What is virtualization?

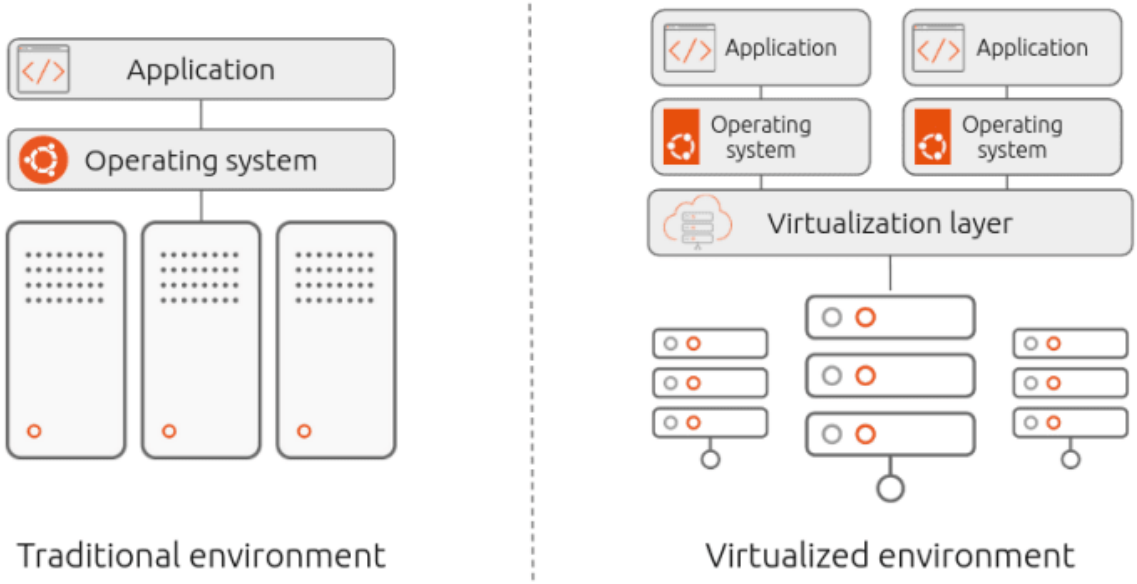
Bare Metal Servers:

Back in the old days, physical servers functioned much like a regular computer would. You had the physical box, you would install an operating system, and then you would install applications on top.

These types of servers are often referred to as 'bare metal servers', as there's nothing in between the actual physical (metal) machine and the operating system. The costs, however, were very high. Not only did you need more and more servers as your business grew, you also needed to have enough space to host them.

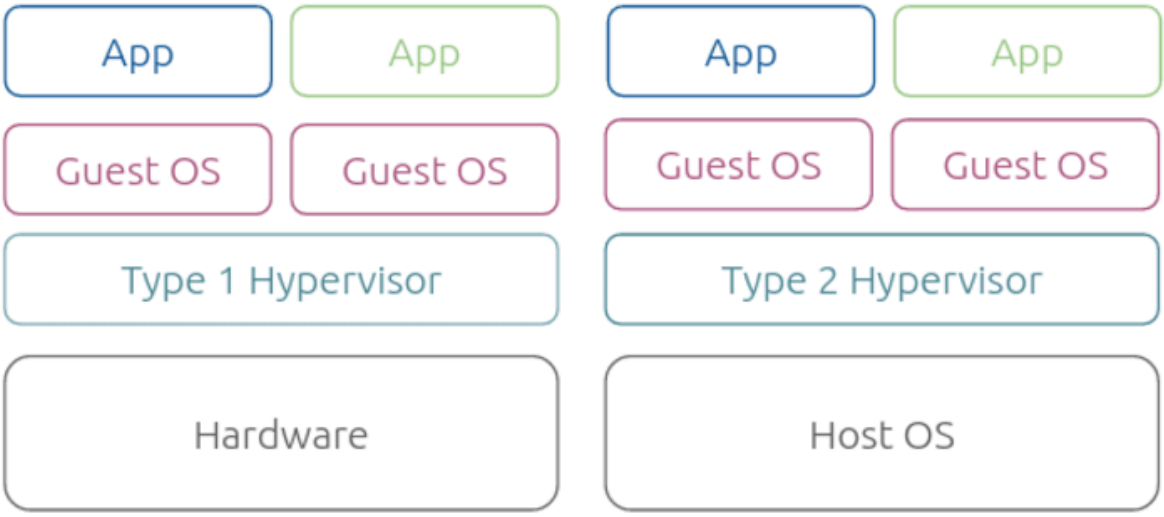
Virtual Servers:

Rather than having the operating system run directly on top of the physical hardware, an additional virtualization layer is added in between, enabling users to deploy multiple virtual servers, each with their own operating system, on one physical machine. This enabled significant savings and optimization for companies, and eventually led to the existence of cloud computing.



The role of a hypervisor:

Virtualization wouldn't be possible without a hypervisor (also known as a virtual machine monitor) – a software layer enabling multiple operating systems to co-exist while sharing the resources of a single hardware host. The hypervisor acts as an intermediary between virtual machines and the underlying hardware, allocating host resources such as memory, CPU, and storage.



There are two main types of hypervisors: Type 1 and Type 2.

- **Type 1 hypervisors**, also known as **bare-metal hypervisors**, run directly on the host's hardware and are responsible for managing the hardware resources and running the virtual machines. Because they run directly on the hardware, they are often more efficient and have a smaller overhead than Type 2 hypervisors. Examples of Type 1 hypervisors include VMware ESXi, Microsoft Hyper-V, and Citrix XenServer.
- **Type 2 hypervisors**, also known as **hosted hypervisors**, run on top of a host operating system and rely on it to provide the necessary hardware resources and support. Because they run on top of an operating system, they are often easier to install and use than Type 1 hypervisors, but they might be less efficient. Examples of Type 2 hypervisors include VMware Workstation and Oracle VirtualBox.

On whole "Virtualization is a technology that allows you to create multiple simulated environments or dedicated resources from a single, physical hardware system" In virtualization, the hypervisor creates an abstraction layer over hardware, so that multiple operating systems can run alongside each other.

Disadvantages:

VM is an entire operating system with its own kernel, hardware drivers, programs, and applications. Spinning up a VM only to isolate a single application is a lot of overhead.

Overhead	High (Full OS per VM)
Boot Time	Slow (Full OS boot)
Portability	Less portable (Heavier, dependent on hypervisor)

What is Containerization?

“Containerization is the packaging together of software code with all its necessary components like libraries, frameworks, and other dependencies so that they are isolated in their own "container".

Containerization is considered to be a lightweight version of virtualization, which virtualizes the operating system instead of hardware. Without the hypervisor, the containers enjoy faster resource provisioning. All the resources (including code, dependencies) that are needed to run the application or microservice are packaged together, so that the applications can run anywhere.

What is a Container?

A container is simply an isolated process with all of the files it needs to run. If you run multiple containers, they all share the same kernel, allowing you to run more applications on less infrastructure.

Containerization is increasingly popular because containers are:

Flexible: Even the most complex applications can be containerized.

Lightweight: Containers leverage and share the host kernel.

Interchangeable: You can deploy updates and upgrades on-the-fly.

Portable: You can build locally, deploy to the cloud, and run anywhere.

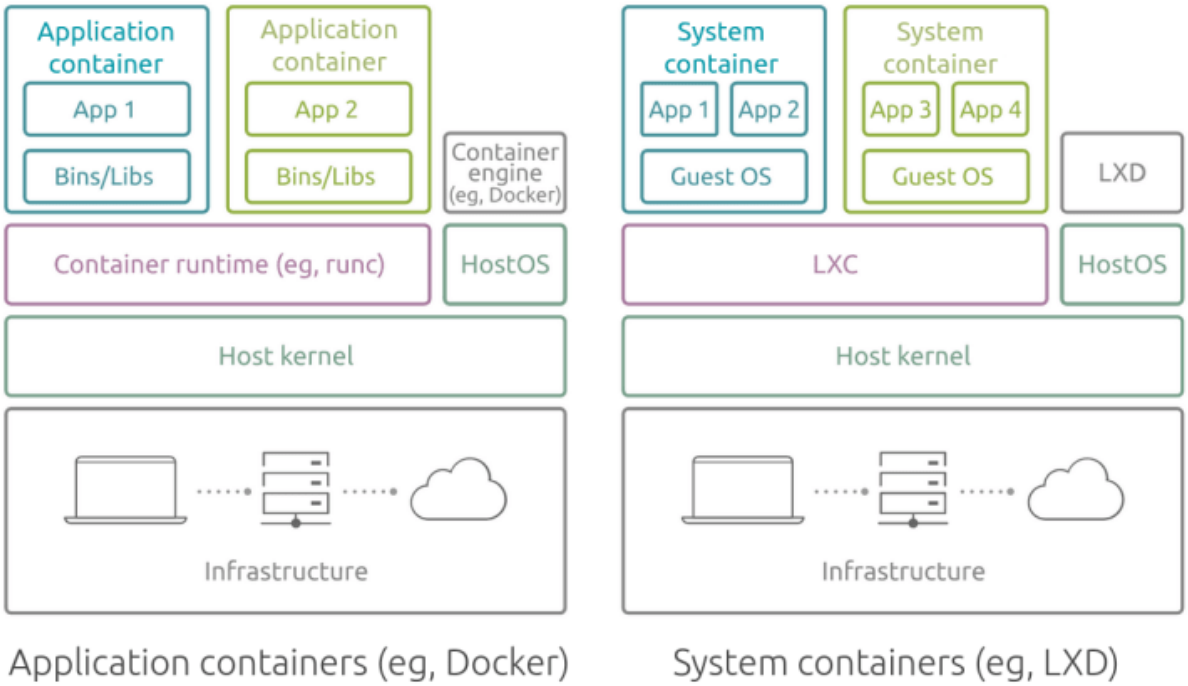
Scalable: You can increase and automatically distribute container replicas.

Stackable: You can stack services vertically and on-the-fly.

Types of Containers:

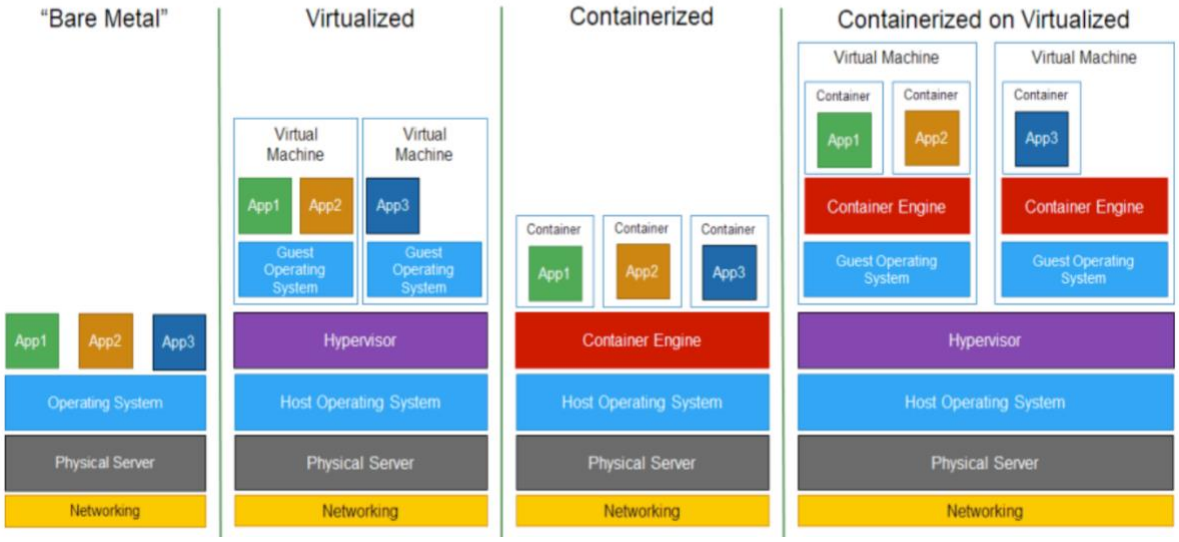
Application vs. system containers

It is relevant to note that there are different types of containers: application, system, embedded. They all rely on the host kernel, offering bare metal performance and no virtualization overhead, but they do so in slightly different ways.



- **Application containers (such as Docker),** also known as process containers, package and run a single process or a service per container. They are packaged along with all the libraries, dependencies, and configuration files they need, allowing them to be run consistently across different environments.
- **System containers (as run by LXD),** on the other hand, are in a way similar to a physical or a virtual machine. They run a full operating system and have the same behaviour and manageability as VMs, without the usual overhead, and with the density and efficiency of containers.

Deployment mechanisms and supportability



What are the different Containerization Tools and Platforms available in the market?

1. Docker:

- **Overview: Docker is the most widely used containerization platform that allows you to package applications and their dependencies into containers. It provides tools to create, run, and manage containers on any system.**
- **Key Features:**
  - **Portable and consistent environments**
  - **Extensive ecosystem and community support**
  - **Docker Hub for storing and sharing container images**
- **Use Cases: Development, testing, CI/CD pipelines, microservices.**

2. Podman:

- **Overview: Podman is a daemonless, open-source container engine developed by Red Hat. It is designed to be compatible with Docker CLI commands but offers additional security by allowing containers to run without root privileges.**
- **Key Features:**
  - **No central daemon, improving security and reducing single points of failure.**
  - **Rootless containers, enhancing security by running without root privileges.**
  - **Docker CLI compatibility, making it easy to switch from Docker to Podman.**

3. Buildah:

- **Overview: Buildah is a tool for building OCI-compatible (Open Container Initiative) container images. It is designed to be a more flexible and efficient alternative to Docker's image-building process.**
- **Key Features:**
  - **Daemonless image building, reducing overhead and improving security.**
  - **Fine-grained control over the image-building process, including the ability to build images from scratch.**
  - **Integration with Podman for running containers built with Buildah.**

5. LXC/LXD:

- **Overview: LXC (Linux Containers) is an OS-level virtualization tool that allows multiple isolated Linux systems (containers) to run on a single Linux kernel. LXD is a container hypervisor built on top of LXC, offering more advanced features.**
- **Key Features:**
  - **Full system containers, allowing for running entire Linux distributions in containers.**
  - **Lightweight and efficient, with lower overhead than traditional VMs.**
  - **Advanced networking and storage management capabilities.**

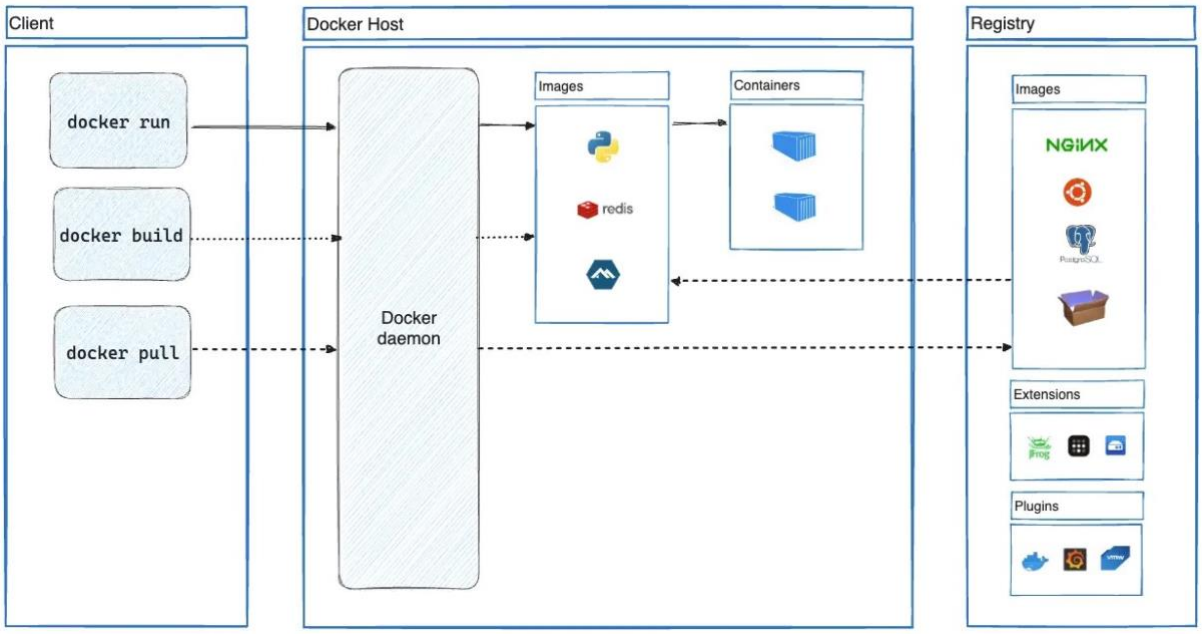
Why Docker?

Docker is a platform for developers and sysadmins to develop, deploy, and run applications with containers. They are

- **Cost-saving: Docker containers use far less memory, especially when compared to their counterparts (virtual machines). Thus, you spend less on IT infrastructure resources.**
- **Flexible resource sharing: Your containerized apps are all running on the same operating system even though your application and its dependencies are isolated from the underlying operating system and other containers by Docker containers.**
- **Multi-Cloud Platforms: Almost all of the main cloud service providers support running Docker, and switching between environments is simple. Thus, you can ship anytime and anywhere.**
- **Configuration and consistent delivery of your applications: Docker offers a faster, more resource-efficient, and standardized way to develop, ship, and run applications. Applications can be distributed on various platforms without worrying about framework, library, and compatibility issues.**
- **Pipelines: Docker allows you to standardize the development and release cycle. This acts as a form of legacy change management for applications and encourages Continuous Integration and Continuous Delivery (CI/CD). Thus, building agile and scalable applications is possible.**

Docker architecture

Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.



The Docker daemon

The Docker daemon (dockerd) 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 looks 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, Docker pulls the required images from your configured registry. When you use the docker push command, Docker pushes your image to your configured registry.

Docker Objects

```
### Summary of Docker Objects

- **Containers**: Running instances of images.
- **Images**: Read-only templates used to create containers.
- **Volumes**: Persistent storage for container data.
- **Networks**: Facilitate communication between containers and with external systems.
- **Registries**: Storage and distribution points for images.
- **Dockerfile**: Defines how to build a Docker image.
- **Docker Compose**: Manages multi-container Docker applications.
```

DOCKER IMAGES

A Docker image, or container image, is a standalone, executable file used to create a container. This container image contains all the libraries, dependencies, and files that the container needs to run. It encapsulates an application and its dependencies into a single unit, which ensures consistency across different environments.

'docker image' command:

The docker image command is part of Docker's newer command structure, introduced to organize and simplify Docker CLI commands related to images. This command is used to manage Docker images, including listing, removing, and inspecting them.

Subcommands

Command	Description
`docker image history`	Show the history of an image
`docker image import`	Import the contents from a tarball to create a filesystem image
`docker image inspect`	Display detailed information on one or more images
`docker image load`	Load an image from a tar archive or STDIN
`docker image prune`	Remove unused images
`docker image rm`	Remove one or more images
`docker image save`	Save one or more images to a tar archive (streamed to STDOUT by default)
`docker image tag`	Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
`docker image ls`	List images
`docker image pull`	Download an image from a registry
`docker image push`	Upload an image to a registry

docker pull:

Pulling a docker image from Docker Registry:

```
ubuntu@ip-172-31-90-246:~$ sudo su -
root@ip-172-31-90-246:~# docker pull tomcat:9.0
9.0: Pulling from library/tomcat
2b3981cac065: Pull complete
3e44a677d4d8: Pull complete
f561a59c5174: Pull complete
8fc851d1d586: Pull complete
4723684ec455: Pull complete
1028279da7ac: Pull complete
4f4fb700ef54: Pull complete
32789e2d4c58: Pull complete
Digest: sha256:de248c13aa49d4b245220b65a0194ed8ea82e19cc063604838bd14ad675afcef
Status: Downloaded newer image for tomcat:9.0
docker.io/library/tomcat:9.0
```

docker images or docker image ls:

Checking the list of Images available:

```
root@ip-172-31-90-246:~# docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
tomcat        9.0       f23130139036   7 days ago       471MB
nginx         latest    a72860cb95fd   7 weeks ago      188MB
hello-world   latest    d2c94e258dcb   15 months ago    13.3kB
```

Login into your Docker Hub:

```
root@ip-172-31-90-246:~# docker login
Log in with your Docker ID or email address to push and pull images from Docker Hub.
You can log in with your password or a Personal Access Token (PAT). Using a limited-s
 at https://docs.docker.com/go/access-tokens/

Username: subbu7677
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credential-stores

Login Succeeded
```

Issue:

Initially when you are using a command directly you will get the following error:

```
docker search tomcat
permission denied while trying to connect to the Docker daemon
socket at unix:///var/run/docker.sock: Get
"http://%2Fvar%2Frun%2Fdocker.sock/v1.46/images/search?
term=tomcat": dial unix /var/run/docker.sock: connect: permission
denied
```

Solution:

Add Your User to the Docker Group:

- **Instead of using sudo every time, you can add your user to the docker group, which grants permission to interact with the Docker daemon without requiring root privileges.**

1. **Add your user to the docker group:**
   sudo usermod -aG docker $USER

```
root@ip-172-31-90-246:~# sudo usermod -aG docker $USER
```

2. **Log out and log back in to apply the group changes, or use the following command to activate the changes without logging out:**
   newgrp docker

```
root@ip-172-31-90-246:~# newgrp docker
```

Push the image to Docker Hub:

1. Tag the Image:

In Docker, the docker tag command is used to create a new tag for an existing Docker image. Tags are essentially aliases or references to a particular image, allowing you to easily manage and organize your Docker images.

docker image tag

Description	Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
Usage	`docker image tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]`
Aliases ?	`docker tag`

Ex: docker tag [IMAGE_ID] subbu7677/tomcat:9.0

```
root@ip-172-31-90-246:~# docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
tomcat        9.0       f23130139036   7 days ago       471MB
nginx         latest    a72860cb95fd   7 weeks ago      188MB
hello-world   latest    d2c94e258dcb   15 months ago    13.3kB
```

```
root@ip-172-31-90-246:~# docker tag f23130139036 subbu7677/tomcat:9.0
```

docker push:

2. Push the Image to Docker Hub:

The docker push command is used to upload a Docker image to a Docker registry. This is typically done to share the image with others or to deploy it to a container orchestration system.

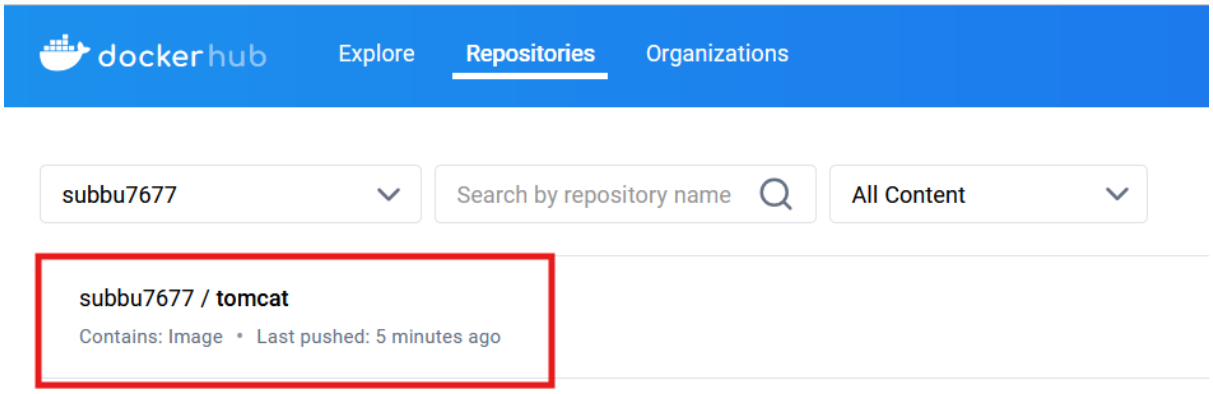
docker image push

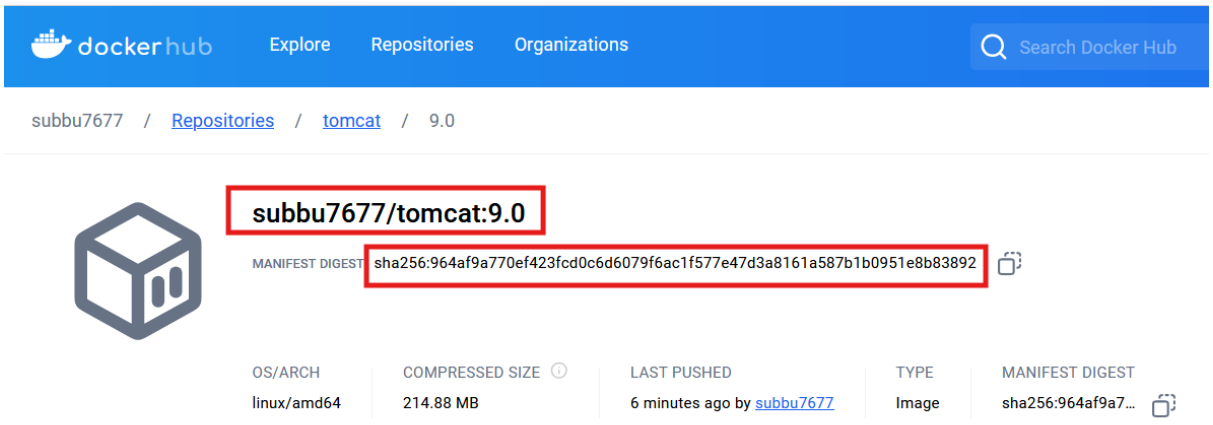
Description	Upload an image to a registry
Usage	`docker image push [OPTIONS] NAME[:TAG]`
Aliases	`docker push`

docker push subbu7677/tomcat:9.0

```
root@ip-172-31-90-246:~# docker push subbu7677/tomcat:9.0
The push refers to repository [docker.io/subbu7677/tomcat]
5f70bf18a086: Mounted from library/tomcat
92be6d7d176d: Mounted from library/tomcat
d6906dc30bee: Mounted from library/tomcat
8c898876677c: Mounted from library/tomcat
d6843b9cd847: Mounted from library/tomcat
c72ec1f3fa99: Mounted from library/tomcat
25224a50eec4: Mounted from library/tomcat
a30a5965a4f7: Mounted from library/tomcat
9.0: digest: sha256:964af9a770ef423fcd0c6d6079f6ac1f577e47d3a8161a587b1b0951e8b83892 size: 2201
root@ip-172-31-90-246:~#
```

3. We can check in the Docker Hub:





To check the list of images are present in your local system:

docker images (or) docker image ls

```
root@ip-172-31-90-246:~# docker image ls
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
tomcat              9.0       f23130139036   7 days ago       471MB
subbu7677/tomcat    9.0       f23130139036   7 days ago       471MB
nginx               latest    a72860cb95fd   7 weeks ago      188MB
hello-world         latest    d2c94e258dcb   15 months ago    13.3kB
root@ip-172-31-90-246:~#
```

docker rm (or) docker rmi:

The commands docker image rm and docker image rmi both serve the purpose of removing Docker images.

The docker rmi command is used to remove one or more Docker images from your local system. This is helpful when you want to free up disk space or remove outdated or unnecessary images.

Removes (and un-tags) one or more images from the host node. If an image has multiple tags, using this command with the tag as a parameter only removes the tag. If the tag is the only one for the image, both the image and the tag are removed.

This does not remove images from a registry. You cannot remove an image of a running container unless you use the -f option. To see all images on a host use the docker image ls command.

```
root@ip-172-31-90-246:/home/ubuntu# docker image ls
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
tomcat              9.0       f23130139036   8 days ago       471MB
subbu7677/tomcat    9.0       f23130139036   8 days ago       471MB
nginx               latest    a72860cb95fd   7 weeks ago      188MB
hello-world         latest    d2c94e258dcb   15 months ago    13.3kB
```

```
root@ip-172-31-90-246:/home/ubuntu# docker rmi hello-world
Error response from daemon: conflict: unable to remove repository reference "hello-world" (must force) - container ad3ab19a08d7 is using its referenced image d2c94e258dcb
root@ip-172-31-90-246:/home/ubuntu# docker rmi hello-world d2c94e258dcb
Error response from daemon: conflict: unable to remove repository reference "hello-world" (must force) - container ad3ab19a08d7 is using its referenced image d2c94e258dcb
Error response from daemon: conflict: unable to delete d2c94e258dcb (must be forced) - image is being used by stopped container ad3ab19a08d7
```

ERROR: (Examples)

1. **Error response from daemon: conflict: unable to remove repository reference "hello-world" (must force) - container ad3ab19a08d7 is using its referenced image d2c94e258dcb**
2. **Error response from daemon: conflict: unable to remove repository reference "nginx" (must force) - container d1262f249ab3 is using its referenced image a72860cb95fd**

Solution:

The error you're encountering indicates that the hello-world image (ID: d2c94e258dcb) is currently being used by a stopped container (ID: ad3ab19a08d7). To remove the image, you must first delete the stopped container or force the removal of the image.

List All Containers:

- **First, list all containers, including stopped ones, to verify the container using the image: docker ps -a**

```
root@ip-172-31-90-246:/home/ubuntu# docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS                     PORTS     NAMES
d1262f249ab3   nginx          "/docker-entrypoint.…"   45 hours ago   Exited (0) 43 hours ago            ngnixcontainer
ad3ab19a08d7   hello-world    "/hello"                 46 hours ago   Exited (0) 46 hours ago            laughing_kapitsa
```

Remove the Stopped Container:

- **If you don't need the stopped container, you can remove it using:**

  docker rm <container id>

```
root@ip-172-31-90-246:/home/ubuntu# docker rm d1262f249ab3
d1262f249ab3
root@ip-172-31-90-246:/home/ubuntu# docker ps -a
CONTAINER ID   IMAGE          COMMAND    CREATED        STATUS                     PORTS     NAMES
ad3ab19a08d7   d2c94e258dcb   "/hello"   46 hours ago   Exited (0) 46 hours ago            laughing_kapitsa
```

After removing the container, try removing the image again: docker rmi <image_name>

```
root@ip-172-31-90-246:/home/ubuntu# docker image ls
REPOSITORY          TAG       IMAGE ID       CREATED        SIZE
tomcat              9.0       f23130139036   8 days ago     471MB
subbu7677/tomcat    9.0       f23130139036   8 days ago     471MB
nginx               latest    a72860cb95fd   7 weeks ago    188MB
root@ip-172-31-90-246:/home/ubuntu# docker rmi nginx
Untagged: nginx:latest
Untagged: nginx@sha256:6af79ae5de407283dcea8b00d5c37ace95441fd58a8b1d2aa1ed93f5511bb18c
Deleted: sha256:a72860cb95fd59e9c696c66441c64f18e66915fa26b249911e83c3854477ed9a
Deleted: sha256:1188c692bee9694c47db34046023dbd938d88f303f216ef689863741b2d1a900
Deleted: sha256:3eefccfd7e5fd8bbb2bd982509dc79206b056f22dd2b14553951a743833b0d09
Deleted: sha256:5234252bfd2bba1548a4998869e9a01aedfe3b319ce61acbe98f8aec223640e7
Deleted: sha256:b292d631e6ca5af8269dc2cf3ec47be1f9faa0865b2aaa794daa2b8c25ea8cb4
Deleted: sha256:beda8840654459fe0efc1cd0bcae6a00b65b469cc999ebc41608b53c51fb93b4
Deleted: sha256:7a69d5090b2e7d873a365c81c590b2e6b87a702178b22b3c32c50d35eb7616fc
Deleted: sha256:e0781bc8667fb5ebf954df4ae52997f6f5568ec9f07e21e5db7c9d324ed41e1f
root@ip-172-31-90-246:/home/ubuntu#
```

Now that image is deleted from your local system.

Force Removal of the Image:

If you prefer to remove the image directly without manually deleting the container, use the -f (force) option: docker rmi -f <image-name>

Ex:

```
root@ip-172-31-90-246:/home/ubuntu# docker rmi -f hello-world
Untagged: hello-world:latest
Untagged: hello-world@sha256:1408fec50309afee38f3535383f5b0941
Deleted: sha256:d2c94e258dcb3c5ac2798d32e1249e42ef01cba4841c22
```

Note:

Scenario-1: If any image is mapped with a container, then it is unable to delete a docker image normally even though that container is in stopped state. But You can delete that image forcefully if the container is in stopped status by using forcefully option.

Ex:

```
root@ip-172-31-90-246:/home/ubuntu# docker images
REPOSITORY        TAG       IMAGE ID       CREATED        SIZE
jenkins           2.361.1   bb6d7d143f5f   3 hours ago    1.11GB
nginx             latest    cd43ca3ebe4a   7 weeks ago    188MB
subbu7677/nginx   latest    cd43ca3ebe4a   7 weeks ago    188MB
sonatype/nexus    latest    c55c6ce14194   2 years ago    464MB
root@ip-172-31-90-246:/home/ubuntu# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
root@ip-172-31-90-246:/home/ubuntu# docker ps -a
CONTAINER ID   IMAGE             COMMAND                  CREATED        STATUS                      PORTS
f8eae0ea3cd5   jenkins:2.361.1   "/usr/bin/tini -- /u…"   2 hours ago    Exited (143) 37 minutes ago
1395f3f3122b   sonatype/nexus    "/bin/sh -c 'java  …"    6 hours ago    Exited (255) 2 hours ago    0.0.0.0:8081->8081/tcp, :::8081->8081/tcp
ad3ab19a08d7   d2c94e258dcb      "/hello"                 3 days ago     Exited (0) 3 days ago
root@ip-172-31-90-246:/home/ubuntu# docker rmi bb6d7d143f5f
Error response from daemon: conflict: unable to delete bb6d7d143f5f (must be forced) - image is being used by stopped container f8eae0ea3cd5
root@ip-172-31-90-246:/home/ubuntu#
```

```
root@ip-172-31-90-246:/home/ubuntu# docker rmi -f bb6d7d143f5f
Untagged: jenkins:2.361.1
Deleted: sha256:bb6d7d143f5f2c643ccecf9e4d2b31b63721f17c2d5c6d48a9944b2f4159de52
root@ip-172-31-90-246:/home/ubuntu# docker images
REPOSITORY        TAG       IMAGE ID       CREATED        SIZE
nginx             latest    cd43ca3ebe4a   7 weeks ago    188MB
subbu7677/nginx   latest    cd43ca3ebe4a   7 weeks ago    188MB
sonatype/nexus    latest    c55c6ce14194   2 years ago    464MB
```

Even though you deleted that image, the container works fine as usual once you started it again.

Ex:

```
root@ip-172-31-90-246:/home/ubuntu# docker rmi -f bb6d7d143f5f
Untagged: jenkins:2.361.1
Deleted: sha256:bb6d7d143f5f2c643ccecf9e4d2b31b63721f17c2d5c6d48a9944b2f4159de52
root@ip-172-31-90-246:/home/ubuntu# docker images
REPOSITORY          TAG       IMAGE ID       CREATED        SIZE
nginx               latest    cd43ca3ebe4a   7 weeks ago    188MB
subbu7677/nginx     latest    cd43ca3ebe4a   7 weeks ago    188MB
sonatype/nexus      latest    c55c6ce14194   2 years ago    464MB
root@ip-172-31-90-246:/home/ubuntu# docker ps -a
CONTAINER ID   IMAGE            COMMAND                  CREATED        STATUS                        PORTS
f8eae0ea3cd5   bb6d7d143f5f     "/usr/bin/tini -- /u…"   2 hours ago    Exited (143) 45 minutes ago
1395f3f3122b   sonatype/nexus   "/bin/sh -c 'java   …"  6 hours ago    Exited (255) 2 hours ago      0.0.0.0:8081->8081/tcp, :
ad3ab19a08d7   d2c94e258dcb     "/hello"                 3 days ago     Exited (0) 3 days ago
root@ip-172-31-90-246:/home/ubuntu# docker start f8eae0ea3cd5
f8eae0ea3cd5
root@ip-172-31-90-246:/home/ubuntu# docker ps
CONTAINER ID   IMAGE            COMMAND                  CREATED        STATUS         PORTS
   NAMES
f8eae0ea3cd5   bb6d7d143f5f     "/usr/bin/tini -- /u…"   3 hours ago    Up 2 minutes   0.0.0.0:8080->8080/tcp, :::8080->8080/tcp
   my-jenkins
root@ip-172-31-90-246:/home/ubuntu# docker stop f8eae0ea3cd5
f8eae0ea3cd5
root@ip-172-31-90-246:/home/ubuntu#
```

Scenario-2: If any image is mapped with a container, then it is unable to delete a docker image normally and forcefully if that container is in running state.

Ex:

```
root@ip-172-31-90-246:/home/ubuntu# docker ps -a
CONTAINER ID   IMAGE            COMMAND                  CREATED        STATUS
f8eae0ea3cd5   bb6d7d143f5f     "/usr/bin/tini -- /u…"   2 hours ago    Exited (143) 45 minutes ago
1395f3f3122b   sonatype/nexus   "/bin/sh -c 'java   …"  6 hours ago    Exited (255) 2 hours ago
ad3ab19a08d7   d2c94e258dcb     "/hello"                 3 days ago     Exited (0) 3 days ago
```

```
root@ip-172-31-90-246:/home/ubuntu# docker start 1395f3f3122b
1395f3f3122b
root@ip-172-31-90-246:/home/ubuntu# docker ps
CONTAINER ID   IMAGE            COMMAND                  CREATED        STATUS         PORTS                                          NAMES
1395f3f3122b   sonatype/nexus   "/bin/sh -c 'java   …"  6 hours ago    Up 9 seconds   0.0.0.0:8081->8081/tcp, :::8081->8081/tcp      nexusapp
root@ip-172-31-90-246:/home/ubuntu# docker image ls
REPOSITORY          TAG       IMAGE ID       CREATED        SIZE
nginx               latest    cd43ca3ebe4a   7 weeks ago    188MB
subbu7677/nginx     latest    cd43ca3ebe4a   7 weeks ago    188MB
sonatype/nexus      latest    c55c6ce14194   2 years ago    464MB
root@ip-172-31-90-246:/home/ubuntu# docker rmi c55c6ce14194
Error response from daemon: conflict: unable to delete c55c6ce14194 (cannot be forced) - image is being used by running container 1395f3f3122b
root@ip-172-31-90-246:/home/ubuntu# docker rmi -f c55c6ce14194
Error response from daemon: conflict: unable to delete c55c6ce14194 (cannot be forced) - image is being used by running container 1395f3f3122b
root@ip-172-31-90-246:/home/ubuntu#
```

Key Points of Docker Image and Container Relationship

1. **Image and Container Independence:**
   - **Running Containers: When a container is running, it uses its own file system, which was initialized from the image. This allows the container to function independently of the image's existence.**
2. **Deleting Images:**

- **Running Containers: You cannot delete an image that is currently in use by a running container. Docker enforces this to prevent disruptions.**
- **Stopped Containers: Docker allows image deletion with forceful (-f) option if the image is only used by stopped containers. The image will be removed if no active containers depend on it.**

To delete all stopped container images at a time:

docker rmi -f $(docker images -q)

```
root@ip-172-31-90-246:/home/ubuntu# docker ps -a
CONTAINER ID   IMAGE             COMMAND                  CREATED        STATUS
f8eae0ea3cd5   bb6d7d143f5f      "/usr/bin/tini -- /u…"   3 hours ago    Exited (143) 48 minutes ago
1395f3f3122b   sonatype/nexus    "/bin/sh -c 'java   …"  7 hours ago    Exited (143) 4 seconds ago
ad3ab19a08d7   d2c94e258dcb      "/hello"                 3 days ago     Exited (0) 3 days ago
root@ip-172-31-90-246:/home/ubuntu# docker images
REPOSITORY        TAG       IMAGE ID       CREATED        SIZE
nginx             latest    cd43ca3ebe4a   7 weeks ago    188MB
subbu7677/nginx   latest    cd43ca3ebe4a   7 weeks ago    188MB
sonatype/nexus    latest    c55c6ce14194   2 years ago    464MB
root@ip-172-31-90-246:/home/ubuntu# docker rmi -f $(docker images -q)
Untagged: nginx:latest
Untagged: subbu7677/nginx:latest
Untagged: subbu7677/nginx@sha256:03bc8cca389b961e1f446706e83c1e565fd4426b0658a5478ad69aa737dc1570
Deleted: sha256:cd43ca3ebe4acdc90375a2ec6f1945e3eee504981eb31b00a7746d041189b57c
Untagged: sonatype/nexus:latest
Untagged: sonatype/nexus@sha256:54702b1a275b8a458ae271cfaa616ccf68b135bcc89c63bf3111ff32fad1c23b
Deleted: sha256:c55c6ce141942bc88ed317420f32e40dc66c8b4fb97a67573babc01e81c6f345
Error response from daemon: No such image: cd43ca3ebe4a:latest
root@ip-172-31-90-246:/home/ubuntu# docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
root@ip-172-31-90-246:/home/ubuntu#
```

How to build a Docker Image: docker build

To build a Docker image that is based on another image pulled from Docker Hub, you'll use a Dockerfile to specify the base image and add your custom layers on top of it.

Note: If you try to build an image without a Dockerfile then you will get the error:

```
root@ip-172-31-90-246:/home/ubuntu# docker image ls
REPOSITORY          TAG       IMAGE ID       CREATED        SIZE
tomcat              9.0       f23130139036   8 days ago     471MB
subbu7677/tomcat    9.0       f23130139036   8 days ago     471MB
root@ip-172-31-90-246:/home/ubuntu# ls -lrth
total 0
```

```
root@ip-172-31-90-246:/home/ubuntu# docker build -t tomcat .
[+] Building 0.1s (1/1) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 2B
ERROR: failed to solve: failed to read dockerfile: open Dockerfile: no such file or directory
```

Create a Dockerfile:

In the Dockerfile, specify the base image and define the instructions to customize it.

Ex-1:

```
root@ip-172-31-90-246:/home/ubuntu# vi dockerfile
root@ip-172-31-90-246:/home/ubuntu# cat dockerfile
FROM tomcat:9.0
```

Now try to build the image from the pulled one

```
root@ip-172-31-90-246:/home/ubuntu# docker build -t tomcat .
[+] Building 0.3s (5/5) FINISHED
 => [internal] load build definition from dockerfile
 => => transferring dockerfile: 53B
 => [internal] load metadata for docker.io/library/tomcat:9.0
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/1] FROM docker.io/library/tomcat:9.0
 => exporting to image
 => => exporting layers
 => => writing image sha256:f231301390368f9114251820c43185785f611af531efd6d6e0e9203de943929c
 => => naming to docker.io/library/tomcat
root@ip-172-31-90-246:/home/ubuntu# docker image ls
REPOSITORY          TAG       IMAGE ID       CREATED       SIZE
tomcat              9.0       f23130139036   8 days ago    471MB
tomcat              latest    f23130139036   8 days ago    471MB
subbu7677/tomcat    9.0       f23130139036   8 days ago    471MB
```

Ex-2: Let's try with another version:

```
root@ip-172-31-90-246:/home/ubuntu# cat dockerfile
FROM tomcat:9.0.93
```

```
root@ip-172-31-90-246:/home/ubuntu# docker build -t subbu7677/tomcatcustomimage .
[+] Building 0.4s (6/6) FINISHED
 => [internal] load build definition from dockerfile
 => => transferring dockerfile: 56B
 => [internal] load metadata for docker.io/library/tomcat:9.0.93
 => [auth] library/tomcat:pull token for registry-1.docker.io
 => [internal] load .dockerignore
 => => transferring context: 2B
 => CACHED [1/1] FROM docker.io/library/tomcat:9.0.93@sha256:de248c13aa49d4b245220b65a0194ed8ea82e19cc063604838bd14ad675afcef
 => exporting to image
 => => exporting layers
 => => writing image sha256:f231301390368f9114251820c43185785f611af531efd6d6e0e9203de943929c
 => => naming to docker.io/subbu7677/tomcatcustomimage
```

Can we save Dockerfile with a custom name?

Yes, you can save a Dockerfile with any name you prefer, but by default, Docker looks for a file named Dockerfile in the current directory when building an image.

If you want to use a Dockerfile with a different name, you need to specify that name when building the image using the -f flag.

1. Create the Dockerfile with a Different Name:

Pull a docker image: Ex: docker pull nginx:1.26.1

```
root@ip-172-31-90-246:/home/ubuntu# docker pull nginx:1.26.1
1.26.1: Pulling from library/nginx
e4fff0779e6d: Pull complete
c4d25407c66a: Pull complete
d396f65a74db: Pull complete
8016a4e1cf5a: Pull complete
b42eeb8aaefe: Pull complete
9e433f77c7eb: Pull complete
f563f99f82c6: Pull complete
Digest: sha256:0f0c707b16468ea9b6cc13a315f29d2c84b0fc53c223ee4b3e8b882506343659
Status: Downloaded newer image for nginx:1.26.1
docker.io/library/nginx:1.26.1
```

You can create a Dockerfile with any name, For example: nginxdockerfile

```
root@ip-172-31-90-246:/home/ubuntu# vi nginxdockerfile
root@ip-172-31-90-246:/home/ubuntu# cat nginxdockerfile
FROM nginx:1.26.1
```

```
root@ip-172-31-90-246:/home/ubuntu# ls -lrth
total 8.0K
-rw-r--r-- 1 root root 19 Aug 14 13:30 dockerfile
-rw-r--r-- 1 root root 19 Aug 14 13:48 nginxdockerfile
root@ip-172-31-90-246:/home/ubuntu# docker build -f nginxdockerfile -t nginx .
[+] Building 5.3s (6/6) FINISHED
 => [internal] load build definition from nginxdockerfile
 => => transferring dockerfile: 61B
 => [internal] load metadata for docker.io/library/nginx:1.26.1
 => [auth] library/nginx:pull token for registry-1.docker.io
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/1] FROM docker.io/library/nginx:1.26.1@sha256:0f0c707b16468ea9b6cc13a315f29d2c84b0fc53c223ee4b3e8b882506343659
 => => resolve docker.io/library/nginx:1.26.1@sha256:0f0c707b16468ea9b6cc13a315f29d2c84b0fc53c223ee4b3e8b882506343659
 => => sha256:8e2a56a4e78b45dfe778be9df9bc92a86c1a3b6a749f841431b9c464d10b4cb6 2.29kB / 2.29kB
 => => sha256:cd43ca3ebe4acdc90375a2ec6f1945e3eee504981eb31b00a7746d041189b57c 7.28kB / 7.28kB
 => => sha256:e4fff0779e6ddd22366469f08626c3ab1884b5cbe1719b26da238c95f247b305 29.13MB / 29.13MB
 => => sha256:c4d25407c66ac8c24f0908e40b9c6ad1e0328cf4c785877b2022b1b57b8b54ef 41.83MB / 41.83MB
 => => sha256:d396f65a74db86979ae69b9cf453b51e8c4423208a8c91c4d0bb142433f9ac35 627B / 627B
 => => sha256:0f0c707b16468ea9b6cc13a315f29d2c84b0fc53c223ee4b3e8b882506343659 10.26kB / 10.26kB
 => => sha256:8016a4e1cf5a5b0f8f34afa403acf394cb22fe780a9b6c5b6af95ced9c0b64f5 956B / 956B
 => => sha256:b42eeb8aaefe6647e820c9a6cb992b38a944e4e5572ad6555764117ad5c5285c 394B / 394B
 => => sha256:9e433f77c7eb74fd172927c7c60ba35c733fcc6a47623818be9752286bca8836 1.21kB / 1.21kB
 => => sha256:f563f99f82c61452eb86ba2f9c25941e4abc1f44cee70d0bef029619c6c49e10 1.40kB / 1.40kB
 => => extracting sha256:e4fff0779e6ddd22366469f08626c3ab1884b5cbe1719b26da238c95f247b305
 => => extracting sha256:c4d25407c66ac8c24f0908e40b9c6ad1e0328cf4c785877b2022b1b57b8b54ef
 => => extracting sha256:d396f65a74db86979ae69b9cf453b51e8c4423208a8c91c4d0bb142433f9ac35
 => => extracting sha256:8016a4e1cf5a5b0f8f34afa403acf394cb22fe780a9b6c5b6af95ced9c0b64f5
 => => extracting sha256:b42eeb8aaefe6647e820c9a6cb992b38a944e4e5572ad6555764117ad5c5285c
 => => extracting sha256:9e433f77c7eb74fd172927c7c60ba35c733fcc6a47623818be9752286bca8836
 => => extracting sha256:f563f99f82c61452eb86ba2f9c25941e4abc1f44cee70d0bef029619c6c49e10
 => exporting to image
 => => exporting layers
 => => writing image sha256:cd43ca3ebe4acdc90375a2ec6f1945e3eee504981eb31b00a7746d041189b57c
 => => naming to docker.io/library/nginx
```

```
root@ip-172-31-90-246:/home/ubuntu# docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
nginx         latest    cd43ca3ebe4a   7 weeks ago    188MB
```

You can build with any custom name also:

```
root@ip-172-31-90-246:/home/ubuntu# docker build -f nginxdockerfile -t subbu7677/nginx .
[+] Building 0.3s (6/6) FINISHED
 => [internal] load build definition from nginxdockerfile
 => => transferring dockerfile: 61B
 => [internal] load metadata for docker.io/library/nginx:1.26.1
 => [auth] library/nginx:pull token for registry-1.docker.io
 => [internal] load .dockerignore
 => => transferring context: 2B
 => CACHED [1/1] FROM docker.io/library/nginx:1.26.1@sha256:0f0c707b16468ea9b6cc13a315f29d2c84b0fc53c223ee4b3e8b882506343659
 => exporting to image
 => => exporting layers
 => => writing image sha256:cd43ca3ebe4acdc90375a2ec6f1945e3eee504981eb31b00a7746d041189b57c
 => => naming to docker.io/subbu7677/nginx
root@ip-172-31-90-246:/home/ubuntu# docker image ls
REPOSITORY        TAG       IMAGE ID       CREATED        SIZE
nginx             latest    cd43ca3ebe4a   7 weeks ago    188MB
subbu7677/nginx   latest    cd43ca3ebe4a   7 weeks ago    188MB
```

Docker Containers

A Docker container is a runtime environment with all the necessary components—like code, dependencies, and libraries—needed to run the application code without using host machine dependencies.

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.

By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.

Key commands:

1. Check Running Containers:

To view the running containers: **docker ps**

This will show details such as the container ID, image, command, status, ports, and names.

```
root@ip-172-31-90-246:/home/ubuntu# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

In this example there are no running containers at the moment.

2. View All Containers:

To view all containers, including stopped ones: **docker ps -a**

```
root@ip-172-31-90-246:/home/ubuntu# docker ps -a
CONTAINER ID   IMAGE          COMMAND     CREATED       STATUS                    PORTS     NAMES
ad3ab19a08d7   d2c94e258dcb   "/hello"    2 days ago    Exited (0) 2 days ago               laughing_kapitsa
root@ip-172-31-90-246:/home/ubuntu#
```

In this example one stopped container is there.

3. Create and Run a Container:

To create and run a new container from an image.

You can create and start a Docker container using the **docker run** command.

docker container run

Description	Create and run a new container from an image
Usage	`docker container run [OPTIONS] IMAGE [COMMAND] [ARG...]`
Aliases ?	`docker run`

- IMAGE: The name of the Docker image to use.
- COMMAND: (Optional) The command to run inside the container.
- ARG...: (Optional) Arguments for the command.

Common Options:

- -d: Run the container in detached mode (in the background).

- -it: Run the container in interactive mode with a terminal.
- --name: Assign a name to the container.
- -p: Map container ports to host ports.
- -v: Mount a volume to persist data.
- --rm: Automatically remove the container when it exits.

Ex: Check list of images available.

```
root@ip-172-31-90-246:/home/ubuntu# docker images
REPOSITORY          TAG       IMAGE ID       CREATED        SIZE
nginx               latest    cd43ca3ebe4a   7 weeks ago    188MB
subbu7677/nginx     latest    cd43ca3ebe4a   7 weeks ago    188MB
sonatype/nexus      latest    c55c6ce14194   2 years ago    464MB
```

You can pull a docker image from docker hub or you can create your own docker image to create and run a container.

Once your image is ready the use docker run command:

Syntax: docker run <option1> <option2> --name <container-name> <image-name>: <version_no>

Note: If version no is 'latest' no need to mention separately.

docker run -v -d -p 8081:8081 --name nexusapp sonatype/nexus

- -d: Run the container in detached mode (in the background).
- -p: Map container ports to host ports.

4. Add an Inbound Rule for port 8081:

Add an Inbound rule to open the port 8081 in the AWS EC2 Instance-Security Group where docker is installed:

Inbound rules

Filter rules

Name	Security group rule ID	Port range	Protocol	Source
–	sgr-0f16fc425e594d6ab	80	TCP	0.0.0.0/0
–	sgr-0ae38a32a084c3887	22	TCP	0.0.0.0/0
–	sgr-0d73845aa0f3d9d7b	8081	TCP	0.0.0.0/0

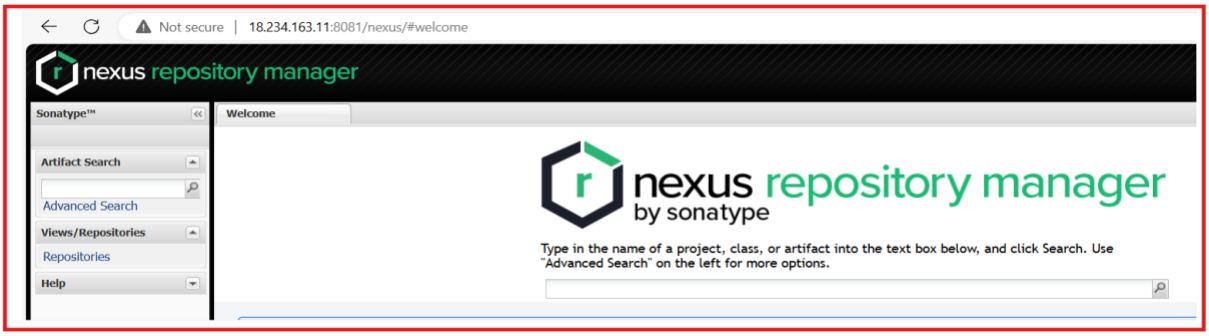
5. Access the Root URL:

Try accessing http://<your-server-ip>:8081/nexus directly.

Nexus Default Path

1. **Default Context Path:**
   - **By default, Sonatype Nexus Repository Manager serves its web application from the /nexus context path. This is defined in the Nexus configuration and Jetty server setup.**
   - **When you access http://18.234.163.11:8081/nexus, you are hitting the default context path for Nexus.**

Ex: http://18.234.163.11:8081/nexus/

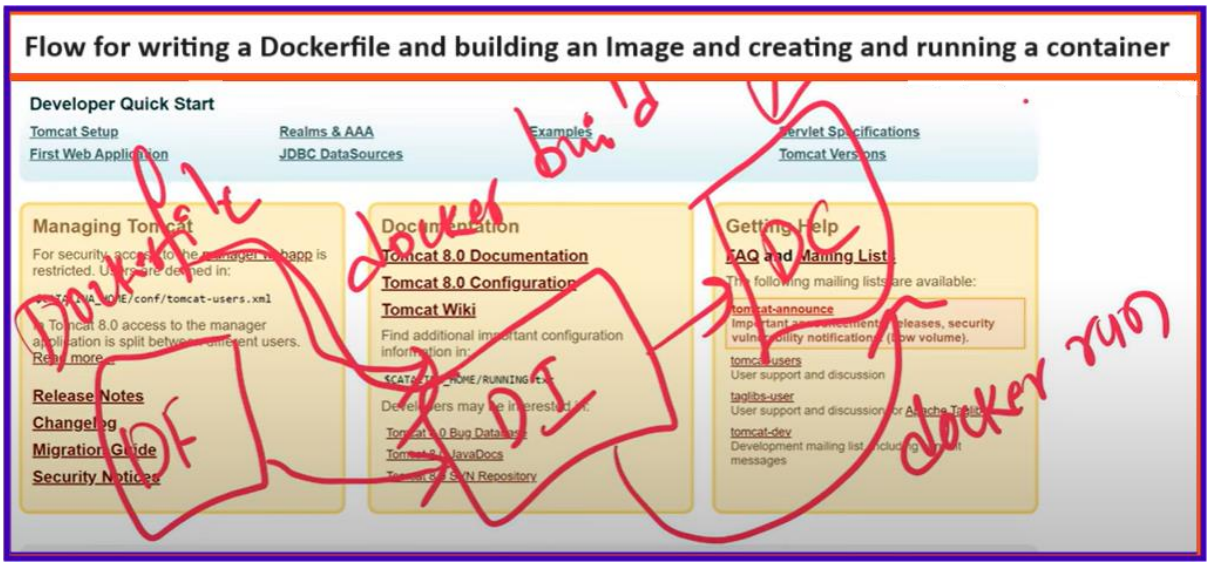


Note:

- Installation of Nexus is to `/opt/sonatype/nexus`. Notably: `/opt/sonatype/nexus/conf/nexus.properties` is the properties file. Parameters (`nexus-work` and `nexus-webapp-context-path`) defined here are overridden in the JVM invocation.
- A persistent directory, `/sonatype-work`, is used for configuration, logs, and storage. This directory needs to be writeable by the Nexus process, which runs as UID 200.
- Environment variables can be used to control the JVM arguments
  - `CONTEXT_PATH`, passed as -Dnexus-webapp-context-path. This is used to define the URL which Nexus is accessed. Defaults to '/nexus'

Building an Image by own:

You can create your own Docker image using a Dockerfile, which is a text document that contains instructions to build an image.



Example Dockerfile for Jenkins:

Here's a `Dockerfile` for Jenkins version `2.361.1`:

dockerfile Copy code

```dockerfile
# Use Jenkins LTS version 2.361.1
FROM jenkins/jenkins:2.361.1

# Switch to the root user to install additional packages
USER root

# Install Docker and other necessary tools
RUN apt-get update && \
    apt-get install -y docker.io maven git

# Switch back to the jenkins user
USER jenkins

# Expose the default Jenkins port
EXPOSE 8080

# Expose the Jenkins agent port
EXPOSE 50000

# Define default Jenkins home directory
VOLUME /var/jenkins_home

# Start Jenkins
CMD ["jenkins"]
```

1. Save the Dockerfile: Save the above content in a file named Dockerfile.

```
root@ip-172-31-90-246:/home/ubuntu# vi JenkinsDockerFile
root@ip-172-31-90-246:/home/ubuntu# ls -lrth
total 16K
-rw-r--r-- 1 root root   19 Aug 14 13:30 dockerfile
-rw-r--r-- 1 root root   19 Aug 14 13:57 nginxdockerfile
-rw-r--r-- 1 root root 1.1K Aug 15 10:52 nexus.properties
-rw-r--r-- 1 root root  483 Aug 15 11:30 JenkinsDockerFile
root@ip-172-31-90-246:/home/ubuntu#
```

```
root@ip-172-31-90-246:/home/ubuntu# cat JenkinsDockerFile
# Use Jenkins LTS version 2.361.1
FROM jenkins/jenkins:2.361.1

# Switch to the root user to install additional packages
USER root

# Install Docker and other necessary tools
RUN apt-get update && \
    apt-get install -y docker.io maven git

# Switch back to the jenkins user
USER jenkins

# Expose the default Jenkins port
EXPOSE 8080

# Expose the Jenkins agent port
EXPOSE 50000

# Define default Jenkins home directory
VOLUME /var/jenkins_home

# Start Jenkins
CMD ["jenkins"]
```

```
root@ip-172-31-90-246:/home/ubuntu# docker image ls
REPOSITORY        TAG       IMAGE ID       CREATED        SIZE
nginx             latest    cd43ca3ebe4a   7 weeks ago    188MB
subbu7677/nginx   latest    cd43ca3ebe4a   7 weeks ago    188MB
sonatype/nexus    latest    c55c6ce14194   2 years ago    464MB
```

2. Build the Docker Image:

Syntax: docker build -f <Custom_Docker_Filename> -t <Tag_Name> .

docker build -f JenkinsDockerFile -t jenkins:2.361.1 .

- **docker build: This is the command used to build a Docker image.**

- **-f JenkinsDockerFile:** This specifies the filename of the Dockerfile. If your Dockerfile is named something other than Dockerfile, you need to use the -f option to point to the correct file. Here, it points to JenkinsDockerFile.
- **-t jenkins:2.361.1:** This tags the resulting Docker image with the name jenkins and the version 2.361.1. The -t option is used to name and optionally tag an image.
- **. (dot):** This specifies the build context, which is the directory containing the Dockerfile and other necessary files. The dot (.) refers to the current directory.

```
root@ip-172-31-90-246:/home/ubuntu# docker build -f JenkinsDockerFile  -t  jenkins:2.361.1 .
[+] Building 74.5s (7/7) FINISHED
 => [internal] load build definition from JenkinsDockerFile
 => => transferring dockerfile: 529B
 => [internal] load metadata for docker.io/jenkins/jenkins:2.361.1
 => [auth] jenkins/jenkins:pull token for registry-1.docker.io
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/2] FROM docker.io/jenkins/jenkins:2.361.1@sha256:5508cb1317aa0ede06cb34767fb1ab3860d1307109
 => => resolve docker.io/jenkins/jenkins:2.361.1@sha256:5508cb1317aa0ede06cb34767fb1ab3860d1307109
 => => sha256:1e010a8344e7540e1092137c97cb7f551abf222fc809ef8e7a3cd2a431efe739 50.67MB / 50.67MB
 => => sha256:1671565cc8df8c365c9b661d3fbc164e73d01f1b0430c6179588428f99a9da2e 55.01MB / 55.01MB
 => => sha256:5508cb1317aa0ede06cb34767fb1ab3860d1307109ade577d5df871f62170214 1.05kB / 1.05kB
 => => sha256:f7406b2e1315a0eb3fce107abb94f5571b4a0180ba4be221acfcfb0fb1977d4b 8.66MB / 8.66MB
 => => sha256:8fceb665e514b324f23dc2535a18ab95b1bdac847f71ae3146a3b00cb9c8cd95 3.25kB / 3.25kB
 => => sha256:729c87ece8d086b05a3a67e1f7b7a7e669c3a50db75ea2440dd6099a3f887111 13.32kB / 13.32kB
 => => sha256:a7516ebe83d28a0ae4fe8babd9142a4ef58adcd28ce0dff073f334f9c873a3ec 1.24kB / 1.24kB
 => => sha256:a51dca64e82b1453f81098eb6afc7ed02b45e36e617dbec445d8a962b8a63533 188B / 188B
 => => sha256:77ef07b6a14139da2df1f61405729961c69f278bbebaefe92ccbfaee9be42013 93.27MB / 93.27MB
 => => extracting sha256:1671565cc8df8c365c9b661d3fbc164e73d01f1b0430c6179588428f99a9da2e
 => => sha256:2ac030a719df05c25f261cc9d3332896768c1acaef7c8a9a9822b6eb2682b36b 194B / 194B
 => => sha256:263bf74244c09fd210857ce76f0071fcac17f72107c8083364feb57b147b7c2b 5.69MB / 5.69MB
 => => sha256:620f54e03b445408dd16affaa767ad0d2ab39734e88a490d0f418a168af5966c 76.76MB / 76.76MB
 => => sha256:59e43d37c90415583a3e91ddc62795d57acaf91488f3b660eb2ef7a49ca6dfc1 1.93kB / 1.93kB
 => => sha256:c9dbe2415122c359ec91408b0ffcec92a867b936c10d84450f0abb0a61c4ba8b 1.17kB / 1.17kB
 => => sha256:2c049b4765e9c609287141bd9590d18a2f167d59d6b15c7f2f78235ee2dc077b 366B / 366B
 => => sha256:c2b2538c867b58c52a3e37b04630052a98b95c36c6101c5a48e58017fabc3e3b 375B / 375B
 => => sha256:57c5d5e596fda31f51c41c550bf0e2b0bed6d7bd12c920b1fc6e396f5d5b9f3b 269B / 269B
 => => extracting sha256:1e010a8344e7540e1092137c97cb7f551abf222fc809ef8e7a3cd2a431efe739
 => => extracting sha256:f7406b2e1315a0eb3fce107abb94f5571b4a0180ba4be221acfcfb0fb1977d4b
 => => extracting sha256:a7516ebe83d28a0ae4fe8babd9142a4ef58adcd28ce0dff073f334f9c873a3ec
 => => extracting sha256:a51dca64e82b1453f81098eb6afc7ed02b45e36e617dbec445d8a962b8a63533
 => => extracting sha256:77ef07b6a14139da2df1f61405729961c69f278bbebaefe92ccbfaee9be42013
 => => extracting sha256:2ac030a719df05c25f261cc9d3332896768c1acaef7c8a9a9822b6eb2682b36b
 => => extracting sha256:263bf74244c09fd210857ce76f0071fcac17f72107c8083364feb57b147b7c2b
 => => extracting sha256:620f54e03b445408dd16affaa767ad0d2ab39734e88a490d0f418a168af5966c
 => => extracting sha256:59e43d37c90415583a3e91ddc62795d57acaf91488f3b660eb2ef7a49ca6dfc1
 => => extracting sha256:c9dbe2415122c359ec91408b0ffcec92a867b936c10d84450f0abb0a61c4ba8b
 => => extracting sha256:2c049b4765e9c609287141bd9590d18a2f167d59d6b15c7f2f78235ee2dc077b
 => => extracting sha256:c2b2538c867b58c52a3e37b04630052a98b95c36c6101c5a48e58017fabc3e3b
 => => extracting sha256:57c5d5e596fda31f51c41c550bf0e2b0bed6d7bd12c920b1fc6e396f5d5b9f3b
 => [2/2] RUN apt-get update &&     apt-get install -y docker.io maven git
 => exporting to image
 => => exporting layers
 => => writing image sha256:3488d95cbeafdd043443a5e244eede1950175ad0e41309e9aed889ebcf469a0b
 => => naming to docker.io/library/jenkins:2.361.1
root@ip-172-31-90-246:/home/ubuntu#
```

Now you can check the available images:

```
root@ip-172-31-90-246:/home/ubuntu# docker image ls
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
jenkins             2.361.1   3488d95cbeaf   6 minutes ago    1.11GB
subbu7677/nginx     latest    cd43ca3ebe4a   7 weeks ago      188MB
nginx               latest    cd43ca3ebe4a   7 weeks ago      188MB
sonatype/nexus      latest    c55c6ce14194   2 years ago      464MB
root@ip-172-31-90-246:/home/ubuntu#
```

Docker image history command:

The docker history command is used to view the history of a Docker image, showing the layers that make up the image and the commands used to create each layer.

Syntax: docker history <image_name>:<tag>

```
root@ip-172-31-90-246:/home/ubuntu# docker history jenkins:2.361.1
IMAGE          CREATED        CREATED BY                                      SIZE      COMMENT
3488d95cbeaf   10 minutes ago CMD ["jenkins"]                                 0B        buildkit.dockerfile.v0
<missing>      10 minutes ago VOLUME [/var/jenkins_home]                      0B        buildkit.dockerfile.v0
<missing>      10 minutes ago EXPOSE map[50000/tcp:{}]                        0B        buildkit.dockerfile.v0
<missing>      10 minutes ago EXPOSE map[8080/tcp:{}]                         0B        buildkit.dockerfile.v0
<missing>      10 minutes ago USER jenkins                                    0B        buildkit.dockerfile.v0
<missing>      10 minutes ago RUN /bin/sh -c apt-get update &&     apt-get…   647MB     buildkit.dockerfile.v0
<missing>      10 minutes ago USER root                                       0B        buildkit.dockerfile.v0
<missing>      23 months ago  LABEL org.opencontainers.image.vendor=Jenkin…   0B        buildkit.dockerfile.v0
<missing>      23 months ago  COPY install-plugins.sh /usr/local/bin/insta…  110B      buildkit.dockerfile.v0
<missing>      23 months ago  ENTRYPOINT ["/usr/bin/tini" "--" "/usr/local…  0B        buildkit.dockerfile.v0
<missing>      23 months ago  COPY jenkins-plugin-cli.sh /bin/jenkins-plug…  323B      buildkit.dockerfile.v0
<missing>      23 months ago  COPY tini-shim.sh /sbin/tini # buildkit        506B      buildkit.dockerfile.v0
<missing>      23 months ago  COPY jenkins.sh /usr/local/bin/jenkins.sh # …  2.15kB    buildkit.dockerfile.v0
<missing>      23 months ago  COPY jenkins-support /usr/local/bin/jenkins-…  6.5kB     buildkit.dockerfile.v0
<missing>      23 months ago  USER jenkins                                   0B        buildkit.dockerfile.v0
<missing>      23 months ago  COPY /javaruntime /opt/java/openjdk # buildk…  101MB     buildkit.dockerfile.v0
<missing>      23 months ago  ENV PATH=/opt/java/openjdk/bin:/usr/local/sb…  0B        buildkit.dockerfile.v0
<missing>      23 months ago  ENV JAVA_HOME=/opt/java/openjdk                 0B        buildkit.dockerfile.v0
<missing>      23 months ago  ENV COPY_REFERENCE_FILE_LOG=/var/jenkins_hom…  0B        buildkit.dockerfile.v0
<missing>      23 months ago  EXPOSE map[50000/tcp:{}]                       0B        buildkit.dockerfile.v0
<missing>      23 months ago  EXPOSE map[8080/tcp:{}]                        0B        buildkit.dockerfile.v0
<missing>      23 months ago  RUN |15 TARGETARCH=amd64 COMMIT_SHA=00d1edcb…  6.27MB    buildkit.dockerfile.v0
```

Check the running containers:

```
root@ip-172-31-90-246:/home/ubuntu# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED       STATUS       PORTS                                          NAMES
1395f3f3122b   sonatype/nexus "/bin/sh -c 'java   …"  3 hours ago   Up 2 hours   0.0.0.0:8081->8081/tcp, :::8081->8081/tcp      nexusapp
root@ip-172-31-90-246:/home/ubuntu# docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED       STATUS                  PORTS                                          NAMES
1395f3f3122b   sonatype/nexus "/bin/sh -c 'java   …"  3 hours ago   Up 2 hours              0.0.0.0:8081->8081/tcp, :::8081->8081/tcp      nexusapp
ad3ab19a08d7   d2c94e258dcb   "/hello"                 2 days ago    Exited (0) 2 days ago                                                  laughing_kapitsa
root@ip-172-31-90-246:/home/ubuntu#
```

Run the Jenkins Container:

Ex:

docker run -d -p 8080:8080 -p 50000:50000 --name my-jenkins jenkins:2.361.1

1. docker run

This is the command used to create and start a new container from a specified Docker image.

2. -d (Detached Mode)

The -d flag runs the container in detached mode, meaning it runs in the background, and you won't see the container's logs in your terminal. This allows you to continue using the terminal while the container runs.

3. -p 8080:8080 (Port Mapping)

The -p option maps a port on the host machine to a port on the container. In this case:

- 8080:8080 maps port 8080 on your host machine to port 8080 inside the container. Jenkins' web interface runs on port 8080 by default, so this allows you to access Jenkins through your host's IP address at http://<host-ip>:8080.

4. -p 50000:50000 (Port Mapping)

Similarly, this maps port 50000 on the host to port 50000 in the container. Jenkins uses this port for connecting to its agents/slaves (nodes). This allows Jenkins agents to connect to the Jenkins master (running in the container) through the host's IP address on port 50000.

5. --name my-jenkins

The --name option assigns a name to the container, making it easier to manage. In this case, the container is named my-jenkins. You can refer to the container by this name in subsequent Docker commands (e.g., docker stop my-jenkins, docker logs my-jenkins).

6. jenkins:2.361.1

This specifies the Docker image from which to create the container. Here, jenkins:2.361.1 refers to the Jenkins Docker image tagged with version 2.361.1.

Run the container:

```
root@ip-172-31-90-246:/home/ubuntu# docker run -d -p 8080:8080 -p 50000:50000 --name my-jenkins jenkins:2.361.1
4eed0d577fa55c90fecec9606bff032f13458e6110238f35986f4d7f786bba57
root@ip-172-31-90-246:/home/ubuntu#
```

Check the list of Containers and start the Jenkins:

```
root@ip-172-31-90-246:/home/ubuntu# docker ps -a
CONTAINER ID   IMAGE              COMMAND                  CREATED              STATUS                      PORTS                                       NAMES
4eed0d577fa5   jenkins:2.361.1    "/usr/bin/tini -- /u…"   About a minute ago   Exited (127) 59 seconds ago                                             my-jenkins
1395f3f3122b   sonatype/nexus     "/bin/sh -c 'java  …"    3 hours ago          Up 3 hours                  0.0.0.0:8081->8081/tcp, :::8081->8081/tcp   nexusapp
ad3ab19a08d7   d2c94e258dcb       "/hello"                 2 days ago           Exited (0) 2 days ago                                                   laughing_kapitsa
root@ip-172-31-90-246:/home/ubuntu# docker start 4eed0d577fa5
4eed0d577fa5
root@ip-172-31-90-246:/home/ubuntu#
```

Due to error in the Dockerfile Jenkins is not started, so I am removing the container and image.

How to stop a container:

```
root@ip-172-31-90-246:/home/ubuntu# docker stop 4eed0d577fa5
4eed0d577fa5
```

How to remove a container:

```
root@ip-172-31-90-246:/home/ubuntu# docker rm 4eed0d577fa5
4eed0d577fa5
root@ip-172-31-90-246:/home/ubuntu#
```

You can remove the docker image by using :

docker rmi <image_id_or_name> or docker rmi -f <image_id_or_name>

Here's the corrected Dockerfile for Jenkins version 2.361.1:

```
# Use Jenkins LTS version 2.361.1
FROM jenkins/jenkins:2.361.1

# Switch to the root user to install additional packages
USER root

# Install Docker and other necessary tools
RUN apt-get update && \
    apt-get install -y docker.io maven git

# Switch back to the jenkins user
USER jenkins

# Expose the default Jenkins port
EXPOSE 8080

# Expose the Jenkins agent port
EXPOSE 50000

# Define default Jenkins home directory
VOLUME /var/jenkins_home

# Start Jenkins using the default startup script
CMD ["/usr/local/bin/jenkins.sh"]
```

Build the image again:

docker build -f JenkinsDockerFile -t jenkins:2.361.1 .

```
root@ip-172-31-90-246:/home/ubuntu# docker build -f JenkinsDockerFile -t jenkins:2.361.1 .
[+] Building 0.6s (7/7) FINISHED
 => [internal] load build definition from JenkinsDockerFile
 => => transferring dockerfile: 580B
 => [internal] load metadata for docker.io/jenkins/jenkins:2.361.1
 => [auth] jenkins/jenkins:pull token for registry-1.docker.io
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/2] FROM docker.io/jenkins/jenkins:2.361.1@sha256:5508cb1317aa0ede06cb34767fb1ab3860d1307109ade577d5df871f62170214
 => CACHED [2/2] RUN apt-get update &&     apt-get install -y docker.io maven git
 => exporting to image
 => => exporting layers
 => => writing image sha256:bb6d7d143f5f2c643ccecf9e4d2b31b63721f17c2d5c6d48a9944b2f4159de52
 => => naming to docker.io/library/jenkins:2.361.1
```

Create and Run the Jenkins container:

docker run -d -p 8080:8080 -p 50000:50000 --name my-jenkins jenkins:2.361.1

```
root@ip-172-31-90-246:/home/ubuntu# docker run -d -p 8080:8080 -p 50000:50000 --name my-jenkins jenkins:2.361.1
f8eae0ea3cd5adfdcca3d1a55ec150bb47be3e78a3ffe969e7898820f079ae58
root@ip-172-31-90-246:/home/ubuntu# docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED         STATUS         PORTS
        NAMES
f8eae0ea3cd5   jenkins:2.361.1    "/usr/bin/tini -- /u…"   6 seconds ago   Up 5 seconds   0.0.0.0:8080->8080/tcp, :::8080->8080/tcp,
0/tcp   my-jenkins
1395f3f3122b   sonatype/nexus     "/bin/sh -c 'java   …"   4 hours ago     Up 3 hours     0.0.0.0:8081->8081/tcp, :::8081->8081/tcp
        nexusapp
root@ip-172-31-90-246:/home/ubuntu#
```

Allow the ports in your AWS EC2 Instance where your Docker is installed:

Inbound rules (5)

Search

Name	Security group rule...	IP version	Type	Protocol	Port range	Source
–	sgr-0f16fc425e594d6ab	IPv4	HTTP	TCP	80	0.0.0.0/0
–	sgr-0ae38a32a084c3887	IPv4	SSH	TCP	22	0.0.0.0/0
–	sgr-0d73845aa0f3d9d7b	IPv4	Custom TCP	TCP	8081	0.0.0.0/0
–	sgr-0ad71cd6818e8e8...	IPv4	Custom TCP	TCP	8080	0.0.0.0/0
–	sgr-0526f40fa9b0f69bb	IPv4	Custom TCP	TCP	50000	0.0.0.0/0

Access the Jenkins on your browser:

Syntax: http://<IP-of-Docker-Server>:8080

http://3.92.228.193:8080

Sign in [Jenkins]

3.92.228.193:8080/login?from=%2F

Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

/var/jenkins_home/secrets/initialAdminPassword

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

Administrator password

If the container uses sh instead of bash, use:

docker exec -it <container_id_or_name> /bin/sh

```
root@ip-172-31-90-246:/home/ubuntu# docker exec -it f8eae0ea3cd5 /bin/sh
$ cd /var/
$ ls -lrth
total 52K
drwxrwxrwt  2 root    root    4.0K Jun 30  2022 tmp
drwxrwsr-x  2 root    staff   4.0K Jun 30  2022 local
drwxr-xr-x  2 root    root    4.0K Jun 30  2022 backups
drwxr-xr-x  2 root    root    4.0K Aug 22  2022 spool
lrwxrwxrwx  1 root    root       4 Aug 22  2022 run -> /run
drwxr-xr-x  2 root    root    4.0K Aug 22  2022 opt
drwxrwsr-x  2 root    mail    4.0K Aug 22  2022 mail
lrwxrwxrwx  1 root    root       9 Aug 22  2022 lock -> /run/lock
drwxr-xr-x  1 root    root    4.0K Aug 15 13:11 log
drwxr-xr-x  1 root    root    4.0K Aug 15 13:11 lib
drwxr-xr-x  1 root    root    4.0K Aug 15 13:11 cache
drwxr-xr-x 12 jenkins jenkins 4.0K Aug 15 14:29 jenkins_home
```

Jenkins Container Homepage: /var/jenkins_home

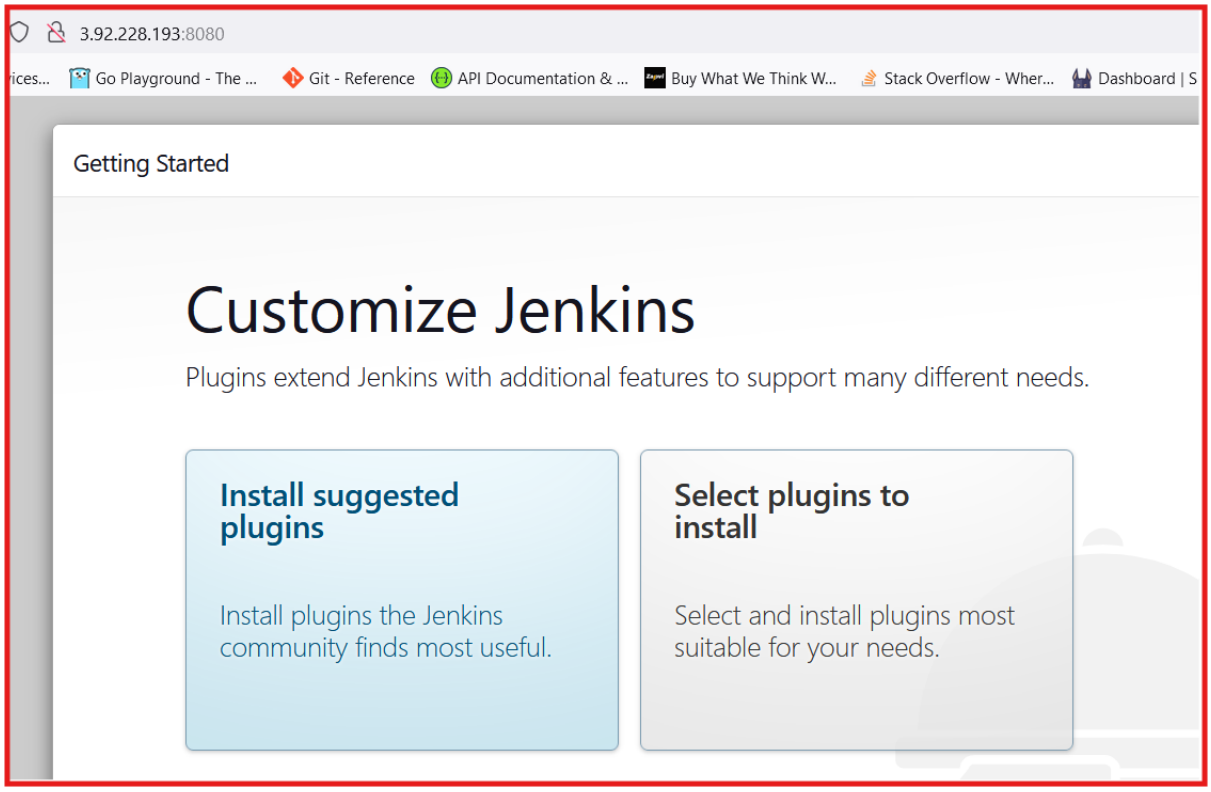
```
$ cd jenkins_home
$ ls -lrth
total 56K
drwxr-xr-x 11 jenkins jenkins 4.0K Aug 15 13:42 war
-rw-r--r--  1 jenkins jenkins   64 Aug 15 13:42 secret.key
-rw-r--r--  1 jenkins jenkins    0 Aug 15 13:42 secret.key.not-so-secret
drwxr-xr-x  2 jenkins jenkins 4.0K Aug 15 13:42 plugins
drwxr-xr-x  2 jenkins jenkins 4.0K Aug 15 13:42 jobs
-rw-r--r--  1 jenkins jenkins  171 Aug 15 13:42 jenkins.telemetry.Correlator.xml
drwxr-xr-x  2 jenkins jenkins 4.0K Aug 15 13:42 nodes
drwxr-xr-x  2 jenkins jenkins 4.0K Aug 15 13:42 userContent
drwxr-xr-x  3 jenkins jenkins 4.0K Aug 15 13:42 users
drwx------  2 jenkins jenkins 4.0K Aug 15 13:42 secrets
-rw-r--r--  1 jenkins jenkins  200 Aug 15 14:29 copy_reference_file.log
-rw-r--r--  1 jenkins jenkins  156 Aug 15 14:29 hudson.model.UpdateCenter.xml
-rw-r--r--  1 jenkins jenkins  907 Aug 15 14:29 nodeMonitors.xml
drwxr-xr-x  2 jenkins jenkins 4.0K Aug 15 14:29 updates
-rw-r--r--  1 jenkins jenkins 1.7K Aug 15 14:29 config.xml
```

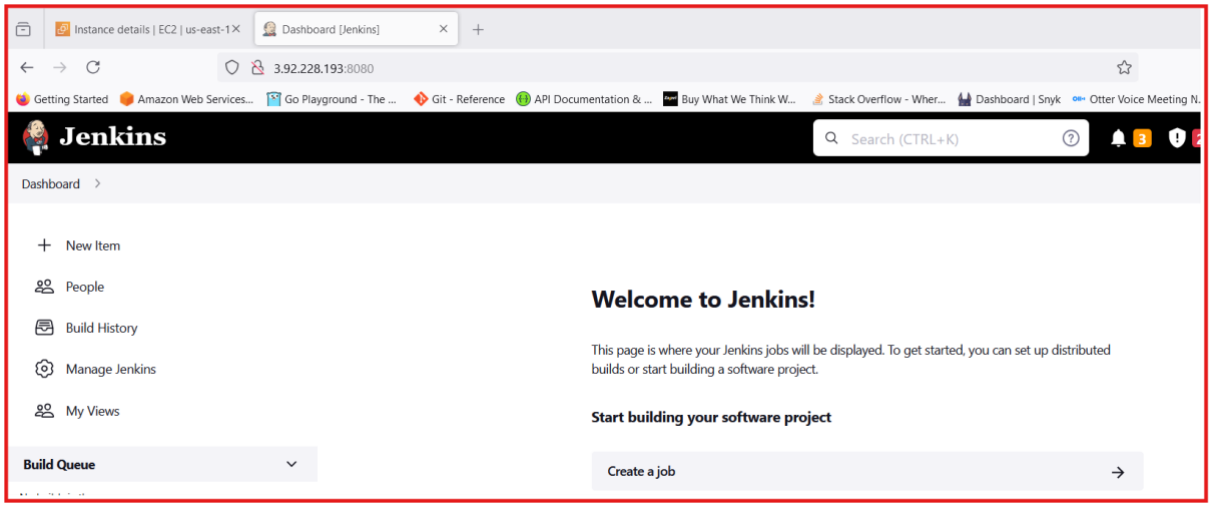
To ensure Jenkins is securely set up by the administrator, a password has been written to the log and this file on the server:

/var/jenkins_home/secrets/initialAdminPassword

```
$ cd secrets
$ ls -lrth
total 12K
-rw-r--r-- 1 jenkins jenkins 256 Aug 15 13:42 master.key
-rw-r--r-- 1 jenkins jenkins  32 Aug 15 13:42 jenkins.model.Jenkins.crumbSalt
-rw-r----- 1 jenkins jenkins  33 Aug 15 13:42 initialAdminPassword
$
```

Once you used the Administrated Password then you can access the Jenkins Homepage:





Like this we can access the container shell.

To exit from a container shell use “exit” command:

```
$ exit
root@ip-172-31-90-246:/home/ubuntu#
```

docker info command:

The docker info command provides detailed information about your Docker installation and its current state. This includes details about Docker's configuration, installed components, running status, and system resources.

```
root@ip-172-31-90-246:/home/ubuntu# docker info
Client: Docker Engine - Community
 Version:    27.1.1
 Context:    default
 Debug Mode: false
 Plugins:
  buildx: Docker Buildx (Docker Inc.)
    Version:  v0.16.1
    Path:     /usr/libexec/docker/cli-plugins/docker-buildx
  compose: Docker Compose (Docker Inc.)
    Version:  v2.29.1
    Path:     /usr/libexec/docker/cli-plugins/docker-compose

Server:
 Containers: 3
  Running: 2
  Paused: 0
  Stopped: 1
 Images: 3
 Server Version: 27.1.1
 Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Using metacopy: false
  Native Overlay Diff: true
  userxattr: false
 Logging Driver: json-file
 Cgroup Driver: systemd
 Cgroup Version: 2
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local splunk syslog
 Swarm: inactive
 Runtimes: io.containerd.runc.v2 runc
 Default Runtime: runc
 Init Binary: docker-init
 containerd version: 2bf793ef6dc9a18e00cb12efb64355c2c9d5eb41
 runc version: v1.1.13-0-g58aa920
 init version: de40ad0
 Security Options:
  apparmor
  seccomp
   Profile: builtin
```

```
  seccomp
   Profile: builtin
  cgroupns
 Kernel Version: 6.8.0-1013-aws
 Operating System: Ubuntu 24.04 LTS
 OSType: linux
 Architecture: x86_64
 CPUs: 1
 Total Memory: 957.4MiB
 Name: ip-172-31-90-246
 ID: 08b0aefb-21d9-47c8-9d00-1e60e803c7af
 Docker Root Dir: /var/lib/docker
 Debug Mode: false
 Username: subbu7677
 Experimental: false
 Insecure Registries:
  127.0.0.0/8
 Live Restore Enabled: false
```

‘Docker inspect’ command:

The docker inspect command provides detailed information about Docker objects such as containers, images, volumes, and networks. It outputs JSON-formatted data, which can be used for in-depth inspection or scripting purposes.

Syntax: docker inspect <object_id_or_name>

We can use this for Docker Images or Containers, Volume, Network

```
root@ip-172-31-90-246:/home/ubuntu# docker image ls
REPOSITORY          TAG         IMAGE ID        CREATED         SIZE
jenkins             2.361.1     bb6d7d143f5f    2 hours ago     1.11GB
nginx               latest      cd43ca3ebe4a    7 weeks ago     188MB
subbu7677/nginx     latest      cd43ca3ebe4a    7 weeks ago     188MB
sonatype/nexus      latest      c55c6ce14194    2 years ago     464MB
```

Ex1: Inspecting a docker image:

```
root@ip-172-31-90-246:/home/ubuntu# docker inspect jenkins:2.361.1
[
    {
        "Id": "sha256:bb6d7d143f5f2c643ccecf9e4d2b31b63721f17c2d5c6d48a9944b2f4159de52",
        "RepoTags": [
            "jenkins:2.361.1"
        ],
        "RepoDigests": [],
        "Parent": "",
        "Comment": "buildkit.dockerfile.v0",
        "Created": "2024-08-15T13:11:44.563845105Z",
        "DockerVersion": "",
        "Author": "",
        "Config": {
            "Hostname": "",
            "Domainname": "",
            "User": "jenkins",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "ExposedPorts": {
                "50000/tcp": {},
                "8080/tcp": {}
            },
```

Ex2: Inspecting a docker container:

```
root@ip-172-31-90-246:/home/ubuntu# docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED        STATUS          PORTS
       NAMES
f8eae0ea3cd5   jenkins:2.361.1    "/usr/bin/tini -- /u…"   2 hours ago    Up 50 minutes   0.0.0.0:8080-
/tcp   my-jenkins
root@ip-172-31-90-246:/home/ubuntu# docker inspect f8eae0ea3cd5
[
    {
        "Id": "f8eae0ea3cd5adfdcca3d1a55ec150bb47be3e78a3ffe969e7898820f079ae58",
        "Created": "2024-08-15T13:42:44.798567699Z",
        "Path": "/usr/bin/tini",
        "Args": [
            "--",
            "/usr/local/bin/jenkins.sh",
            "/usr/local/bin/jenkins.sh"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 1362,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2024-08-15T14:29:12.448994234Z",
            "FinishedAt": "2024-08-15T14:26:15.202918754Z"
```

Key Sections

- **State:** Current state of the container (running, stopped, etc.).
- **Config:** Configuration details such as environment variables, commands, and volumes.
- **NetworkSettings:** Network details including IP addresses, ports, and network configurations.
- **Mounts:** Information on mounted volumes and their configurations.

How to list of a docker images:

docker history:

The docker history command provides a detailed history of an image, showing the layers that make up the image along with their metadata. This is useful for understanding how an image was built, which layers are present, and for diagnosing issues with image construction.

Syntax: **docker history <image_id_or_name>**

Ex: docker history jenkins:2.361.1

```
root@ip-172-31-90-246:/home/ubuntu# docker history jenkins:2.361.1
IMAGE          CREATED        CREATED BY                                      SIZE      COMMENT
bb6d7d143f5f   2 hours ago    CMD ["/usr/local/bin/jenkins.sh"]               0B        buildkit.dockerfile.v0
<missing>      2 hours ago    VOLUME [/var/jenkins_home]                      0B        buildkit.dockerfile.v0
<missing>      2 hours ago    EXPOSE map[50000/tcp:{}]                        0B        buildkit.dockerfile.v0
<missing>      2 hours ago    EXPOSE map[8080/tcp:{}]                         0B        buildkit.dockerfile.v0
<missing>      2 hours ago    USER jenkins                                    0B        buildkit.dockerfile.v0
<missing>      2 hours ago    RUN /bin/sh -c apt-get update &&     apt-get…   647MB     buildkit.dockerfile.v0
<missing>      2 hours ago    USER root                                       0B        buildkit.dockerfile.v0
<missing>      23 months ago  LABEL org.opencontainers.image.vendor=Jenkin…   0B        buildkit.dockerfile.v0
<missing>      23 months ago  COPY install-plugins.sh /usr/local/bin/insta…   110B      buildkit.dockerfile.v0
<missing>      23 months ago  ENTRYPOINT ["/usr/bin/tini" "--" "/usr/local…   0B        buildkit.dockerfile.v0
<missing>      23 months ago  COPY jenkins-plugin-cli.sh /bin/jenkins-plug…   323B      buildkit.dockerfile.v0
<missing>      23 months ago  COPY tini-shim.sh /sbin/tini # buildkit         506B      buildkit.dockerfile.v0
<missing>      23 months ago  COPY jenkins.sh /usr/local/bin/jenkins.sh # …   2.15kB    buildkit.dockerfile.v0
<missing>      23 months ago  COPY jenkins-support /usr/local/bin/jenkins-…   6.5kB     buildkit.dockerfile.v0
<missing>      23 months ago  USER jenkins                                    0B        buildkit.dockerfile.v0
<missing>      23 months ago  COPY /javaruntime /opt/java/openjdk # buildk…   101MB     buildkit.dockerfile.v0
<missing>      23 months ago  ENV PATH=/opt/java/openjdk/bin:/usr/local/sb…   0B        buildkit.dockerfile.v0
<missing>      23 months ago  ENV JAVA_HOME=/opt/java/openjdk                 0B        buildkit.dockerfile.v0
<missing>      23 months ago  ENV COPY_REFERENCE_FILE_LOG=/var/jenkins_hom…   0B        buildkit.dockerfile.v0
<missing>      23 months ago  EXPOSE map[50000/tcp:{}]                        0B        buildkit.dockerfile.v0
<missing>      23 months ago  EXPOSE map[8080/tcp:{}]                         0B        buildkit.dockerfile.v0
<missing>      23 months ago  RUN |15 TARGETARCH=amd64 COMMIT_SHA=00d1edcb…   6.27MB    buildkit.dockerfile.v0
<missing>      23 months ago  ARG PLUGIN_CLI_URL=https://github.com/jenkin…   0B        buildkit.dockerfile.v0
<missing>      23 months ago  ARG PLUGIN_CLI_VERSION=2.12.8                   0B        buildkit.dockerfile.v0
<missing>      23 months ago  RUN |13 TARGETARCH=amd64 COMMIT_SHA=00d1edcb…   0B        buildkit.dockerfile.v0
<missing>      23 months ago  ENV JENKINS_INCREMENTALS_REPO_MIRROR=https:/…   0B        buildkit.dockerfile.v0
<missing>      23 months ago  ENV JENKINS_UC_EXPERIMENTAL=https://updates.…   0B        buildkit.dockerfile.v0
<missing>      23 months ago  ENV JENKINS_UC=https://updates.jenkins.io       0B        buildkit.dockerfile.v0
<missing>      23 months ago  RUN |13 TARGETARCH=amd64 COMMIT_SHA=00d1edcb…   93.5MB    buildkit.dockerfile.v0
<missing>      23 months ago  ARG JENKINS_URL=https://repo.jenkins-ci.org/…   0B        buildkit.dockerfile.v0
<missing>      23 months ago  ARG JENKINS_SHA=1163c4554dc93439c5eef02b06a8…   0B        buildkit.dockerfile.v0
<missing>      23 months ago  ENV JENKINS_VERSION=2.361.1                     0B        buildkit.dockerfile.v0
<missing>      23 months ago  ARG JENKINS_VERSION                             0B        buildkit.dockerfile.v0
<missing>      23 months ago  RUN |10 TARGETARCH=amd64 COMMIT_SHA=00d1edcb…   0B        buildkit.dockerfile.v0
```

Docker's home directory structure

Docker typically stores its data in the /var/lib/docker directory.

```
root@ip-172-31-90-246:/home/ubuntu# cd /var/lib/docker/
root@ip-172-31-90-246:/var/lib/docker# ls -lrth
total 44K
drwx------  4 root root 4.0K Aug 12 14:27 plugins
-rw-------  1 root root   36 Aug 12 14:27 engine-id
drwx------  3 root root 4.0K Aug 12 14:27 image
drwxr-x---  3 root root 4.0K Aug 12 14:27 network
drwx------  2 root root 4.0K Aug 12 14:27 swarm
drwx--x--x  5 root root 4.0K Aug 15 13:10 buildkit
drwx--x---  5 root root 4.0K Aug 15 13:42 containers
drwx------  2 root root 4.0K Aug 15 14:26 runtimes
drwx-----x  6 root root 4.0K Aug 15 14:26 volumes
drwx--x--- 48 root root 4.0K Aug 15 14:26 overlay2
drwx------  2 root root 4.0K Aug 15 14:26 tmp
root@ip-172-31-90-246:/var/lib/docker#
```

Here's a breakdown of the subdirectories within /var/lib/docker:

- **/var/lib/docker/aufs (for systems using the AUFS storage driver):**
  - **Contains metadata and data related to AUFS layers.**
- **/var/lib/docker/containers:**
  - **Contains directories for each container identified by its container ID. These directories store container logs and configuration files.**
- **/var/lib/docker/image:**
  - **Stores images. The subdirectories here will depend on the storage driver being used (e.g., aufs, overlay2).**
- **/var/lib/docker/overlay2 (for systems using the Overlay2 storage driver):**
  - **Contains the overlay file system used by Docker. This directory stores image layers and file system changes.**
- **/var/lib/docker/volumes:**
  - **Contains data for Docker volumes. Each volume has its own directory where data is stored.**
- **/var/lib/docker/network:**
  - **Contains network configuration data for Docker.**
- **/var/lib/docker/swarm (if Docker Swarm mode is used):**

    - **Stores data related to Docker Swarm mode, including cluster state and configuration.**
- **/var/lib/docker/tmp:**
    - **Temporary files used by Docker.**