80:80 –name <**mynginix**> nginx**

**docker run -d -p 80:80 --name testnignix nginx**

This command does the following:

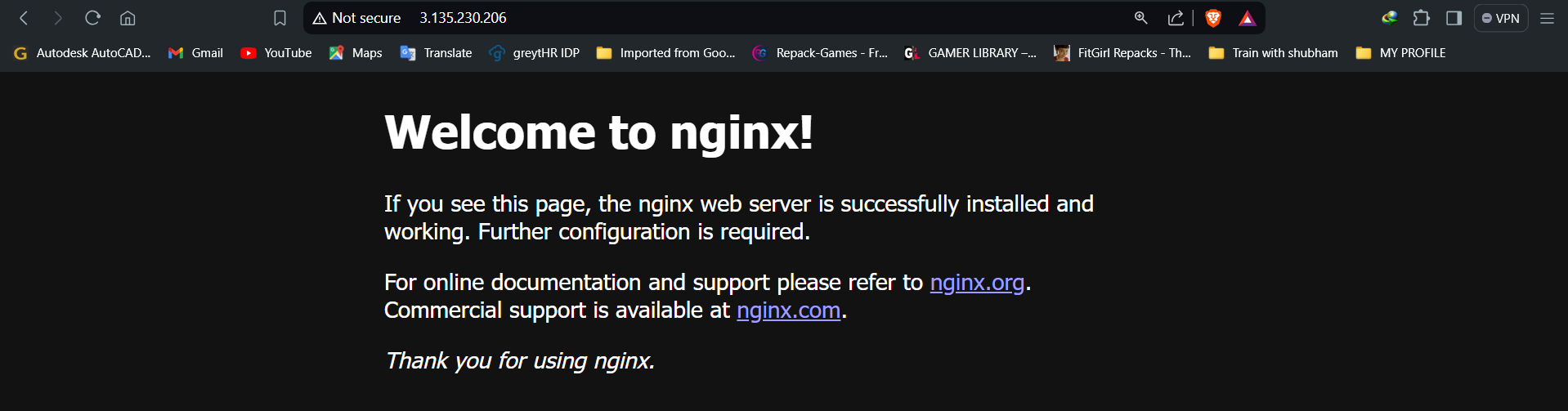
* **-d** : Run the container in the background.
* **-p 80:80** : Map port 80 from the host to port 80 in the container.
* **--name <**mynginx**>** : Assign the name "mynginx" to the container.
* **nginx** : Use the official Nginx image from Docker Hub.





### Access Nginx in the Browser

Open your web browser and navigate to **http://<**your-ec2-instance-ip**>**. You should see the default Nginx welcome page.



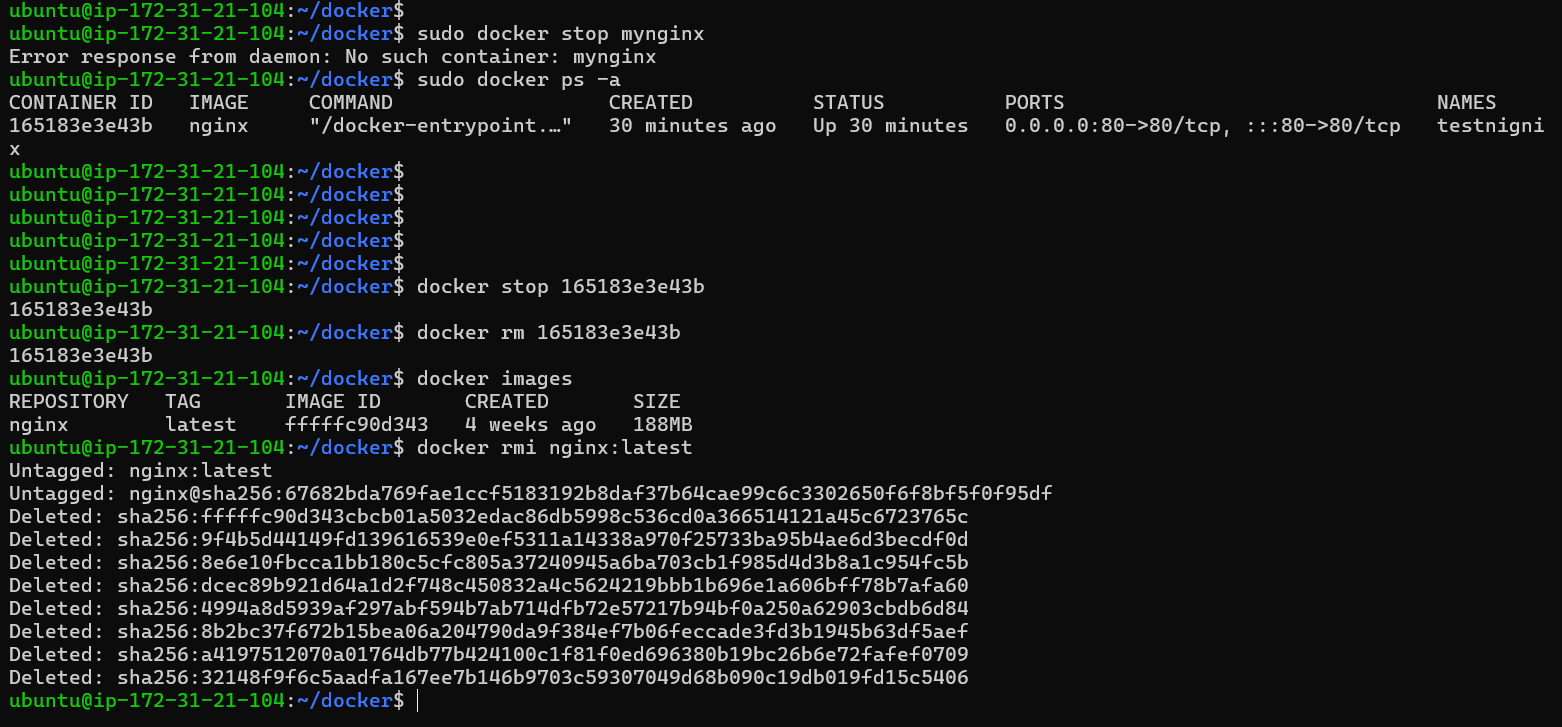
### Cleanup (Optional)

If you want to stop and remove the container:

**docker stop** container\_id

**docker rm** container\_id

**docker rmi** image\_name:tag



Remember to replace *YourKeyPair.pem* and your-ec2-instance-ip with your actual key pair and EC2 instance information.

This is a basic example, but it demonstrates the process of installing Docker on an EC2 instance, running a container, and accessing the application. Depending on your application requirements, you may need to expose additional ports or configure other Docker options.

## Build a Docker image



#### HOW TO BUILD DOCKER IMAGE, RUN CONTAINER WITH IT AND OPEN THE WEBSITE PAGE:

Creating and testing a Flask application using Docker can streamline your development process, ensuring consistency across different environments. This guide will walk you through the steps to set up and test a Flask application using Docker. Let's get started!

Before we begin, make sure you have the following installed:

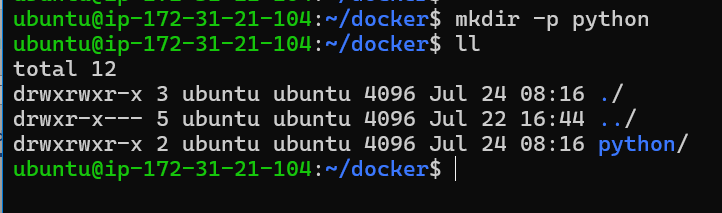
* Install Docker on your AWS machine
* AWS EC2 Machine

For the flask app, here we are using python:3.11-alpine3.20

Supported tags and respective Dockerfile links

1. Set Up Your Project Directory

mkdir -p <your Directory name>



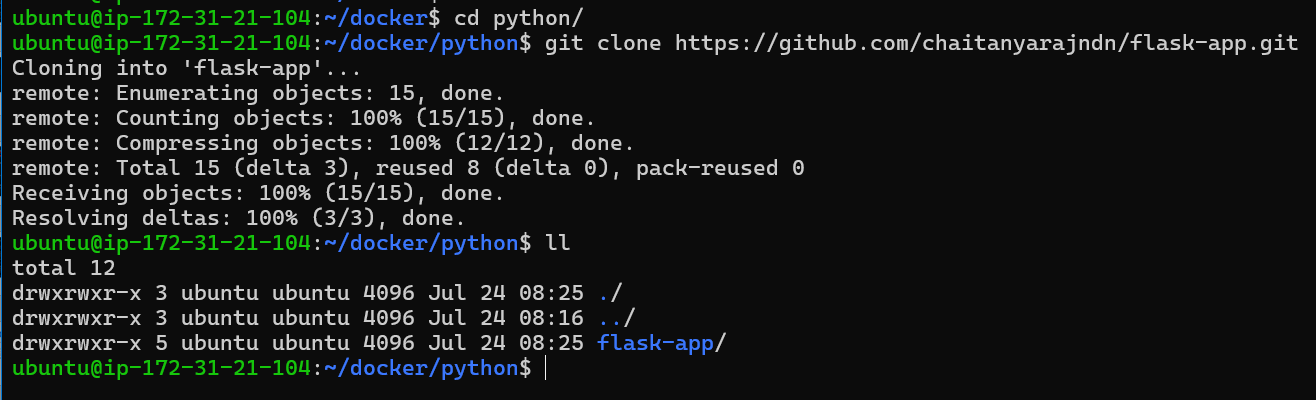
1. Clone repository of GitHub on that created directory:

This GitHub repository contains the flask app codes which can be copied to local directory and build the image with flask app.

<https://github.com/dheeruyadav54/flask-app.git>

<https://github.com/chaitanyarajndn/flask-app.git>

git clone https://github.com/chaitanyarajndn/flask-app.git



1. Create a Dockerfile.

A Dockerfile is a script that contains a set of instructions for building a Docker image. Docker images are lightweight, standalone, and executable packages that include everything needed to run a piece of software, including the code, runtime, libraries, and system tools. Dockerfiles are used to automate the process of creating Docker images.

Here's a basic example of a Dockerfile:

# Use an official Python runtime as a parent image

FROM python:3.9-alpine3.19

# Set the working directory in the container

WORKDIR /app

# Copy the current directory contents into the container at /app

COPY . /app

# Install any needed packages specified in requirements.txt

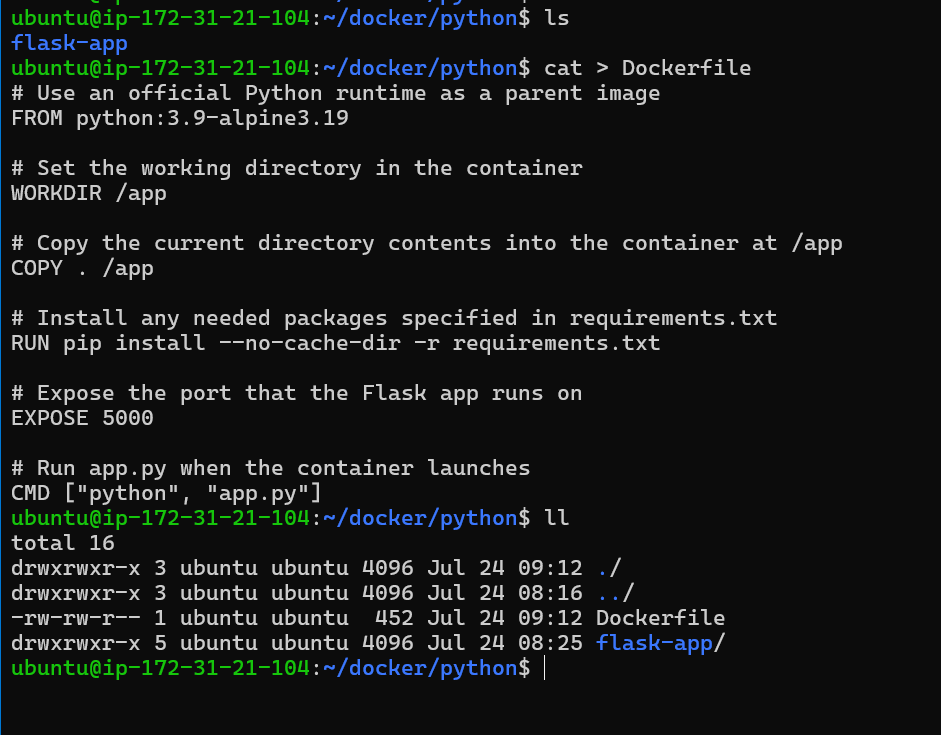
RUN pip install --no-cache-dir -r requirements.txt

# Expose the port that the Flask app runs on

EXPOSE 5000

# Run app.py when the container launches

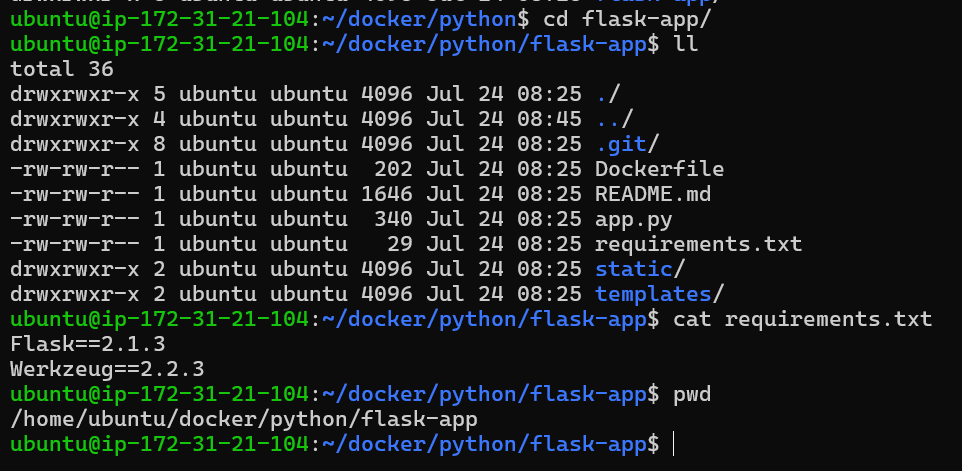
CMD ["python", "app.py"]



Now, let's break down each part of this Dockerfile:

**Explanation:**

1. **Base Image**: python:3.9-alpine3.19 is used as the base image. Alpine Linux provides a lightweight base image with smaller size and enhanced security features compared to standard Linux distributions.
2. **Working Directory**: WORKDIR /app sets the working directory inside the container to /app. This is where your application code will be copied and where the Python interpreter will execute.
3. **Copy Files**: COPY . /app copies all files from your local directory into the /app directory inside the container. This includes your Flask application code (app.py and possibly other files) and requirements.txt.

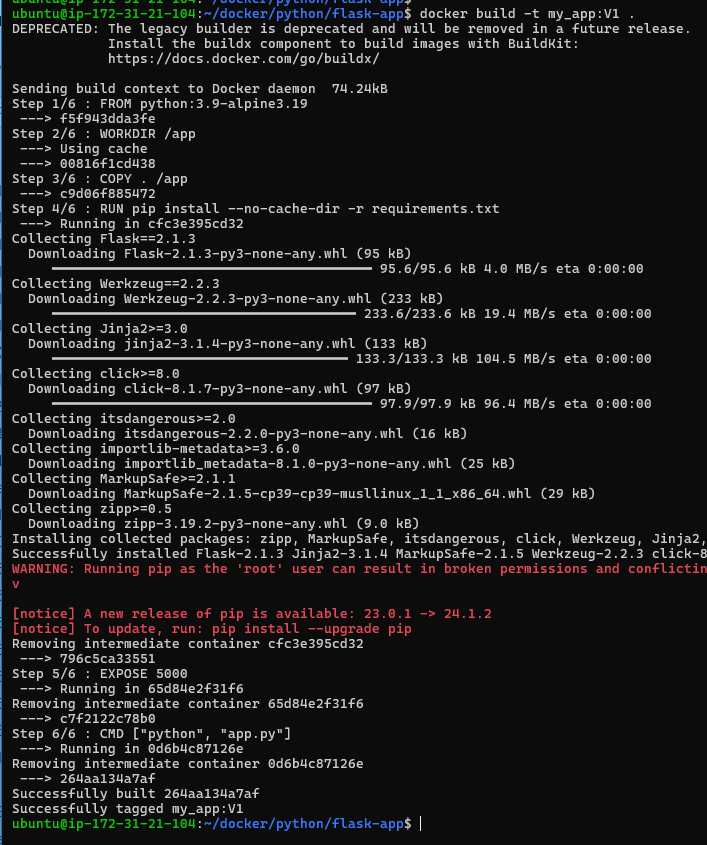


1. **Install Dependencies**: RUN pip install --no-cache-dir -r requirements.txt installs the Python dependencies listed in requirements.txt. The --no-cache-dir option prevents caching of installed packages, reducing the final image size.
2. **Expose Port**: EXPOSE 5000 documents that the container listens on port 5000 at runtime. This does not actually publish the port; it is informational and typically used when running the container with -p or -P to map ports.
3. **CMD**: CMD ["python", "app.py"] specifies the default command to run when the container starts. It runs python app.py, assuming app.py is your Flask application entry point script.
4. Build Your Docker image

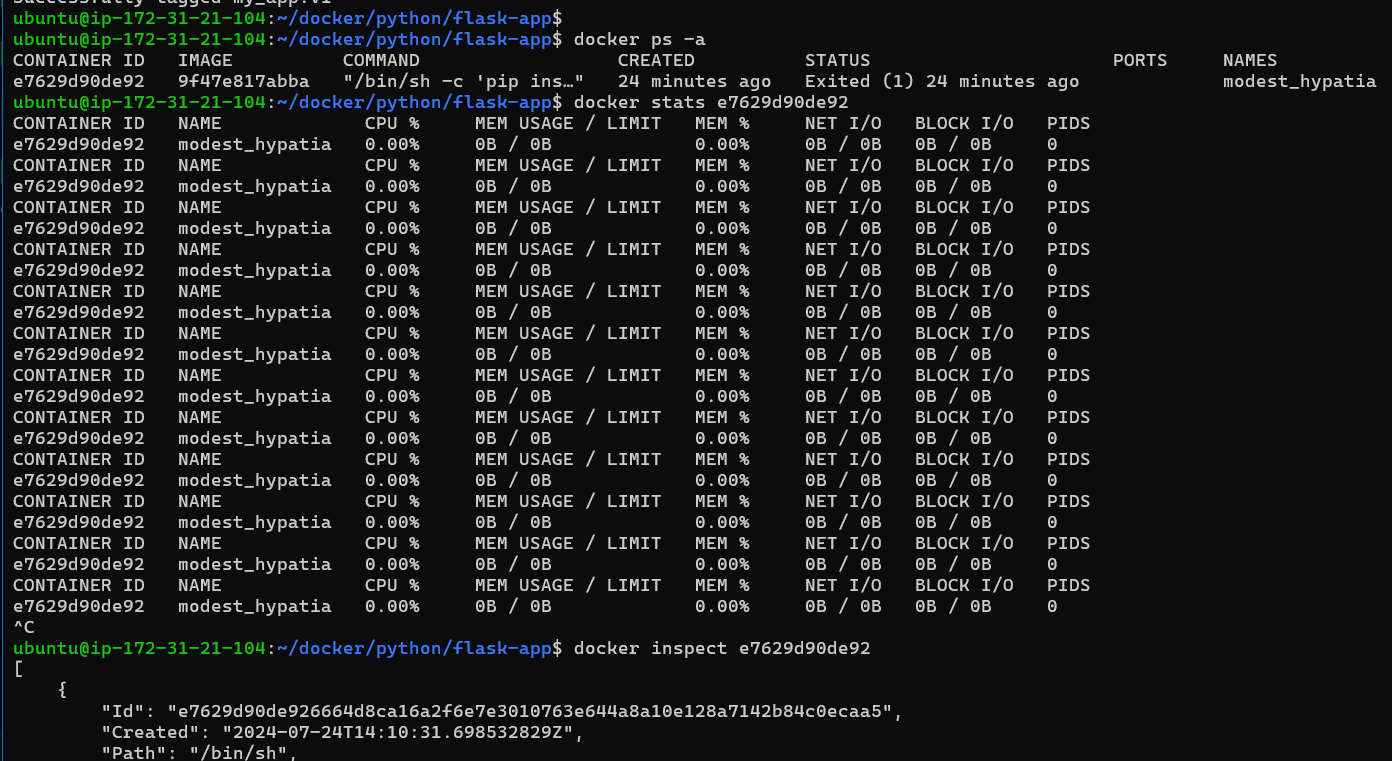
docker build -t my\_app:V1 .

Let's break down the command:

* -t my\_app:V1: This specifies the tag (my\_app:V1) you want to give to your Docker image. Replace my\_app with your preferred image name and V1 with the version tag you want to use.
* .: This dot (.) at the end of the command denotes the build context. It tells Docker to use the current directory (where your Dockerfile is located) as the build context. Make sure you run this command from the directory where your Dockerfile (my\_flask\_app/Dockerfile) is located.



1. Verify the container has been created.



1. Run your container

docker run -p 8080:5000 --name test\_flask my\_app:V1

1. Now ✅ Testing Your Flask App

🌐 Access Your Flask Application

Copy public IP of your Instance and paste on tab

127.152.55.44:8080

### Building a Python Web App

# Uninstall & Remove DOCKER

// check docker is installed or not

**dpkg -l | grep -i docker**

// remove volume , network , container and image files

**sudo docker volume prune -f**

**sudo docker network prune -f**

**sudo docker container prune -f**

**sudo docker image prune -a**

// remove docker

**sudo apt purge docker**

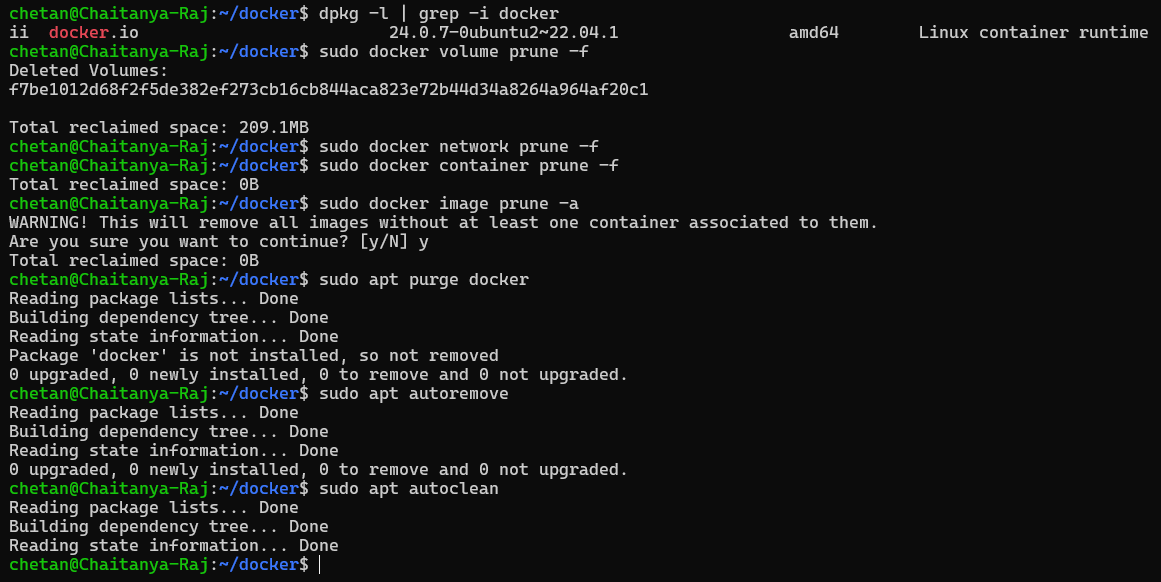
**OR**

**sudo apt remove docker**

// remove dependency packages related to docker

**sudo apt autoremove**

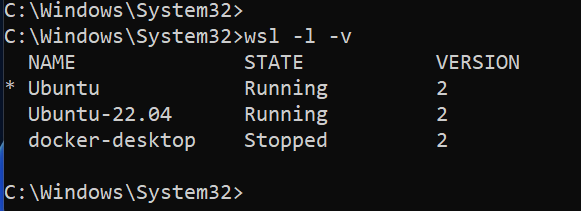
**sudo apt autoclean**



To remove **docker-desktop** from **Windows OS** Subsystem for Linux (WSL), Run command prompt as administrator.

* Run command to see which app are running under WSL:

**wsl -l -v**

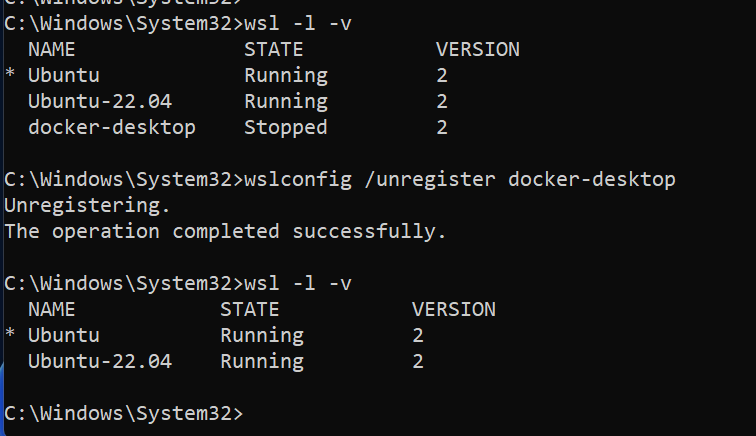


* Type below command to uninstall Docker:

**wslconfig /unregister docker-desktop**

**//OR//**

**wsl –-unregister docker-desktop**



# ERROR AND ITS SOLUTION

## Unauthorized: incorrect username or password

**LOGIN TO DOCKER USING DOCKER LOGIN COMMAND.**

It seems like Docker is unable to pull the MySQL image because it's encountering an authorization issue with the Docker registry. The error message "unauthorized: incorrect username or password" suggests that Docker is not able to authenticate your credentials correctly.

Log in to Docker:

**docker login**

