**what is docker ?**

docker is open source tool for developing, shipping, and running applications. Dockers offers you to sepreate your application

from your infrastructure so you can deliver software quickly. with docker, you can manage your infrastructure

in the same ways you manage your applications.

Features:

Easy and Faster Configuration

This is a key feature of docker that helps us to configure the system easily and faster.

We can deploy our code in less time and effort. As Docker can be used in a wide variety of environments, the requirements of the infrastructure are no longer linked with the environment of the application.

Increase productivity

By easing technical configuration and rapid deployment of application. No doubt it has increase productivity. Docker not only helps to execute the application in isolated environment but also it has reduced the resources.

### Application Isolation

It provides containers that are used to run applications in isolation environment. Each container is independent to another and allows us to execute any kind of application.

### Swarm

It is a clustering and scheduling tool for Docker containers. Swarm uses the Docker API as its front end, which helps us to use various tools to control it. It also helps us to control a cluster of Docker hosts as a single virtual host. It's a self-organizing group of engines that is used to enable pluggable backends.

### Routing Mesh

It routes the incoming requests for published ports on available nodes to an active container. This feature enables the connection even if there is no task is running on the node.

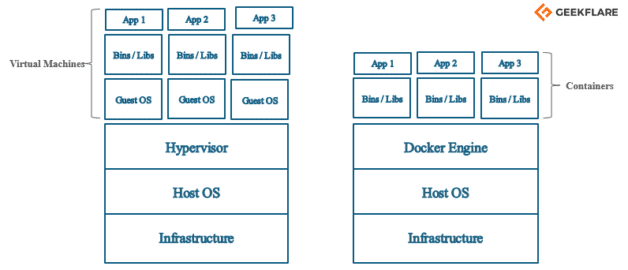
### Services

Services is a list of tasks that lets us specify the state of the container inside a cluster. Each task represents one instance of a container that should be running and Swarm schedules them across nodes.

### Security Management

It allows us to save secrets into the swarm itself and then choose to give services access to certain secrets.

It includes some important commands to the engine like secret inspect, secret create etc



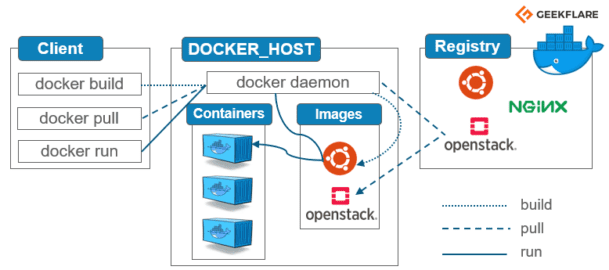
Difference between virulization and containerization

In vitalization perform hardware into software. guest OS is present using hypervisor communicate with host OS to guest OS.Euch guest OS have fix memory on that your application is running.

In Containerzation guest OS is not present On docker engine place yout container that used hose OS memory

Container is lightweight, flexible and execute faster.it is platform independent.

**Docker Architecture:**



### Docker Engine

It is the core part of the whole Docker system. Docker Engine is an application which follows **client-server architecture**. It is installed on the host machine. There are three components in the Docker Engine:

* **Server**: It is the docker daemon called **dockerd**. It can create and manage docker images. Containers, networks, etc.
* **Rest API**: It is used to instruct docker daemon what to do.
* **Command Line Interface (CLI)**: It is a client which is used to enter [docker commands](https://geekflare.com/docker-commands/).

### Docker Client

Docker users can interact with Docker through a client. When any docker commands runs, the client sends them to dockerd daemon, which carries them out. Docker API is used by Docker commands. Docker client can communicate with more than one daemon.

### Docker Registries

It is the location where the Docker images are stored. It can be a public docker registry or a private docker registry. Docker Hub is the default place of docker images, its stores’ public registry. You can also create and run your own [private registry](https://geekflare.com/docker-private-registry-ubuntu/).

When you execute docker pull or docker run commands, the required docker image is pulled from the configured registry. When you execute docker push command, the docker image is stored on the configured registry.

### Docker Objects

When you are working with Docker, you use images, containers, volumes, networks; all these are Docker objects.

#### Images

Docker images are read-only templates with instructions to create a docker container. Docker image can be pulled from a Docker hub and used as it is, or you can add additional instructions to the base image and create a new and modified docker image. You can create your own docker images also using a [dockerfile](https://geekflare.com/dockerfile-tutorial/" \t "_blank). Create a dockerfile with all the instructions to create a container and run it; it will create your custom docker image.

Docker image has a base layer which is read-only, and the top layer can be written. When you edit a dockerfile and rebuild it, only the modified part is rebuilt in the top layer.

#### Containers

After you run a docker image, it creates a docker container. All the applications and their environment run inside this container. You can use Docker API or CLI to start, stop, delete a docker container.

Below is a sample command to run a ubuntu docker container:

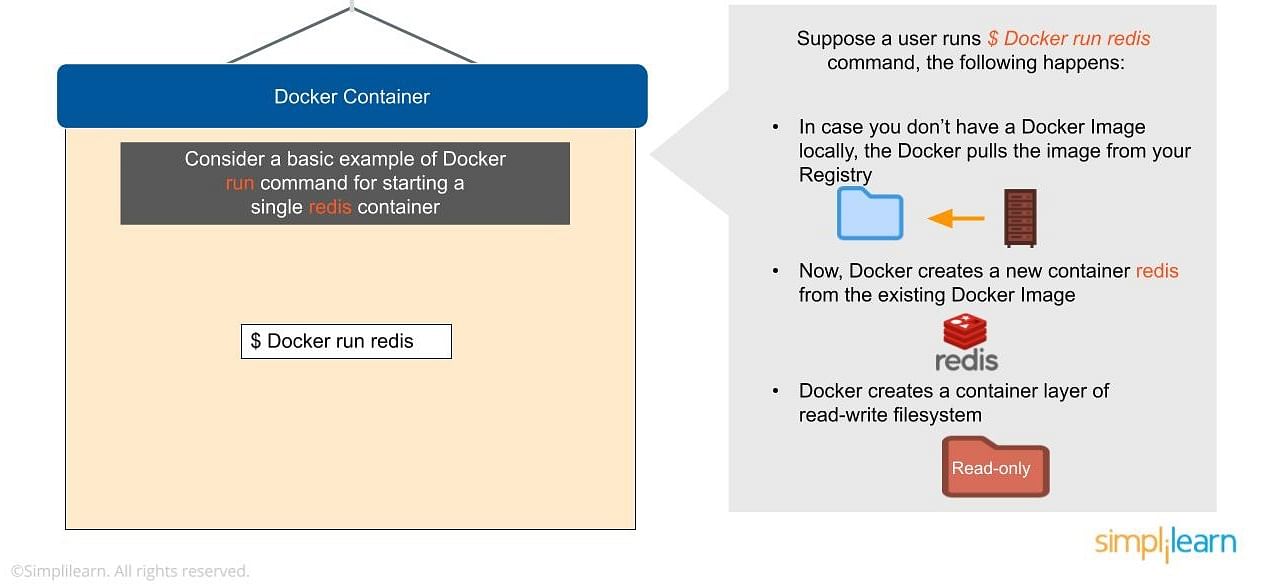
docker run -i -t ubuntu /bin/bash

#### Volumes

The persisting data generated by docker and used by Docker containers are stored in Volumes. They are completely managed by docker through docker CLI or Docker API. Volumes work on both Windows and Linux containers. Rather than persisting data in a container’s writable layer, it is always a good option to use volumes for it. Volume’s content exists outside the lifecycle of a container, so using volume does not increase the size of a container

The [Docker container](https://www.simplilearn.com/tutorials/docker-tutorial/what-is-docker-container) is an executable package of applications and its dependencies bundled together; it gives all the instructions for the solution you’re looking to run. It’s really lightweight due to the built-in structural redundancy. The container is also inherently portable. Another benefit is that it runs completely in isolation. Even if you are running a container, it’s guaranteed not to be impacted by any host OS securities or unique setups, unlike with a virtual machine or a non containerized environment. The memory for a Docker environment can be shared across multiple containers, which is really useful, especially when you have a virtual machine that has a defined amount of memory for each environment.

The container is built using Docker images, and the command to run those images is Run. Let’s go through the basic steps of running a Docker image in this tutorial on Docker.



## **Docker Storage**

A container is volatile it means whenever you remove or kill the container then all of its data will be lost from it. If you want to persist the container data use the docker storage concept.

You can store data within the writable layer of a container but it requires a storage driver. In terms of persistent storage, Docker offers the following options:

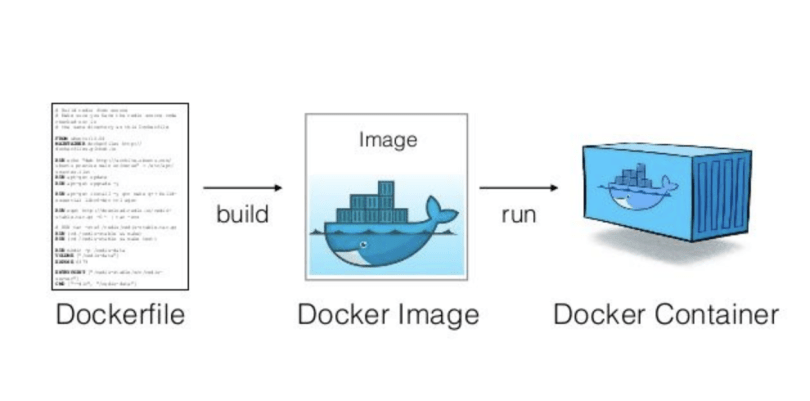
* **Data Volumes**: Data Volumes provide the ability to create persistent storage, with the ability to rename volumes, list volumes, and also list the container that is associated with the volume. Data Volumes sit on the host file system, outside the containers copy on write mechanism and are fairly efficient.
* **Data-Volume Container**: A Data Volume Container is an alternative approach wherein a dedicated container hosts a volume and to mount that volume to other containers. In this case, the volume container is independent of the application container and therefore can be shared across more than one container.
* **Bind Mounts**: Another option is to mount a host’s local directory into a container. In the previously mentioned cases, the volumes would have to be within the Docker volumes folder, whereas when it comes to Directory Mounts any directory on the Host machine can be used as a source for the volume.

.

## **How Does Docker Work?**

Docker works via a Docker engine that is composed of two key elements: a server and a client; and the communication between the two is via REST API. The server communicates the instructions to the client. On older Windows and Mac systems, you can take advantage of the Docker Toolbox, which allows you to control the Docker engine using Compose and Kitematic.

Now that we have learned about Docker, its advantages, and how it works, our next focus in this article is to learn the various components of Docker.



### **Docker Compose**

[Docker-compose](https://www.simplilearn.com/tutorials/docker-tutorial/docker-compose) is designed for running multiple containers as a single service. It does so by running each container in isolation but allowing the containers to interact with one another. As noted earlier, you would write the compose environments using YAML.

So in what situations might you use Docker compose? An example would be if you are running an Apache server with a single database and you need to create additional containers to run additional services without having to start each one separately. ou would write a set of files using Docker compose to do that.

# **Docker Useful Commands**

Docker is natively Linux based software so that it provides commands to interact and work in the client-server environment.

Here, we have listed some important and useful Docker commands.

### Check Docker version

1. $ docker version

It shows docker version for both client and server. As given in the following image.

### Build Docker Image from a Dockerfile

1. $ docker build -t image-name docker-file-location

### Run Docker Image

1. $ docker run -d image-name

**-d** : It is used to create a daemon process.

### Check available Docker images

1. $ docker images

### Check for latest running container

1. $ docker ps -l

**-l** : it is used to show latest available container.

### Check all running containers

**-t**: it is used to tag Docker image with the provided name.

### Check all running containers

1. $ docker ps -a

**-a** : It is used to show all available containers.

### Stop running container

1. $ docker stop container\_id

**container\_id** : It is an Id assigned by the Docker to the container.

### Delete an image

1. $ docker rmi image-name

### Delete all images

1. $ docker rmi $(docker images -q)

### Delete all images forcefully

1. $ docker rmi -r $(docker images -q)

**-r** : It is used to delete image forcefully.

### Delete all containers

1. $ docker rm $(docker ps -a -q)

### Enter into Docker container

1. $ docker exec -it container-id bash