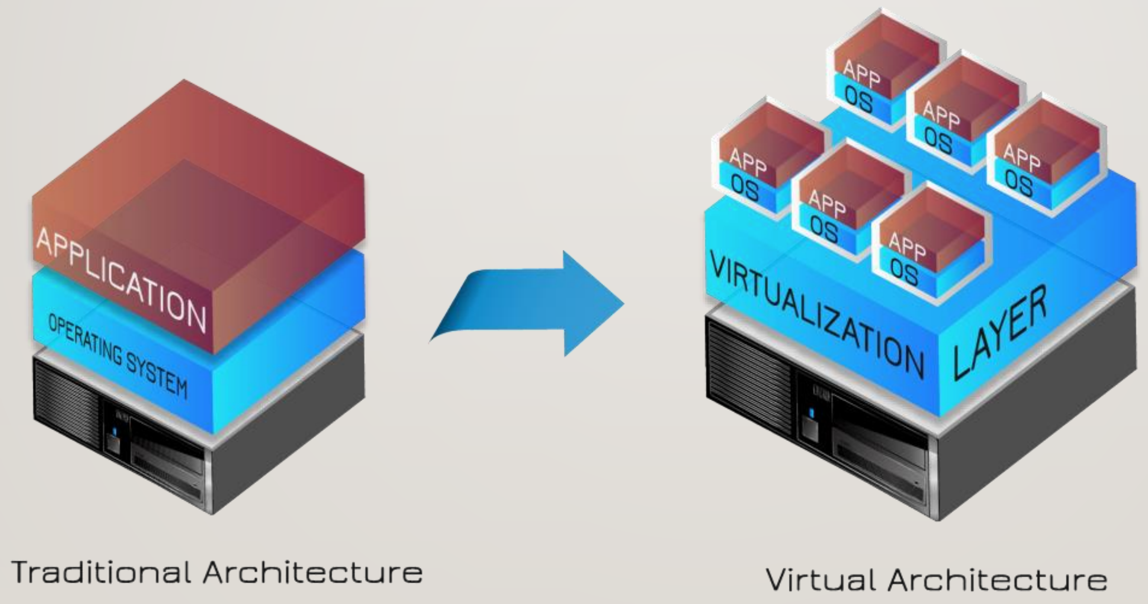
Getting Started with Docker
Module -1

Agenda

Introduction to Virtualization Technologies

- Today's x86 computer hardware was designed to run a single operating system and a single application, leaving most machines vastly underutilized.
- Virtualization lets you run multiple virtual machines on a single physical machine, with each virtual machine sharing the resources of that one physical computer across multiple environments.
- Different virtual machines can run different operating systems and multiple applications on the same physical computer.

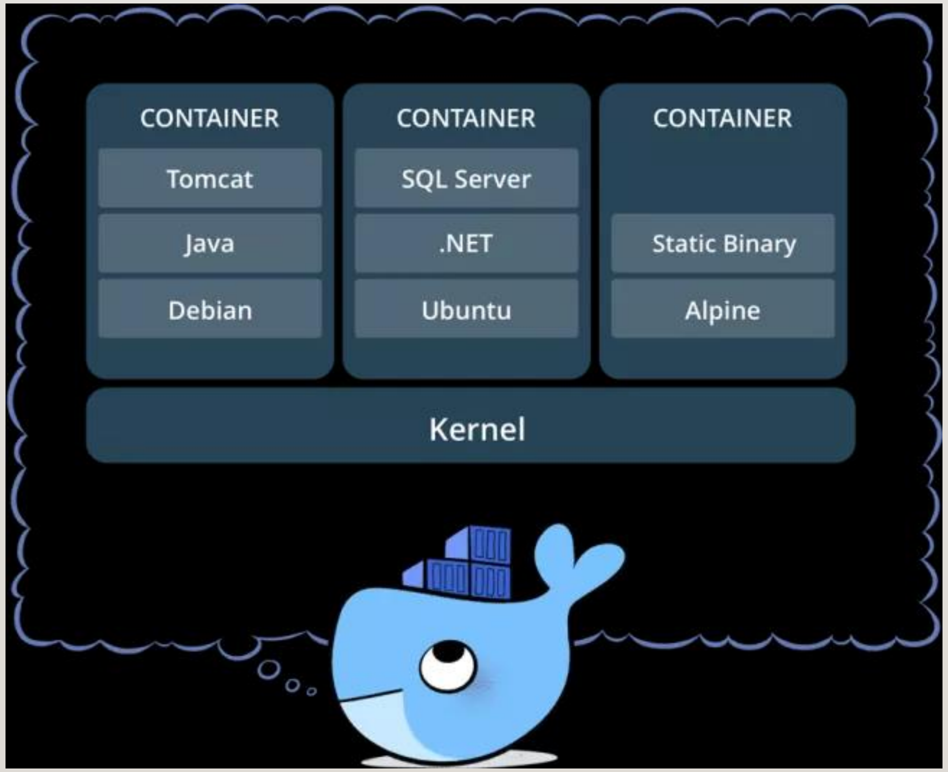


Benefits of Virtualization

Improve Effectiveness and Reduce Costs	**Increased Business Agility**	**Maximize Business Continuity**
• Maximize Asset Utilization • Reduced Power and cooling needs • Lower Capital and Operational expenditure • Lower Carbon footprint • Increased IT staff productivity • Reduced Software Licensing ,hardware management costs	• Dynamically scale up or back • Rapid response to changing business needs • Faster provisioning of services and infrastructure	• Optimize Availability • Application Isolation • Centralized Management • Simplify disaster recovery • Increase security

What is Docker

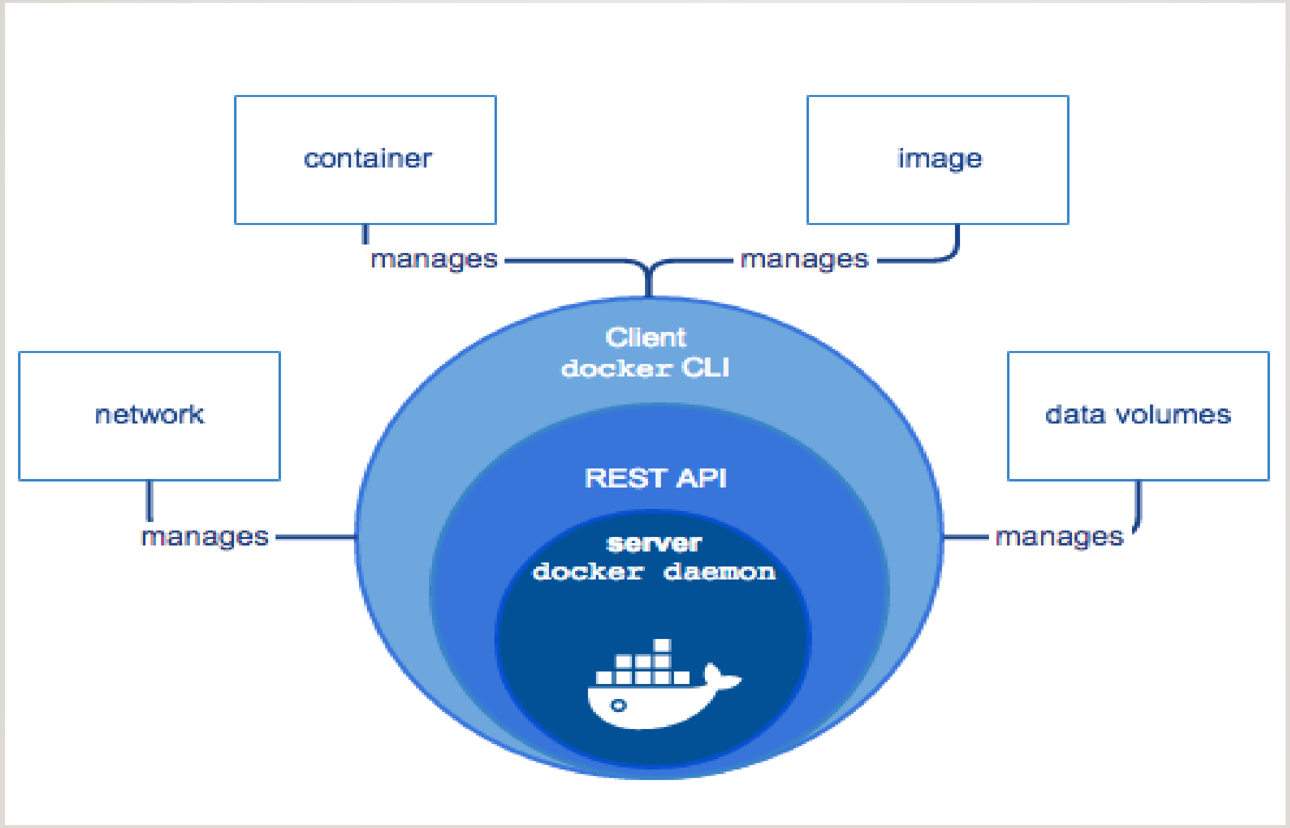
- Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.



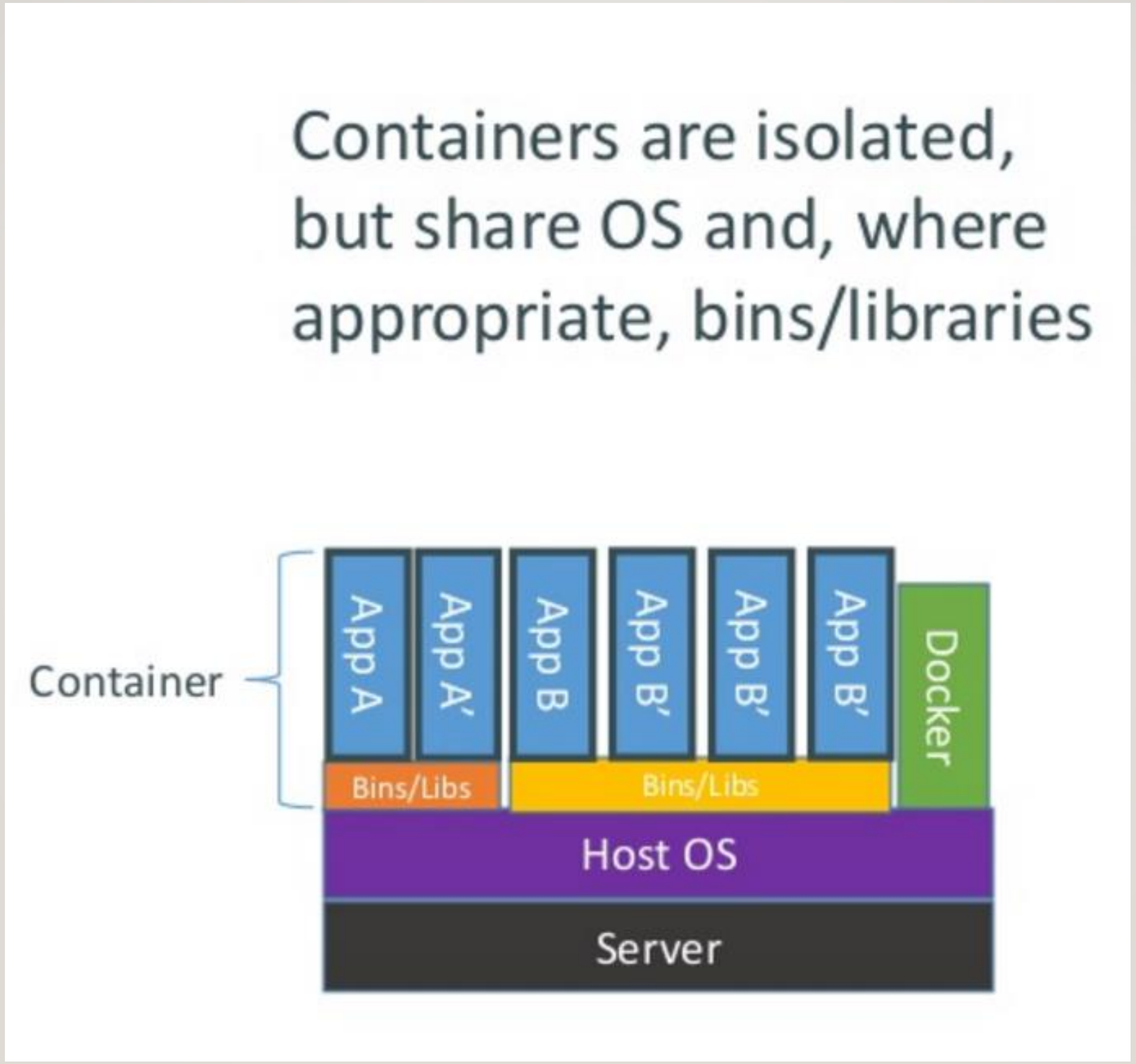
Docker Engine

Docker Engine is a client-server application with these major components:

- A server which is a type of long-running program called a daemon process (the dockerd command).
- A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
- A command line interface (CLI) client (the docker command).



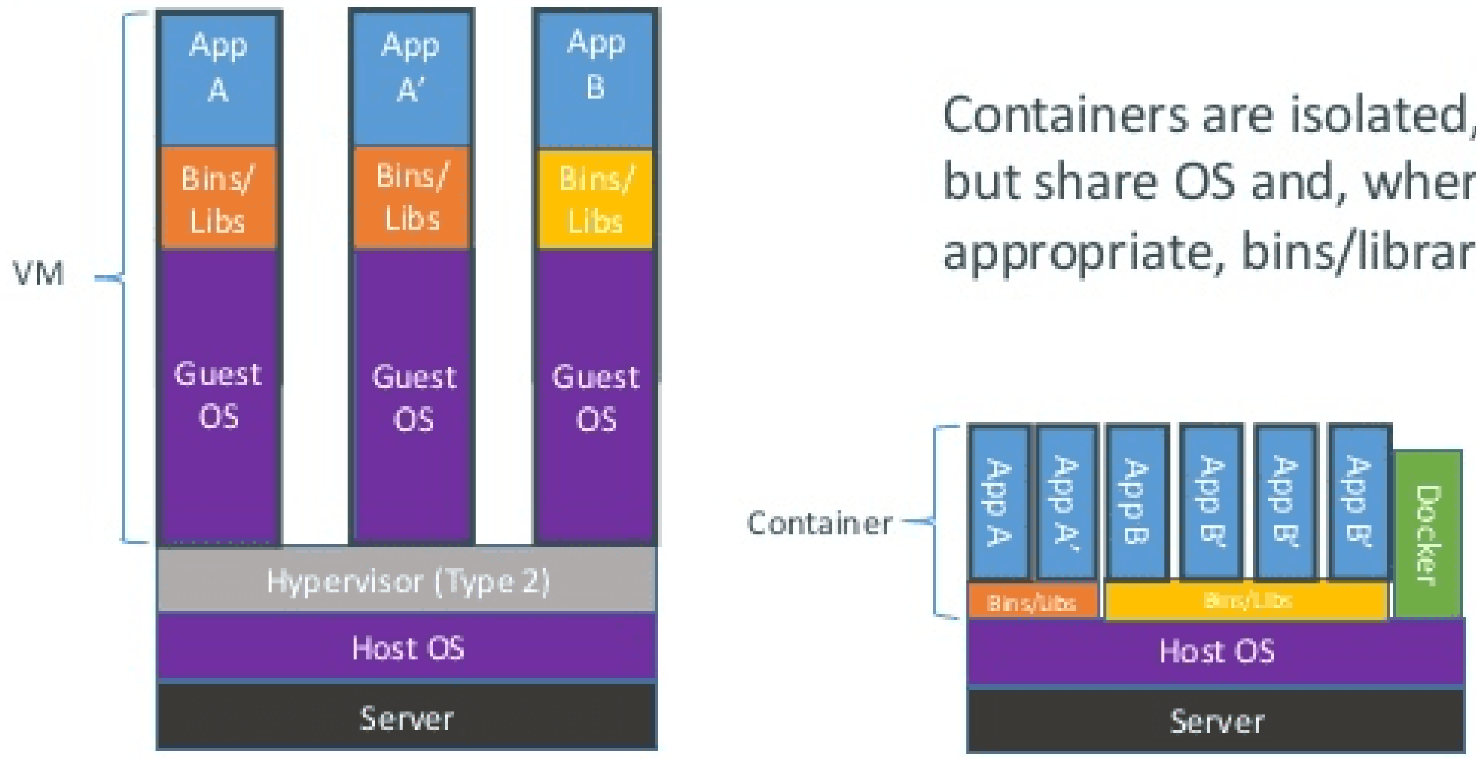
How does Docker Containers Work ?


Containers are isolated, but share OS and, where appropriate, bins/libraries

Comparison between Containers & Virtual Machines

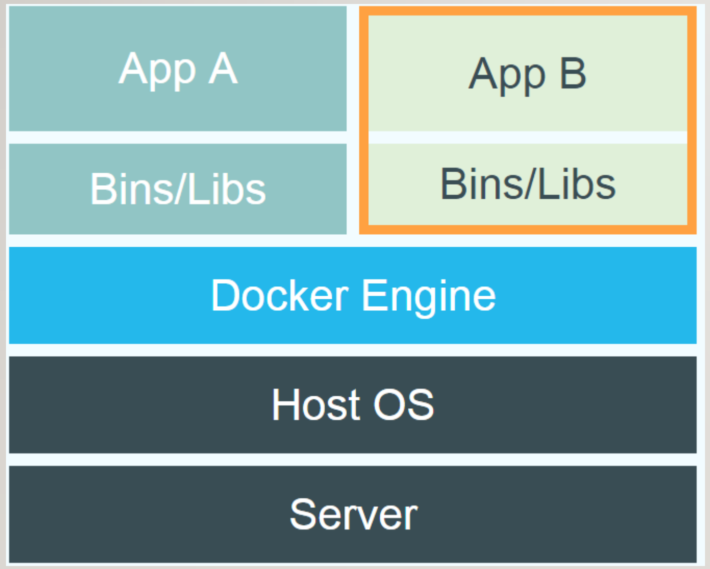
Containers vs. VMs

Containers are isolated, but share OS and, where appropriate, bins/libraries

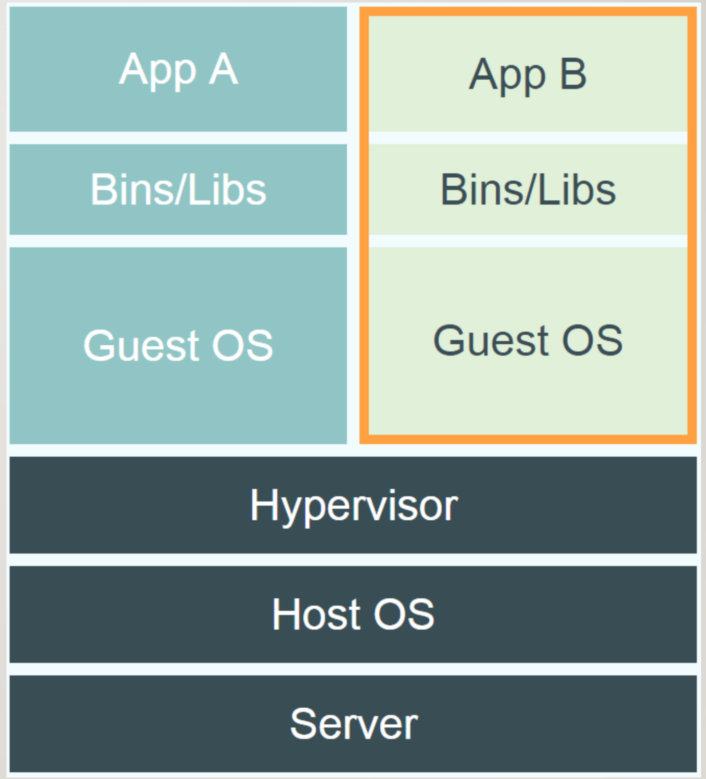


Differences between Dockers and Virtual Machines

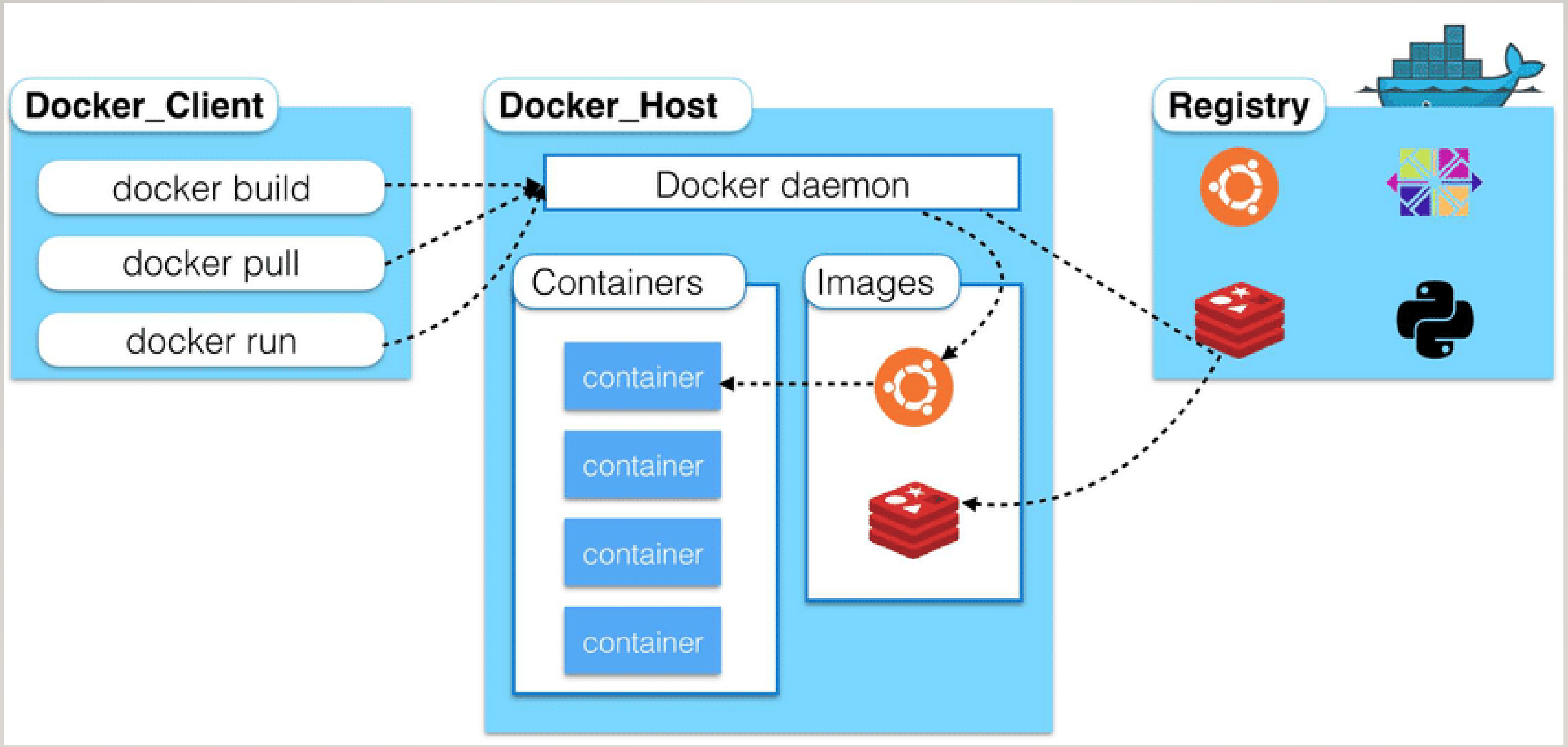
Dockers



Virtual Machines



Docker Architecture



Important Docker Concepts

The Docker daemon

- The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

The Docker Client

- The Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

Docker Registries :

- A Docker registry stores Docker images. Docker Store and Docker Hub are public registries that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run your own private registry. If you use Docker Datacenter (DDC), it includes Docker Trusted Registry (DTR).
- When you use the docker pull or docker run commands, the required images are pulled from your configured registry. When you use the docker push command, your image is pushed to your configured registry.

Docker Installation

- **Docker for Linux :**

- Docker was originally a Linux application
- It uses the kernel container functionality
- It requires a 64 bit installation using a kernel version 3.10 or later
- Docker runs on many popular Linux distributions
- It is available as RPM, APT, or binary versions

Docker Installation

Docker for Mac OS

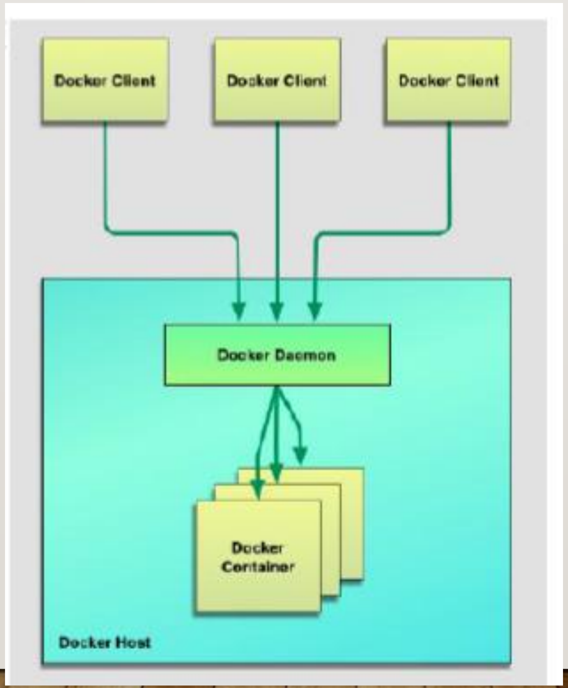
- Docker runs natively on OS X
- Is built on the xhyve hypervisor
- Requires a 2010 or newer Mac with Intel MMU and EPT support
- Requires OS X 10.10.3 Yosemite or newer
- Requires at least 4GB of RAM
- Docker instances can't be accessed remotely due to limited network support

Docker for Windows:

- Docker runs natively on Windows
- Requires later versions of Windows 10 Pro or Enterprise
- Docker requires the Hyper-V package
- This is Microsoft's hypervisor for Windows
- It virtualizes the Docker environment and Linux kernel specific features
- Docker can't run alongside VirtualBox VMs

Docker Toolbox

- For computers not meeting these specification, Docker Toolbox is available.
- It requires a 64 bit operating system
- It is a local installation of Docker-machine, a client program to control the Docker daemon, and a local VirtualBox compatible version
- A local install of VM such as VirtualBox hosts Linux which runs Docker Engine
- A remote computer prepared to run Docker Engine



Hello World Docker Container

- Docker allows us to run applications inside containers. Running an application inside a container takes a single command: **docker run**.

```
$ docker run hello-world


Hello from Docker!
This message shows that your installation appears to be working correctly.


To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.
```

Deep Dive into Containers

- Containers are lightweight because they don’t need the extra load of a hypervisor, but run directly within the host machine’s kernel. This means you can run more containers on a given hardware combination than if you were using virtual machines. You can even run Docker containers within host machines that are actually virtual machines.

- Docker provides tooling and a platform to manage the lifecycle of your containers:

- Develop your application and its supporting components using containers.

- The container becomes the unit for distributing and testing your application.

Underlying Container Technology

- Docker is written in Go and takes advantage of several features of the Linux kernel to deliver its functionality.

Namespaces :

- Docker uses a technology called namespaces to provide the isolated workspace called the container. When you run a container, Docker creates a set of namespaces for that container.
- These namespaces provide a layer of isolation. Each aspect of a container runs in a separate namespace and its access is limited to that namespace.
- Docker Engine uses namespaces such as the following on Linux:
- The pid namespace: Process isolation (PID: Process ID).
- The net namespace: Managing network interfaces (NET: Networking).
- The ipc namespace: Managing access to IPC resources (IPC: InterProcess Communication).
- The mnt namespace: Managing filesystem mount points (MNT: Mount).
- The uts namespace: Isolating kernel and version identifiers. (UTS: Unix Timesharing System).

Underlying Container Technology

Control groups

- Docker Engine on Linux also relies on another technology called control groups (cgroups). A cgroup limits an application to a specific set of resources. Control groups allow Docker Engine to share available hardware resources to containers and optionally enforce limits and constraints. For example, you can limit the memory available to a specific container.

Union file systems

- Union file systems, or UnionFS, are file systems that operate by creating layers, making them very lightweight and fast. Docker Engine uses UnionFS to provide the building blocks for containers. Docker Engine can use multiple UnionFS variants, including AUFS, btrfs, vfs, and DeviceMapper.

Container format

- Docker Engine combines the namespaces, control groups, and UnionFS into a wrapper called a container format. The default container format is libcontainer. In the future, Docker may support other container formats by integrating with technologies such as BSD Jails or Solaris Zones.

Docker Port Mapping and Docker Logs Command

- By default, when you create a container, it does not publish any of its ports to the outside world. To make a port available to services outside of Docker, or to Docker containers which are not connected to the container's network, use the --publish or -p flag. This creates a firewall rule which maps a container port to a port on the Docker host. Here are some examples.

Flag value	**Description**
-p 8080:80	Map TCP port 80 in the container to port 8080 on the Docker host.
-p 8080:80/udp	Map UDP port 80 in the container to port 8080 on the Docker host.
-p 8080:80/tcp -p 8080:80/udp	Map TCP port 80 in the container to TCP port 8080 on the Docker host, and map UDP port 80 in the container to UDP port 8080 on the Docker host.

Docker Logs

- Fetch the logs of a container

```
docker logs [OPTIONS] CONTAINER
```

- The docker logs command batch-retrieves logs present at the time of execution.

- In order to retrieve logs before a specific point in time, run:

- `$ docker run --name test -d busybox sh -c "while true; do $(echo date); sleep 1; done"`
- `$ date`

```
Tue 14 Nov 2017 16:40:00 CET
```

- `$ docker logs -f --until=2s`

```
Tue 14 Nov 2017 16:40:00 CET
Tue 14 Nov 2017 16:40:01 CET
Tue 14 Nov 2017 16:40:02 CET
```

KNOWLEDGE CHECK

Thank You