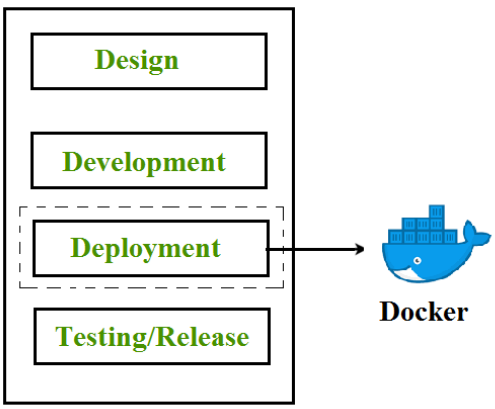
Containerization using Docker

Docker is the containerization platform that is used to package your application and all its dependencies together in the form of containers to make sure that your application works seamlessly in any environment which can be developed or tested or in production. Docker is a tool designed to make it easier to create, deploy, and run applications by using containers.

Docker is the world's leading software container platform. It was launched in 2013 by a company called Dotcloud, Inc which was later renamed Docker, Inc. It is written in the Go language. It has been just six years since Docker was launched yet communities have already shifted to it from VMs. Docker is designed to benefit both developers and system administrators making it a part of many DevOps toolchains. Developers can write code without worrying about the testing and production environment. Sysadmins need not worry about infrastructure as Docker can easily scale up and scale down the number of systems. Docker comes into play at the deployment stage of the software development cycle.

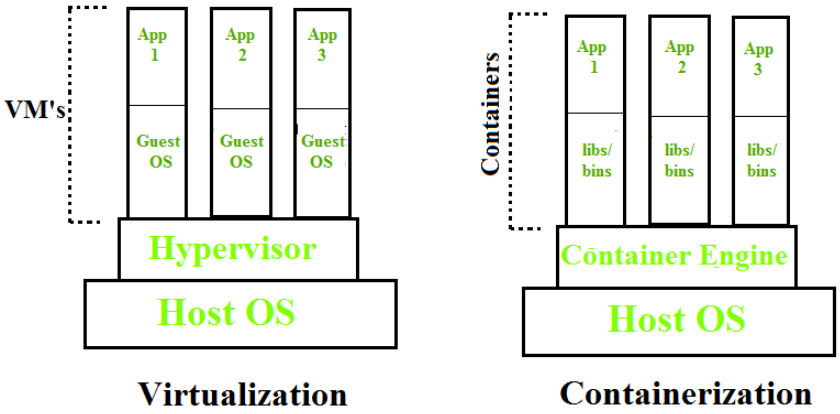


Containerization

Containerization is OS-based virtualization that creates multiple virtual units in the userspace

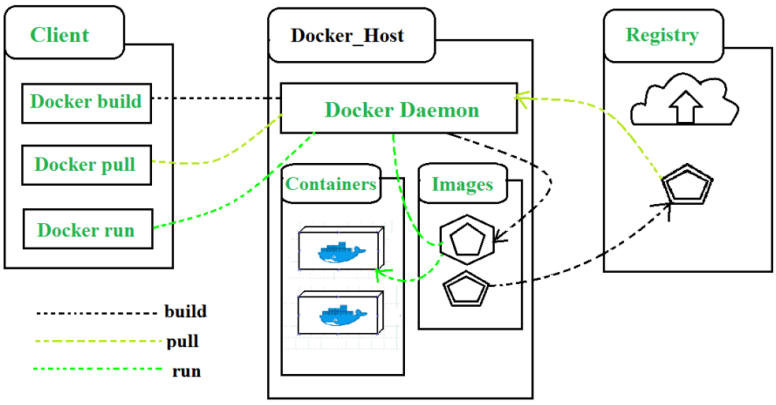
Virtualization provides a different level of abstraction in terms of virtualization and isolation when compared with hypervisors. Hypervisors use a lot of hardware which results in overhead in terms of virtualizing hardware and virtual device drivers. A full operating system (e.g -Linux, Windows) runs on top of this virtualized hardware in each virtual machine instance.
But in contrast, containers implement isolation of processes at the operating system level, thus avoiding such overhead. These containers run on top of the same shared operating system kernel of the underlying host machine and one or more processes can be run within each container. In containers you don't have to pre-allocate any RAM, it is allocated dynamically during the creation of containers while in VMs you need to first pre-allocate the memory and then create the virtual machine. Containerization has better resource utilization compared to VMs and a short boot-up process. It is the next evolution in virtualization.

Containers can run virtually anywhere, greatly easy development and deployment: on Linux, Windows, and Mac operating systems; on virtual machines or bare metal, on a developer's machine or in data centers on-premises; and of course, in the public cloud. Containers virtualize CPU, memory, storage, and network resources at the OS level, providing developers with a sandboxed view of the OS logically isolated from other applications. Docker is the most popular open-source container format available and is supported on Google Cloud Platform and by Google Kubernetes Engine.



Docker Architecture

Docker architecture consists of Docker client, Docker Daemon running on Docker Host, and Docker Hub repository. Docker has client-server architecture in which the client communicates with the Docker Daemon running on the Docker Host using a combination of REST APIs, Socket IO, and TCP. If we have to build the Docker image, then we use the client to execute the build command to Docker Daemon then Docker Daemon builds an image based on given inputs and saves it into the Docker registry. If you don't want to create an image then just execute the pull command from the client and then Docker Daemon will pull the image from the Docker Hub finally if we want to run the image then execute the run command from the client which will create the container.

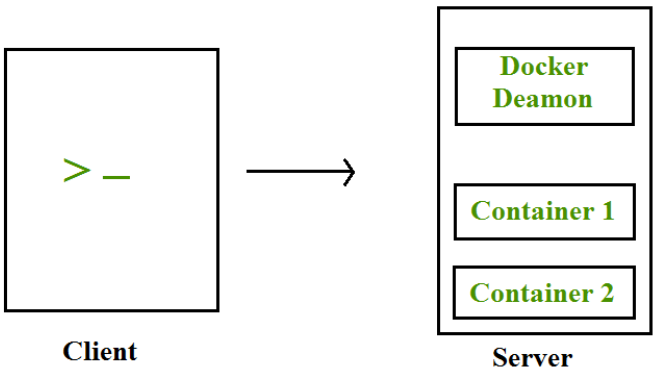


Components of Docker

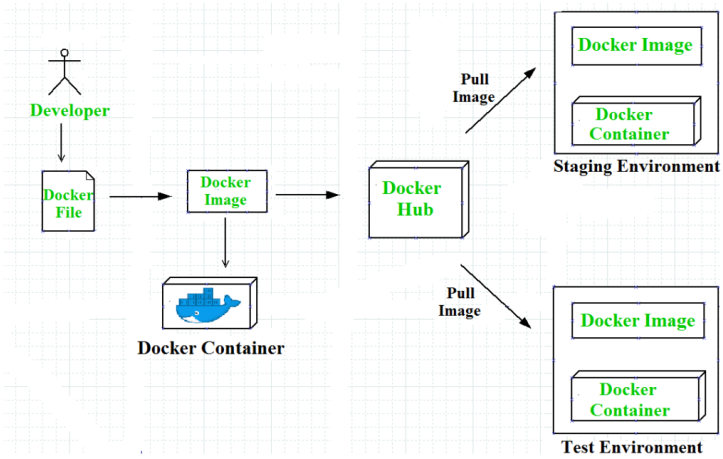
The main components of Docker include – Docker clients and servers, Docker images, Dockerfile, Docker Registries, and Docker containers. These components are explained in detail in the below section :

1. **Docker Clients and Servers**– Docker has a client-server architecture. The Docker Daemon/Server consists of all containers. The Docker Daemon/Server receives the request from the Docker client through CLI or REST APIs and thus processes the request accordingly.

Docker client and Daemon can be present on the same host or different host.



1. **Docker Images**– Docker images are used to build docker containers by using a read-only template. The foundation of every image is a base image eg. base images such as – ubuntu14.04 LTS, and Fedora 20. Base images can also be created from scratch and then required applications can be added to the base image by modifying it thus this process of creating a new image is called “committing the change”.
2. **Docker File**– Dockerfile is a text file that contains a series of instructions on how to build your Docker image. This image contains all the project code and its dependencies. The same Docker image can be used to spin ‘n’ number of containers each with modification to the underlying image. The final image can be uploaded to Docker Hub and shared among various collaborators for testing and deployment. The set of commands that you need to use in your Docker File is FROM, CMD, ENTRYPOINT, VOLUME, ENV, and many more.
3. **Docker Registries**– Docker Registry is a storage component for Docker images. We can store the images in either public/private repositories so that multiple users can collaborate in building the application. Docker Hub is Docker’s cloud repository. Docker Hub is called a public registry where everyone can pull available images and push their images without creating an image from scratch.
4. **Docker Containers**– Docker Containers are runtime instances of Docker images. Containers contain the whole kit required for an application, so the application can be run in an isolated way. For eg.- Suppose there is an image of Ubuntu OS with NGINX SERVER when this image is run with the docker run command, then a container will be created and NGINX SERVER will be running on Ubuntu OS.



Docker Compose

Docker Compose is a tool with which we can create a multi-container application. It makes it easier to configure and
run applications made up of multiple containers. For example, suppose you had an application that required WordPress and MySQL, you could create one file which would start both the containers as a service without the need to start each one separately. We define a multi-container application in a YAML file. With the docker-compose-up command, we can start the application in the foreground. Docker-compose will look for the docker-compose. YAML file in the current folder to start the application. By adding the -d option to the docker-compose-up command, we can start the application in the background. Creating a docker-compose. YAML file for WordPress application :

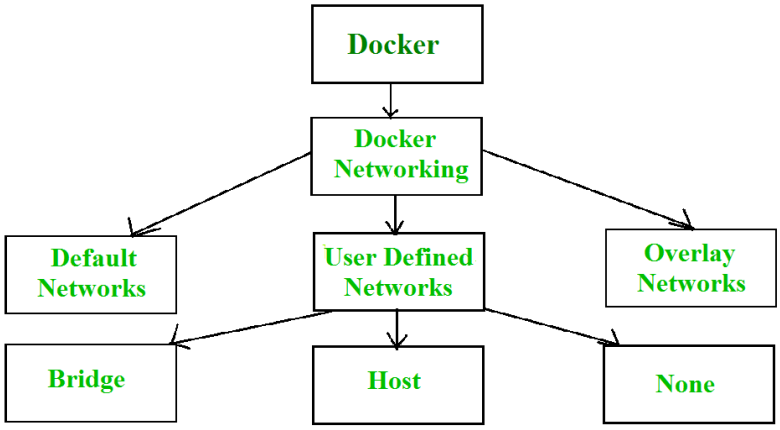
```
#cat docker-compose.yaml
version: ’2’
services:
db:
image: mysql:5.7
volumes:db_data:/var/lib/mysql
restart: always
environment:
MYSQL_ROOT_PASSWORD: WordPress
MYSQL_DATABASE: WordPress
MYSQL_USER: WordPress
MYSQL_PASSWORD: WordPress
WordPress:
depends_on:
- DB
image: WordPress:latest
ports:
- "8000:80"
restart: always
environment:
WORDPRESS_DB_HOST: db:3306

WORDPRESS_DB_PASSWORD: wordpress
volumes:
db_data:
```

In this docker-compose. YAML file, we have the following ports section for the WordPress container, which means that we are going to map the host’s 8000 port with the container’s 80 port. So that host can access the application with its IP and port no.

Docker Networks

When we create and run a container, Docker by itself assigns an IP address to it, by default. Most of the time, it is required to create and deploy Docker networks as per our needs. So, Docker let us design the network as per our requirements. There are three types of Docker networks- default networks, user-defined networks, and overlay networks.



To get a list of all the default networks that Docker creates, we run the command shown below –

```
$ docker network ls

NETWORK ID          NAME                DRIVER
b48564s6sf7bd       bridge              bridge
fe56b3e2dd73d       docker_gwbridge     bridge
5d989edbddbs9       host                host
98absh67bs8m2       none                null
```

There are three types of networks in Docker –

1. **Bridged network**: When a new Docker container is created without the –network argument, Docker by default connects the container with the bridge network. In bridged networks, all the containers in a single host can connect through their IP addresses. A Bridge network is created when the span of Docker hosts is one i.e. when all containers run on a single host. We need an overlay network to create a network that has a span of more than one Docker host.
2. **Host network**: When a new Docker container is created with the –network=host argument it pushes the container into the host network stack where the Docker daemon is running. All interfaces of the host are accessible from the container which is assigned to the host network.
3. **None network**: When a new Docker container is created with the –network=none argument it puts the Docker container in its network stack. So, in this none network, no IP addresses are assigned to the container, because of which they cannot communicate with each other.

We can assign any one of the networks to the Docker containers. The –network option of the ‘docker run’ command is used to assign a specific network to the container.

```
$docker run --network ="network name"
```

To get detailed information about a particular network we use the command-

```
$docker network inspect "network name"
```

Advantages of Docker –

Docker has become popular nowadays because of the benefits provided by Docker containers. The main advantages of Docker are:

1. **Speed** – The speed of Docker containers compared to a virtual machine is very fast. The time required to build a container is very fast because they are tiny and lightweight. Development, testing, and deployment can be done faster as containers are small. Containers can be pushed for testing once they have been built and then from there on to the production environment.
2. **Portability** – The applications that are built inside docker containers are extremely portable. These portable applications can easily be moved anywhere as a single element and their performance also remains the same.
3. **Scalability** – Docker has the ability that it can be deployed on several physical servers, data servers, and cloud platforms. It can also be run on every Linux machine. Containers can easily be moved from a cloud environment to a local host and from there back to the cloud again at a fast pace.
4. **Density** – Docker uses the resources that are available more efficiently because it does not use a hypervisor. This is the reason that more containers can be run on a single host as compared to virtual machines. Docker Containers have higher performance because of their high density and no overhead wastage of resources.

Last Updated : 11 Jul, 2022

Previous

How to Use Local Docker Images With Minikube?

Next

Virtualisation with Docker Containers

Similar Reads

Load Balancing Flask Application using Nginx and Docker

Installing Helm & Kubernetes in Docker

How to commit your own Docker Customized Image from container?

Docker compose tool to run multi container applications

How to call 'npm start' though docker ?

Next.js Docker Images

Use of Docker Playground

How to find record using any key-value pair information of record in your local/custom database using Node.js ?

Largest Rectangle Area under Histogram using JavaScript | Without using Stacks

Build an E-Commerce Web Application using HTML CSS PHP and hosted using XAMPP

Complete Tutorials

JavaScript Project Ideas with Source Code

Onsen UI

React Material UI

NuxtJS

D3.js

rohnux_26

Follow

Article Tags : Web Technologies

Additional Information

Python Interview Question

DevOps

Git

AWS

Docker

Kubernetes

Azure

GCP

DevOps Roadmap

System Design

What is System Design

Monolithic and Distributed SD

High Level Design or HLD

Low Level Design or LLD

Crack System Design Round

System Design Interview Questions

Grokking Modern System Design

NCERT Solutions

Class 12

Class 11

Class 10

Class 9

Class 8

Complete Study Material

Commerce

Accountancy

Business Studies

Indian Economics

Macroeconomics

Microeconimics

Statistics for Economics

UPSC Study Material

Polity Notes

Geography Notes

History Notes

Science and Technology Notes

Economy Notes

Ethics Notes

Previous Year Papers

Engineering Maths

Competitive Programming

Top DS or Algo for CP

Top 50 Tree

Top 50 Graph

Top 50 Array

Top 50 String

Top 50 DP

Top 15 Websites for CP

JavaScript

TypeScript

ReactJS

NextJS

AngularJS

NodeJS

Express.js

Lodash

Web Browser

School Subjects

Mathematics

Physics

Chemistry

Biology

Social Science

English Grammar

Management & Finance

Management

HR Managament

Income Tax

Finance

Economics

SSC/ BANKING

SSC CGL Syllabus

SBI PO Syllabus

SBI Clerk Syllabus

IBPS PO Syllabus

IBPS Clerk Syllabus

SSC CGL Practice Papers

Colleges

Indian Colleges Admission & Campus Experiences

Top Engineering Colleges

Top BCA Colleges

Top MBA Colleges

Top Architecture College

Choose College For Graduation

Companies

IT Companies

Software Development Companies

Artificial Intelligence(AI) Companies

CyberSecurity Companies

Service Based Companies

Product Based Companies

PSUs for CS Engineers

Preparation Corner

Company Wise Preparation

Preparation for SDE

Experienced Interviews

Internship Interviews

Competitive Programming

Aptitude Preparation

Puzzles

Exams

JEE Mains

JEE Advanced

GATE CS

NEET

UGC NET

More Tutorials

Software Development

Software Testing

Product Management

SAP

SEO

Linux

Excel

Write & Earn

Write an Article

Improve an Article

Pick Topics to Write

Share your Experiences

Internships