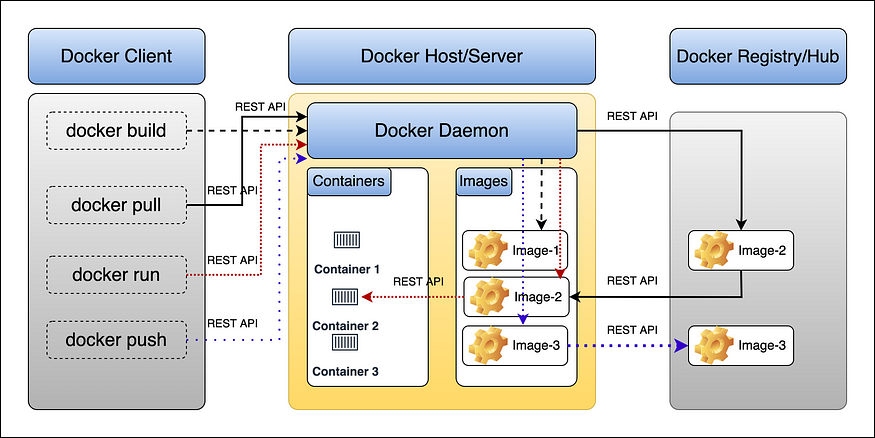
**Docker**

Docker is a set of Platforms as a service (PaaS) products that use Operating system-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries, and configuration files; they can communicate with each other through well-defined channels. All containers are run by a single operating system kernel and therefore use fewer resources than a virtual machine.

Docker gained its popularity due to its impact on the software development and deployment. The following are the some of the main reasons for docker becoming popular:

1. **Portability:**Docker facilitates the developers in packaging their applications with all dependencies into a single lightweight containers. It facilities in ensuring the consistent performance across the different computing environments.
2. **Reproducibility:**Through encapsulating the applications with their dependencies within a container it ensures in software setups remaining consistent across the development, testing and production environments.
3. **Efficiency:** Docker through its container based architecture it optimizes the resource utilization. It allows the developers to run the multiple isolated applications on a single host system.
4. **Scalability:**Docker’s scalability features facilitated the developers in making easier of their applications handling at time of workloads increment.

# Docker Architecture:



* **Docker uses a client-server architecture.**
* The Docker client talks to the Docker daemon(**dockerd)**, **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, which lets you work with applications consisting of a set of containers.

## Let’s understand Some Docker terminologies :

## 1. The Docker Daemon:

1. The Docker daemon (**dockerd**) listens for Docker API requests (docker build, docker run, docker push, docker pull, etc).
2. Manages Docker objects such as **images**, **containers**, **networks**, and **volumes**. A daemon can also communicate with other daemons to manage Docker services.

## 2. The Docker Client:

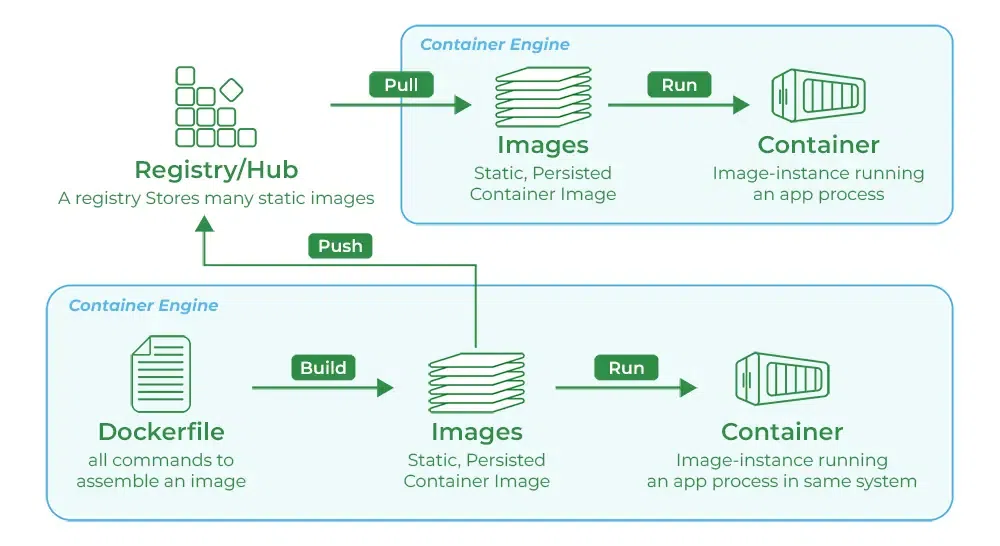
1. The Docker client (**docker**) is the primary way that many Docker users interact with Docker.
2. When you use commands such as **docker run**, the client sends this command to the **dockerd**, which carries them out and processes it.
3. Internally the docker command uses the Docker REST APIs.
4. The Docker client can communicate with more than one daemon.

## 3. Docker Host

* Docker Host is used to provide an environment to execute and run containerized applications.
* It contains the docker **daemon**, **images**, **containers**, **networks**, **volumes**, and **storage**.

## 4. Docker Registries:

1. A Docker registry stores Docker images.
2. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry.
3. When you use the **docker pull** or **docker run** commands, the required images are pulled from your configured registry. When you use the **docker push** command, your image is pushed to your configured registry.



## 5. Docker Objects:

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, etc these are called docker objects.

**Advantages of Docker**

There are the following advantages of Docker -

* It runs the container in seconds instead of minutes.
* It uses less memory.
* It provides lightweight virtualization.
* It does not a require full operating system to run applications.
* It uses application dependencies to reduce the risk.
* Docker allows you to use a remote repository to share your container with others.
* It provides continuous deployment and testing environment.

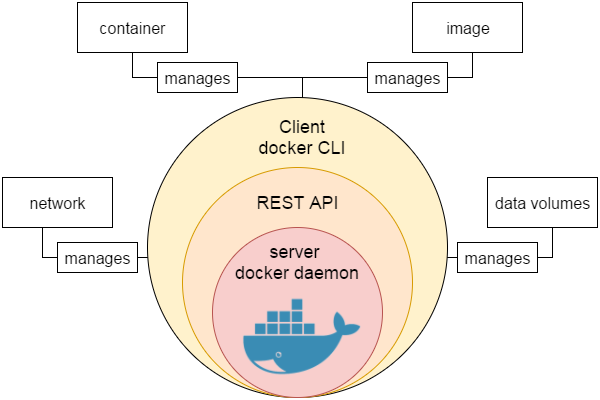
**Disadvantages of Docker**

There are the following disadvantages of Docker -

* It increases complexity due to an additional layer.
* In Docker, it is difficult to manage large amount of containers.
* Some features such as container self -registration, containers self-inspects, copying files form host to the container, and more are missing in the Docker.
* Docker is not a good solution for applications that require rich graphical interface.
* Docker provides cross-platform compatibility means if an application is designed to run in a Docker container on Windows, then it can't run on Linux or vice versa.

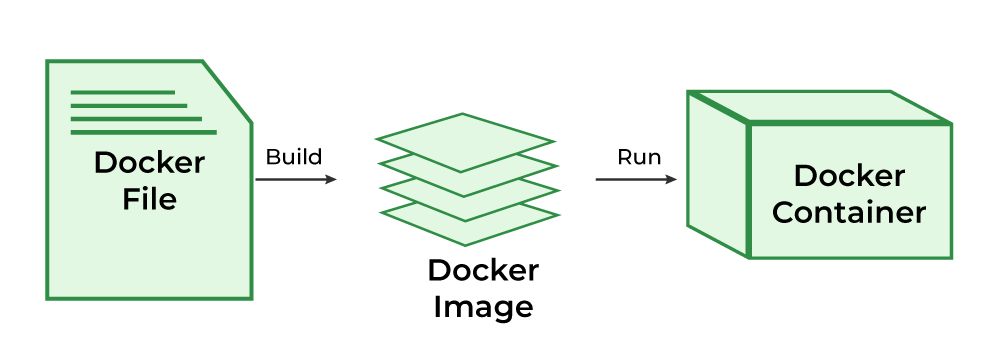
**Docker Engine**

It is a client server application that contains the following major components.

* A server which is a type of long-running program called a daemon process.
* The REST API is used to specify interfaces that programs can use to talk to the daemon and instruct it what to do.
* A command line interface client.
* 

## **What is Dockerfile?**

The Dockerfile uses DSL (Domain Specific Language) and contains instructions for generating a Docker image. Dockerfile will define the processes to quickly produce an image. While creating your application, you should create a Dockerfile in order since the Docker daemon runs all of the instructions from top to bottom.



## **What is Docker Image?**

An artifact with several layers and a lightweight, compact stand-alone executable package that contains all of the components required to run a piece of software, including the code, a runtime, libraries, environment variables, and configuration files is called a [Docker image](https://www.geeksforgeeks.org/what-is-docker-images/).

## What is Docker Container?

**A container is a runtime instance of an image.** Containers make development and deployment more efficient since they contain all the dependencies and parameters needed for the application it runs completely isolated from the host environment.

## Dockerfile commands/Instructions

### **1. FROM**

Represents the base image(OS), which is the command that is executed first before any other commands.

#### **Syntax:**

*FROM <ImageName>*

**Example:**The base image will be ubuntu:19.04

### **2. COPY**

The copy command is used to copy the file/folders to the image while building the image.

#### **Syntax:**

*COPY <Source> <Destination>*

**Example:** Copying the .war file to the .war/ .jar files from java application output from directory

COPY target/java-web-app.war /usr/local/tomcat/webapps/java-web-app.war

### **3. ADD**

While creating the image, we can download files from distant HTTP/HTTPS destinations using the ADD command.

#### **Syntax:**

*ADD <URL>*

**Example:**Try to download Jenkins using ADD command

ADD https://get.jenkins.io/war/2.397/jenkins.war

### **4. RUN**

Scripts and commands are run with the RUN instruction. The execution of RUN commands or instructions will take place while you create an image on top of the prior layers (Image).

#### **Syntax:**

*RUN < Command + ARGS>*

**Example:**

RUN touch file

### **5. CMD**

The main purpose of the CMD command is to start the process inside the container and it can be overridden.

#### **Syntax:**

*CMD [command + args]*

**Example:**Starting Jenkins

CMD ["java","-jar", "Jenkins.war"]

### **6. ENTRYPOINT**

A container that will function as an executable is configured by ENTRYPOINT. When you start the Docker container, a command or script called ENTRYPOINT is executed. It can’t be overridden.The only difference between CMD and ENTRYPOINT  is CMD can be overridden and ENTRYPOINT can’t.

#### **Syntax:**

*ENTRYPOINT [command + args]*

**Example:** Executing the **echo command.**

ENTRYPOINT["echo","Welcome "]

### **7. MAINTAINER**

By using the MAINTAINER command we can identify the author/owner of the Dockerfile and we can set our own author/owner for the image.

#### **Syntax:**

*MAINTAINER <NAME>*

**Example:** Setting the author for the image as a Macys author.

MAINTAINER macys author

## Docker Login Commands

| **Name** | **Command** |
| --- | --- |
| **Log in to a Registry** | docker login |
| **Logout from a Registry** | docker logout |

## Image Management Commands

Docker images are self-contained software packages that contain all the necessary components to run an application. These components include the code, runtime, system tools, system libraries, and settings. Docker images are lightweight and easy to use.

| **Name** | **Command** |
| --- | --- |
| Build an image | docker build -t <image\_name> |
| **Pulling an Image** | docker image pull nginx |
| **Pulling an Image Example** | docker image pull <Name of The Image>:<Tag> |

## Image Transfer Commands

| **Name** | **Command** |
| --- | --- |
| **Pushing an Image** | docker image push <usernameofregistry:Imagename: tag> |
| **Pushing an Image Example** | docker image push eon01/nginx localhost:5000/myadmin/nginx |

## Docker Hub Commands

Docker Hub is a service provided by Docker for finding and sharing container images with your team. Learn more and find images at “***https://hub.docker.com”.***

| **Name** | **Command** |
| --- | --- |
| **Login into Docker** | -docker login -u <username> |
| **Publish an image to Docker Hub** | -docker push <username>/<image\_name> |
| **Search Hub for an image** | -docker search <image\_name> |
| **Pull an image from a Docker Hub** | -docker pull <image\_name> |

## General Docker Commands

| **Name** | **Command** |
| --- | --- |
| **Start the docker daemon** | docker -d |
| **Get help with Docker. Can also use –help on all subcommands** | docker –help |
| **Display system-wide information** | docker info |

## Containers Management Commands

### **CONTAINERS**

A docker image’s runtime instance is referred to as a container. The container remains consistent regardless of the infrastructure in use. This isolation of software from its environment guarantees uniformity in function, even in cases where there are discrepancies between development and staging.

| **Name** | **Command** |
| --- | --- |
| **Starting Containers** | docker container start nginx |
| **Stopping Containers** | docker container stop nginx |
| **Restarting Containers** | docker container restart nginx |
| **Pausing Containers** | docker container pause nginx |
| **Unpausing Containers** | docker container unpause nginx |
| **Blocking a Container** | docker container wait nginx |
| **Sending SIGKILL Containers** | docker container kill nginx |
| **Sending another signal** | docker container kill -s HUP nginx |
| **Connecting to an Existing Container** | docker container attach nginx |
| **Check the Containers** | docker ps |
| **To see all running containers** | docker container ls |
| **Container Logs** | docker logs infinite |
| **‘tail -f’ Containers’ Logs** | docker container logs infinite -f |
| **Inspecting Containers** | docker container inspect infinite |
| **Inspecting Containers for certain** | docker container inspect –format ‘{{ .NetworkSettings.IPAddress }}’ $(docker ps -q) |
| **Containers Events** | docker system events infinite |
| **docker system events infinite** | docker container port infinite |
| **Running Processes** | docker container top infinite |
| **Container Resource Usage** | docker container stats infinite |
| **Inspecting changes to files or directories on a container’s filesystem** | docker container diff infinite |

## Docker Image Management Commands

| **Name** | **Command** |
| --- | --- |
| **Listing Images** | docker image ls |
| **Building Images** | docker build. |
| **From a Remote GIT Repository** | docker build github.com/creack/docker-firefox |
| **Instead of Specifying a Context, You Can Pass a Single Dockerfile in the URL or Pipe the File in via STDIN** | docker build – < Dockerfile |
| **Building and Tagging** | docker build -t eon/infinite. |
| **Building a Dockerfile while Specifying the Build Context** | docker build -f myOtherDockerfile. |
| **Building from a Remote Dockerfile URI** | curl example.com/remote/Dockerfile | docker build -f – . |
| **Removing an Image** | docker image rm nginx |
| **Loading a Tarred Repository from a File or the Standard Input Stream** | docker image load < ubuntu.tar.gz |
| **Saving an Image to a Tar Archive** | docker image save busybox > ubuntu.tar |
| **Showing the History of an Image** | docker image history |
| **Creating an Image From a Container** | docker container commit nginx |
| **Tagging an Image** | docker image tag nginx eon01/nginx |
| **Pushing an Image** | docker image push eon01/nginx |