Docker Terminology

Docker client – command line that can be installed on Mac OS X or Windows to control most of the Docker workflow and talk to Docker servers  
Docker server – Docker running in daemon mode on a Linux server. The servers starts, runs, and stops/destroys containers.  
Docker images – layered filesystems described via metadata that represent all the dependencies necessary to run an application.  
Docker repository – the library of available Docker images, can be public (Docker Hub) or private  
Docker container – a single instance of a Linux container built from the Docker image. There can be many containers built from a single Docker image.  
Docker host – the underlying Linux operating system that hosts the Docker daemon  
In addition to the core lingo, there are some core technologies that need to be understood as well:

Docker Hub – public repository of Docker images  
Docker Trusted Registry – private image repository  
Docker Toolbox – specialized installer for Docker tools on Mac and Windows  
Docker Machine – automated Docker provisioning  
Docker Swarm – host clustering and container scheduling  
Docker Compose – define multi-container applications  
Docker Registry – open source Docker image distribution  
Docker Engine – creates and runs Docker containers  
Kitematic – desktop Docker GUI for Mac  
Notary – trusted content distribution for signing images  
RunC – universal container runtime  
Libnetwork – programming interface for the Container Network Model (CNM)

**Image** – A binary file that includes everything needed to run a Docker **container**, including metadata describing capabilities and dependencies. All images are immutable, but images are comprised of multiple layers, so modifications and additions can be made by adding a layer. Image layering is realized by using [union mounting](https://en.wikipedia.org/wiki/Union_mount). Common underlying layers can be shared among images in the same Docker Engine. A base operating system layer might be shared by most, or all of the images on a given host. This sharing allows most images to be quite small relative to something like a VM disk image.

**Container** – A runtime instance of a Docker **image**. A container is created using the **docker run** command. Multiple containers can run concurrently using the same Docker image. A container is a lightweight isolation mechanism for processes. Multiple containers can run on a single host with no knowledge of other containers. Containers have no access to the processes, files, or networks of other containers (or standalone processes) on the host.

**Registry**– A place where **images** are stored. Images in a registry are typically available for general use. When a **container**is started, it will **pull** its required image from a registry. The most obvious example of a registry is [Docker Hub](https://hub.docker.com/), although there are other third party registries like [Google Container Registry](https://cloud.google.com/container-registry/), [Amazon EC2](https://aws.amazon.com/ecr/), and [Quay](https://quay.io/). It’s also possible to deploy a private registry of your own (e.g., using Docker Trusted Registry). A registry supports searching for images.

**Repository**– A set of Docker **images**. The images in a repository have the same name and provide the same functionality. They may vary by version (e.g., 1.1, 1.2, etc.) or by composition (e.g., one may be based on [Ubuntu](https://www.ubuntu.com/) and another on Alpine Linux). The separate images in a repository are distinguished by their **tags**. For example, Docker Hub hosts an **nginx** repository which contains the images: nginx:1.13, nginx:1.12, nginx:1.13-alpine … one or more of the images in a repository may be **pushed** to a **registry**.

**Tag** – Label attached to a Docker **image** in a **repository**.

**Build** – The process of creating an **image** from a **Dockerfile**.

**Pull** – The process of retrieving/obtaining an **image** from a **registry**.

**Push** – The process of delivering/publishing an **image** to a **registry**.

**Compose** – A tool for defining and running multi-**container** applications. The containers are all defined and configured from data in a single file. All the containers can be started or stopped with a single command.

**Dockerfile** – A text document that contains the execute commands needed to build a Docker **image**. An image is produced from a Dockerfile using the **docker build** command.

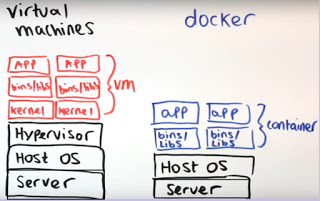
**Docker Hub** – Public registry of Docker images.

**Docker Swarm** – A native clustering system for Docker, not to be confused with Swarm Mode. This is Docker’s first container orchestration implementation. If you are in doubt about whether to use “swarm” or “swarm mode”, Docker recommends that you try “swarm mode” first.

**Swarm Mode** – Cluster management and orchestration features integrated into the Docker Engine. This is Docker’s latest implementation of orchestration and cluster management — available in Docker Engine v1.12 and later.

Docker is a containerisation platform which package our application and all its dependencies together in form of containers to ensure that our application works seamlessly in any environment be it Development, Test or Production.

Docker is not just another stack in virtual machine (VM) concept. To relate containerisation is just virtualisation at the OS level.

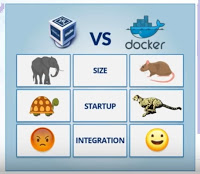


Why Docker?

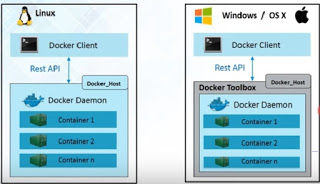
Containers on same OS kernel are lighter & smaller.

Containerisation brings process level isolation.

Better resource Utilisation compared to VM.

Short Boot-up process (1/20th of a second). 

Docker Engine

Communication happens between docker client & Docker Daemon by combination of REST API, Socket.IO and TCP. 

Docker Daemon:

It run on host Machine.

It creates and manages dockers object such as images, containers, Networks, Data, Volumes etc.

User does not directly interact with the docker client.

Docker Client:

Primary user interface to docker.

It accepts commands from the users and communicates back and forth with dockers daemon.

Docker Images:

Images are used to create docker containers.

Dockers provides a simple way to build new images or update existing images,

Docker images are the 'Build' component of dockers.

Docker Registries:

Registries store images.

These are public or private stored from which we upload/downloads images.

This can be done on docker hub (which is docker version of GITHUB)

Docker Registry are 'distribution' component of docker.

Docker Containers:

Containers are created from docker images.

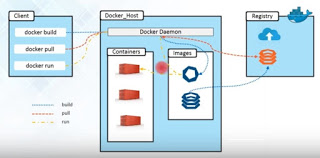
It holds everything needed for an application to run.

Each containers is an isolated and secure application platform.

Docker containers are the 'run' components of docker.

Docker Architecture:

It has following component:

build--- pull --- distribute --- run

**Container image**: A package with all the dependencies and information needed to create a container. An image includes all the dependencies (such as frameworks) plus deployment and execution configuration to be used by a container runtime. Usually, an image derives from multiple base images that are layers stacked on top of each other to form the container’s filesystem. An image is immutable once it has been created.

**Container**: An instance of a Docker image. A container represents the execution of a single application, process, or service. It consists of the contents of a Docker image, an execution environment, and a standard set of instructions. When scaling a service, you create multiple instances of a container from the same image. Or a batch job can create multiple containers from the same image, passing different parameters to each instance.

**Tag**: A mark or label you can apply to images so that different images or versions of the same image (depending on the version number or the target environment) can be identified.

**Dockerfile**: A text file that contains instructions for how to build a Docker image.

**Build**: The action of building a container image based on the information and context provided by its Dockerfile, plus additional files in the folder where the image is built. You can build images with the Docker docker build command.

**Repository (repo)**: A collection of related Docker images, labeled with a tag that indicates the image version. Some repos contain multiple variants of a specific image, such as an image containing SDKs (heavier), an image containing only runtimes (lighter), etc. Those variants can be marked with tags. A single repo can contain platform variants, such as a Linux image and a Windows image.

**Registry**: A service that provides access to repositories. The default registry for most public images is [Docker Hub](https://hub.docker.com/) (owned by Docker as an organization). A registry usually contains repositories from multiple teams. Companies often have private registries to store and manage images they’ve created. Azure Container Registry is another example.

**Docker Hub**: A public registry to upload images and work with them. Docker Hub provides Docker image hosting, public or private registries, build triggers and web hooks, and integration with GitHub and Bitbucket.

**Azure Container Registry**: A public resource for working with Docker images and its components in Azure. This provides a registry that is close to your deployments in Azure and that gives you control over access, making it possible to use your Azure Active Directory groups and permissions.

**Docker Trusted Registry (DTR)**: A Docker registry service (from Docker) that can be installed on-premises so it lives within the organization’s datacenter and network. It is convenient for private images that should be managed within the enterprise. Docker Trusted Registry is included as part of the Docker Datacenter product. For more information, see [Docker Trusted Registry (DTR)](https://docs.docker.com/docker-trusted-registry/overview/).

**Docker Community Edition (CE)**: Development tools for Windows and macOS for building, running, and testing containers locally. Docker CE for Windows provides development environments for both Linux and Windows Containers. The Linux Docker host on Windows is based on a [Hyper-V](https://www.microsoft.com/en-us/server-cloud/solutions/virtualization.aspx) virtual machine. The host for Windows Containers is directly based on Windows. Docker CE for Mac is based on the Apple Hypervisor framework and the [xhyve hypervisor](https://github.com/mist64/xhyve), which provides a Linux Docker host virtual machine on Mac OS X. Docker CE for Windows and for Mac replaces Docker Toolbox, which was based on Oracle VirtualBox.

**Docker Enterprise Edition (EE)**: An enterprise-scale version of Docker tools for Linux and Windows development.

**Compose**: A command-line tool and YAML file format with metadata for defining and running multi-container applications. You define a single application based on multiple images with one or more .yml files that can override values depending on the environment. After you have created the definitions, you can deploy the whole multi-container application with a single command (docker-compose up) that creates a container per image on the Docker host.

**Cluster**: A collection of Docker hosts exposed as if it were a single virtual Docker host, so that the application can scale to multiple instances of the services spread across multiple hosts within the cluster. Docker clusters can be created with Docker Swarm, Mesosphere DC/OS, Kubernetes, and Azure Service Fabric. (If you use Docker Swarm for managing a cluster, you typically refer to the cluster as a swarm instead of a cluster.)

**Orchestrator**: A tool that simplifies management of clusters and Docker hosts. Orchestrators enable you to manage their images, containers, and hosts through a command line interface (CLI) or a graphical UI. You can manage container networking, configurations, load balancing, service discovery, high availability, Docker host configuration, and more. An orchestrator is responsible for running, distributing, scaling, and healing workloads across a collection of nodes. Typically, orchestrator products are the same products that provide cluster infrastructure, like Mesosphere DC/OS, Kubernetes, Docker Swarm, and Azure Service Fabric.