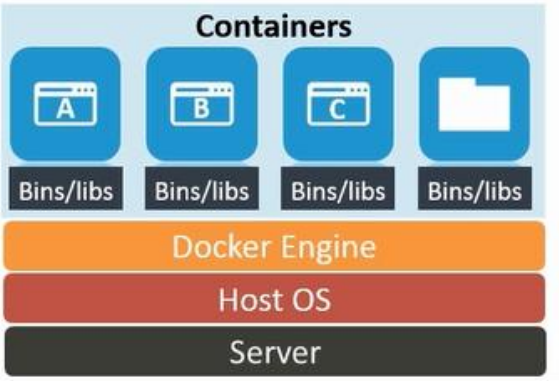
Introduction to Docker

- Docker is an open-source project that automates deployment and execution of template applications inside software containers
- Solomon Hykes started Docker as an internal project within dotCloud, a platform-as-a-service company
- Docker was released as open source platform in March 2013

- The fundamental philosophy in Docker is “Develop; Ship; and Run Any Application, Anywhere”
- Docker helps in rapidly assembling applications from components eliminating friction and dependencies that come while shipping code to varied environments
- Docker allows code to be tested and deployed into production as fast as possible

- Docker is based on open source container virtualization technology
- It is a SaaS service for sharing and managing application stacks



Need for Docker

- Quicker App Delivery
  - A more complete environment is provided to developers and QA
  - Increased visibility of changes is made available to stakeholders
  - Container launch times are much faster compared to traditional deployment and execution. In case of the latter, all dependencies may have to be plugged in independently, which can slow down execution speeds. Containers on the other hand “know” the needed dependencies as part of their packaging and resolve them much faster.

Need for Docker

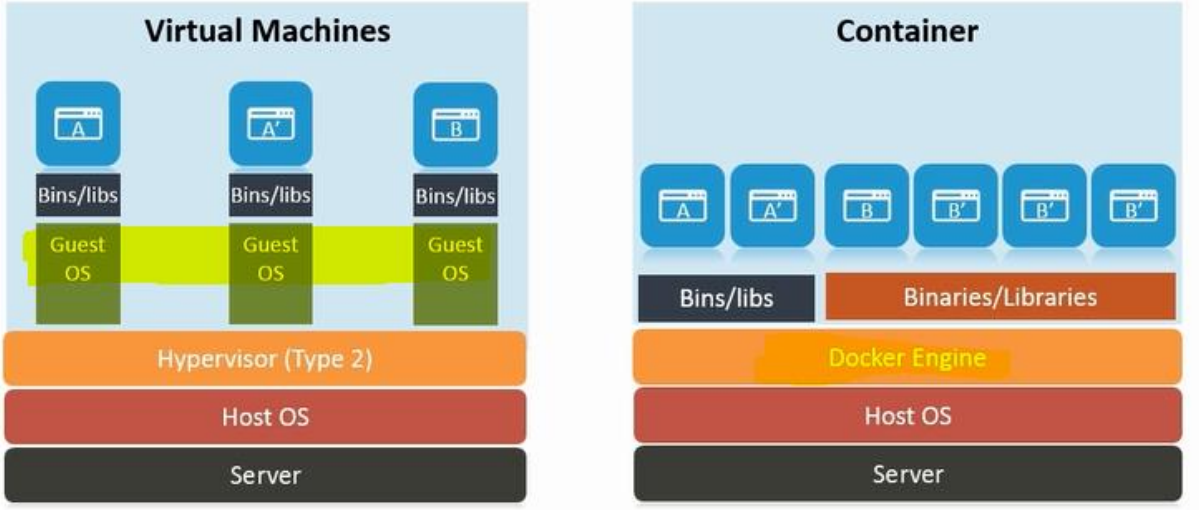
- Deployability and scalability
  - Containers are deployable on any platform that supports kernel services, thereby making executable applications easier to port
  - It is easier to move applications across Docker containers as well because essentially all of them are packaged and run the same way
  - Docker containers are lightweight, with only essential dependencies plugged in. Also, once launched and executed, containers are easy to tear down

- Makes application management easier
  - Containers provide the ability to make incremental changes in application behavior and execution. The file system used by Docker containers allows changes to be stored as layers
  - Consumers have a choice of updating images from a central Docker repository; hence, chances of working with a specific image becomes easier and also, the risk of using a wrong image is reduced

Docker versus Virtual Machines

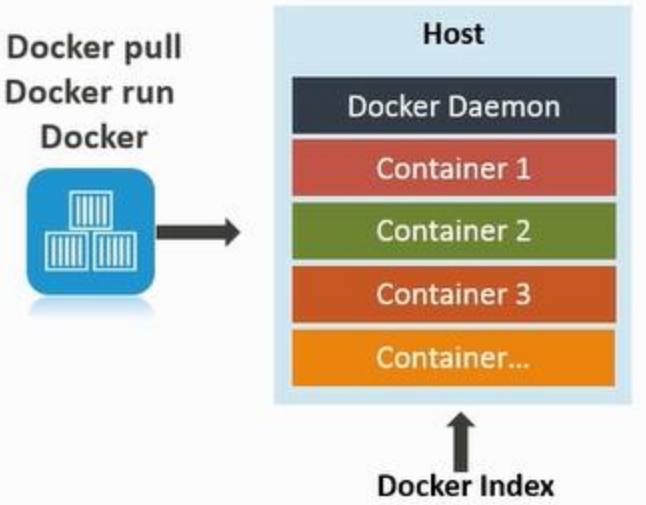
- Virtual machines
  - Virtual machines possess a full operating system with memory and device drivers
  - They need a Hypervisor that manages running of an individual OS
  - Every OS runs as an separate entity from the host operating system

Docker Containers	Virtual Machines
Share a single kernel	Use multiple kernels for each OS instance
Faster and less resource heavy	Heavy-weight and use a lot of system resources
Quick start up and executed within Docker engine	Execute within hypervisor as a separate process



Basic Docker Architecture

- Docker Architecture
  - Docker uses a client-server architecture



In a Host, we may have many containers, each has its own app configs.

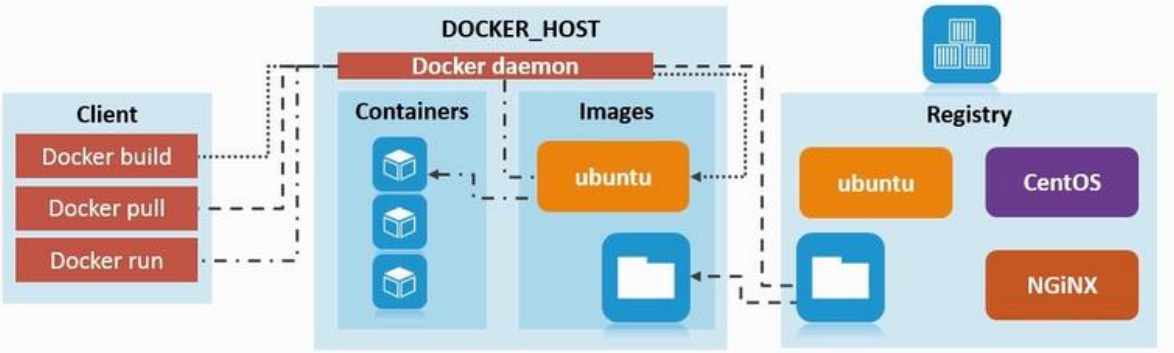
Docker Daemon will be accessed by docker client (pull etc)

End user never talks to Daemon, always talks to Client.

- Docker Daemon
  - Docker client talks to the Docker Daemon
  - Docker Daemon runs on a host machine and does all the heavy lifting required
  - A user does not directly interact with a Daemon

- Docker architecture ensures that:
  - It is operating system agnostic – all that it needs is an OS that can run Docker essential application-related resources are packaged to run an image within a Docker container
  - Docker containers have all the needed internal libraries, network information, and execution environment for an application to be executed

Docker Internals



Client: set of libs and commands, it talks to daemon on Docker Host.

Host has two things, one contains Containers and another is images. On the top we have daemon.

Third is Docker Hub, we can download templates of choice.

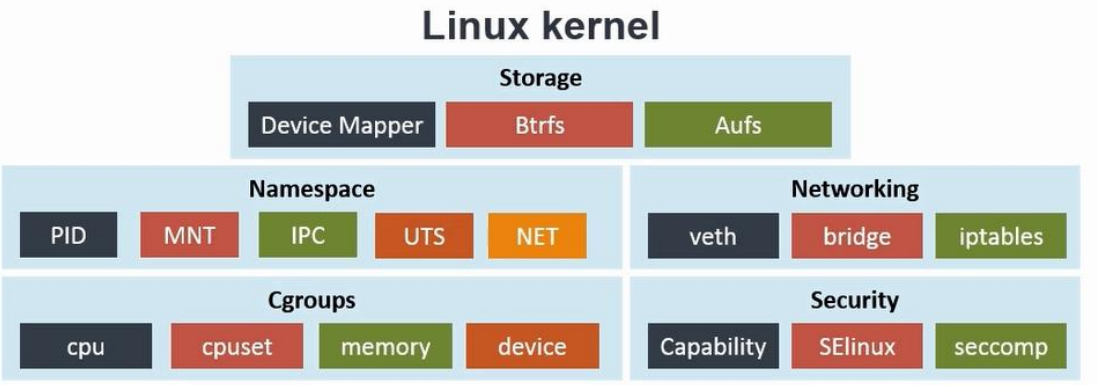
- Docker Client
  - Docker client is in the form of Docker binary that interfaces with Docker containers
  - An application per se cannot talk to a Docker container directly
  - It can run in an interactive mode, communicating back and forth with a Docker daemon

- Docker Host
  - Docker containers reside and run on top of Linux kernel services
  - Docker clients communicate with the containers through daemons that pass commands and results between them
  - Docker containers work on application images pulled from local or remote registries
- Docker Registry
  - Docker registry is the central repository of all image templates that can be pulled into a Docker container
  - Registries can be updated with modified images, which can be subsequently requested for by Docker clients for execution

Namespaces

- Docker containers are essentially LXC containers, and namespaces are an integral part of Linux container architecture
- LXC containers are an OS-level virtualization environment for running multiple isolated Linux systems (containers) on a single Linux control host. Docker containers inherit properties of LXC containers, such a Namespaces, Cgroups, storage, and so on

- Namespaces provide a way of isolating containers



All these are shipped by Linux by default and will be used by Docker. (inheritance).

Storage, namespace, networking, control groups and security.

- Namespaces provide a way of isolating containers
- Processes running within a container cannot affect processes running outside it

- pid namespace – used for process isolation
- net namespace – used for managing network interfaces
- ipc namespace – used for managing interprocess communication
- mnt namespace – used for managing mounting points
- uts namespace – used for isolating kernel

Control Groups

- Control groups have been part of Linux kernel since 2006, starting from kernel version 2.6.24
- Control groups were built upon the notion of restricting resource access to a group of processes

- Design goals of Control Groups (Cgroups)
  - Resource limitation: Specific groups can be set not to exceed a configured memory limit including file system cache
  - Prioritization: Certain groups may be configured to get a larger share of CPU utilization or disk I/O throughput, depending on processes

- Accounting: Measures how much resources certain systems can use in a given set of conditions
- Control: Controls freezing groups of processes from execution, check pointing (for rollback), and restarting (if necessary)

Union Systems

- Union File System (UFS) is a storage system which implements a policy mounting file systems one over the other
- Its allows files and directories (known as branches) to be transparently overlaid to form a single cohesive file system
- When two branches have the same file name, the branch that was last mounted gets priority

- Docker uses Union File Systems because they are fast and lightweight
- They are building blocks of Docker containers. Each update or edit to an image is mounted as a separate layer
- Docker makes use of several union file system variants, including AUFS, btrfs, vfs, and DeviceMapper