Kunal Kushwaha

Getting started with Docker

Kunal Kushwaha

Jan 21, 2022 • 7 min read

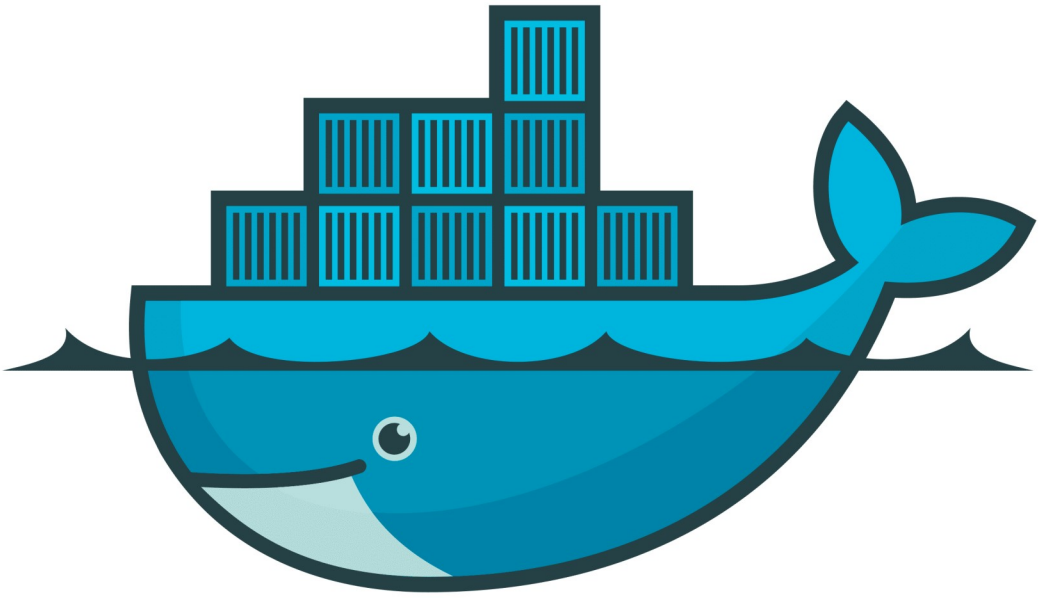


Table of contents

Subscribe

Has it ever happened with you that you create an application which runs like a charm on your machine but when you pass it on to someone else, for some reason it doesn't quite work on their machine?

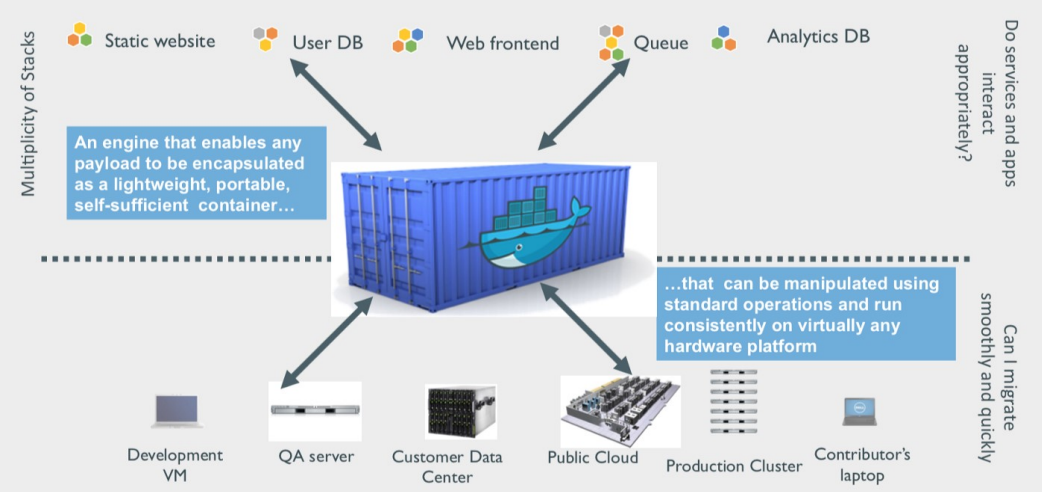
What is Docker?

Docker is a container platform that allows you to build, test and deploy applications quickly. A developer defines all the applications and it's dependencies in a Dockerfile which is then used to build Docker images that defines a Docker container. Doing this ensures that your application will run in any environment.

Why use Docker?

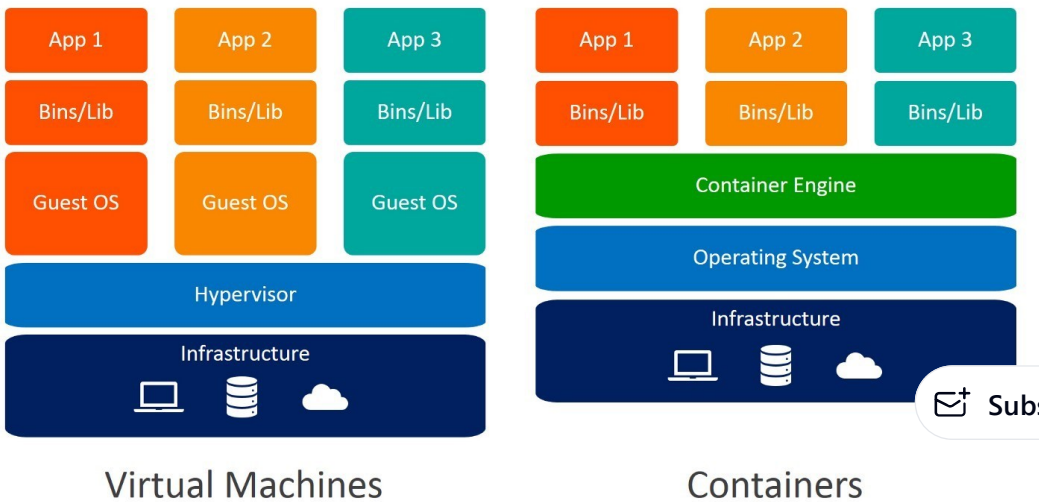
Using Docker can help you ship your code faster, gives you control over your applications. You can deploy applications on containers that make it easier for them to be deployed, scaled, perform rollbacks and ider issues. It also helps in saving money by utilizing resources. Docker-based applications can be seamlessly moved from local development machines

to production deployments. You can use Docker for Microservices, Data Processing, Continuous Integration and Delivery, Containers as a Service.



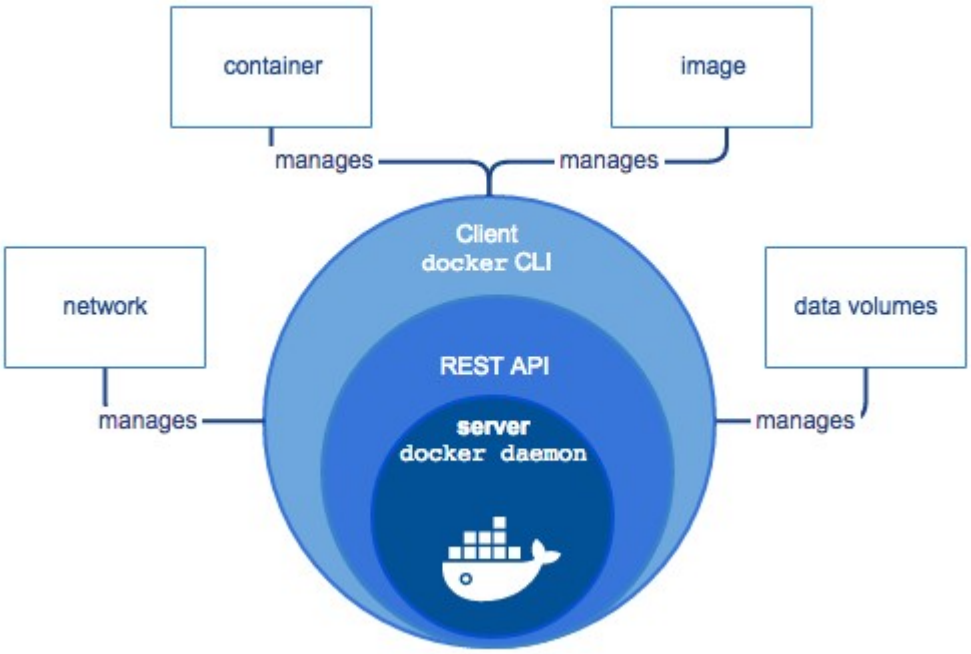
How does it work?

Containerization vs Virtualization



An application on a VM requires a guest OS and thus an underlying hypervisor to run. Hypervisor is used to create multiple machines on a host operating system and it manages virtual machines. These virtual machines have their own operating system and do not use the host's operating system. They have some space allocated. In the software world, containerization is an efficient method for deploying applications. A container encapsulates an application with its own operating environment. It can be placed on any host machine without special configuration, removing the issue of dependencies. VM is hardware virtualization, whereas containerization is OS virtualization. In comparison, virtualization is the creation of a virtual version of something such as an operating system, server or a storage device or network resources. Essentially, containerization is a lightweight approach to virtualization.

Docker Architecture



Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers.

For download and installation refer: https://docs.docker.com

Docker daemon

It listens to the API requests being made through the Docker client and manages Docker objects such as images, containers, networks, and volumes.

Docker client

This is what you use to interact with Docker. When you run a command using docker, the client sends the command to the daemon, which carries them out. The Docker client can communicate with more than one daemon.

Docker registries

This is where Docker images are stored. Docker Hub is a public registry that anyone can use. When you pull an image, Docker by default looks for it in the public registry and saves the image on your local system on DOCKER_HOST. You can also store images on your local machine or push them to the public registry.

Dockerfile

Describes steps to create a Docker image. It's like a recipe with all ingredients and steps necessary in making your dish. This file can be used to create Docker Image. These images can be pulled to create containers in any environment. These images can also be store online at docker hubs. When you run docker image you get docker containers. The container will have the application with all its dependencies.

- Create a file named 'Dockerfile'
- By default on building, docker searches for 'Dockerfile' `$ docker build -t myimage:1.0 .`
- During building of the image, the commands in RUN section of Dockerfile will get executed. `$ docker run ImageID`
- The commands in CMD section of Dockerfile will get executed when you create a container out of the image.

Dockerfile Example

```
FROM ubuntu
MAINTAINER kunal kushwaha <kunal@gmail.com>
RUN apt-get update
CMD [“echo”, “Hello World”]
```

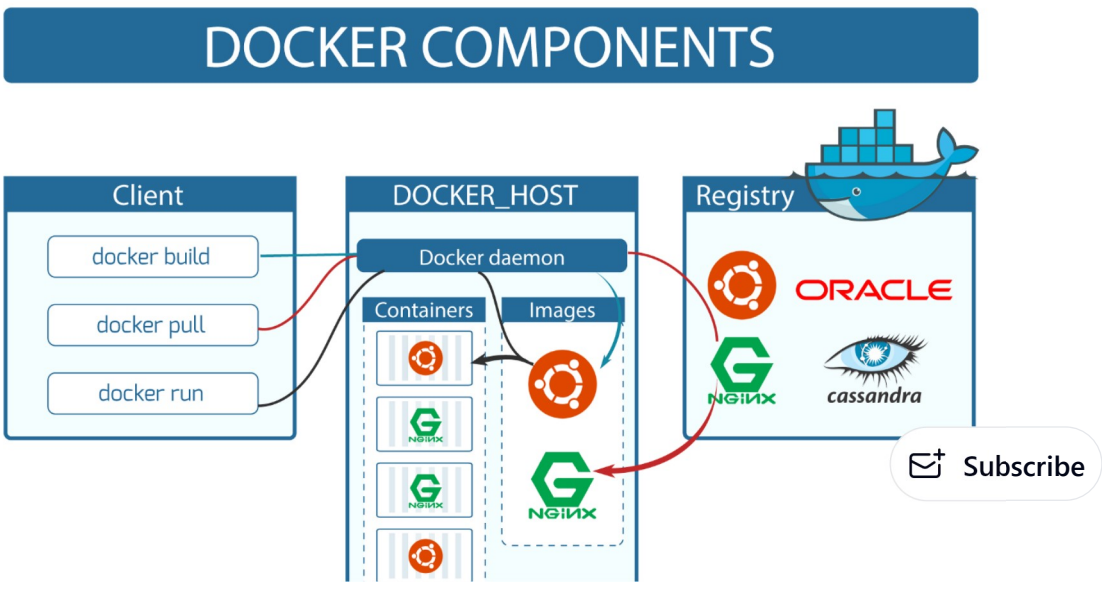
Docker Image

A Docker Image is a file that defines a Docker Container. It is similar in concept to a snapshot of a VM. A container that moves from one Docker environment to another with the same OS will work without changes because the image includes all of the dependencies needed to execute the code. Docker Image is run to create a docker container. Images are immutable. Once built, the files making up an image do not change. Images can be stored locally or remote locations like hub.docker.com.

Basic Commands

```
$ docker pull ubuntu:18.04 (18.04 is tag/version (explained below)
$ docker images (Lists Docker Images)
$ docker run image (creates a container out of an image)
$ docker rmi image (deletes a Docker Image if no container is usir
$ docker rmi $(docker images -q) (deletes all Docker images)
```

Docker can build images automatically by reading instructions from a DockerFile. A single image can be used to create multiple containers.



Images are built in layers. Each layer is an immutable file, but is a collection of files and directories. The last layer can be used to write out data to. Layers receive an ID, calculated via a SHA 256 hash of the layer contents. Thus, if the layer contents change- Notice the IMAGE ID below and the Hash Values given above, the first 12 characters of the hash are equal to the IMAGE ID, the SHA 256 hash changes as well. Note: The Image ID listed by docker commands (ie 'docker images') is the first 12 characters of the hash. These hash values are referred to by 'tag' names.

Listing Hash Values of Docker Images

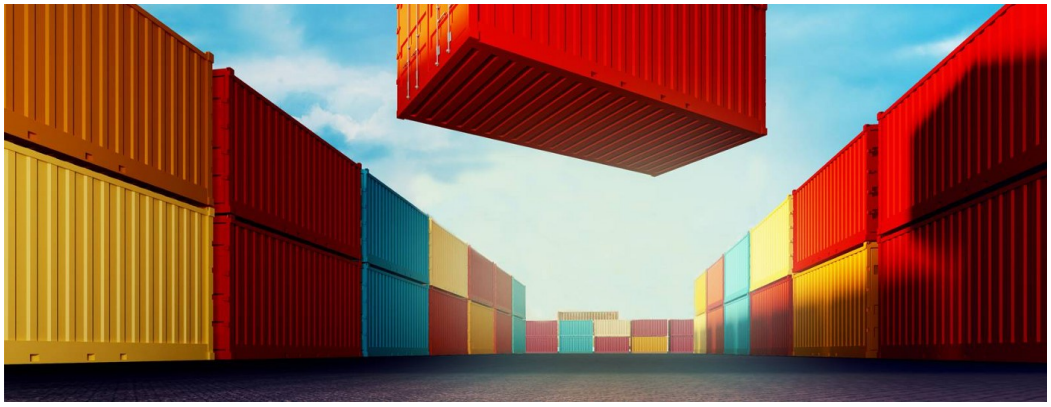
```
$ docker images -q --no-trunc
sha256:3556258649b2ef23a41812be17377d32f568ed9f45150a26466d2ea26d9
sha256:9f38484d220fa527b1fb19747638497179500a1bed8bf0498eb7882292
sha256:fce289e99eb9bca977dae136fbe2a82b6b7d4c372474c9235adc174167
```

- Notice the IMAGE ID below and the Hash Values given above, the first 12 characters of the hash are equal to the IMAGE ID

```
$ docker images
REPOSITORY   TAG      IMAGE ID
ubuntu       18.04    3556258649b2
centos       latest   9f38484d220f
hello-world  latest   fce289e99eb9
```

The format of a full tag name is: [REGISTRYHOST/] [USERNAME/]NAME[:TAG] For Registry Host 'registry.hub.docker.com' is inferred. For ':TAG' — 'latest' is default, and inferred Example: registry.hub.docker.com/mongo:latest

Containers



A container is a runnable instance of an image. This is where your application is running. You can manage containers using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state. If we delete a container the data will be lost! Because when the container went down and we brought it back up, the last layer got created again as a new layer. This helps in development if you don't want to store record for each test. To be persistent, use volumes to store data.

```
$ docker inspect ubuntu:18.04
“RootFS”: {
    “Type”: “layers”,
    “Layers”: [
“sha256:543791078bdb84740cb5457abbea10d96dac3dea8c07d6dc173f734c26
“sha256:c56e09e1bd18e5e41afb1fd16f5a211f533277bdae6d5d8ae96a248214
“sha256:a31dbd3063d77def5b2562dc8e14ed3f578f1f90a89670ae620fd62ae7
“sha256:b079b3fa8d1b4b30a71a6e81763ed3da1327abaf0680ed3ed9f00ad1d5
    ]
},
```

Basic Commands

```
$ docker ps (list all containers)
$ docker run ImageName/ID (checks locally for image, if not availa
$ docker start ContainerName/ID
$ docker kill ContainerName/ID (Stops a running container)
$ docker rm ContainerName/ID (Deletes a stopped container)
$ docker rm $(docker ps -a -q) (Delete all stopped containers)
```


Recap with Demo:

- Pulling an image from Docker registry

```
$ docker images
REPOSITORY      TAG      IMAGE ID     CREATED      SIZE
$ docker pull ubuntu:18.04
18.04: Pulling from library/ubuntu
7413c47ba209: Pull complete
0fe7e7cbb2e8: Pull complete
1d425c982345: Pull complete
344da5c95cec: Pull complete
Digest:sha256:c303f19cfe9ee92badbbbd7567bc1ca47789f79303ddcef56f77
Status: Downloaded newer image for ubuntu:18.04
$ docker images
REPOSITORY     TAG        IMAGE ID          CREATED         SIZE
ubuntu         18.04      3556258649b2      9 days ago      64.2MB
```

- Running image to create container (-it stands for interactive mode)

```
$ docker run -it ubuntu:18.04
root@4183618bcf17:/# ls
bin boot dev etc home lib lib64 media mnt opt proc root run sbin sr
root@4183618bcf17:/# exit
exit
```

- Listing Containers

```
$ docker ps -a
CONTAINER ID   IMAGE          COMMAND       CREATED         STATUS
4183618bcf17   ubuntu:18.04   “/bin/bash”   4 minutes ago   Exited
```

What now?

Imagine you have a large number of containers running and it’s getting difficult to manage all of them. The answer? Container-orchestration system. It helps in automating application deployment, scaling, and management. One of the very widely used COE is Kubernetes.