

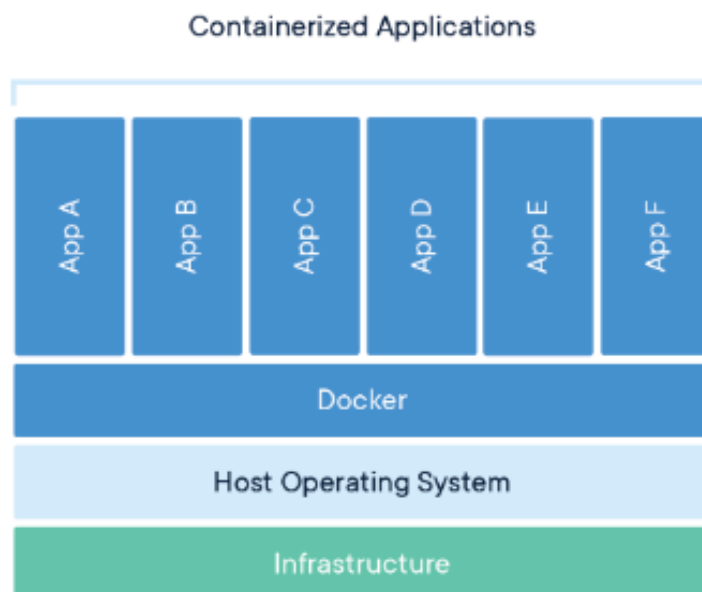
# What is Docker ?

it is a containerization is a concept or technology and **Docker Implements Containerization**.(or) Docker is a containerization platform that provides easy way to containerize your applications, which means, using Docker you can build container images, run the images to create containers and also push these containers to container registries such as DockerHub, Quay.io and so on.



## What is a container ?

A container is a standard unit of software that **packages up code and all its dependencies** so the application runs quickly and reliably from one computing environment to another. Some key points about containers:

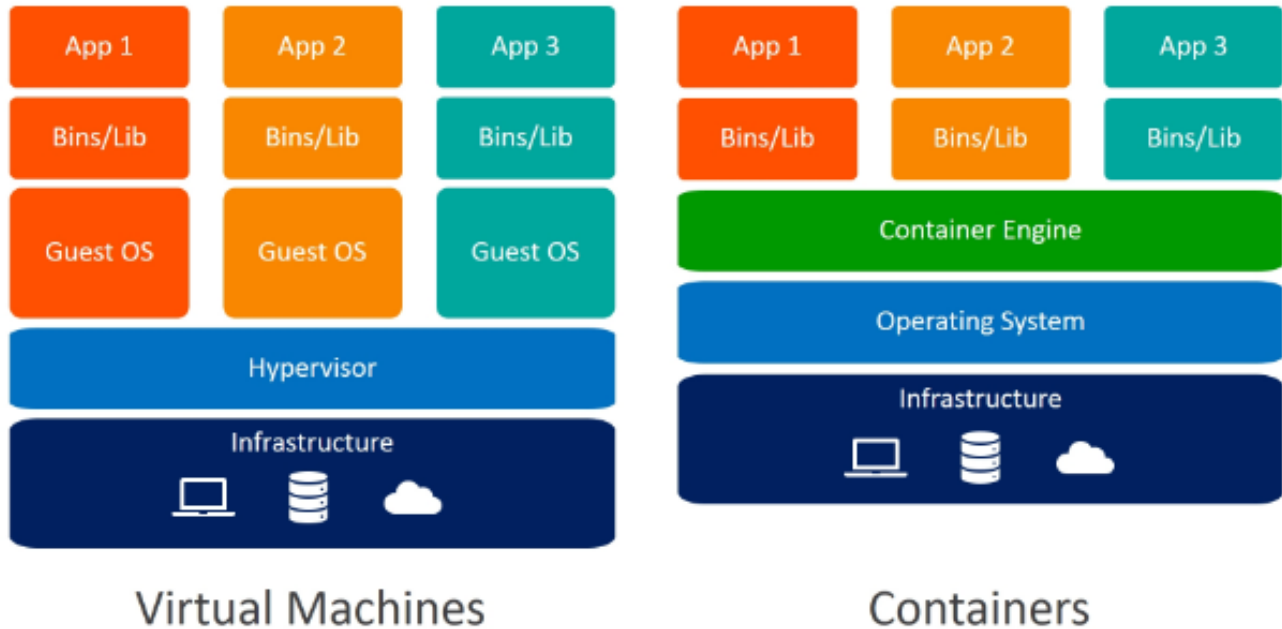


- **Containers** virtualize the operating system, not the hardware. They run as isolated processes on the host operating system.
- All the **dependencies** like libraries, frameworks, binaries, configuration files etc. are bundled inside the container image.
- Containers share the host OS kernel and therefore do not require an entire guest OS to be loaded. This makes them very **lightweight** and fast.
- Common container formats are Docker containers and OCI containers. **Docker** is the most popular containerization platform.
- Containers provide **portable**, isolated environments for applications to run without interference between deployments or infrastructure changes.
- Multiple containers can run on the same machine and share the OS kernel while being **isolated** from each other.
- Containers are **immutable** and stateless in nature. Any data generated is stored externally.
- Containers are spun up from **images** that define the dependencies. Images are reusable, shareable units.
- Container **orchestration platforms** like Kubernetes are used to manage and scale container deployments.

**containers package code and dependencies together into standardized units for developing, shipping and deploying applications rapidly and reliably across environments. They provide consistent, isolated execution environments for apps.**

## **What distinguishes containers from virtual machines?**

- **Virtual Machines (VMs)** fully simulate operating systems, providing robust isolation and supporting diverse OSes, but use more resources and are less portable.
- **Containers**, on the other hand, share the host OS, ensuring efficient resource usage and easy portability for individual apps, making them ideal for modern apps, microservices, and scalability.



The main differences between containers and virtual machines are:

- Containers virtualize the **operating system** whereas VMs virtualize the **hardware**.
- Containers run as **isolated processes on top of a single host** operating system kernel. VMs run a **full guest operating system** kernel on virtualized hardware.
- Containers are **more lightweight and share resources** from the host OS like memory, storage, CPU etc. VMs **dedicate a portion of these resources** per VM.
- Containers **start very quickly, in seconds**, while VMs require **several minutes typically to start up**.
- The image footprint of **containers is small (MBs) compared to VM images (GBs)**.
- Containers are ideal for isolated, **portable application deployment**. VMs provide **full machine virtualization**.
- It takes relatively **fewer resources to run 1000 containers** than 1000 VMs on the same hardware.
- **Containers are immutable and stateless by design**. VMs maintain state like **permanent storage, user accounts, configs etc.**
- Docker and podman are popular container runtimes. Hypervisors like KVM, Xen, VirtualBox etc. run VMs.
- Kubernetes, Docker Swarm provide container orchestration at scale. VMs have hypervisors for management.

# Why are containers light weight ?

Because containers **utilize OS virtualization and share resources**, they provide the **same isolated environment as VMs but in a fraction of the size and with much lower startup times and resource usage**. This makes them very lightweight.

docker hub

ubuntu

Explore

Repositories

Organizations

Help

Upgrade

prudhvivardhan

Explore

Official Images

ubuntu

ubuntu

DOCKER OFFICIAL IMAGE

1B+10K+

Ubuntu is a Debian-based Linux operating system based on free software.

docker pull ubuntu

Overview

Tags

Sort byNewestFilter Tags

TAG

latest

Last pushed 7 days ago by doijanky

DIGEST

56887c5194fd

c835a4f2a632

cf3cc0848a5d

+2 more...

OS/ARCH

linux/amd64

linux/arm/v7

linux/arm64/v8

VULNERABILITIES

0 H2 M9 L

0 H2 M9 L

0 H2 M9 L

COMPRESSED SIZE

28.17 MB

24.93 MB

26.08 MB

docker pull ubuntu:latest

localhost:8888/notebooks/ Docker ( Prudhvi Vardhan Notes).ipynb

4/18

- Containers **share the host OS kernel** instead of needing the entire guest OS image loaded. This avoids duplication and reduces size.
- Containers run as **isolated user space instances** on top of the host OS. They don't emulate underlying hardware like VMs, avoiding overhead.
- Containers package only the **application code, libraries, and dependencies**. The base OS files are shared from the host. This minimized duplication.
- The container format is simply a **bundle of the layered filesystem, configuration files and execution environment**. This is megabytes in size versus GBs for VM images.
- Containers **don't need to boot up an entire OS at start**. The host OS handles boot tasks so containers start almost instantly.
- Resources like **CPU, memory and storage are shared across containers** to maximize efficiency. There's no pre-allocation per VM.
- You can **run many more containers than VMs on the same hardware** since they utilize fewer resources individually.
- Containers are designed to be **ephemeral and stateless**. Their processes run then terminate without persisting state.
- Container images are constructed in **layers, allowing reuse of common layers** across images to reduce duplication.

## What are the files and folders typically found within container base images?

The base image of a container provides the operating system files and folders that serve as the foundation for building containerized applications. base images contain the core filesystem, binaries, libraries, and configurations to launch containers with a basic Linux environment on top of which application images are built. The structure mirrors a standard Linux filesystem layout.

**Here are some key files and folders you can expect to find inside container base images:**

- **/bin** - Contains basic Linux binaries/executables like bash, cat, ls etc. needed to provide a Linux environment.
- **/etc** - System configuration files for services, networking, filesystems, hosts etc.
- **/lib** - Essential shared libraries and kernel modules required by binaries in /bin/ and /sbin/.
- **/sbin** - System binaries/executives, usually for admin tasks like fsck, init, iptables etc.
- **/opt** - Optional software packages added by the base image creator, like Java, Ruby, Python etc.
- **/root** - Home directory for the root user.

- **/tmp** - Temporary folder for transient files created by apps running in container.
- **/var** - Variable data like logs, databases, web app data files etc. Persists beyond container lifecycle.
- **/usr/bin** - Common Linux user binaries and executables.
- **/usr/lib** - Shared libraries for binaries located under /usr/bin/
- **/usr/share** - Architecture-independent data files used by apps and binaries under /usr/.
- **/usr/src** - Source code for software included in the base image.

## INSTALL DOCKER

the detailed steps to build a Docker image from installation:

1. **Install Docker:** If you haven't already, start by installing Docker on your system. You can download the appropriate installer for your operating system from the [Docker website](https://www.docker.com/get-started) (<https://www.docker.com/get-started>).

## Installation:

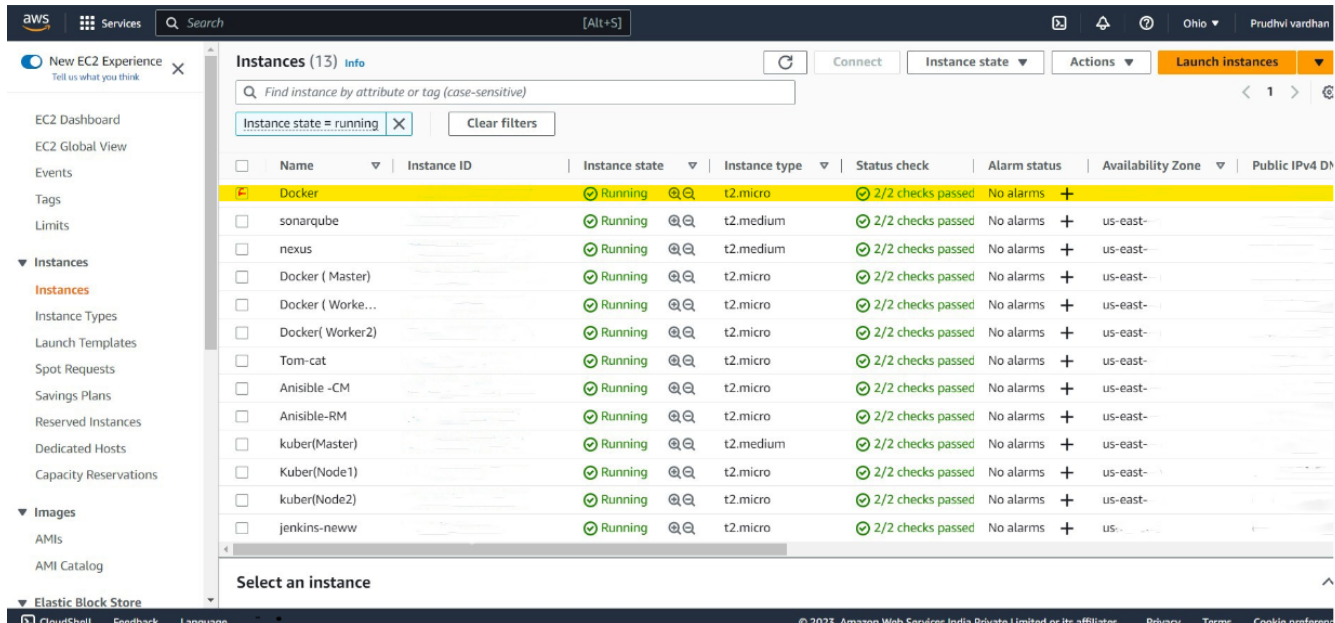
- <https://docs.docker.com/>



### 2. Connect to EC2 Instance:

- Open your terminal.
- Use the `ssh` command to connect to your EC2 instance. Replace `your-key.pem` with the path to your private key and `ec2-user` with the appropriate user for your instance:

```
ssh -i /path/to/your-key.pem ec2-user@your-instance-ip
```



### 3. Update Packages:

- Update the package list and install the required packages:

```
sudo yum update -y
sudo yum install -y docker
```

### 4. Start Docker and Enable on Boot:

- Start the Docker service:

```
sudo service docker start
```

- Enable Docker to start on boot:

```
sudo chkconfig docker on
```

### 5. Add User to Docker Group (Optional):

- By default, Docker commands require root privileges. To run Docker commands without sudo, add your user to the docker group:

```
sudo usermod -aG docker $USER
```

- To apply the group changes, either log out and log back in or run:

```
newgrp docker
```

### Build, Run, and Push Docker Image:

#### 1. Create a Directory:

- Create a directory where you'll organize your Docker image-related files.

#### 2. Create a Dockerfile:

- Inside the directory, create a file named Dockerfile.

#### 3. Write Dockerfile Instructions:

- Write the instructions in the `Dockerfile` to define your image. For example:

```
FROM ubuntu:latest
RUN apt-get update && apt-get install -y curl
CMD ["curl", "http://www.example.com"]
```

```
FROM ubuntu:latest

# set the working directory in the image
WORKDIR /app

# copy the files from the host file system to the image file system
COPY . /app

# Install the necessary packages
RUN apt-get update && apt-get install -y python3 python3-pip

# set environment variables
ENV NAME world

# Run a command to start the application
CMD ["python3", "app.py"]
```

```
ubuntu@ip-172-31-39-12:~$ sudo systemctl start docker
ubuntu@ip-172-31-39-12:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2023-08-23 04:28:08 UTC; 33min ago
     TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
    Main PID: 2299 (dockerd)
       Tasks: 9
      Memory: 240.3M
         CPU: 11.047s
```

```
ubuntu@ip-172-31-39-12:~$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
```

#### 4. Build Docker Image:

- Build the Docker image using the `docker build` command:

```
docker build -t my-image:latest .
```

#### 5. Run Docker Container:

- Run a container based on the built image:

```
docker run my-image:latest
```



## 6. Tag the Image:

- Tag your image to prepare it for pushing to Docker Hub:

```
docker tag my-image:latest prudhvivardhan/my-first-docker-file:latest
```

```
Step 5/6 : ENV NAME world
---> Running in 9e15de276206
Removing intermediate container 9e15de276206
---> c5d274cb8f80
Step 6/6 : CMD ["python3", "app.py"]
---> Running in e82a94b44bbb
Removing intermediate container e82a94b44bbb
---> 3f5c3f8f5263
Successfully built 3f5c3f8f5263
Successfully tagged prudhvivardhan/my-first-docker-file:latest
```

```
ubuntu@ip-172-31-39-1:~/Docker-Zero-to-Hero/examples/first-docker-file$ docker run -it prudhvivardhan/my-first-docker-file
Hello world
ubuntu@ip-172-31-39-1:~/Docker-Zero-to-Hero/examples/first-docker-file$ docker images
REPOSITORY          TAG          IMAGE ID          CREATED          SIZE
prudhvivardhan/my-first-docker-file  latest      3f5c3f8f5263     3 minutes ago   469MB
ubuntu               latest      01f29b872827     2 weeks ago     77.8MB
hello-world          latest      9c7a54a9a43c     3 months ago    13.3kB
```

## 7. Push Docker Image:


- Log in to Docker Hub (if not already):


```
docker login
```

- Push the tagged image to Docker Hub:


```
docker push prudhvivardhan/my-first-docker-file:latest
```

```
ubuntu@ip-172-31-39-1:~/Docker-Zero-to-Hero/examples/first-docker-file$ docker push prudhvivardhan/my-first-docker-file:latest
The push refers to repository [docker.io/prudhvivardhan/my-first-docker-file]
97a46f576943: Pushing [=====] 62.6MB/391MB
97046aea9ec7: Pushed
299aa167a669: Pushed
bce45ce613d3: Mounted from library/ubuntu
```



[Explore](#)
[Repositories](#)
[Organizations](#)
[Help](#)
[Upgrade](#)

prudhvivardhan

[prudhvivardhan](#)
[Repositories](#)
[my-first-docker-file](#)
[latest](#)



**prudhvivardhan/my-first-docker-file:latest**

DIGEST: sha256:4850d209fa2599beb81

OS/ARCH	COMPRESSED SIZE	LAST PUSHED	TYPE
linux/amd64	176.94 MB	a few seconds ago by <a href="#">prudhvivardhan</a>	Image

[Delete Tag](#)

[Image Layers](#)
[Vulnerabilities](#)

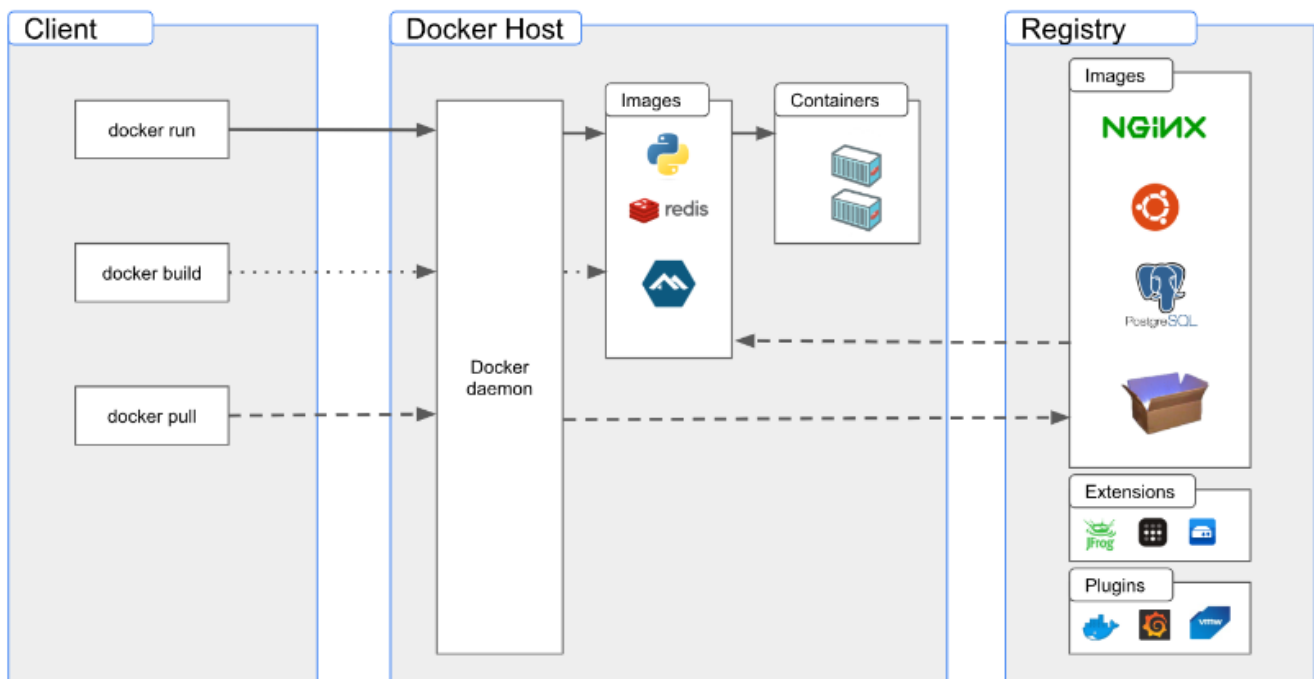
```

ubuntu@ip-172-31-39-...:~/Docker-Zero-to-Hero/examples/first-docker-file$ docker rmi -f prudhivardhan/my-first-docker-file
Untagged: prudhivardhan/my-first-docker-file:latest
Deleted: sha256:1b0de56ff8c32f820358e6550a4dbb8dad57c2575968d1a91dad9a7e8e0c263f
Deleted: sha256:399fa8282f479f32681a1e76f439d95f94df9aad770c56cec8f0acb4fb22e217
Deleted: sha256:c3ed5e9da92c85841fe30463fd0c36e3ccb6130fb03cac60411acaa2026c0bf7
Deleted: sha256:54ec72da3ef9eb661dfce158959beed748691714f7ed416eba0035264efb36d3
Deleted: sha256:8fc1060933c0623f993e39344f305dcdeff3a70d4b6aa0c24461fc53e2d9643c

```

## Explain Docker architecture in Detail ?

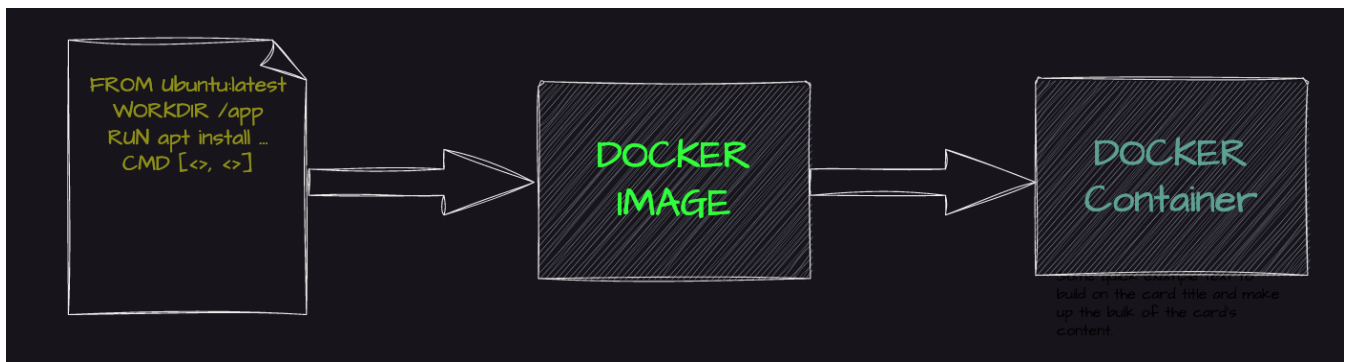
Docker's architecture is designed to provide an efficient and consistent way to package, distribute, and run applications using containers. It promotes isolation, portability, and scalability while making it easier to manage complex application deployments.



1. **Docker Client:** The Docker client is a **command-line tool or API interface** that users interact with to manage Docker containers and images. It sends commands to the Docker daemon for execution.
2. **Docker Daemon:** The Docker daemon (or Docker engine) is the **core background process** responsible for building, running, and managing containers. It listens to Docker API requests from the Docker client, handles **container lifecycle**, and manages the container's **file system, networks, and storage**.
3. **Container Images:** Images are **read-only templates** that define the application and its dependencies. They include everything needed to run a container, such as code, runtime, system tools, libraries, and settings. Images are built from a set of instructions specified in a **Dockerfile**.
4. **Docker Registry:** A Docker registry is a **repository** that stores and distributes Docker images. Docker Hub is a popular public registry, but you can also set up **private registries**. You can push and pull images to and from registries to share or distribute container images.

5. **Docker Container:** A container is a **runtime instance of an image**. It encapsulates the application and its dependencies, running in an isolated environment. Containers **share the host OS kernel** but have their own isolated file system, processes, and network.
6. **Docker Compose:** Docker Compose is a tool for defining and running **multi-container Docker applications**. It uses a **YAML file** to define the services, networks, and volumes needed for an application and then starts and manages those containers as a single application.
7. **Docker Swarm:** Docker Swarm is Docker's native **clustering and orchestration solution**. It allows you to create and manage a **cluster of Docker nodes**, enabling features like **load balancing, scaling, rolling updates, and service discovery**.
8. **Docker CLI:** The Docker **command-line interface (CLI)** is used to interact with Docker

## There are three important things in Docker

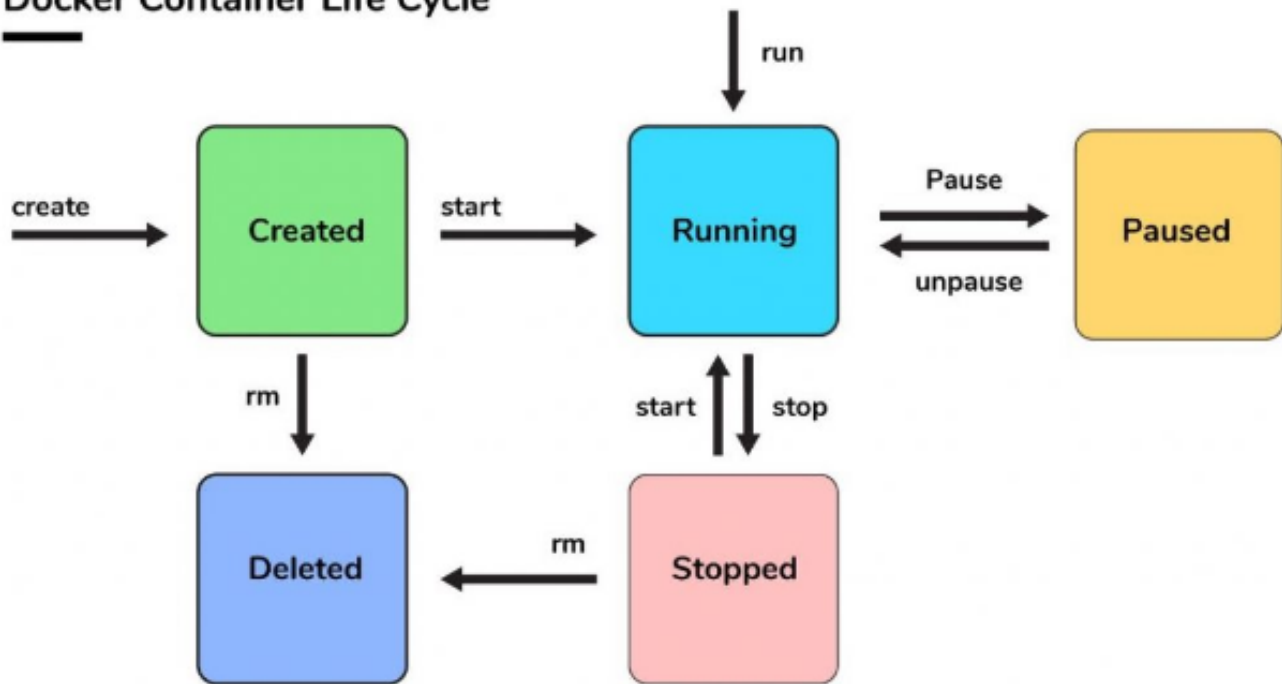


- **docker build** -> builds docker images from Dockerfile
- **docker run** -> runs container from docker images
- **docker push** -> push the container image to public/private registries to share the docker images.

## What are the main stages in the lifecycle of a Docker container?

The Docker container lifecycle involves creating images, running containers, managing their states, and updating as needed. This process allows for flexibility, scalability, and consistent deployment across different environments. The lifecycle of a Docker container involves several stages, from creating an image to running and managing containers. Here's a breakdown of the Docker container lifecycle:

## Docker Container Life Cycle



### Image Creation:

- **Dockerfile:** Create a Dockerfile outlining build instructions, including selecting a base image, copying files, installing software, and configuring settings.

### Container Creation:

- **Run Container:** Use `docker run` to create a container based on an image. Configure options like ports, volumes, and environment variables.

### Container Management:

- **Interact with Container:** Communicate with running containers through the terminal, attaching, executing commands, and monitoring output.
- **Background Execution:** Launch containers in the background with `-d` flag for daemon mode.

### Container States:

- **Running:** Container actively executes its defined process.
- **Paused:** Containers' processes are paused without stopping the container.
- **Stopped:** Containers are halted using `docker stop`.
- **Exited:** Containers that have stopped or were manually terminated.

### Container Removal:

- **Remove Container:** Use `docker rm` to eliminate stopped containers. `-f` flag removes running containers forcefully.

### Image Push and Pull:

- **Push Image:** Share images using `docker push` to a registry.
- **Pull Image:** Fetch images from a registry using `docker pull`.

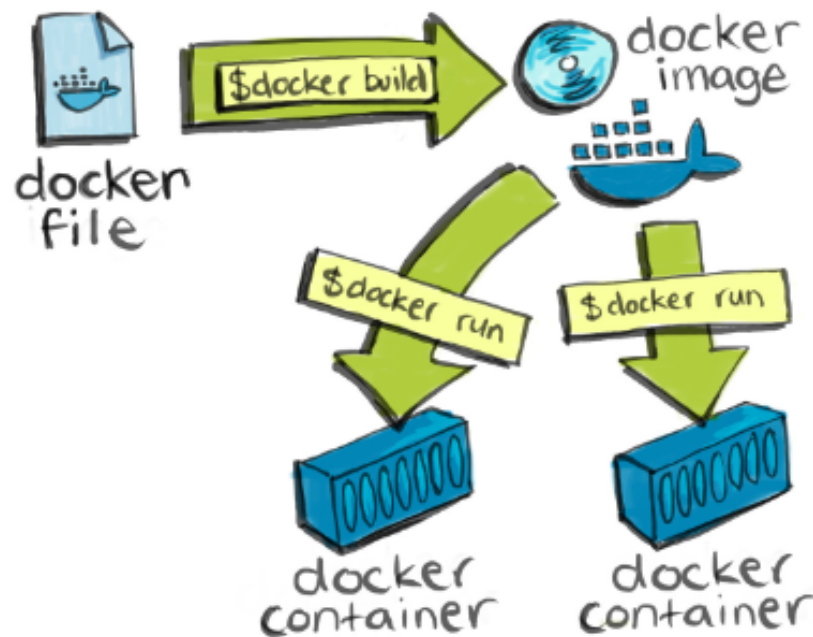
### Container Updates:

- **Image Updates:** Modify the Dockerfile and rebuild images to update dependencies or code.
- **Container Redeployment:** Stop, remove existing containers, and create new ones from updated images.

## Basic Docker commands

These commands cover the basics of managing images and containers using Docker. As you become more comfortable with Docker, you can explore more advanced features and options.

- Remember to replace `<image-name>`, `<tag>`, `<container-id>`, and other placeholders with actual values. This cheat sheet provides a quick reference to essential Docker commands and concepts, but there are many more advanced features and options to explore



### Docker Basics:

- `docker --version` : Display Docker version.
- `docker info` : Display system-wide information.
- `docker login` : Log in to a Docker registry.
- `docker logout` : Log out from a Docker registry.
- `docker search <image-name>` : Search for Docker images on Docker Hub.

### Image Management:

- `docker images` : List images.
- `docker pull <image-name>` : Pull an image from Docker Hub.
- `docker build -t <image-name>:<tag> <path>` : Build an image from a Dockerfile.

### Container Management:

- `docker ps` : List running containers.
- `docker ps -a` : List all containers.
- `docker run -it <image-name>` : Run a container interactively.
- `docker start <container-id>` : Start a stopped container.
- `docker stop <container-id>` : Stop a running container.
- `docker restart <container-id>` : Restart a container.
- `docker rm <container-id>` : Remove a stopped container.
- `docker rm -f <container-id>` : Forcefully remove a running container.

### Container Interaction:

- `docker exec -it <container-id> <command>` : Run a command in a running container.
- `docker logs <container-id>` : Fetch logs from a container.
- `docker cp <local-path> <container-id>:<container-path>` : Copy files between host and container.

### Networking:

- `docker network ls` : List Docker networks.
- `docker network create <network-name>` : Create a Docker network.
- `docker run --network <network-name> ...` : Run a container in a specific network.

### Image and Container Cleanup:

- `docker image prune` : Remove unused images.
- `docker container prune` : Remove all stopped containers.
- `docker system prune` : Remove unused containers, networks, and images.

### Compose (Multi-Container Applications):

- `docker-compose up` : Create and start containers defined in a Compose file.
- `docker-compose down` : Stop and remove containers defined in a Compose file.

# Docker Cheat Sheet

1. **docker attach:** Attach local standard input, output, and error streams to a running container.
2. **docker build:** Build an image from a Dockerfile.
3. **docker commit:** Create a new image from a container's changes.
4. **docker cp:** Copy files/folders between a container and the local filesystem.
5. **docker create:** Create a new container but do not start it.
6. **docker diff:** Show changes to files/folders in a container's filesystem.
7. **docker events:** Get real-time events from the server.
8. **docker exec:** Run a command in a running container.
9. **docker export:** Export a container's filesystem as a tar archive.
10. **docker history:** Show the history of an image.
11. **docker images:** List images.
12. **docker import:** Import the contents from a tarball to create a filesystem image.
13. **docker info:** Display system-wide information.
14. **docker inspect:** Display detailed information on one or more containers, images, networks, or volumes.
15. **docker kill:** Kill a running container.
16. **docker load:** Load an image from a tar archive or STDIN.
17. **docker login:** Log in to a Docker registry.
18. **docker logout:** Log out from a Docker registry.
19. **docker logs:** Fetch the logs of a container.
20. **docker pause:** Pause all processes within a container.
21. **docker port:** List port mappings or a specific mapping for a container.
22. **docker ps:** List containers.

23. **docker pull:** Pull an image or a repository from a registry.
24. **docker push:** Push an image or a repository to a registry.
25. **docker rename:** Rename a container.
26. **docker restart:** Restart a container.
27. **docker rm:** Remove one or more containers.
28. **docker rmi:** Remove one or more images.
29. **docker run:** Run a command in a new container.
30. **docker save:** Save one or more images to a tar archive (streamed to STDOUT by default).
31. **docker search:** Search the Docker Hub for images.
32. **docker start:** Start one or more stopped containers.
33. **docker stats:** Display a live stream of container(s) resource usage statistics.
34. **docker stop:** Stop one or more running containers.
35. **docker tag:** Create a tag TARGET\_IMAGE that refers to SOURCE\_IMAGE.
36. **docker top:** Display the running processes of a container.
37. **docker unpause:** Unpause all processes within a container.
38. **docker update:** Update configuration of one or more containers.
39. **docker version:** Show the Docker version information.
40. **docker volume:** Manage volumes.
41. **docker wait:** Block until a container stops, then print the container's exit code.

### Docker Swarm:

1. `docker swarm init` : Initialize a Docker swarm.
2. `docker swarm join` : Join a Docker swarm as a node.
3. `docker service create` : Create a new service.
4. `docker service ls` : List services.
5. `docker service ps` : List the tasks of a service.
6. `docker service inspect` : Display detailed information about a service.

### Docker Compose:

1. `docker-compose up` : Start services defined in a Compose file.



2. `docker-compose down` : Stop and remove containers defined in a Compose file.
3. `docker-compose ps` : List services defined in a Compose file.

### Docker Network:

1. `docker network create` : Create a Docker network.
2. `docker network ls` : List Docker networks.
3. `docker network inspect` : Display detailed information about a network.

### Docker Volume:

1. `docker volume create` : Create a Docker volume.
2. `docker volume ls` : List Docker volumes.
3. `docker volume inspect` : Display detailed information about a volume.

### Image Build and Push:

1. `docker build -t <image-name>:<tag> <path>` : Build an image from a Dockerfile.
2. `docker tag <source-image> <target-image>` : Tag an image with a new name.
3. `docker push <image-name>:<tag>` : Push an image to a registry.

### Container Interaction:

1. `docker exec -it <container-id> <command>` : Run a command in a running container interactively.
2. `docker logs -f <container-id>` : Stream container logs in real-time.
3. `docker cp <local-path> <container-id>:<container-path>` : Copy files between host and container.

### Miscellaneous:

1. `docker version` : Show Docker version information.
2. `docker system df` : Display a summary of Docker disk usage.
3. `docker system events` : Display system-wide events.
4. `docker system prune -a` : Remove all unused images, containers, networks, and volumes.

-- Prudhvi Vardhan ( [LinkedIn](#) )