Prakash-Ethans

Docker

In simple words, Docker is a software containerization platform, meaning you can build your application, package them along with their dependencies into a container and then these containers can be easily shipped to run on other machines.

For example: Lets consider a linux based application which has been written both in Ruby and Python. This application requires a specific version of linux, Ruby and Python. In order to avoid any version conflicts on user's end, a linux docker container can be created with the required versions of Ruby and Python installed along with the application. Now the end users can use the application easily by running this container without worrying about the dependencies or any version conflicts.

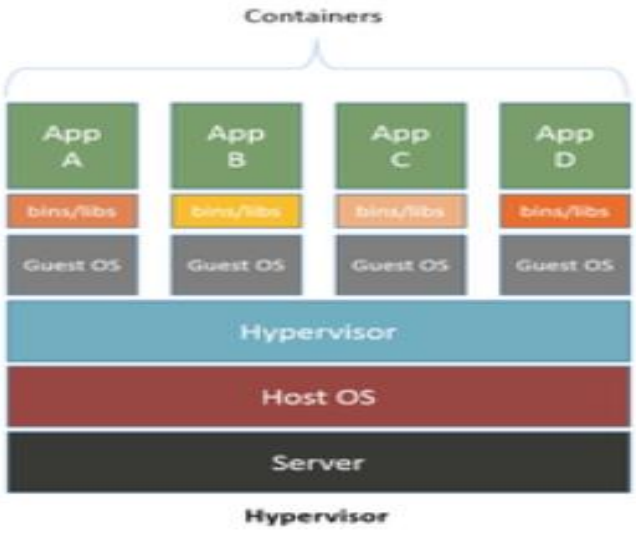
These containers use Containerization which can be considered as an evolved version of Virtualization. The same task can also be achieved using Virtual Machines, however it is not very efficient.

I generally receive a question at this point, i.e. what is the difference between Virtualization and Containerization? These two terms are very similar to each other. So, let me first tell you What is Virtualization?

What is Virtualization?

Virtualization is the technique of importing a Guest operating system on top of a Host operating system. This technique was a revelation at the beginning because it allowed developers to run multiple operating systems in different virtual machines all running on the same host. This eliminated the need for extra hardware resource. The advantages of Virtual Machines or Virtualization are:

- Multiple operating systems can run on the same machine
- Maintenance and Recovery were easy in case of failure conditions
- Total cost of ownership was also less due to the reduced need for infrastructure



In the diagram on the right, you can see there is a host operating system on which there are 3 guest operating systems running which is nothing but the virtual machines.

As you know nothing is perfect, Virtualization also has some shortcomings. Running multiple Virtual Machines in the same host operating system leads to performance degradation. This is because of the guest OS running on top of the host OS, which will have its own kernel and set of libraries and dependencies. This takes up a large chunk of system resources, i.e. hard disk, processor and especially RAM.

Another problem with Virtual Machines which uses virtualization is that it takes almost a minute to boot-up. This is very critical in case of real-time applications.

Following are the disadvantages of Virtualization:

- Running multiple Virtual Machines leads to unstable performance
- Hypervisors are not as efficient as the host operating system
- Boot up process is long and takes time

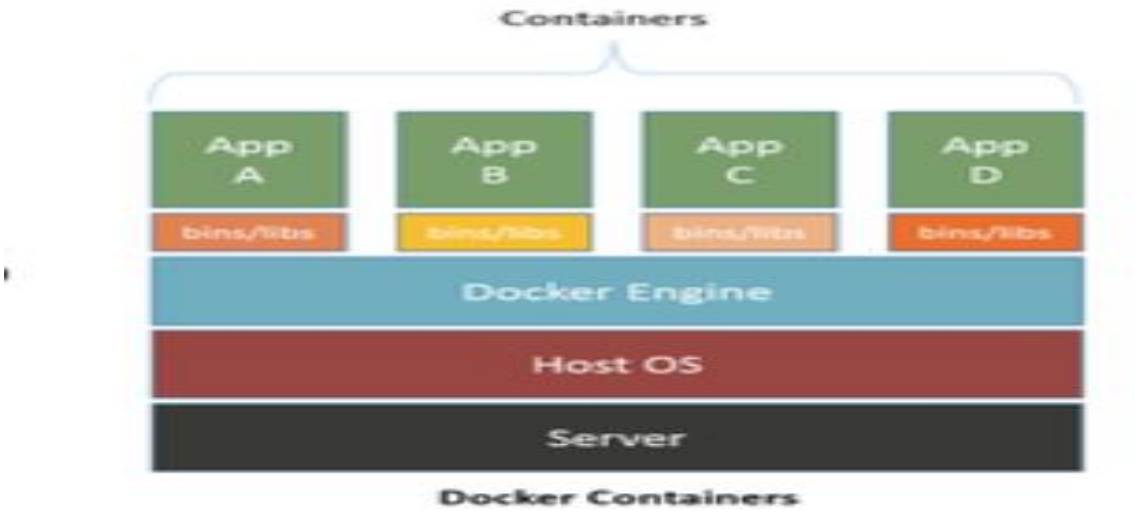
These drawbacks led to the emergence of a new technique called Containerization. Now let me tell you about Containerization.

What is Containerization?

Containerization is the technique of bringing virtualization to the operating system level. While Virtualization brings abstraction to the hardware, Containerization brings abstraction to the operating system. Do note that Containerization is also a type of Virtualization. Containerization is however more efficient because there is no guest OS here and utilizes a host's operating system, share relevant libraries & resources as and when needed unlike virtual machines. Application specific binaries and libraries of containers run on the host kernel, which makes processing and execution very fast. Even booting-up a container takes only a fraction of a second. Because all the containers share, host operating system and holds only the application related binaries & libraries. They are lightweight and faster than Virtual Machines.

Advantages of Containerization over Virtualization:

- Containers on the same OS kernel are lighter and smaller
- Better resource utilization compared to VMs
- Boot-up process is short and takes few seconds

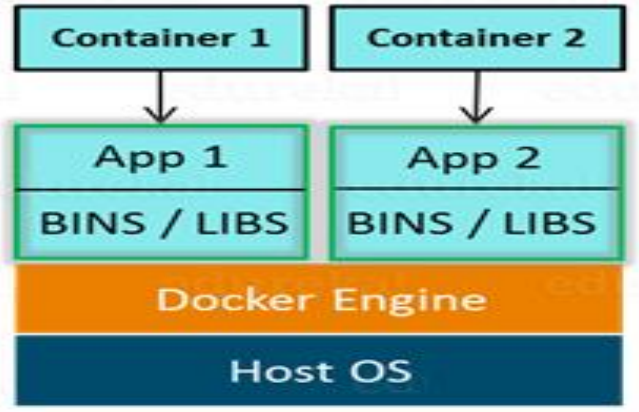


In the diagram on the right, you can see that there is a host operating system which is shared by all the containers. Containers only contain application specific libraries which are separate for each container and they are faster and do not waste any resources.

All these containers are handled by the containerization layer which is not native to the host operating system. Hence a software is needed, which can enable you to create & run containers on your host operating system.

Introduction To Docker

Docker is a containerization platform that packages your application and all its dependencies together in the form of Containers to ensure that your application works seamlessly in any environment.



As you can see in the diagram on the above, each application will run on a separate container and will have its own set of libraries and dependencies. This also ensures that there is process level isolation, meaning each application is independent of other applications, giving developers surety that they can build applications that will not interfere with one another.

As a developer, I can build a container which has different applications installed on it and give it to my QA team who will only need to run the container to replicate the developer environment.

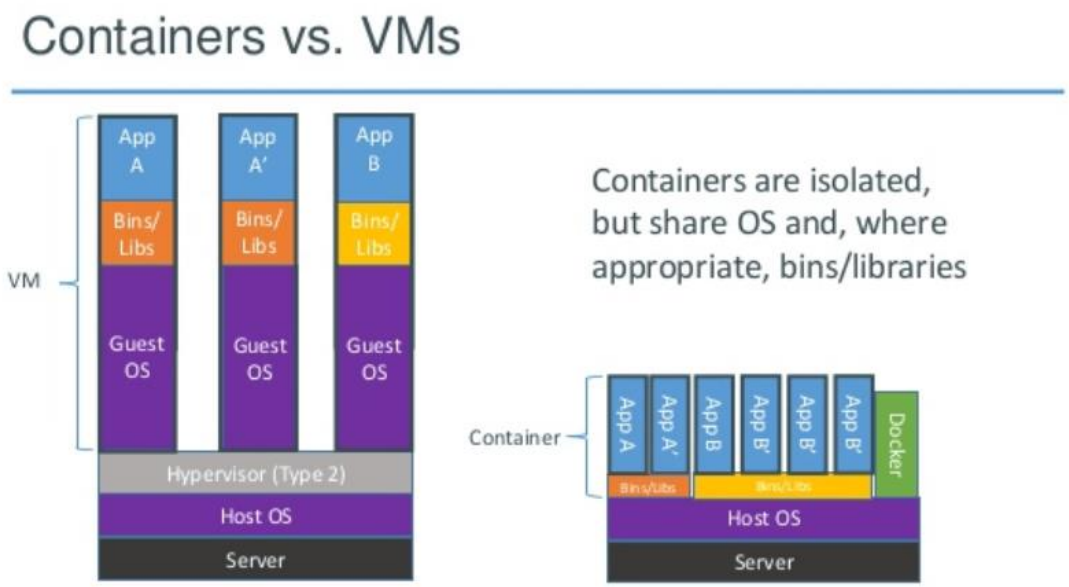
Benefits of Docker

Now, the QA team need not install all the dependent software and applications to test the code and this helps them save lots of time and energy. This also ensures that the working environment is consistent across all the individuals involved in the process, starting from development to deployment. The number of systems can be scaled up easily and the code can be deployed on them effortlessly.

Virtualization vs Containerization

Virtualization and Containerization both let you run multiple operating systems inside a host machine.

Virtualization deals with creating many operating systems in a single host machine. Containerization on the other hand will create multiple containers for every type of application as required.



As we can see from the image, the major difference is that there are multiple Guest Operating Systems in Virtualization which are absent in Containerization. The best part of Containerization is that it is very light weight as compared to the heavy virtualization.

VM and Docker Understanding:

- Virtual Machines are slow and takes a lot of time to boot.
- Containers are fast and boots quickly as it uses host operating system and shares the relevant libraries.
- Containers does not waste or block host resources unlike virtual machines.
- Containers have isolated libraries and binaries specific to the application they are running.
- Containers are handled by Containerization engine.
- Docker is one of the containerization platforms which can be used to create and run containers.

What is Docker & Docker Container ?

What is Docker? – Docker is a containerization platform that packages your application and all its dependencies together in the form of a docker container to ensure that your application works seamlessly in any environment.

What is Container? – Docker Container is a standardized unit which can be created on the fly to deploy a particular application or environment. It could be an Ubuntu container, CentOs container, etc. to full-fill the requirement from an operating system point of view. Also, it could be an application-oriented container like CakePHP container or a Tomcat-Ubuntu container etc.

Let's understand it with an example:

- A company needs to develop a Java Application. In order to do so the developer will setup an environment with tomcat server installed in it. Once the application is developed, it needs to be tested by the tester. Now the tester will again set up tomcat environment from the scratch to test the application. Once the application testing is done, it will be deployed on the production server. Again the production needs an environment with tomcat installed on it, so that it can host the Java application. If you see the same tomcat environment setup is done thrice. There are some issues that I have listed below with this approach:
- 1) There is a loss of time and effort.
- 2) There could be a version mismatch in different setups i.e. the developer & tester may have installed tomcat 7, however the system admin installed tomcat 9 on the production server.

How Docker container can be used to prevent this loss.

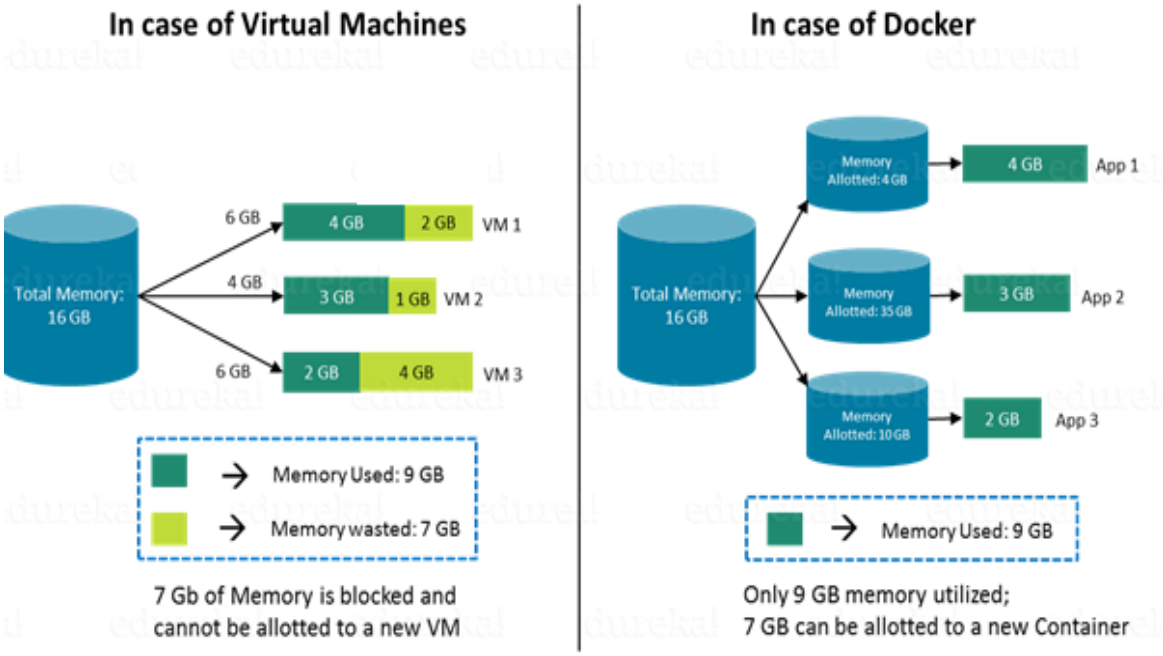
- In this case, the developer will create a tomcat docker image (A Docker Image is nothing but a blueprint to deploy multiple containers of the same configurations) using a base image like Ubuntu, which is already existing in Docker Hub (Docker Hub has some base docker images available for free) . Now this image can be used by the developer, the tester and the

system admin to deploy the tomcat environment. This is how docker container solves the problem.

- However, now you would think that this can be done using Virtual Machines as well. However, there is catch if you choose to use virtual machine. Let's see a comparison between a Virtual machine and Docker Container to understand this better.

Virtual Machine and Docker Container are compared on the following three parameters:

- Size – This parameter will compare Virtual Machine & Docker Container on their resource they utilize.
- Startup – This parameter will compare on the basis of their boot time.
- Integration – This parameter will compare on their ability to integrate with other tools with ease.



Consider a situation depicted in the above image. I have a host system with 16 Gigabytes of RAM and I have to run 3 Virtual Machines on it. To run the Virtual

Machines in parallel, I need to divide my RAM among the Virtual Machines. Suppose I allocate it in the following way:

- 6 GB of RAM to my first VM,
- 4 GB of RAM to my second VM, and
- 6 GB to my third VM.

In this case, I will not be left with anymore RAM even though the usage is:

- My first VM uses only **4 GB** of RAM – Allotted **6 GB** – **2 GB** Unused & Blocked
- My second VM uses only **3 GB** of RAM – Allotted **4 GB** – **1 GB** Unused & Blocked
- My third VM uses only **2 GB** of RAM – Allotted **6 GB** – **4 GB** Unused & Blocked

So, how can I avoid this problem?

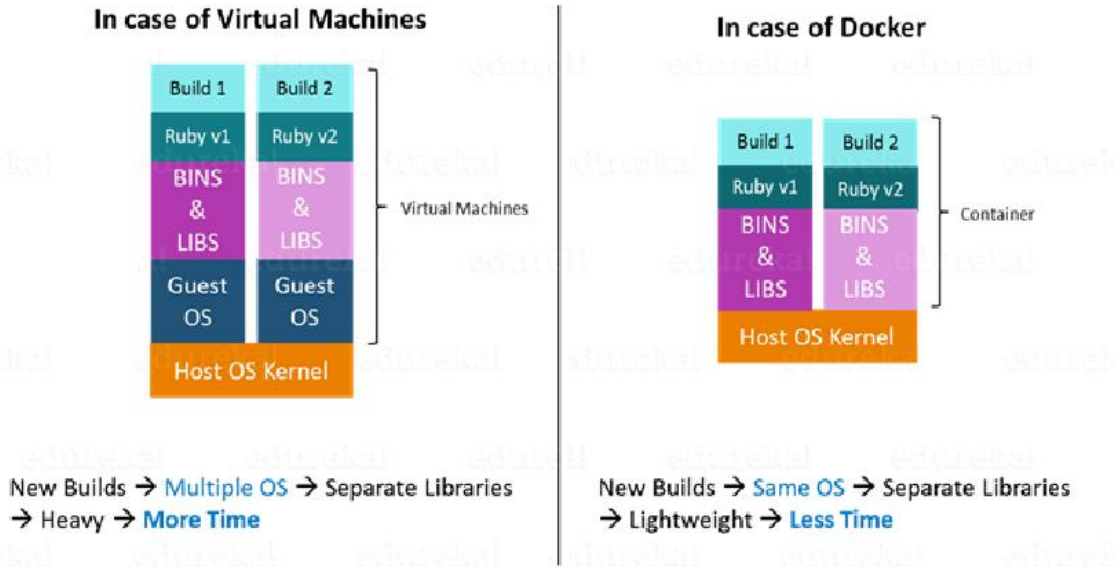
If I use Docker, my CPU will allocate exactly the amount of memory that is required by the Docker Container.

- My first container will use only **4 GB** of RAM – Allotted **4 GB** – **0 GB** Unused & Blocked
- My second container will use only **3 GB** of of RAM – Allotted **3 GB** – **0 GB** Unused & Blocked
- My third container will use only **2 GB** of RAM – Allotted **2 GB** – **0 GB** Unused & Blocked

Since there is no allocated memory (RAM) which is unused, I save **7 GB** (**16** – **4 – 3 – 2**) of RAM by using Docker Container. I can even create additional containers from the leftover RAM and increase my productivity.

So here Docker Container clearly wins over Virtual machine as I can efficiently use my resources as per my need

Start-Up



When it comes to start-up, Virtual Machine takes a lot of time to boot up because the guest operating system needs to start from scratch, which will then load all the binaries and libraries. This is time consuming and will prove very costly at times when quick startup of applications is needed. In case of Docker Container, since the container runs on your host OS, you can save precious boot-up time. This is a clear advantage over Virtual Machine.

Consider a situation where I want to install two different versions of Ruby on my system. If I use Virtual Machine, I will need to set up 2 different Virtual Machines to run the different versions. Each of these will have its own set of binaries and libraries while running on different guest operating systems. Whereas if I use Docker Container, even though I will be creating 2 different containers where each container will have its own set of binaries and libraries, I will be running them on my host operating system. Running them straight on my Host operating system makes my Docker Containers lightweight and faster.

So Docker Container clearly wins again from Virtual Machine based on Startup parameter.

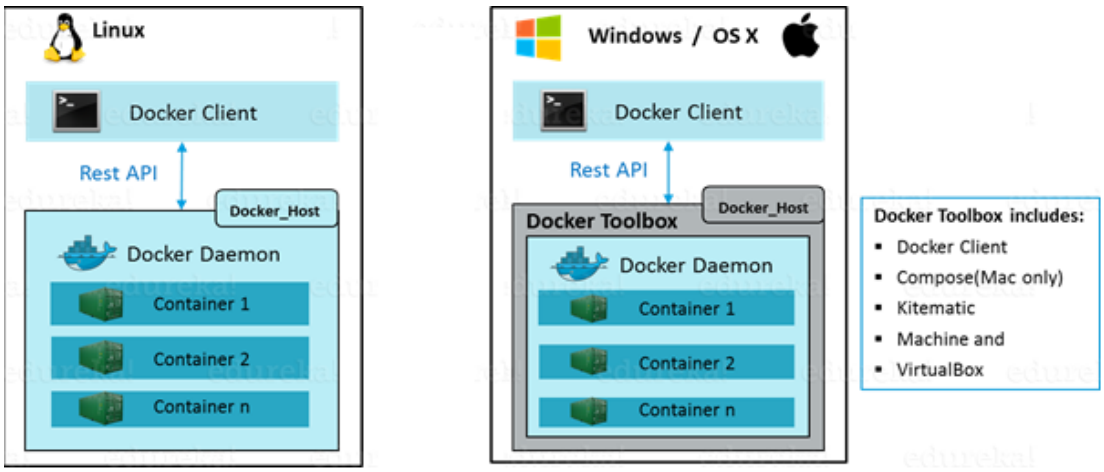
Docker is designed to benefit both Developers and System Administrators, making it a part of many DevOps toolchains. Developers can write their code without worrying about the testing or the production environment and system administrators need not worry about infrastructure as Docker can easily scale up and scale down the number of systems for deploying on the servers.

What is Docker Engine?

Now I will take you through Docker Engine which is the heart of the Docker system.

Docker Engine is simply the docker application that is installed on your host machine. It works like a client-server application which uses:

- A **server** which is a type of long-running program called a daemon process
- A command line interface (CLI) **client**
- REST API is used for communication between the CLI client and Docker Daemon



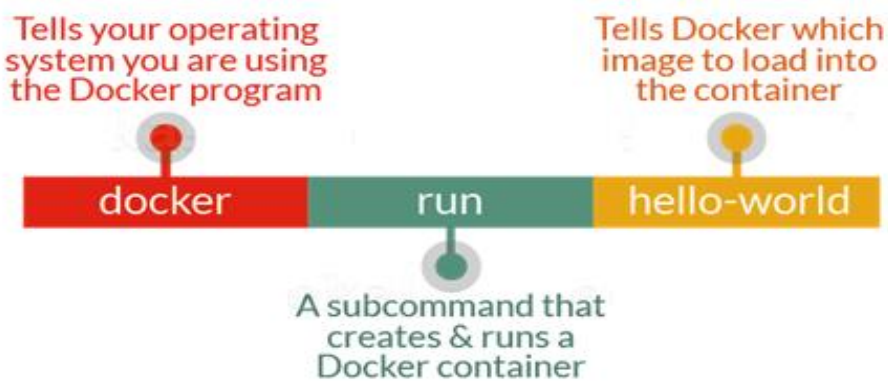
As per the above image, in a Linux Operating system, there is a Docker client which can be accessed from the terminal and a Docker Host which runs the Docker Daemon. We build our Docker images and run Docker containers by passing commands from the CLI client to the Docker Daemon.

However, in case of Windows/Mac there is an additional Docker Toolbox component inside the Docker host. This Docker Toolbox is an installer to quickly and easily install and setup a Docker environment on your Windows/iOS. Docker Toolbox installs Docker Client, Machine, Compose (Mac only), Kitematic and VirtualBox.

Let's now understand three important terms, i.e. **Docker Images**, **Docker Containers** and **Docker Registry**.

What is Docker Image?

Docker Image can be compared to a template which is used to create Docker Containers. They are the building blocks of a Docker Container. These Docker Images are created using the build command. These Read only templates are used for creating containers by using the run command. We will explore Docker commands in depth in the “Docker Commands blog”.

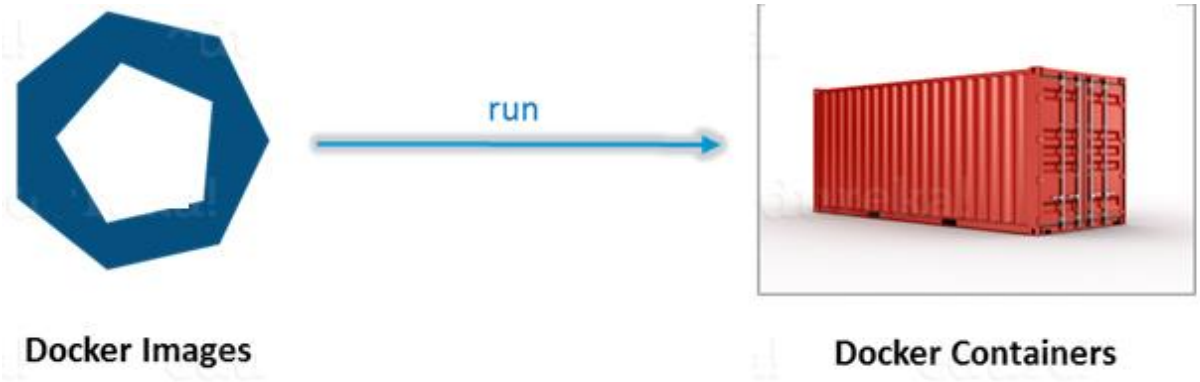


Docker lets people (or companies) create and share software through Docker images. Also, you don’t have to worry about whether your computer can run the software in a Docker image — a Docker container *can always run it*.

I can either use a ready-made docker image from docker-hub or create a new image as per my requirement. In the Docker Commands blog we will see how to create your own image.

What is Docker Container?

Containers are the ready applications created from Docker Images or you can say a Docker Container is a running instance of a Docker Image and they hold the entire package needed to run the application. This happens to be the ultimate utility of Docker.

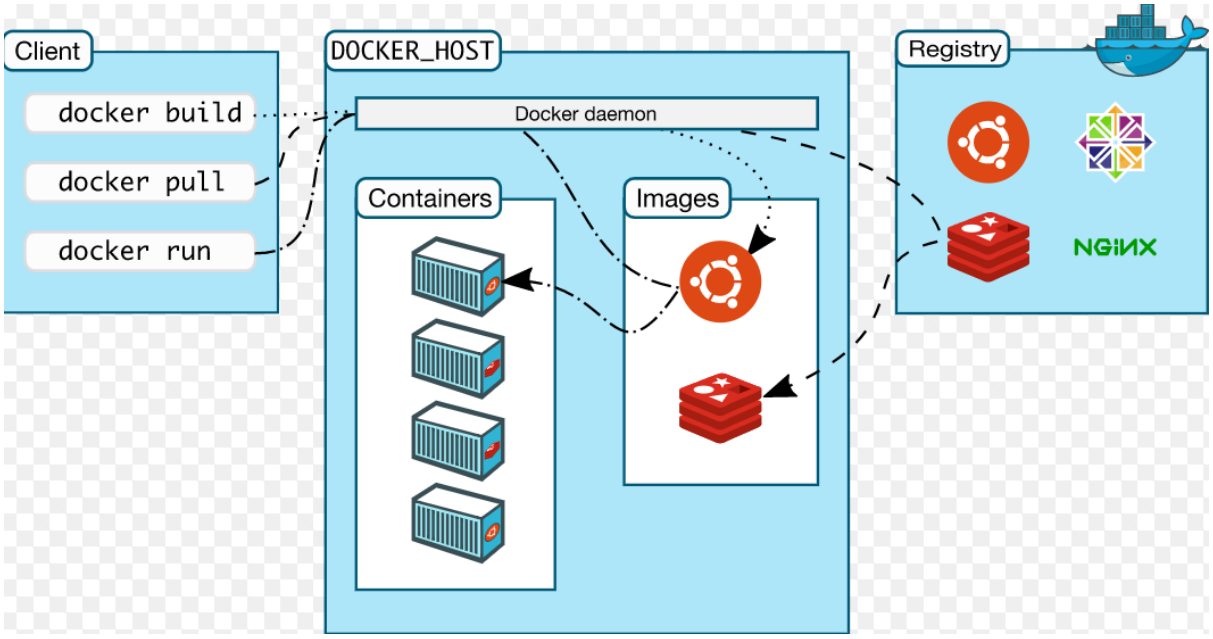


What is Docker Registry?

Finally, Docker Registry is where the Docker Images are stored. The Registry can be either a user's local repository or a public repository like a Docker Hub allowing multiple users to collaborate in building an application. Even with multiple teams within the same organization can exchange or share containers by uploading them to the Docker Hub. Docker Hub is Docker's very own cloud repository similar to GitHub.

What is Docker Architecture?

Docker Architecture includes a Docker client – used to trigger Docker commands, a Docker Host – running the Docker Daemon and a Docker Registry – storing Docker Images. The Docker Daemon running within Docker Host is responsible for the images and containers.



- To build a Docker Image, we can use the CLI (client) to issue a build command to the Docker Daemon (running on Docker_Host). The Docker Daemon will then build an image based on our inputs and save it in the Registry, which can be either Docker hub or a local repository
- If we do not want to create an image, then we can just pull an image from the Docker hub, which would have been built by a different user
- Finally, if we have to create a running instance of my Docker image, we can issue a run command from the CLI, which will create a Docker Container.