XFLOW RESEARCH

Docker Containers/Volumes

docker (April 2024)

Table of Contents

Introduction

Docker is an open-source platform that automates the deployment of applications inside software containers. Containers are lightweight, portable, and self-sufficient execution environments that encapsulate application code, runtime, system tools, libraries, and sett
Docker simplifies the process of developing, shipping, and running applications by providi
standardized way to package and deploy them in isolated environments.This ensures that
application will run on any environment.

How Does Docker Work?

- Docker works by utilizing containerization technology to create isolated environmen called containers.
- Containers share the host system's kernel but are isolated from each other, providir consistency across different environments.

Purpose of Docker

The primary purpose of Docker is to streamline the software development and deploymer
workflow. It aims to improve consistency, scalability, and efficiency by enabling developer
build, ship, and run applications in lightweight, portable containers. Docker facilitates fast
development cycles, easier collaboration, and greater flexibility in managing application dependencies and environments.

Why Do We Need Docker?

Environment Consistency

Docker ensures consistency between development, testing, and production environments Since containers encapsulate the entire runtime environment, applications run the same regardless of where they are deployed.

Dependency Management

Docker eliminates "it works on my machine" issues by packaging applications with their dependencies. This makes it easier to manage dependencies and avoids conflicts betwee different versions.

Scalability

Docker allows applications to be scaled quickly and efficiently. Containers can be easily replicated and distributed across multiple hosts, enabling horizontal scaling to meet varyi workload demands.

Resource Efficiency

Containers share the host system's kernel and resources, making them lightweight compa virtual machines. This results in faster startup times and better resource utilization.

When to Use Docker?

Microservices Architecture

Docker is well-suited for microservices architecture, where applications are divided into smaller, independent services that can be developed, deployed, and scaled separately.

Continuous Integration/Continuous Deployment (CI/CD)

Docker streamlines the CI/CD process by providing a consistent environment for building, testing, and deploying applications.

DevOps Practices

Docker facilitates DevOps practices such as infrastructure like code, version control, and automated testing by standardizing the deployment process.

Hybrid and Multi-Cloud Environments

Docker enables portability across different cloud providers and on-premises environments making it easier to migrate and scale applications.

Main Features of Docker

Containerization

Docker enables the packaging of applications and their dependencies into containers, ens
consistency and portability.

Image-based

Docker uses images as templates for creating containers. Images are read-only and conta
the necessary components for running an application.

Lightweight

Containers are lightweight compared to virtual machines, as they share the host system's
kernel and resources.

Scalable

Docker allows applications to be scaled quickly and efficiently by replicating containers ac
multiple hosts.

Components of Docker

Docker Engine

Docker Engine is the core component of Docker, responsible for building, running, and managing containers.

Docker Daemon

Docker Daemon (dockerd) is a background service that manages Docker objects, such as images, containers, networks, and volumes.

Docker CLI

Docker CLI (Command Line Interface) is a tool used to interact with the Docker Engine anc manage Docker objects.

Docker Images

Docker Images are read-only templates used to create containers. They contain the applic code, runtime, libraries, and dependencies.

Docker Client

Docker Client (docker) is a command-line tool used to interact with the Docker daemon ar manage Docker objects.

Docker Registry

Docker Registry stores Docker images, allowing users to share and distribute their images publicly or privately.

Docker Desktop

Docker Desktop is a desktop application that provides an easy-to-use interface for manag
Docker containers and images on Windows and macOS operating systems. It includes Doc
Engine, Docker CLI, Docker Compose, and Docker Kubernetes Service (Docker Desktop fo
Windows only).

Docker Hub

- Docker Hub is a cloud-based registry service that allows users to store and share Do
images.
- It provides access to a vast collection of pre-built images for various software stacks
allows users to publish their images for public or private use.

Docker Orchestration

Orchestration Tools

Orchestration tools like Docker Swarm and Kubernetes help manage and scale containeriz
applications across multiple hosts.

Docker Swarm

Docker Swarm is Docker's built-in orchestration tool, designed to manage clusters of Dock
hosts and deploy services across them.

Dockerfile

What is a Dockerfile?

- A Dockerfile is a text document that contains instructions for building a Docker imag
- Dockerfile defines the environment, dependencies, and commands needed to run a
  application inside a container.

How Does a Dockerfile Work?

- Dockerfile consists of a series of commands that are executed sequentially to build
  image.
- Commands like `FROM, RUN, COPY` and `CMD` are commonly used to define the image's base, install dependencies, copy files, and set the default command, respectively.

Benefits of Docker

Consistency

Docker ensures consistency between development, testing, and production environments
reducing the risk of deployment errors.

Scalability

Docker enables applications to be scaled quickly and efficiently by replicating containers a
multiple hosts.

Resource Efficiency

Containers are lightweight and share the host system's resources, resulting in faster start
times and better resource utilization.

Drawbacks of Docker

Learning Curve

Docker has a steep learning curve, especially for beginners, due to its complex concepts a
terminology.

Security Concerns

Improperly configured Docker containers can pose security risks, such as privilege escalat
and container breakouts.

Docker Concepts

- **Images:** Docker images are read-only templates used to create containers. They co the application code, runtime, libraries, and other dependencies.
- **Containers:** Containers are lightweight, portable, and self-sufficient execution environments created from Docker images. They run instances of applications in isolated environments.

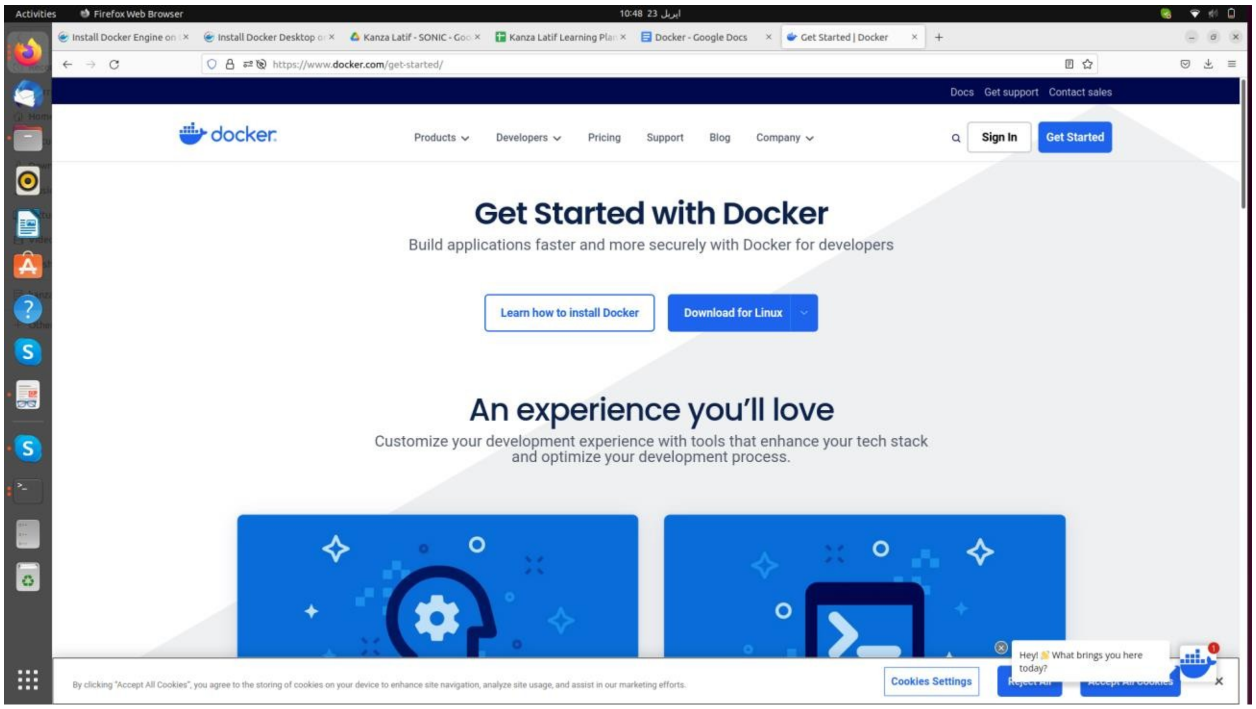
Basic Docker Commands

- `docker pull <image>`: Pull an image from a Docker registry (e.g., Docker Hub).
- `docker run <image>`: Create and start a new container based on the specified image.
- `docker ps`: List running containers.
- `docker exec -it <container> <command>`: Execute a command inside a running container.
- `docker stop <container>`: Stop a running container.
- `docker rm <container>:` Remove a container.

Installation Steps

1. Download Docker Desktop

Visit the official Docker website (https://www.docker.com/) and you’ll see the “Get Startec
button on the top right side of the Screen. Click on it.



Get Started with Docker

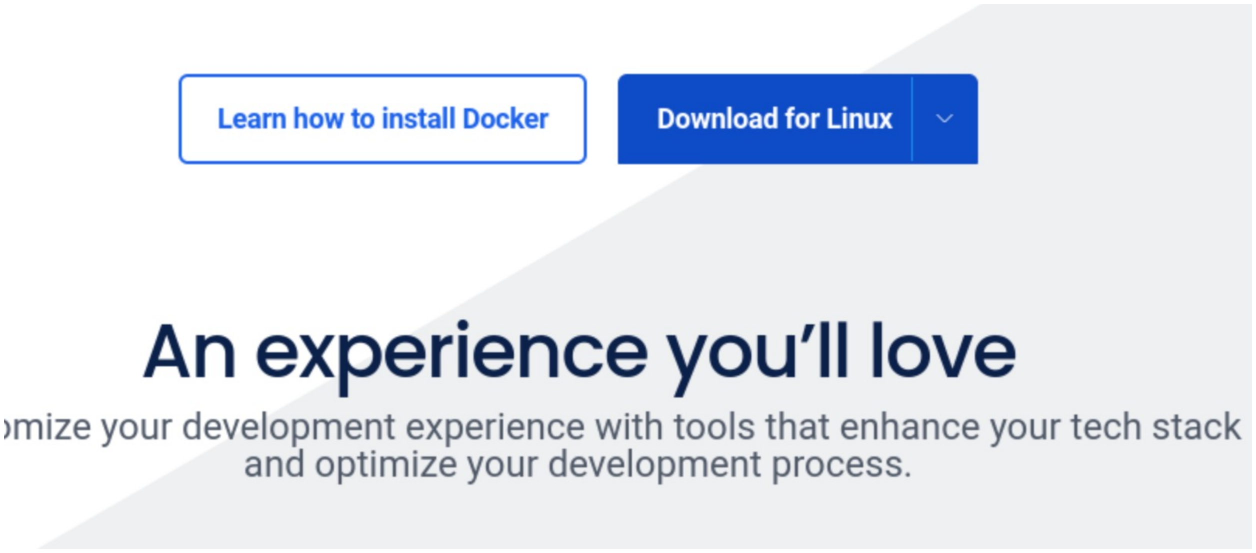
Build applications faster and more securely with Docker for developers

An experience you’ll love

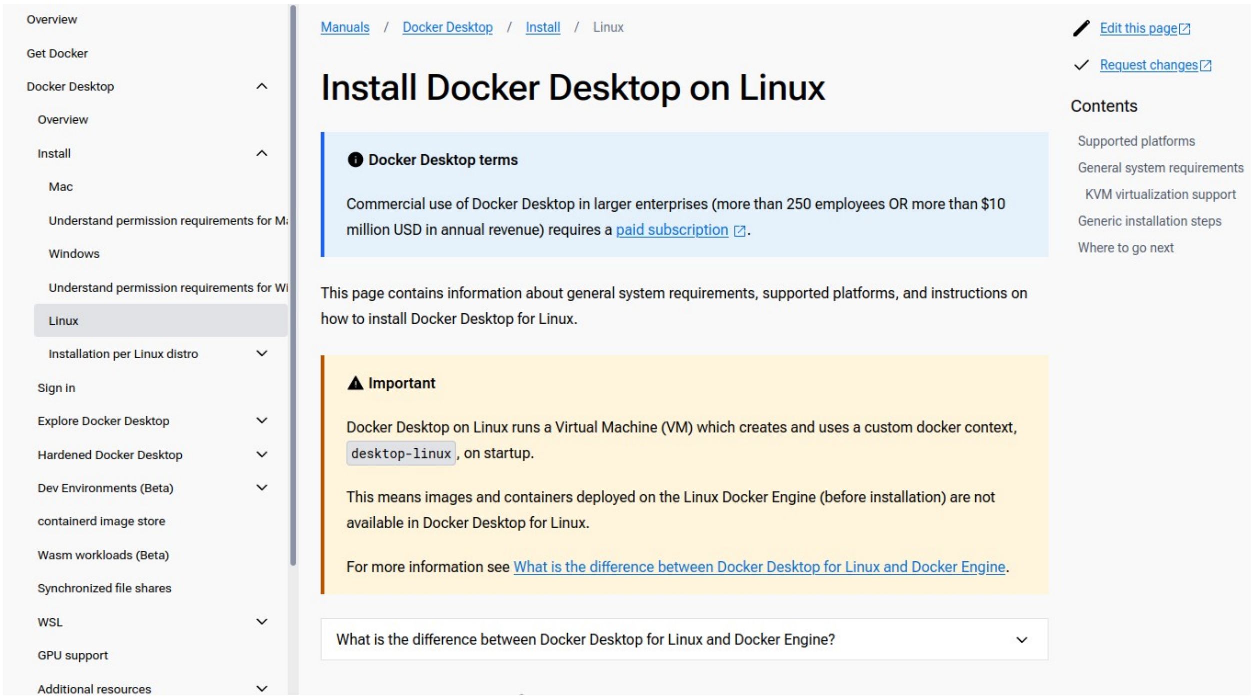
Customize your development experience with tools that enhance your tech stack and optimize your development process.

By clicking “Accept All Cookies”, you agree to the storing of cookies on your device to enhance site navigation, analyze site usage, and assist in our marketing efforts.

2. After that, you’ll see the option to choose your operating system. Download the Doc
   Desktop application for your operating system.As I was installing Docker Desktop or
   linux machine, I chose `“Download for LInux”` as follows:


Learn how to install Docker
Download for Linux
An experience you'll love
›mize your development experience with tools that enhance your tech stack and optimize your development process.

3. After clicking on the Download button, you'll be redirected to the next page



Install Docker Desktop on Linux

Docker Desktop terms

Commercial use of Docker Desktop in larger enterprises (more than 250 employees OR more than $10 million USD in annual revenue) requires a paid subscription.

This page contains information about general system requirements, supported platforms, and instructions on how to install Docker Desktop for Linux.

Important

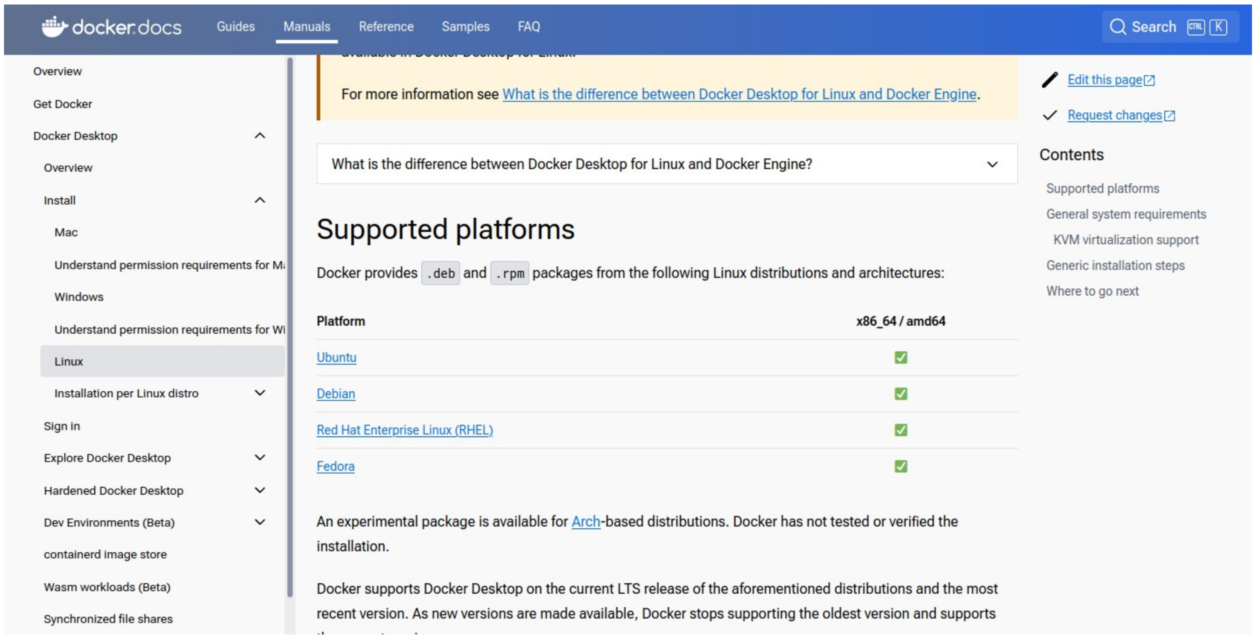
Docker Desktop on Linux runs a Virtual Machine (VM) which creates and uses a custom docker context, `desktop-linux`, on startup.

This means images and containers deployed on the Linux Docker Engine (before installation) are not available in Docker Desktop for Linux.

For more information see What is the difference between Docker Desktop for Linux and Docker Engine.

What is the difference between Docker Desktop for Linux and Docker Engine?

4. Scroll down and click on "Ubuntu"



For more information see What is the difference between Docker Desktop for Linux and Docker Engine.

What is the difference between Docker Desktop for Linux and Docker Engine?

Supported platforms

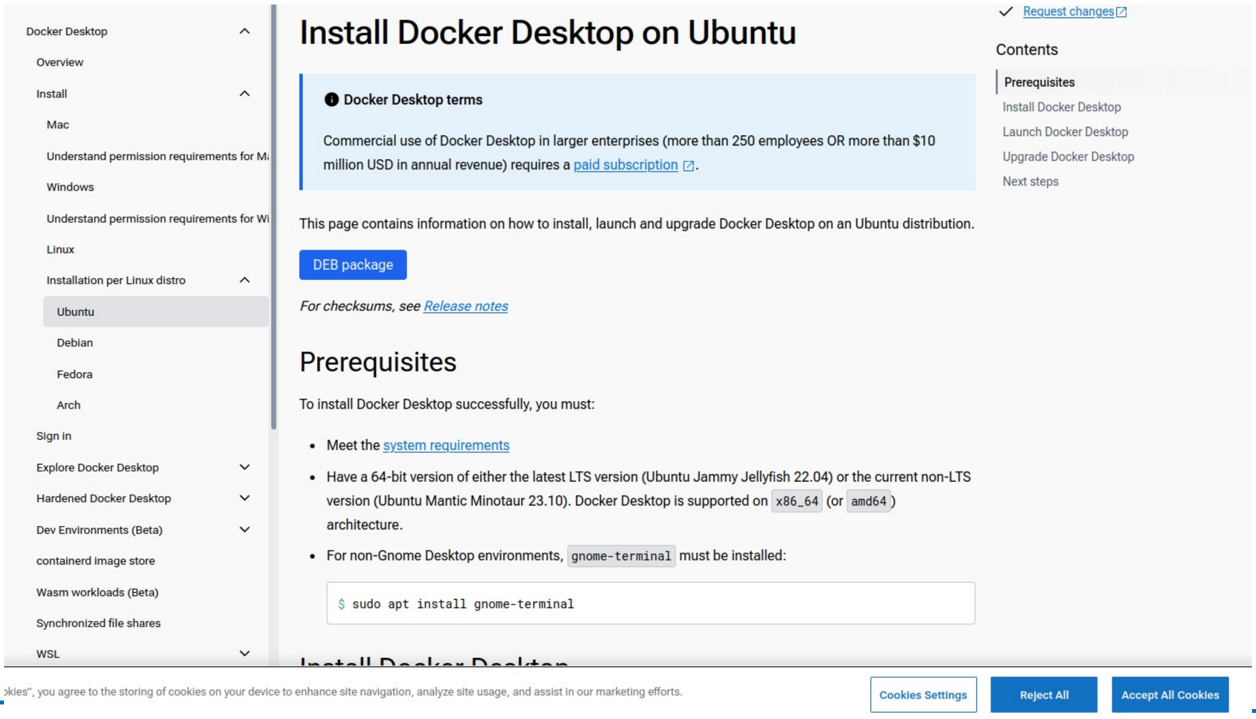
Docker provides `.deb` and `.rpm` packages from the following Linux distributions and architectures:

Platform	x86_64 / amd64
Ubuntu	✓
Debian	✓
Red Hat Enterprise Linux (RHEL)	✓
Fedora	✓

An experimental package is available for Arch-based distributions. Docker has not tested or verified the installation.

Docker supports Docker Desktop on the current LTS release of the aforementioned distributions and the most recent version. As new versions are made available, Docker stops supporting the oldest version and supports

5. You'll see the following page when you click on ubuntu, you need to click on "DEB Package". It will be successfully installed.



Install Docker Desktop on Ubuntu

Docker Desktop terms

Commercial use of Docker Desktop in larger enterprises (more than 250 employees OR more than $10 million USD in annual revenue) requires a paid subscription.

This page contains information on how to install, launch and upgrade Docker Desktop on an Ubuntu distribution.

For checksums, see Release notes

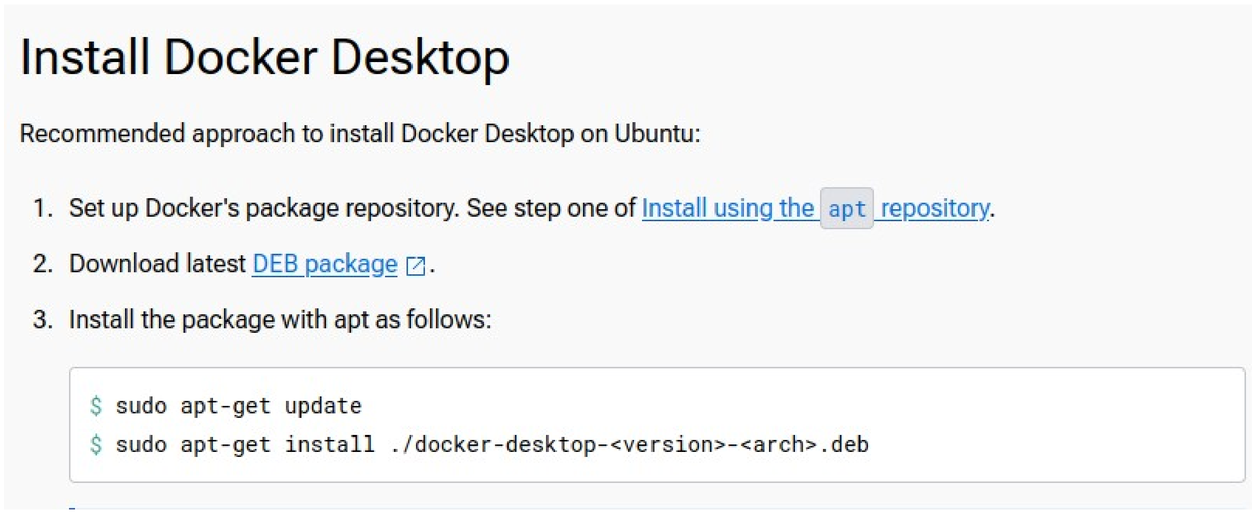
Prerequisites

To install Docker Desktop successfully, you must:

- Meet the system requirements
- Have a 64-bit version of either the latest LTS version (Ubuntu Jammy Jellyfish 22.04) or the current non-LTS version (Ubuntu Mantic Minotaur 23.10). Docker Desktop is supported on `x86_64` (or `amd64`) architecture.
- For non-Gnome Desktop environments, `gnome-terminal` must be installed:

```
$ sudo apt install gnome-terminal
```

okies", you agree to the storing of cookies on your device to enhance site navigation, analyze site usage, and assist in our marketing efforts.

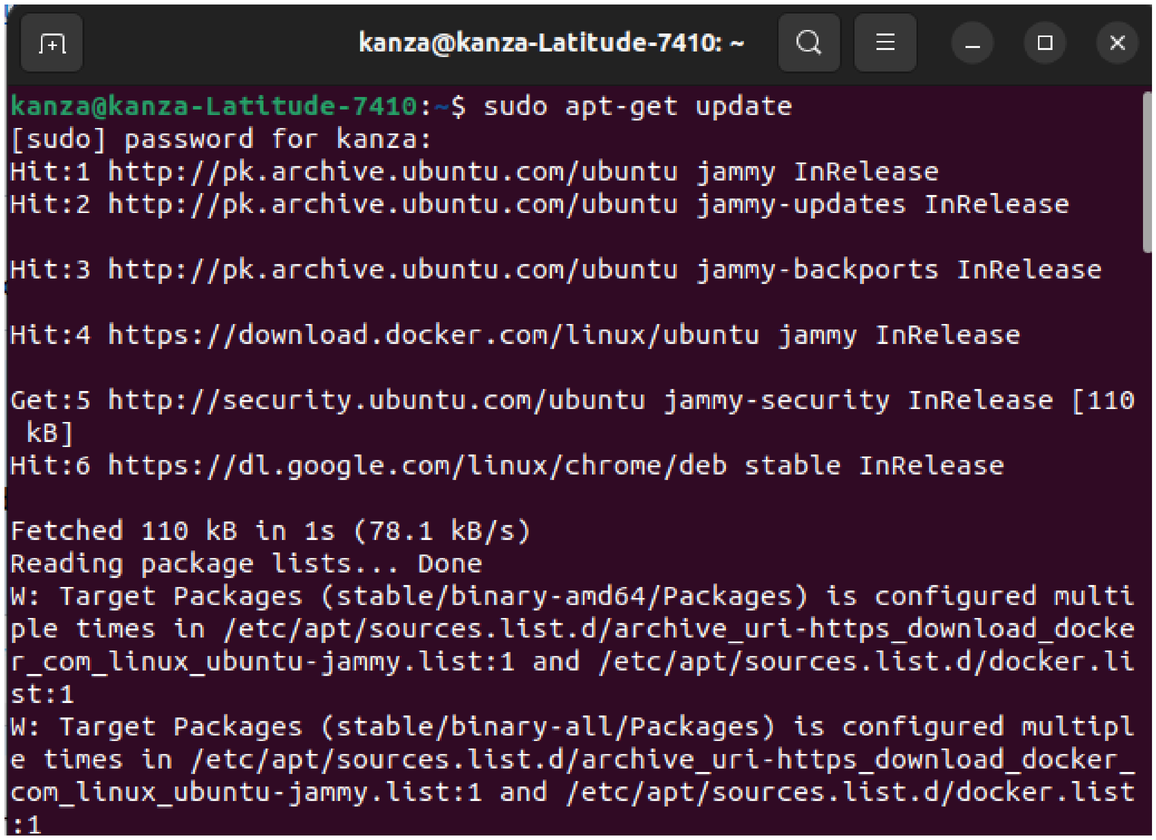

Install Docker Desktop

Recommended approach to install Docker Desktop on Ubuntu:

1. Set up Docker's package repository. See step one of Install using the `apt` repository.
2. Download latest DEB package.
3. Install the package with apt as follows:

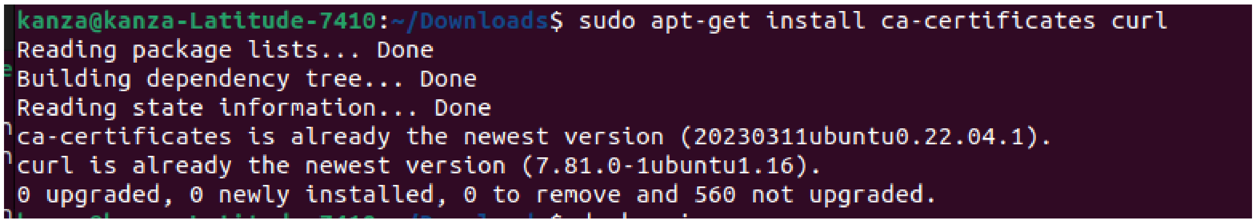
```
$ sudo apt-get update
$ sudo apt-get install ./docker-desktop-<version>-<arch>.deb
```

6. Scroll down and you’ll see the following page and these commands will appear
7. Run the first command i.e. `sudo apt-get update`



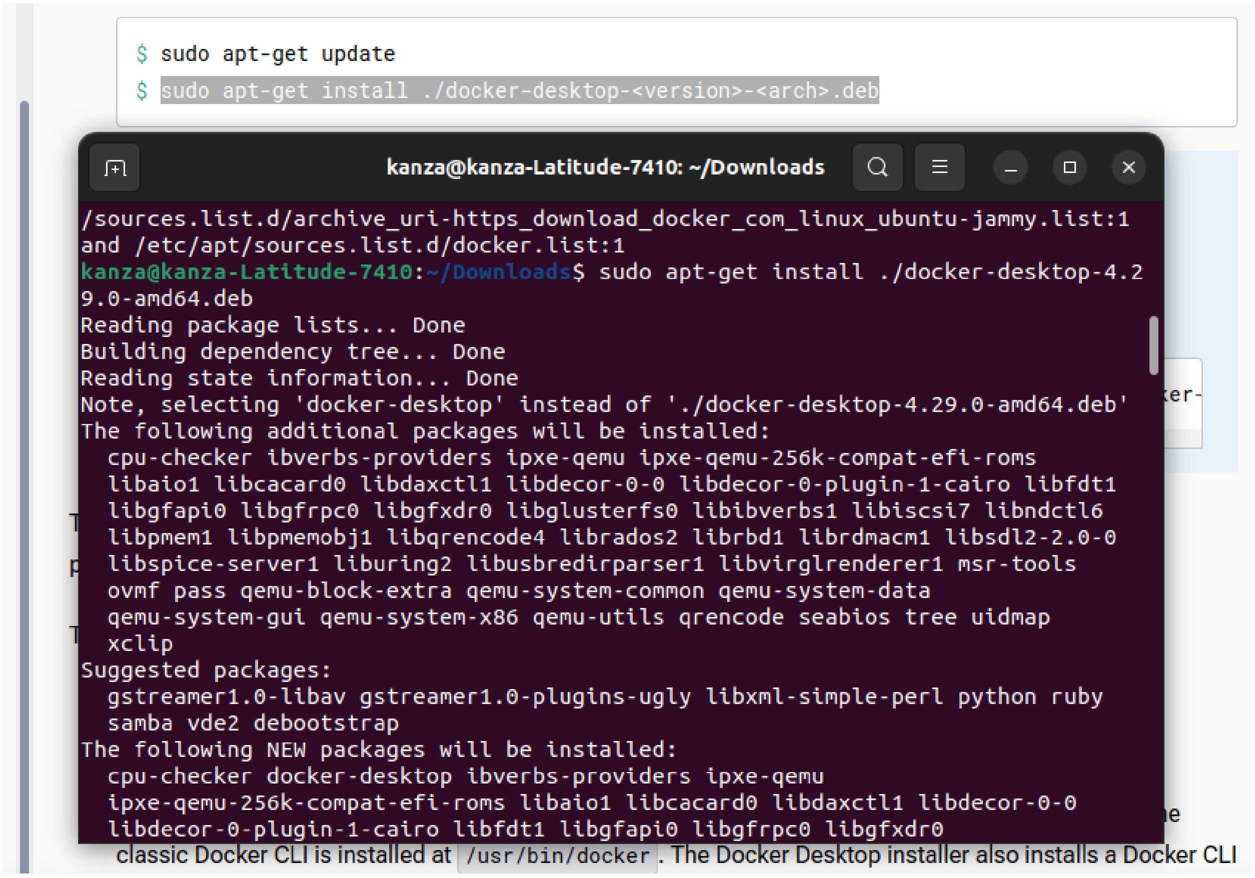
```
kanza@kanza-Latitude-7410:~$ sudo apt-get update
[sudo] password for kanza:
Hit:1 http://pk.archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 http://pk.archive.ubuntu.com/ubuntu jammy-updates InRelease

Hit:3 http://pk.archive.ubuntu.com/ubuntu jammy-backports InRelease

Hit:4 https://download.docker.com/linux/ubuntu jammy InRelease

Get:5 http://security.ubuntu.com/ubuntu jammy-security InRelease [110
 kB]
Hit:6 https://dl.google.com/linux/chrome/deb stable InRelease

Fetched 110 kB in 1s (78.1 kB/s)
Reading package lists... Done
W: Target Packages (stable/binary-amd64/Packages) is configured multi
ple times in /etc/apt/sources.list.d/archive_uri-https_download_docke
r_com_linux_ubuntu-jammy.list:1 and /etc/apt/sources.list.d/docker.li
st:1
W: Target Packages (stable/binary-all/Packages) is configured multipl
e times in /etc/apt/sources.list.d/archive_uri-https_download_docker_
com_linux_ubuntu-jammy.list:1 and /etc/apt/sources.list.d/docker.list
:1
```

8. Now, you need to install the prerequisites by running the following command:



```
kanza@kanza-Latitude-7410:~/Downloads$ sudo apt-get install ca-certificates curl
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
ca-certificates is already the newest version (20230311ubuntu0.22.04.1).
curl is already the newest version (7.81.0-1ubuntu1.16).
0 upgraded, 0 newly installed, 0 to remove and 560 not upgraded.
```

9. Now , we are going to install docker by using the command as shown below. You’ll h
   to make sure to add the correct name of the .deb file that you downloaded earlier.

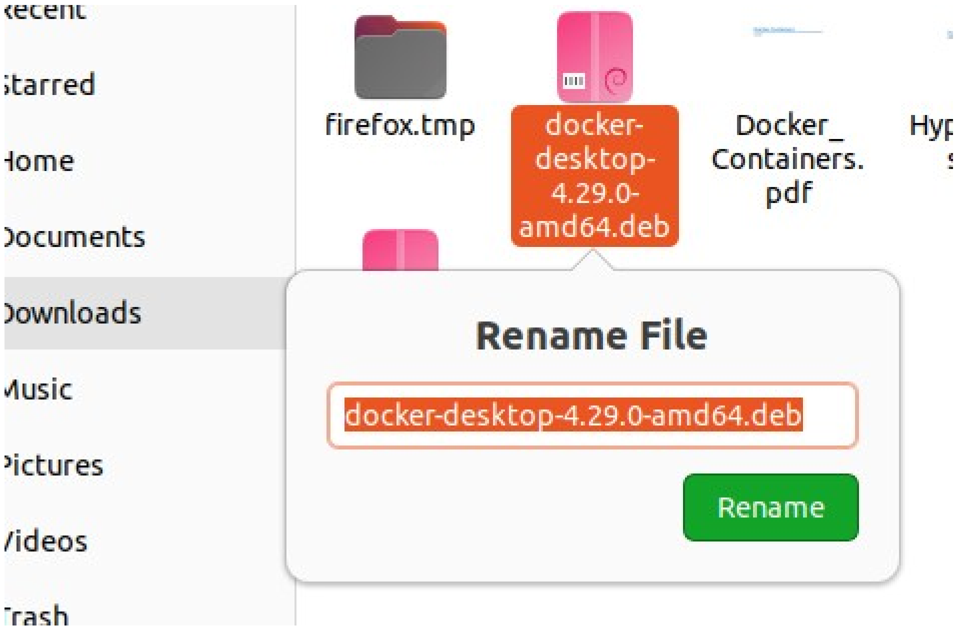


```
$ sudo apt-get update
$ sudo apt-get install ./docker-desktop-<version>-<arch>.deb
```

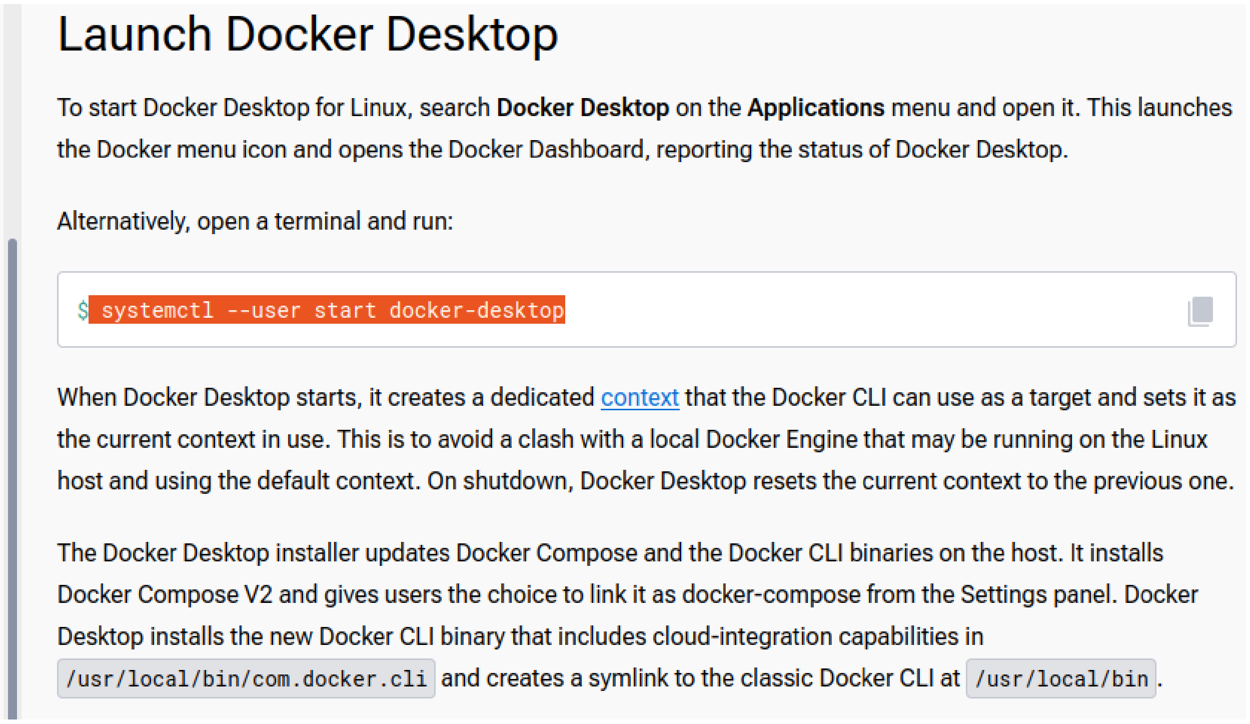
kanza@kanza-Latitude-7410: ~/Downloads

```
/sources.list.d/archive_uri-https_download_docker_com_linux_ubuntu-jammy.list:1
and /etc/apt/sources.list.d/docker.list:1
kanza@kanza-Latitude-7410:~/Downloads$ sudo apt-get install ./docker-desktop-4.2
9.0-amd64.deb
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Note, selecting 'docker-desktop' instead of './docker-desktop-4.29.0-amd64.deb'
The following additional packages will be installed:
  cpu-checker ibverbs-providers ipxe-qemu ipxe-qemu-256k-compat-efi-roms
  libaio1 libcacard0 libdaxctl1 libdecor-0-0 libdecor-0-plugin-1-cairo libfdt1
  libgfapi0 libgfrpc0 libgfxdr0 libglusterfs0 libibverbs1 libiscsi7 libndctl6
  libpmem1 libpmemobj1 libqrencode4 librados2 librbd1 librdmacm1 libsdl2-2.0-0
  libspice-server1 liburing2 libusbredirparser1 libvirglrenderer1 msr-tools
  ovmf pass qemu-block-extra qemu-system-common qemu-system-data
  qemu-system-gui qemu-system-x86 qemu-utils qrencode seabios tree uidmap
  xclip
Suggested packages:
  gstreamer1.0-libav gstreamer1.0-plugins-ugly libxml-simple-perl python ruby
  samba vde2 debootstrap
The following NEW packages will be installed:
  cpu-checker docker-desktop ibverbs-providers ipxe-qemu
  ipxe-qemu-256k-compat-efi-roms libaio1 libcacard0 libdaxctl1 libdecor-0-0
  libdecor-0-plugin-1-cairo libfdt1 libgfapi0 libgfrpc0 libgfxdr0
```

classic Docker CLI is installed at `/usr/bin/docker`. The Docker Desktop installer also installs a Docker CLI



10.Once installed, launch Docker Desktop



Launch Docker Desktop

To start Docker Desktop for Linux, search **Docker Desktop** on the **Applications** menu and open it. This launches the Docker menu icon and opens the Docker Dashboard, reporting the status of Docker Desktop.

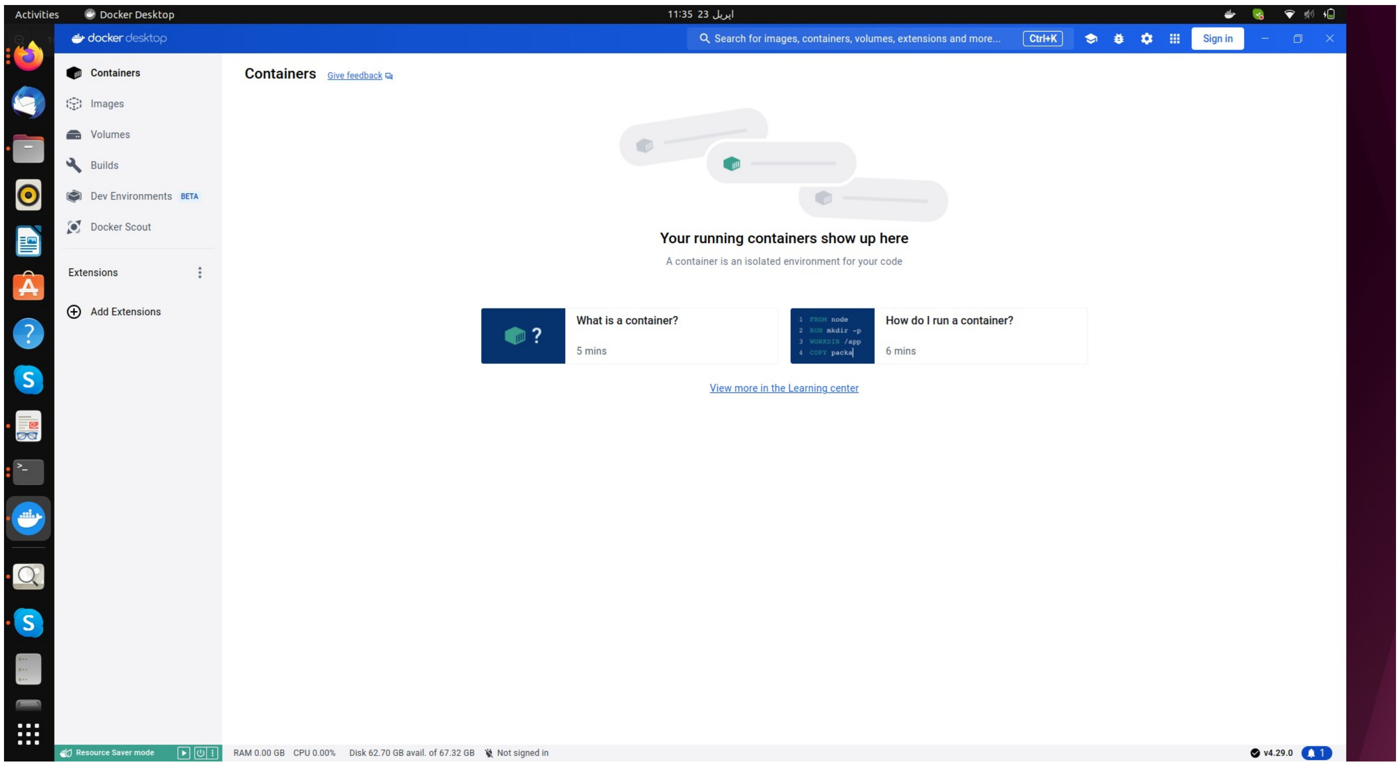
Alternatively, open a terminal and run:

```
$ systemctl --user start docker-desktop
```

When Docker Desktop starts, it creates a dedicated context that the Docker CLI can use as a target and sets it as the current context in use. This is to avoid a clash with a local Docker Engine that may be running on the Linux host and using the default context. On shutdown, Docker Desktop resets the current context to the previous one.

The Docker Desktop installer updates Docker Compose and the Docker CLI binaries on the host. It installs Docker Compose V2 and gives users the choice to link it as docker-compose from the Settings panel. Docker Desktop installs the new Docker CLI binary that includes cloud-integration capabilities in `/usr/local/bin/com.docker.cli` and creates a symlink to the classic Docker CLI at `/usr/local/bin`.

11.Sign in with your Docker Hub account credentials or use it without signing in



Your running containers show up here

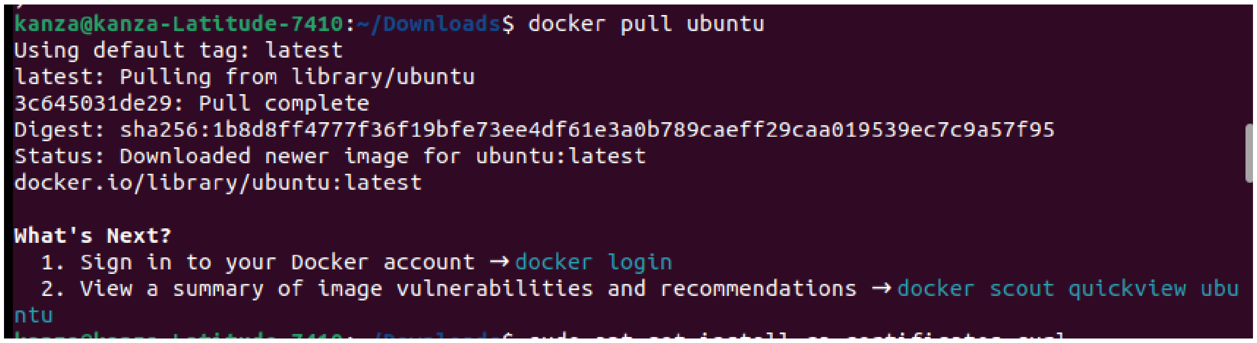
A container is an isolated environment for your code

Pulling a Ubuntu Image from Docker Hub

Pulling the Ubuntu Image

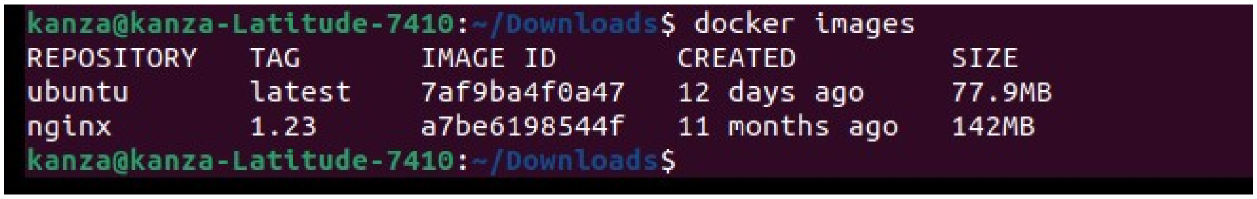
- Open a terminal or command prompt on your system.
- Use the following command to pull the Ubuntu image from Docker Hub:

```
docker pull ubuntu
```

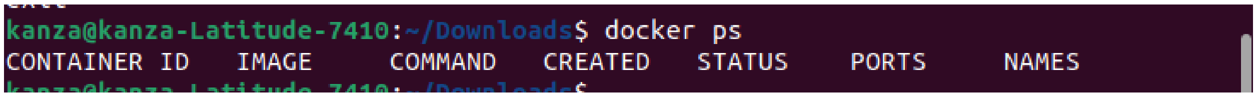


```
kanza@kanza-Latitude-7410:~/Downloads$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
3c645031de29: Pull complete
Digest: sha256:1b8d8ff4777f36f19bfe73ee4df61e3a0b789caeff29caa019539ec7c9a57f95
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest

What's Next?
  1. Sign in to your Docker account → docker login
  2. View a summary of image vulnerabilities and recommendations → docker scout quickview ubu
ntu
```

- This command will download the latest version of the Ubuntu image from the Docke Hub repository to your local machine.



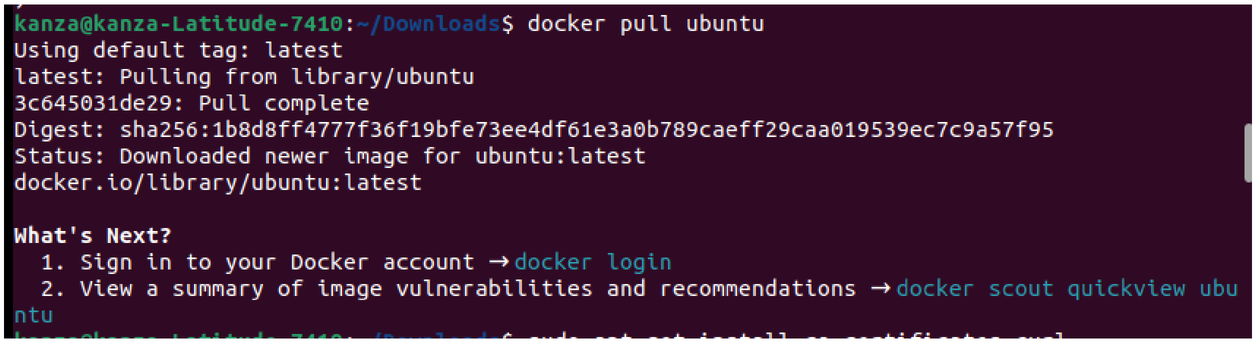
```
kanza@kanza-Latitude-7410:~/Downloads$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
ubuntu        latest    7af9ba4f0a47   12 days ago      77.9MB
nginx         1.23      a7be6198544f   11 months ago    142MB
kanza@kanza-Latitude-7410:~/Downloads$
```



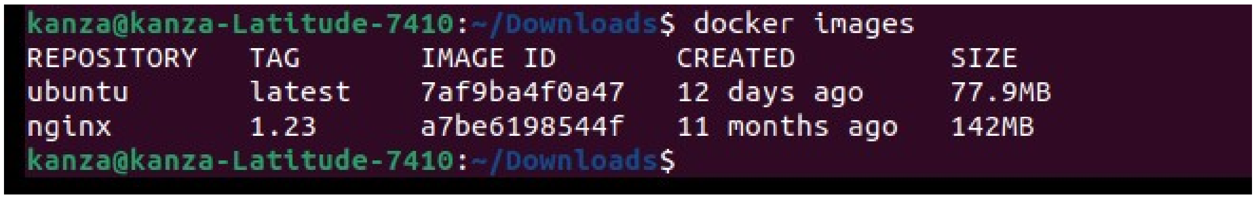
```
kanza@kanza-Latitude-7410:~/Downloads$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

Docker Images

The following command provides the list of all the docker images available on the system
also shows the repository, tag, image id, size and when it was created.



```
kanza@kanza-Latitude-7410:~/Downloads$ docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
3c645031de29: Pull complete
Digest: sha256:1b8d8ff4777f36f19bfe73ee4df61e3a0b789caeff29caa019539ec7c9a57f95
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest

What's Next?
  1. Sign in to your Docker account → docker login
  2. View a summary of image vulnerabilities and recommendations → docker scout quickview ubu
ntu
```



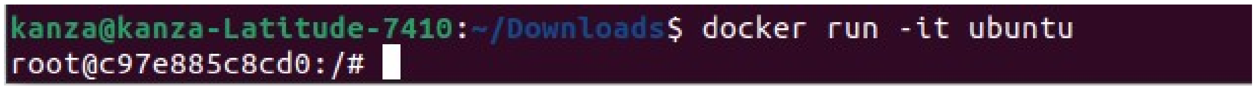
```
kanza@kanza-Latitude-7410:~/Downloads$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
ubuntu        latest    7af9ba4f0a47   12 days ago     77.9MB
nginx         1.23      a7be6198544f   11 months ago   142MB
kanza@kanza-Latitude-7410:~/Downloads$
```

Ubuntu Container in Docker

Starting a Docker Container

- After pulling the Ubuntu image, you can start a container using the following comma

```
docker run -it ubuntu
```



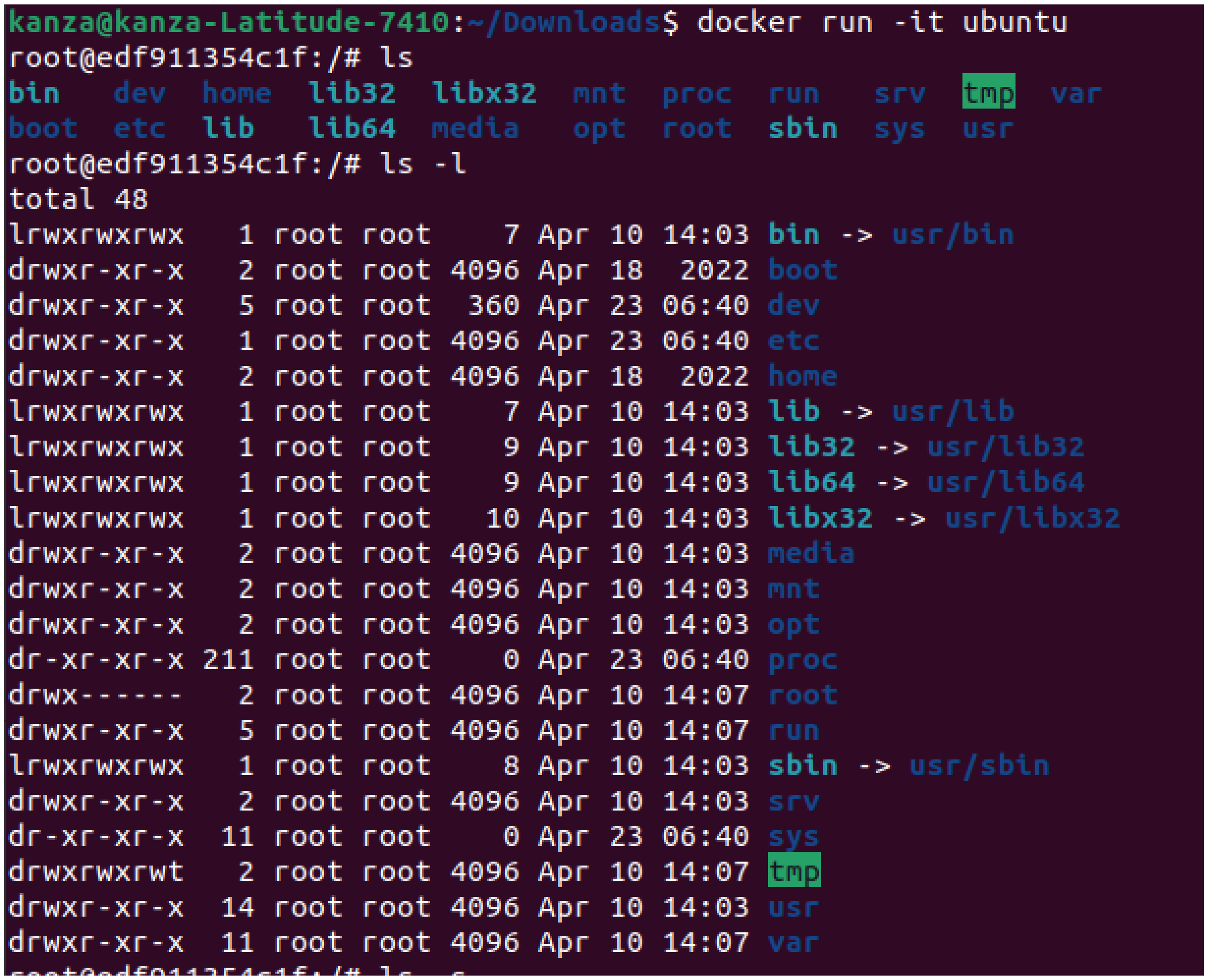
```
kanza@kanza-Latitude-7410:~/Downloads$ docker run -it ubuntu
root@c97e885c8cd0:/# 
```

Running Commands in a Docker Container

This command will start a new Ubuntu container in interactive mode, allowing you to run commands within the container.

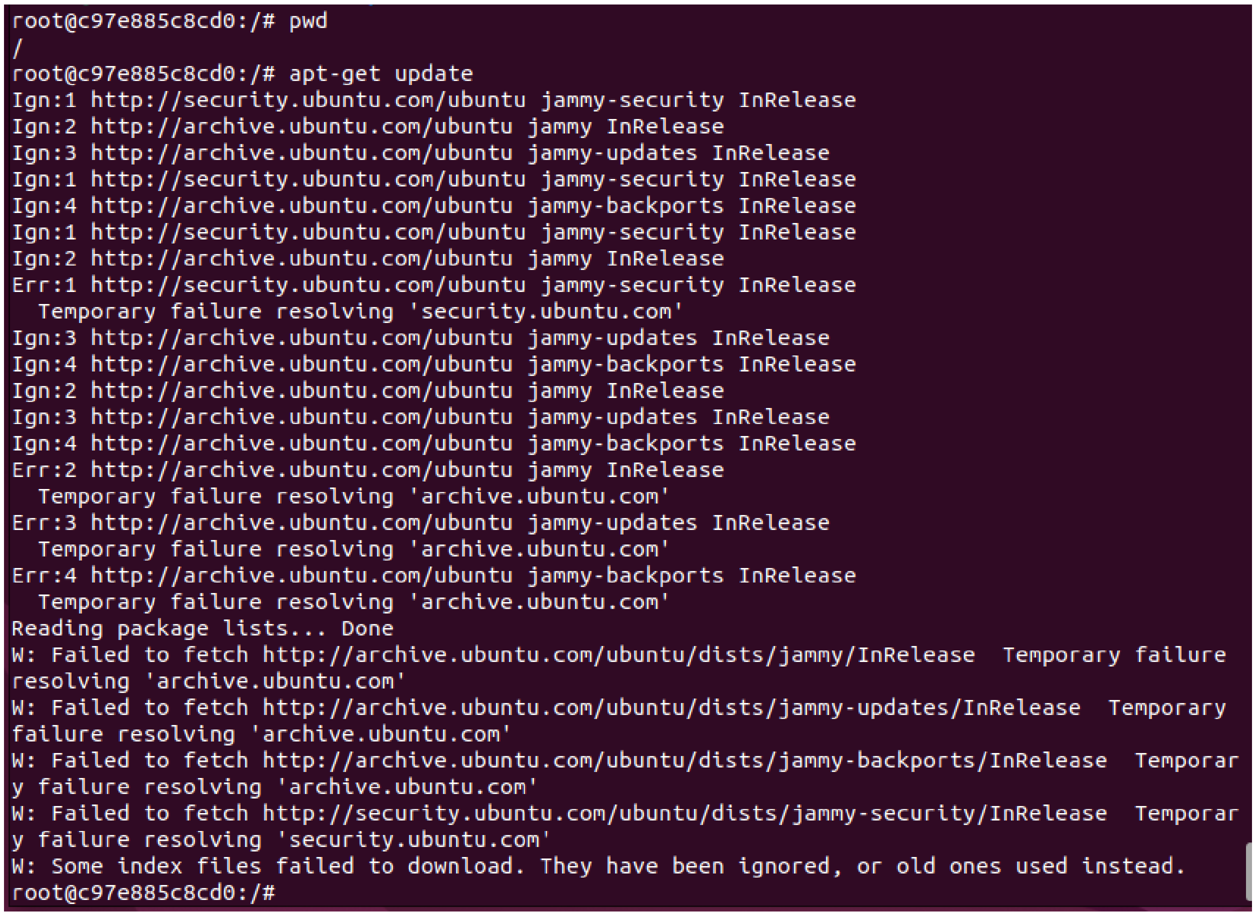
Running Bash Commands

- Once inside the container, you can run various Bash commands to interact with the Ubuntu environment. For example:
- `‘ls’`: List files and directories in the current directory.
- `‘ls -l’`: Lists detailed information about files and directories within the container's filesystem, including permissions, owner, group, size, modification date, and filenam



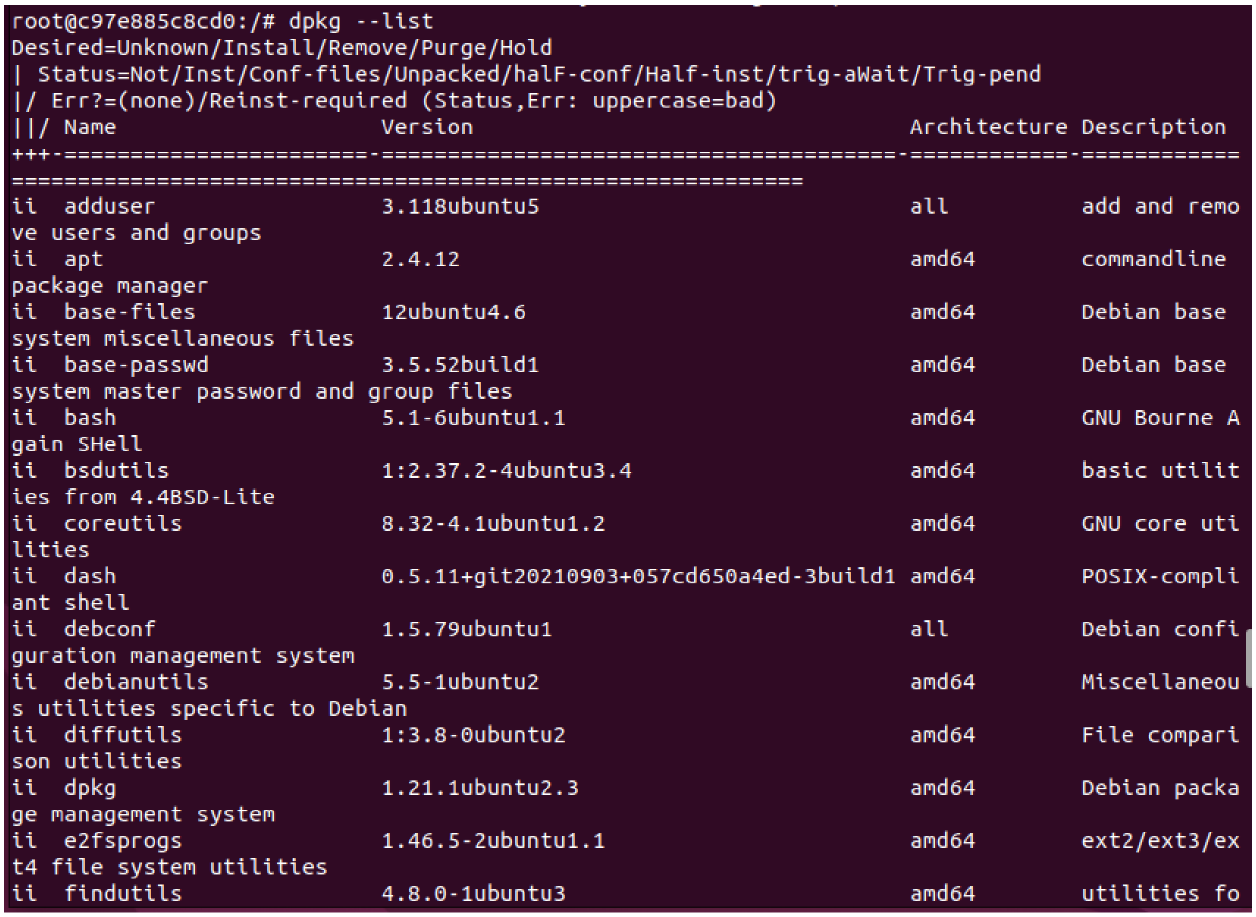
```
kanza@kanza-Latitude-7410:~/Downloads$ docker run -it ubuntu
root@edf911354c1f:/# ls
bin   dev  home  lib32  libx32  mnt    proc  run   srv  tmp  var
boot  etc  lib   lib64  media   opt    root  sbin  sys  usr
root@edf911354c1f:/# ls -l
total 48
lrwxrwxrwx   1 root root    7 Apr 10 14:03 bin -> usr/bin
drwxr-xr-x   2 root root 4096 Apr 18  2022 boot
drwxr-xr-x   5 root root  360 Apr 23 06:40 dev
drwxr-xr-x   1 root root 4096 Apr 23 06:40 etc
drwxr-xr-x   2 root root 4096 Apr 18  2022 home
lrwxrwxrwx   1 root root    7 Apr 10 14:03 lib -> usr/lib
lrwxrwxrwx   1 root root    9 Apr 10 14:03 lib32 -> usr/lib32
lrwxrwxrwx   1 root root    9 Apr 10 14:03 lib64 -> usr/lib64
lrwxrwxrwx   1 root root   10 Apr 10 14:03 libx32 -> usr/libx32
drwxr-xr-x   2 root root 4096 Apr 10 14:03 media
drwxr-xr-x   2 root root 4096 Apr 10 14:03 mnt
drwxr-xr-x   2 root root 4096 Apr 10 14:03 opt
dr-xr-xr-x 211 root root    0 Apr 23 06:40 proc
drwx------   2 root root 4096 Apr 10 14:07 root
drwxr-xr-x   5 root root 4096 Apr 10 14:07 run
lrwxrwxrwx   1 root root    8 Apr 10 14:03 sbin -> usr/sbin
drwxr-xr-x   2 root root 4096 Apr 10 14:03 srv
dr-xr-xr-x  11 root root    0 Apr 23 06:40 sys
drwxrwxrwt   2 root root 4096 Apr 10 14:07 tmp
drwxr-xr-x  14 root root 4096 Apr 10 14:03 usr
drwxr-xr-x  11 root root 4096 Apr 10 14:07 var
```

- `’pwd’`: Print the current working directory.
- `‘apt-get update’`: Update package lists within the container.



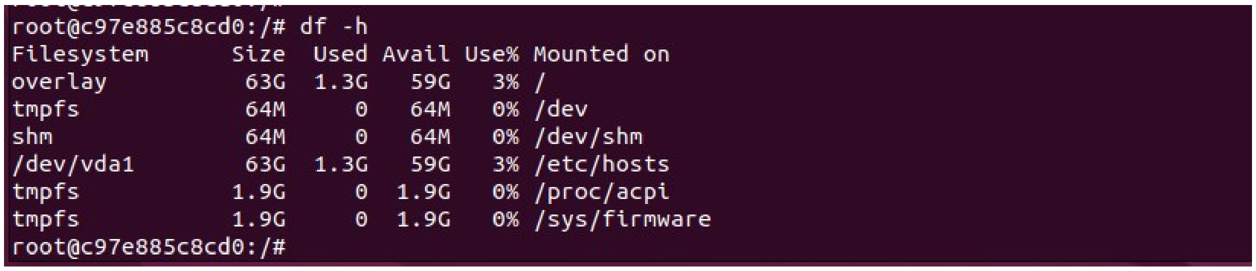
```
root@c97e885c8cd0:/# pwd
/
root@c97e885c8cd0:/# apt-get update
Ign:1 http://security.ubuntu.com/ubuntu jammy-security InRelease
Ign:2 http://archive.ubuntu.com/ubuntu jammy InRelease
Ign:3 http://archive.ubuntu.com/ubuntu jammy-updates InRelease
Ign:1 http://security.ubuntu.com/ubuntu jammy-security InRelease
Ign:4 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Ign:1 http://security.ubuntu.com/ubuntu jammy-security InRelease
Ign:2 http://archive.ubuntu.com/ubuntu jammy InRelease
Err:1 http://security.ubuntu.com/ubuntu jammy-security InRelease
  Temporary failure resolving 'security.ubuntu.com'
Ign:3 http://archive.ubuntu.com/ubuntu jammy-updates InRelease
Ign:4 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Ign:2 http://archive.ubuntu.com/ubuntu jammy InRelease
Ign:3 http://archive.ubuntu.com/ubuntu jammy-updates InRelease
Ign:4 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Err:2 http://archive.ubuntu.com/ubuntu jammy InRelease
  Temporary failure resolving 'archive.ubuntu.com'
Err:3 http://archive.ubuntu.com/ubuntu jammy-updates InRelease
  Temporary failure resolving 'archive.ubuntu.com'
Err:4 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
  Temporary failure resolving 'archive.ubuntu.com'
Reading package lists... Done
W: Failed to fetch http://archive.ubuntu.com/ubuntu/dists/jammy/InRelease  Temporary failure
resolving 'archive.ubuntu.com'
W: Failed to fetch http://archive.ubuntu.com/ubuntu/dists/jammy-updates/InRelease  Temporary
failure resolving 'archive.ubuntu.com'
W: Failed to fetch http://archive.ubuntu.com/ubuntu/dists/jammy-backports/InRelease  Temporar
y failure resolving 'archive.ubuntu.com'
W: Failed to fetch http://security.ubuntu.com/ubuntu/dists/jammy-security/InRelease  Temporar
y failure resolving 'security.ubuntu.com'
W: Some index files failed to download. They have been ignored, or old ones used instead.
root@c97e885c8cd0:/#
```

- `‘apt-get install <package_name>’`: Install packages inside the container.
- `‘dpkg --list’:` List Installed Packages



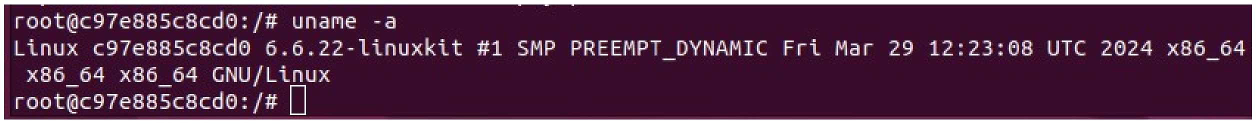
```
root@c97e885c8cd0:/# dpkg --list
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name                    Version                                 Architecture Description
+++-=======================-=======================================-============-============
============================================================
ii  adduser                 3.118ubuntu5                            all          add and remo
ve users and groups
ii  apt                     2.4.12                                  amd64        commandline
package manager
ii  base-files              12ubuntu4.6                             amd64        Debian base
system miscellaneous files
ii  base-passwd             3.5.52build1                            amd64        Debian base
system master password and group files
ii  bash                    5.1-6ubuntu1.1                          amd64        GNU Bourne A
gain SHell
ii  bsdutils                1:2.37.2-4ubuntu3.4                     amd64        basic utilit
ies from 4.4BSD-Lite
ii  coreutils               8.32-4.1ubuntu1.2                       amd64        GNU core uti
lities
ii  dash                    0.5.11+git20210903+057cd650a4ed-3build1 amd64        POSIX-compli
ant shell
ii  debconf                 1.5.79ubuntu1                           all          Debian confi
guration management system
ii  debianutils             5.5-1ubuntu2                            amd64        Miscellaneou
s utilities specific to Debian
ii  diffutils               1:3.8-0ubuntu2                          amd64        File compari
son utilities
ii  dpkg                    1.21.1ubuntu2.3                         amd64        Debian packa
ge management system
ii  e2fsprogs               1.46.5-2ubuntu1.1                       amd64        ext2/ext3/ex
t4 file system utilities
ii  findutils               4.8.0-1ubuntu3                          amd64        utilities fo
```

- `‘df -h’:` Check Disk Usage



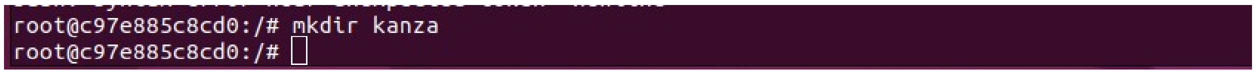
```
root@c97e885c8cd0:/# df -h
Filesystem      Size  Used Avail Use% Mounted on
overlay          63G  1.3G   59G   3% /
tmpfs            64M     0   64M   0% /dev
shm              64M     0   64M   0% /dev/shm
/dev/vda1        63G  1.3G   59G   3% /etc/hosts
tmpfs           1.9G     0  1.9G   0% /proc/acpi
tmpfs           1.9G     0  1.9G   0% /sys/firmware
root@c97e885c8cd0:/#
```

- ‘uname -a’: Check System Information



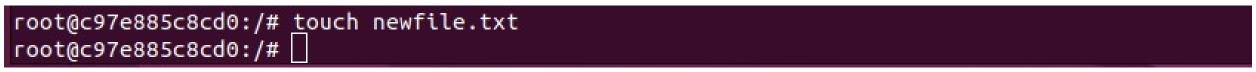
```
root@c97e885c8cd0:/# uname -a
Linux c97e885c8cd0 6.6.22-linuxkit #1 SMP PREEMPT_DYNAMIC Fri Mar 29 12:23:08 UTC 2024 x86_64
 x86_64 x86_64 GNU/Linux
root@c97e885c8cd0:/# 
```

- ‘cd <directory_path>’: Change Directory
- ‘mkdir <directory_name>’: creates a new directory



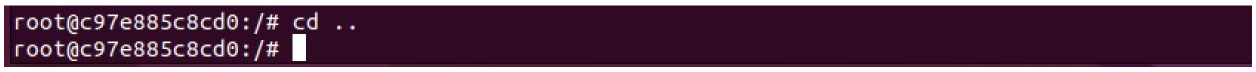
```
root@c97e885c8cd0:/# mkdir kanza
root@c97e885c8cd0:/# 
```

- ‘touch <filename>’: creates a new file in the current directory



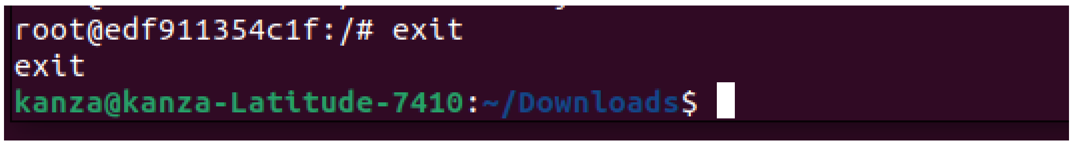
```
root@c97e885c8cd0:/# touch newfile.txt
root@c97e885c8cd0:/# 
```

- ‘cd ..’: moves to the previous directory



```
root@c97e885c8cd0:/# cd ..
root@c97e885c8cd0:/# 
```

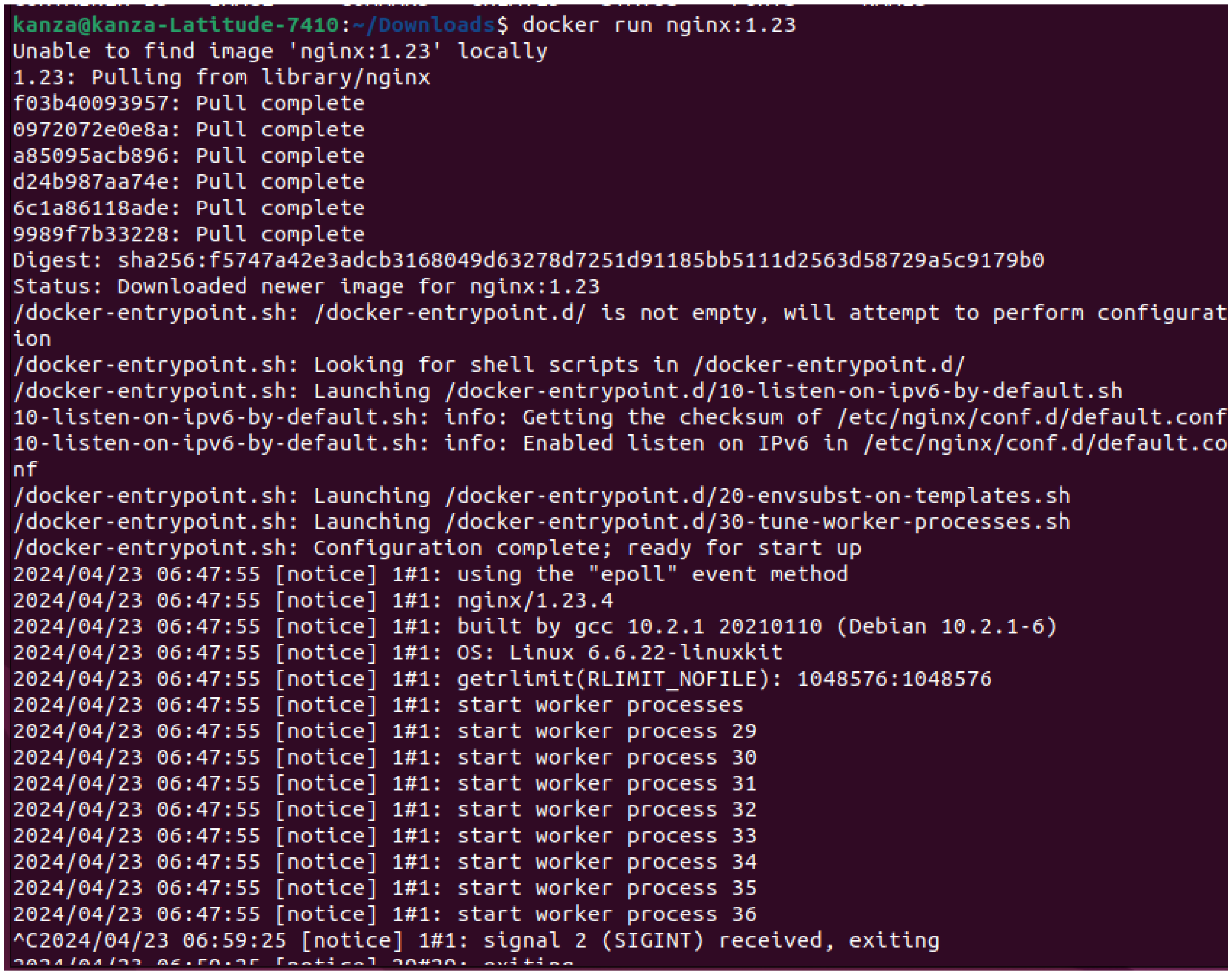
- ‘exit’: exits the docker container



```
root@edf911354c1f:/# exit
exit
kanza@kanza-Latitude-7410:~/Downloads$ 
```

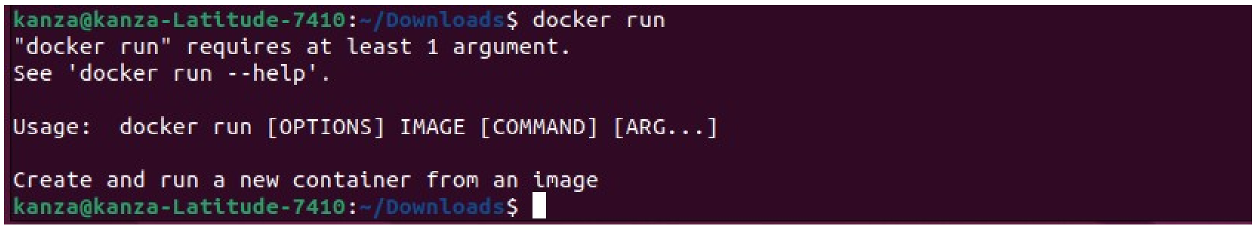
Running Docker image with a tag

While running the docker image, the following command is used and a tag is given at the
shown below i.e. nginx:1.23.



```
kanza@kanza-Latitude-7410:~/Downloads$ docker run nginx:1.23
Unable to find image 'nginx:1.23' locally
1.23: Pulling from library/nginx
f03b40093957: Pull complete
0972072e0e8a: Pull complete
a85095acb896: Pull complete
d24b987aa74e: Pull complete
6c1a86118ade: Pull complete
9989f7b33228: Pull complete
Digest: sha256:f5747a42e3adcb3168049d63278d7251d91185bb5111d2563d58729a5c9179b0
Status: Downloaded newer image for nginx:1.23
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configurat
ion
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.co
nf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/04/23 06:47:55 [notice] 1#1: using the "epoll" event method
2024/04/23 06:47:55 [notice] 1#1: nginx/1.23.4
2024/04/23 06:47:55 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian 10.2.1-6)
2024/04/23 06:47:55 [notice] 1#1: OS: Linux 6.6.22-linuxkit
2024/04/23 06:47:55 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/04/23 06:47:55 [notice] 1#1: start worker processes
2024/04/23 06:47:55 [notice] 1#1: start worker process 29
2024/04/23 06:47:55 [notice] 1#1: start worker process 30
2024/04/23 06:47:55 [notice] 1#1: start worker process 31
2024/04/23 06:47:55 [notice] 1#1: start worker process 32
2024/04/23 06:47:55 [notice] 1#1: start worker process 33
2024/04/23 06:47:55 [notice] 1#1: start worker process 34
2024/04/23 06:47:55 [notice] 1#1: start worker process 35
2024/04/23 06:47:55 [notice] 1#1: start worker process 36
^C2024/04/23 06:59:25 [notice] 1#1: signal 2 (SIGINT) received, exiting
```

If I run it without giving the tag, it throws an error

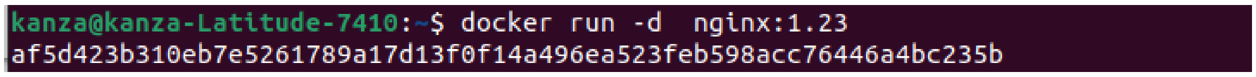


```
kanza@kanza-Latitude-7410:~/Downloads$ docker run
"docker run" requires at least 1 argument.
See 'docker run --help'.

Usage:  docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

Create and run a new container from an image
kanza@kanza-Latitude-7410:~/Downloads$ 
```

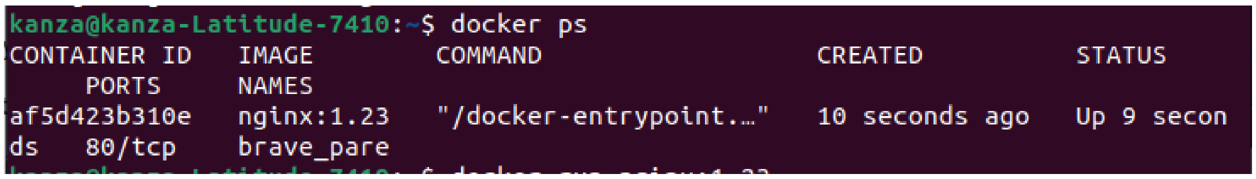
Docker Image in Detached Mode

This command will start a new container based on the nginx:1.23 image

Command:`docker run -d IMAGE_NAME`



```
kanza@kanza-Latitude-7410:~$ docker run -d  nginx:1.23
af5d423b310eb7e5261789a17d13f0f14a496ea523feb598acc76446a4bc235b
```



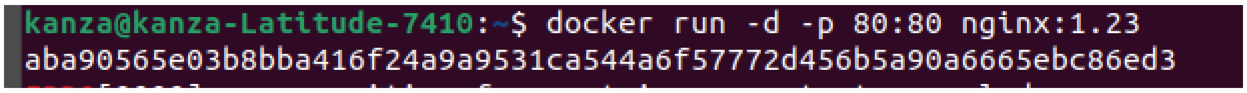
```
kanza@kanza-Latitude-7410:~$ docker ps
CONTAINER ID   IMAGE        COMMAND                  CREATED          STATUS
      PORTS     NAMES
af5d423b310e   nginx:1.23   "/docker-entrypoint.…"   10 seconds ago   Up 9 secon
ds    80/tcp    brave_pare
```

Docker Image with Port Mapping

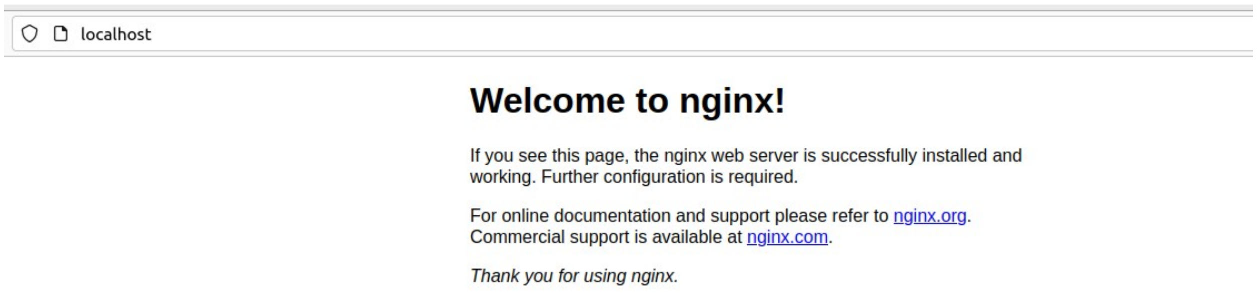
command:

```
docker run -d -p 80:80 nginx:1.25
```

This command will start a new container and will map port 80 of the host and container.



```
kanza@kanza-Latitude-7410:~$ docker run -d -p 80:80 nginx:1.23
aba90565e03b8bba416f24a9a9531ca544a6f57772d456b5a90a6665ebc86ed3
```



localhost

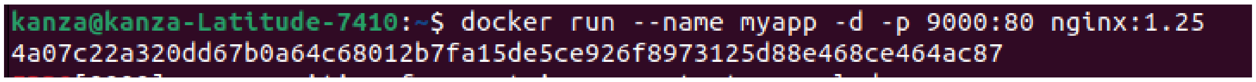
Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

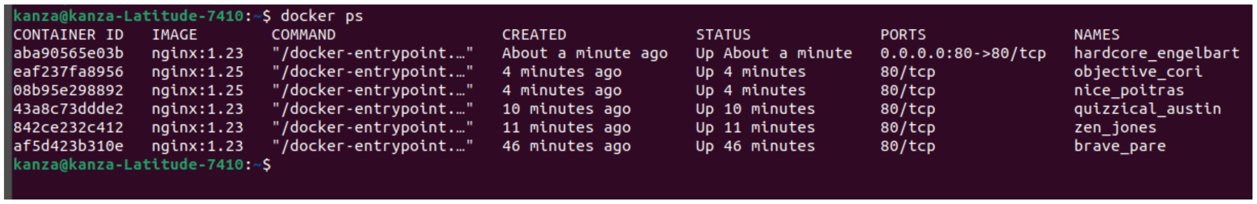
For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Containers running at present

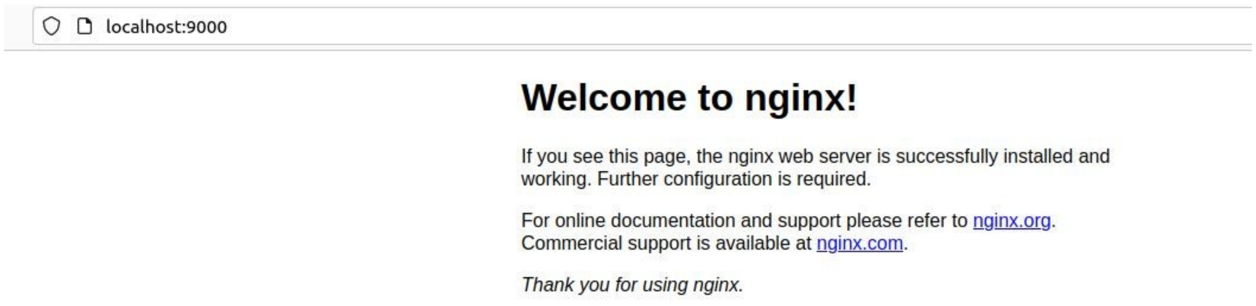


```
kanza@kanza-Latitude-7410:~$ docker run --name myapp -d -p 9000:80 nginx:1.25
4a07c22a320dd67b0a64c68012b7fa15de5ce926f8973125d88e468ce464ac87
```



```
kanza@kanza-Latitude-7410:~$ docker ps
CONTAINER ID   IMAGE        COMMAND                  CREATED              STATUS              PORTS                NAMES
aba90565e03b   nginx:1.23   "/docker-entrypoint.…"   About a minute ago   Up About a minute   0.0.0.0:80->80/tcp   hardcore_engelbart
eaf237fa8956   nginx:1.25   "/docker-entrypoint.…"   4 minutes ago        Up 4 minutes        80/tcp               objective_cori
08b95e298892   nginx:1.25   "/docker-entrypoint.…"   4 minutes ago        Up 4 minutes        80/tcp               nice_poitras
43a8c73ddde2   nginx:1.23   "/docker-entrypoint.…"   10 minutes ago       Up 10 minutes       80/tcp               quizzical_austin
842ce232c412   nginx:1.23   "/docker-entrypoint.…"   11 minutes ago       Up 11 minutes       80/tcp               zen_jones
af5d423b310e   nginx:1.23   "/docker-entrypoint.…"   46 minutes ago       Up 46 minutes       80/tcp               brave_pare
kanza@kanza-Latitude-7410:~$
```

You can access the Nginx server by visiting `http://localhost:9000` in your web browser, assuming you are running Docker on your local machine and port 9000 is access


localhost:9000

Welcome to nginx!

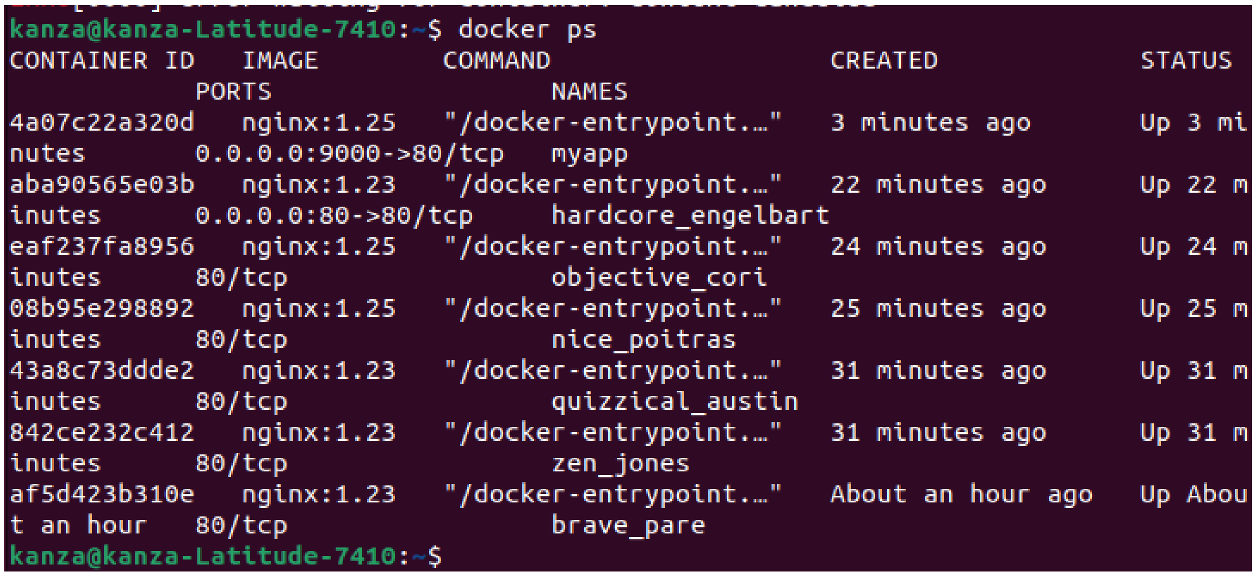
If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

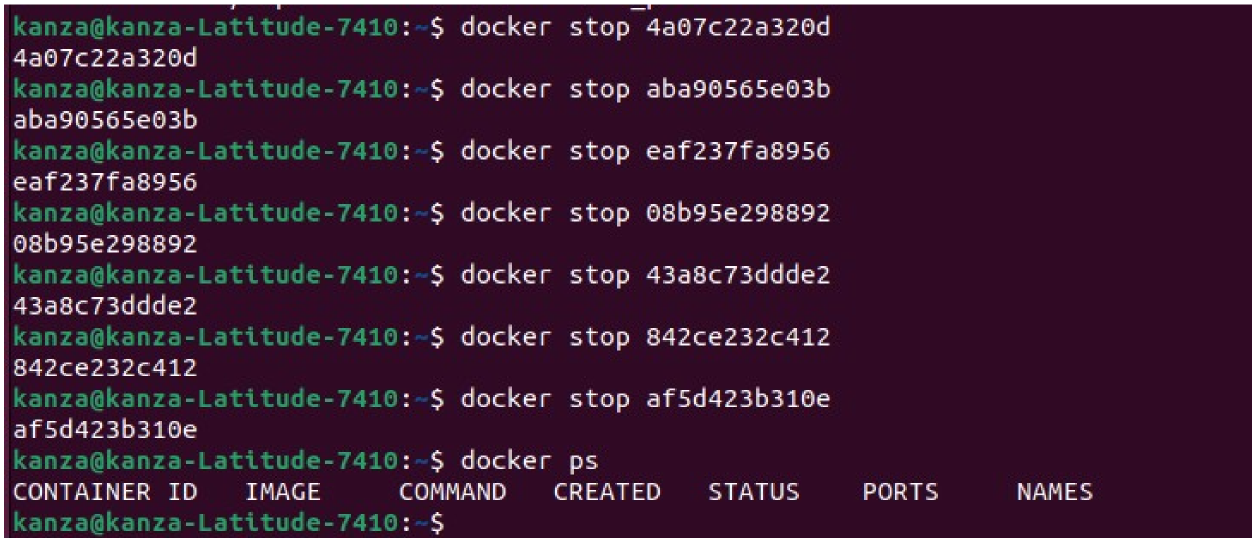
Thank you for using nginx.

- Now stop all the running containers once you’re done with your task.

  First check all the running containers:



```
kanza@kanza-Latitude-7410:~$ docker ps
CONTAINER ID   IMAGE         COMMAND                  CREATED             STATUS
               PORTS                  NAMES
4a07c22a320d   nginx:1.25    "/docker-entrypoint.…"   3 minutes ago       Up 3 mi
nutes          0.0.0.0:9000->80/tcp   myapp
aba90565e03b   nginx:1.23    "/docker-entrypoint.…"   22 minutes ago      Up 22 m
inutes         0.0.0.0:80->80/tcp     hardcore_engelbart
eaf237fa8956   nginx:1.25    "/docker-entrypoint.…"   24 minutes ago      Up 24 m
inutes         80/tcp                 objective_cori
08b95e298892   nginx:1.25    "/docker-entrypoint.…"   25 minutes ago      Up 25 m
inutes         80/tcp                 nice_poitras
43a8c73ddde2   nginx:1.23    "/docker-entrypoint.…"   31 minutes ago      Up 31 m
inutes         80/tcp                 quizzical_austin
842ce232c412   nginx:1.23    "/docker-entrypoint.…"   31 minutes ago      Up 31 m
inutes         80/tcp                 zen_jones
af5d423b310e   nginx:1.23    "/docker-entrypoint.…"   About an hour ago   Up Abou
t an hour      80/tcp                 brave_pare
kanza@kanza-Latitude-7410:~$
```



```
kanza@kanza-Latitude-7410:~$ docker stop 4a07c22a320d
4a07c22a320d
kanza@kanza-Latitude-7410:~$ docker stop aba90565e03b
aba90565e03b
kanza@kanza-Latitude-7410:~$ docker stop eaf237fa8956
eaf237fa8956
kanza@kanza-Latitude-7410:~$ docker stop 08b95e298892
08b95e298892
kanza@kanza-Latitude-7410:~$ docker stop 43a8c73ddde2
43a8c73ddde2
kanza@kanza-Latitude-7410:~$ docker stop 842ce232c412
842ce232c412
kanza@kanza-Latitude-7410:~$ docker stop af5d423b310e
af5d423b310e
kanza@kanza-Latitude-7410:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
kanza@kanza-Latitude-7410:~$
```

Install Dependencies

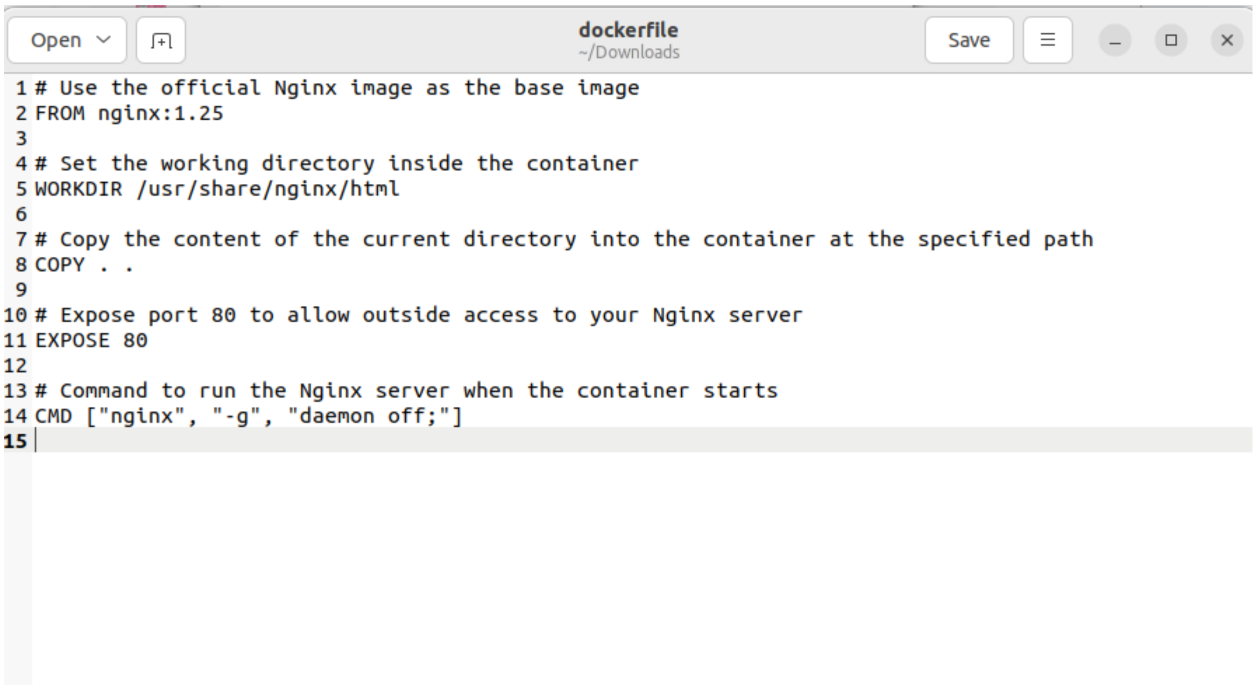
Command:

```
sudo apt-get update && sudo apt-get install -y \
python3 \
python3-pip \
&& sudo rm -rf /var/lib/apt/lists/*
```

Creating a Custom Docker Container

Dockerfile Creation

- Dockerfile is a text document that contains all the commands a user could call on th command line to assemble an image.
- Create a new file named ‘Dockerfile’ in a directory of your choice.



```
# Use the official Nginx image as the base image
FROM nginx:1.25

# Set the working directory inside the container
WORKDIR /usr/share/nginx/html

# Copy the content of the current directory into the container at the specified path
COPY . .

# Expose port 80 to allow outside access to your Nginx server
EXPOSE 80

# Command to run the Nginx server when the container starts
CMD ["nginx", "-g", "daemon off;"]
```

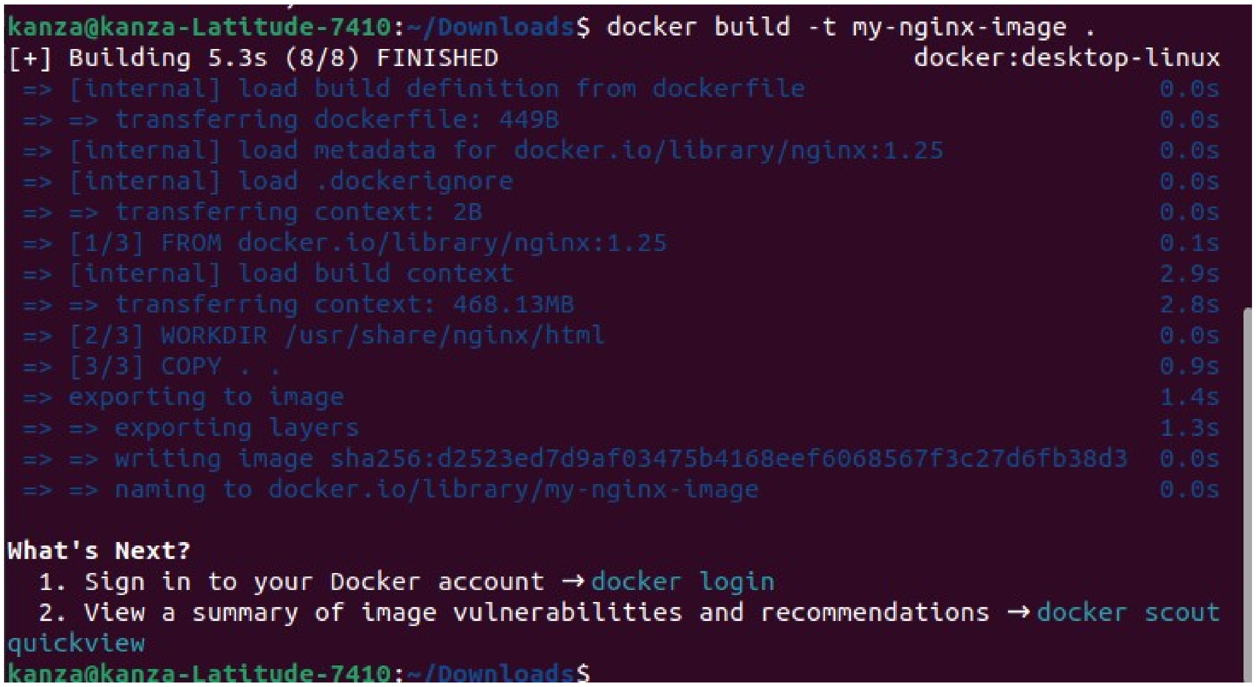
To use this Dockerfile, follow these steps

1. Save the Dockerfile in your project directory.
2. Place any files you want to include in the Docker image (e.g., HTML files) in the sam directory

Build the Docker image

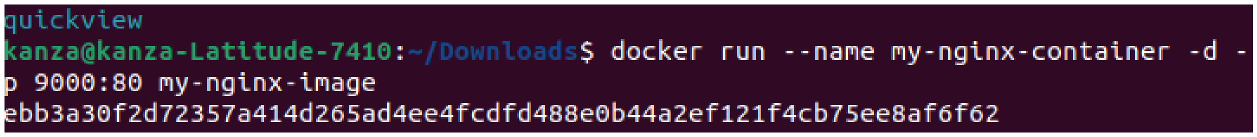
using the docker build command:

```
docker build -t my-nginx-image
```

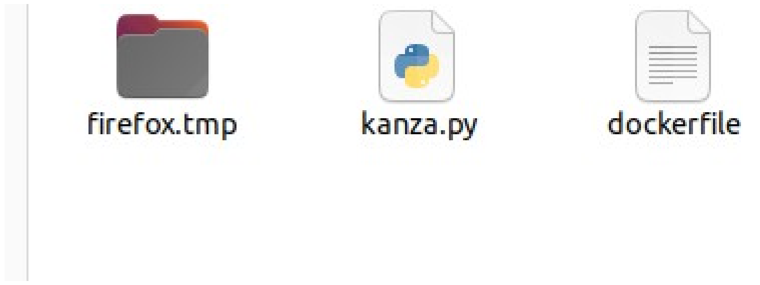


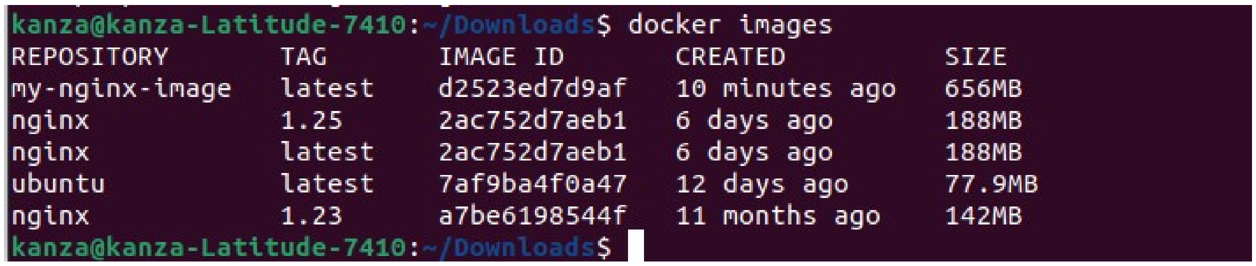
```
kanza@kanza-Latitude-7410:~/Downloads$ docker build -t my-nginx-image .
[+] Building 5.3s (8/8) FINISHED                             docker:desktop-linux
 => [internal] load build definition from dockerfile                        0.0s
 => => transferring dockerfile: 449B                                        0.0s
 => [internal] load metadata for docker.io/library/nginx:1.25               0.0s
 => [internal] load .dockerignore                                           0.0s
 => => transferring context: 2B                                             0.0s
 => [1/3] FROM docker.io/library/nginx:1.25                                 0.1s
 => [internal] load build context                                           2.9s
 => => transferring context: 468.13MB                                       2.8s
 => [2/3] WORKDIR /usr/share/nginx/html                                     0.0s
 => [3/3] COPY . .                                                          0.9s
 => exporting to image                                                      1.4s
 => => exporting layers                                                     1.3s
 => => writing image sha256:d2523ed7d9af03475b4168eef6068567f3c27d6fb38d3   0.0s
 => => naming to docker.io/library/my-nginx-image                           0.0s

What's Next?
  1. Sign in to your Docker account → docker login
  2. View a summary of image vulnerabilities and recommendations → docker scout
quickview
kanza@kanza-Latitude-7410:~/Downloads$
```

Once the image is built, you can run a container based on this image using the docker run
command, like we did earlier:



```
quickview
kanza@kanza-Latitude-7410:~/Downloads$ docker run --name my-nginx-container -d -
p 9000:80 my-nginx-image
ebb3a30f2d72357a414d265ad4ee4fcdfd488e0b44a2ef121f4cb75ee8af6f62
```





```
kanza@kanza-Latitude-7410:~/Downloads$ docker images
REPOSITORY       TAG       IMAGE ID       CREATED          SIZE
my-nginx-image   latest    d2523ed7d9af   10 minutes ago   656MB
nginx            1.25      2ac752d7aeb1   6 days ago       188MB
nginx            latest    2ac752d7aeb1   6 days ago       188MB
ubuntu           latest    7af9ba4f0a47   12 days ago      77.9MB
nginx            1.23      a7be6198544f   11 months ago    142MB
kanza@kanza-Latitude-7410:~/Downloads$
```

Docker Volumes

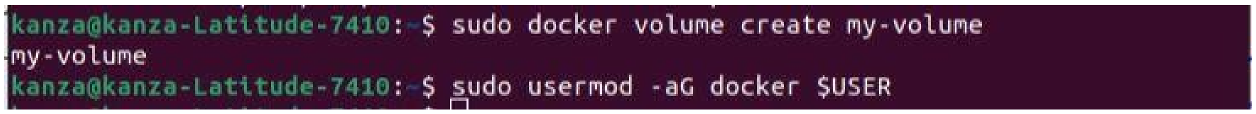
Understanding Docker Volumes

- Docker volumes provide a way to persist data generated by and used by Docker containers.
- Volumes can be used to share files between the file system of a container and the l file system of your machine even if the container is stopped or removed.

Using Docker Volumes

- To create a volume, you can use the following command:

```
docker volume create <volume_name>
```



```
kanza@kanza-Latitude-7410:~$ sudo docker volume create my-volume
my-volume
kanza@kanza-Latitude-7410:~$ sudo usermod -aG docker $USER
```

Benefits of Docker Volumes

Data Persistence

Volumes ensure that data created or modified inside a Docker container persists even aft container is stopped or removed.

Share Data Between Containers

Volumes enable multiple containers to share and access the same data, facilitating collaboration and data consistency.

Performance

Docker volumes offer better performance compared to bind mounts, especially on platfor like Docker for Mac and Docker for Windows.

Types of Docker Volumes

Docker volumes are a crucial aspect of containerized applications, providing a way to mar
and persist data beyond the lifecycle of containers. Understanding the different types of D
volumes helps in choosing the appropriate storage solution for your application's needs.

1. Named Volumes

Named volumes are explicitly created and managed by Docker. They are identified by a u
defined name and are stored within Docker's managed storage directory on the host macl
Named volumes offer several advantages:

Advantages

Ease of Management

Named volumes are easy to create, manage, and reference using their assigned names.

Isolation

Each named volume is isolated from others, ensuring data integrity and security.

Integration with Docker CLI

Docker CLI commands can be used to create, inspect, and manage named volumes effort

Example:

```
docker volume create my_volume

docker run -v my_volume:/data my_image
```

2. Bind Mounts

Bind mounts link a directory or file on the host machine to a directory in the container. Wi
bind mounts, the content of the specified host directory or file is directly accessible from t
container, and any changes made within the container are reflected on the host and vice
Key features of bind mounts include:

Key Features

Real-Time Synchronization

Data changes are synchronized in real-time between the host and container.

Flexibility

Bind mounts allow access to host directories or files, enabling seamless integration with t
host filesystem.

Performance

Bind mounts may offer better performance compared to named volumes, especially for I/
intensive operations.

Example:

```
docker run -v /host/directory:/container/directory my_image
```

3. Anonymous Volumes

Anonymous volumes are automatically created by Docker and are managed internally. Th
primarily used when a container needs access to a volume, but the specific details of the
volume are unimportant or temporary. Anonymous volumes have the following characteri

Characteristics

Automatic Creation

Docker automatically generates an anonymous volume when one is needed by a containe

Managed Internally

Docker manages the lifecycle and storage of anonymous volumes, relieving users of expli
management tasks.

Transient Nature

Anonymous volumes are typically used for temporary data storage and are discarded whe
associated container is removed.

Example:

```
docker run -v /container/directory my_image
```

Choosing the Right Type

Selecting the appropriate type of Docker volume depends on factors such as data persiste
requirements, ease of management, and performance considerations. Consider the follow
guidelines when choosing a volume type:

Named Volumes

Ideal for long-term data storage, sharing data between containers, and simplified manage

Bind Mounts

Suitable for accessing host files or directories directly from containers and achieving real-
synchronization.

Anonymous Volumes

Use for temporary or disposable data storage needs, where explicit management is not necessary.

Conclusion

Understanding the nuances of each type of Docker volume empowers developers and operators to make informed decisions regarding data storage and management within containerized environments. By leveraging the appropriate volume type, applications can achieve reliable data persistence and efficient data access.

Usage

Creating a Container with a Volume

```
docker run -v my_volume:/data my_image
```

Mounting a Local Directory as a Volume

```
docker run -v /host/directory:/container/directory my_image
```

Listing Volumes

```
docker volume ls
```

Inspecting a Volume

```
docker volume inspect my_volume
```

Conclusion

Docker volumes are a powerful feature for managing data in Docker containers. By understanding the types of volumes available and how to use them effectively, you can er data persistence and efficient data management in your Dockerized applications.