Docker is a containerization platform

History before Containerization

Before containerization came into the picture, the leading way to isolate, organize applications and their dependencies was to place each and every application in its own virtual machine. These machines run multiple applications on the same physical hardware, and this process is nothing but **Virtualization**.

But virtualization had few drawbacks such as the virtual machines were bulky in size, running multiple virtual machines lead to unstable performance, boot up process would usually take a long time and VM's would not solve the problems like portability, software updates, or continuous integration and continuous delivery.

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

Containerization

Containerization is a type of Virtualization which brings virtualization to the operating system level. While Virtualization brings abstraction to the hardware, Containerization brings abstraction to the operating system moving ahead, it's time that you understand the reasons to use containers.

Reasons to use Containers

Following are the reasons to use containers:

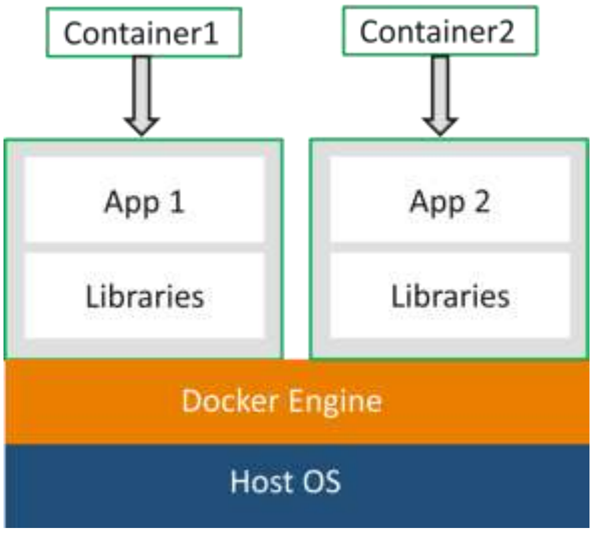
- Containers have no guest OS and use the host's operating system. So, they share relevant libraries & resources as and when needed.
- Processing and execution of applications are very fast since applications specific binaries and libraries of containers run on the host kernel.

- Booting up a container takes only a fraction of a second, and also containers are lightweight and faster than Virtual Machines.

Now, that you have understood what containerization is and the reasons to use containers, it's the time you understand our main concept here.

What is Docker?

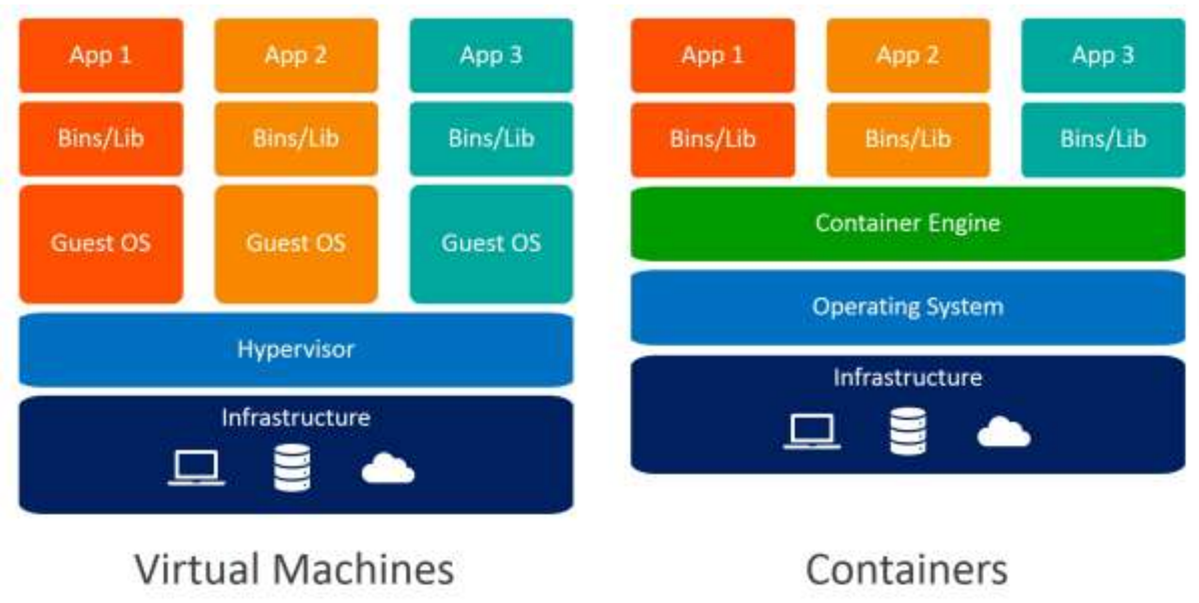
Docker is a platform which packages an application and all its dependencies together in the form of containers. This containerization aspect ensures that the application works in any environment.



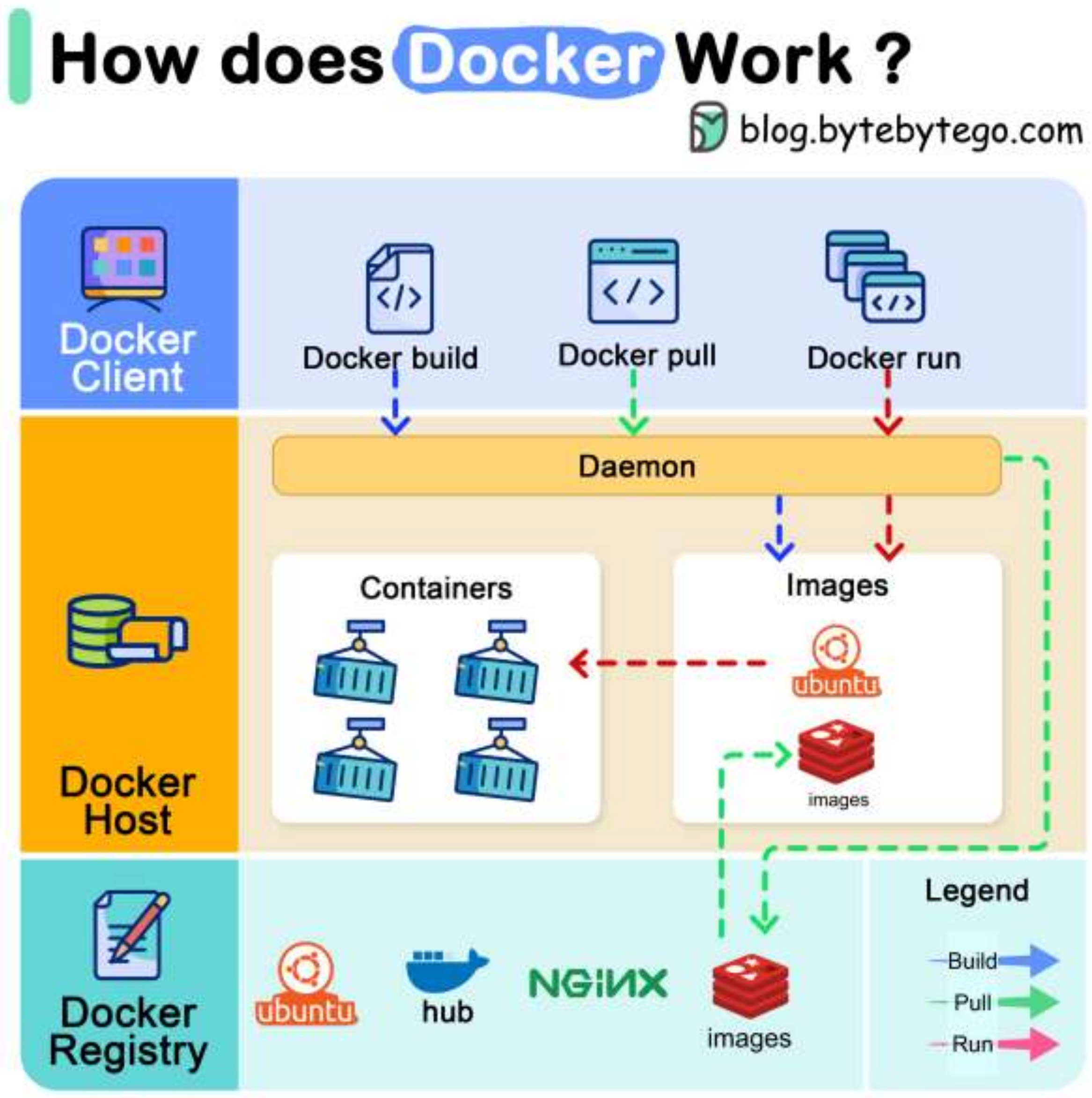
As you can see in the diagram, each and every application runs on separate containers and has its own set of dependencies & libraries. This makes sure that each application is independent of other applications, giving developers surety that they can build applications that will not interfere with one another.

So a developer can build a container having different applications installed on it and give it to the QA team. Then the QA team would only need to run the container to replicate the developer's environment.

Virtualization vs Containerization

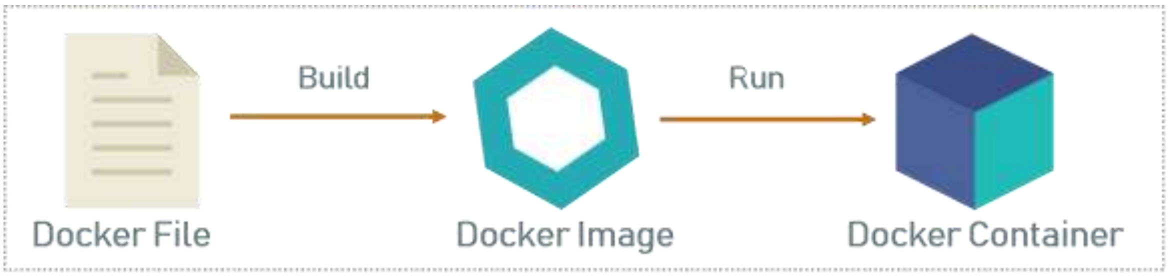


basic concepts of Docker, such as Dockerfile, images & containers.



Dockerfile, Images & Containers

Dockerfile, Docker Images & Docker Containers are three important terms that you need to understand while using Docker.



As you can see in the above diagram when the Dockerfile is built, it becomes a Docker Image and when we run the Docker Image then it finally becomes a Docker Container.

Refer below to understand all the three terms.

Dockerfile: A Dockerfile is a text document which contains all the commands that a user can call on the command line to assemble an image. So, Docker can build images automatically by reading the instructions from a Dockerfile. You can use `docker build` to create an automated build to execute several command-line instructions in succession.

Docker Image: In layman terms, Docker Image can be compared to a template which is used to create Docker Containers. So, these read-only templates are the building blocks of a Container. You can use `docker run` to run the image and create a container.

Docker Images are stored in the Docker Registry. It can be either a user's local repository or a public repository like a Docker Hub which allows multiple users to collaborate in building an application.

Docker Container: It is a running instance of a Docker Image as they hold the entire package needed to run the application. So, these are basically the ready applications created from Docker Images which is the ultimate utility of Docker.

Docker Java Application Example

As, we have mentioned earlier that docker can execute any application.

Here, we are creating a Java application and running by using the docker. This example includes the following steps.

1. **Create a directory**

Directory is required to organize files. Create a director by using the following command

1. $ mkdir java-docker-app

2. **Create a Java File**

   Now create a Java file. Save this file as **Hello.java** file.

   // Hello.java

```
class Hello{
public static void main(String[] args){
System.out.println("This is java app \n by using Docker");
}
}
```

Save it inside the directory **java-docker-app** as Hello.java.

3. **Create a Dockerfile**

   After creating a Java file, we need to create a Dockerfile which contains instructions for the Docker. Dockerfile does not contain any file extension. So, save it simple with **Dockerfile** name.

   // Dockerfile

```
FROM java:8
COPY . /var/www/java
WORKDIR /var/www/java
RUN javac Hello.java
CMD ["java", "Hello"]
```

Write all instructions in uppercase because it is convention. Put this file inside **java-docker-app** directory. Now we have Dockerfile parallel to Hello.java inside the **java-docker-app** directory.

See, your folder inside must look like the below.

Dockerfile
Hello.java

4. **Build Docker Image**

   After creating Dockerfile, we are changing working directory

1. $ cd   java-docker-app

docker build -t java-app .

docker run java-app