Docker Fundamentals

.NET CORE

Docker provides the ability to package and run an application in a loosely isolated environment called a ***container****. You can run many containers simultaneously on a given host. Containers don't need a hypervisor and run directly within the host machine's kernel and can even run within virtual host machines.*

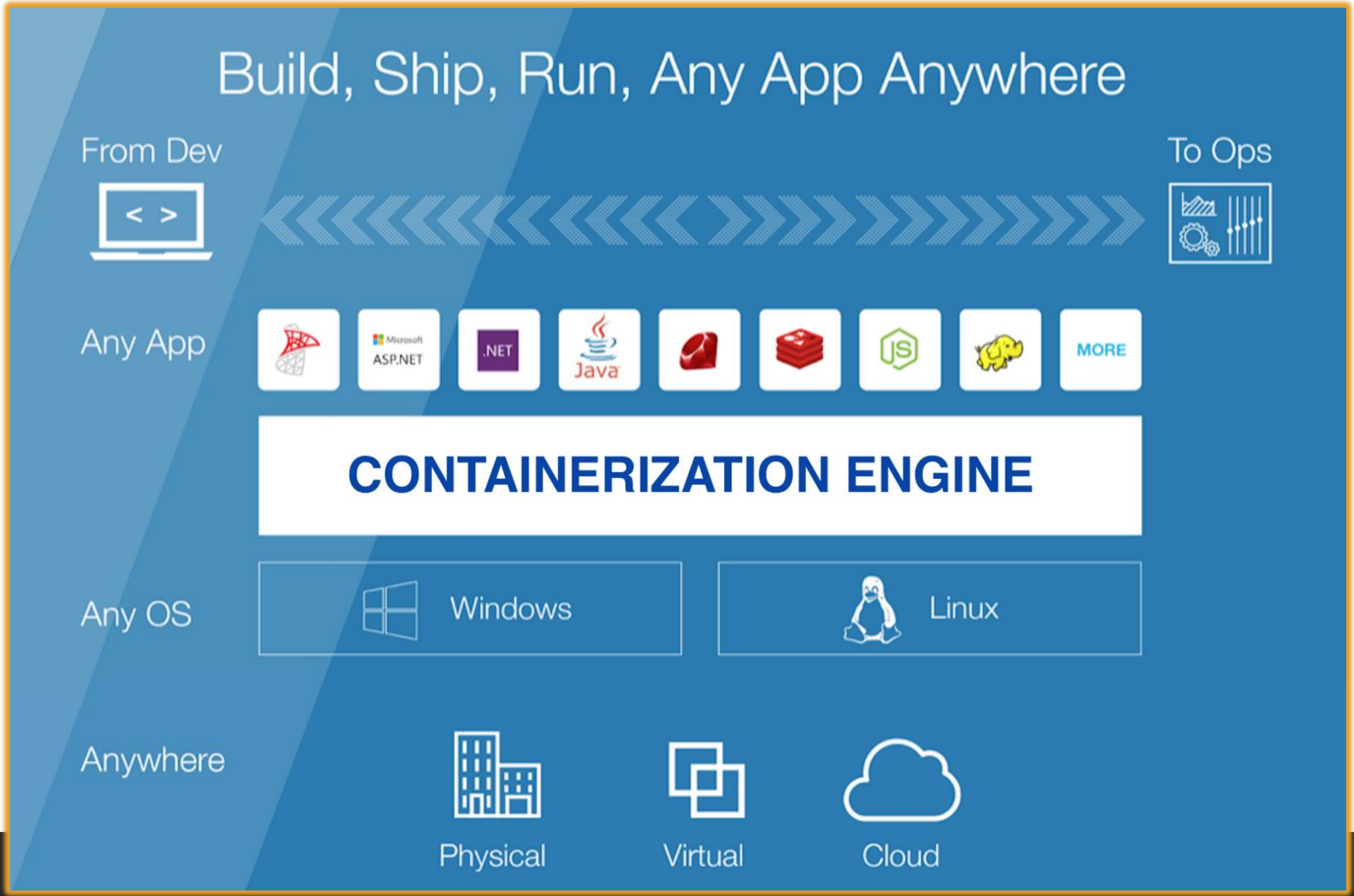
HTTPS://DOCS.DOCKER.COM/ENGINE/DOCKER-OVERVIEW/

What is Containerization?

https://hackernoon.com/what-is-containerization-83ae53a709a6

Containerization involves bundling an application together with all its related configuration files, libraries, and dependencies required for it to run efficiently and bug-free across different computing environments.

The most popular *containerization* ecosystems are *Docker* and Kubernetes.



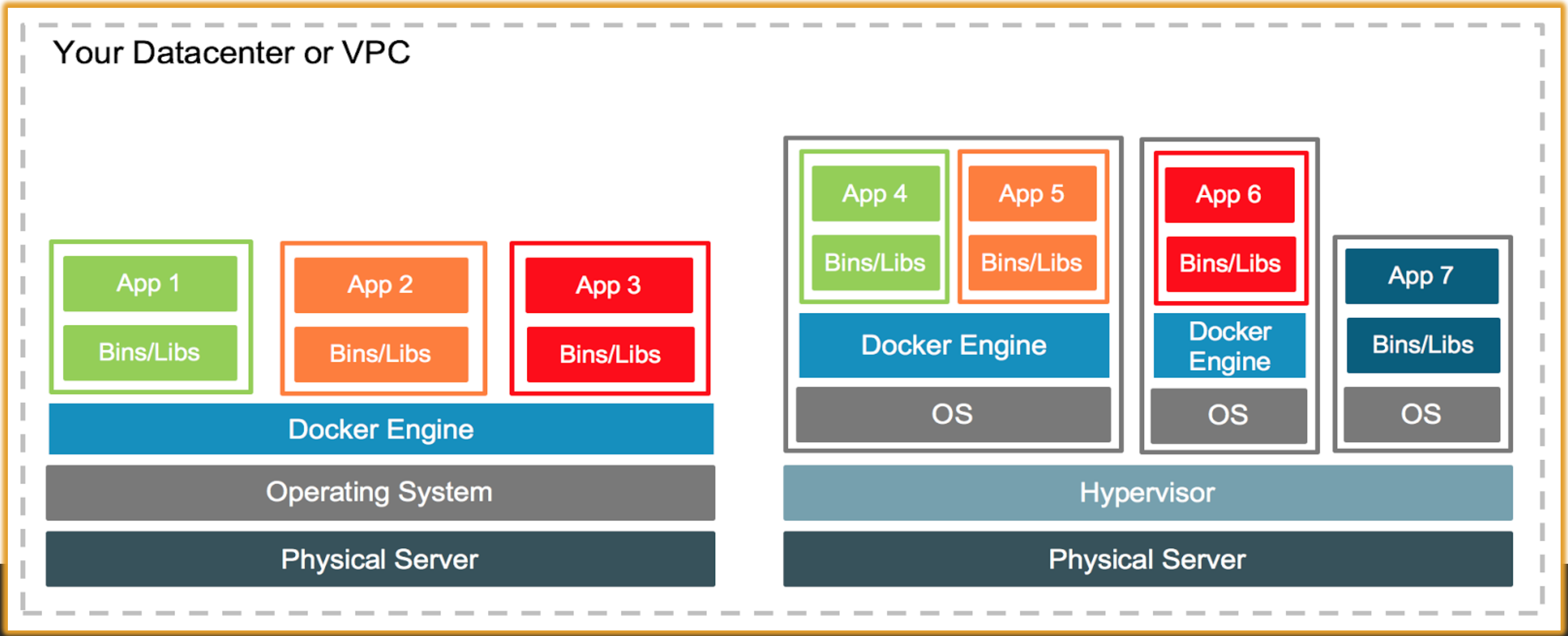
What is Virtualization?

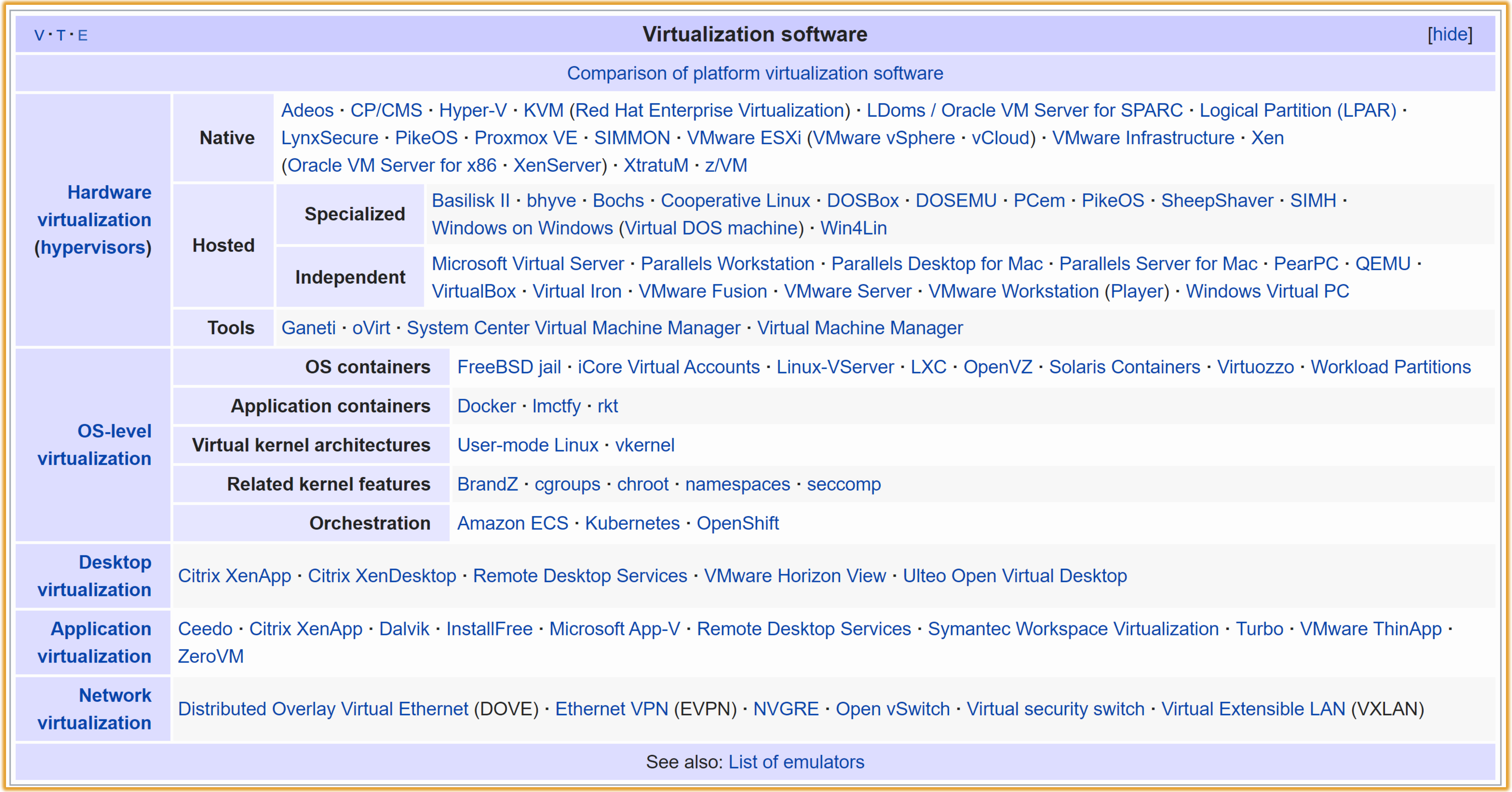
https://en.wikipedia.org/wiki/Virtualization#Hardware_virtualization
https://www.docker.com/blog/containers-and-vms-together/

In computing, virtualization refers to the act of creating a virtual (rather than actual) version of something.

Hardware ***virtualization*** or platform ***virtualization*** refers to the creation of a virtual machine (an application) that simulates a real computer with an operating system.

Software executed on a virtual machine is separated from the underlying hardware resources.





V · T · E

Virtualization software

[hide]

Comparison of platform virtualization software

Hardware virtualization (hypervisors)	Native		Adeos · CP/CMS · Hyper-V · KVM (Red Hat Enterprise Virtualization) · LDoms / Oracle VM Server for SPARC · Logical Partition (LPAR) · LynxSecure · PikeOS · Proxmox VE · SIMMON · VMware ESXi (VMware vSphere · vCloud) · VMware Infrastructure · Xen (Oracle VM Server for x86 · XenServer) · XtratuM · z/VM
	Hosted	Specialized	Basilisk II · bhyve · Bochs · Cooperative Linux · DOSBox · DOSEMU · PCem · PikeOS · SheepShaver · SIMH · Windows on Windows (Virtual DOS machine) · Win4Lin
		Independent	Microsoft Virtual Server · Parallels Workstation · Parallels Desktop for Mac · Parallels Server for Mac · PearPC · QEMU · VirtualBox · Virtual Iron · VMware Fusion · VMware Server · VMware Workstation (Player) · Windows Virtual PC
	Tools		Ganeti · oVirt · System Center Virtual Machine Manager · Virtual Machine Manager
OS-level virtualization	OS containers		FreeBSD jail · iCore Virtual Accounts · Linux-VServer · LXC · OpenVZ · Solaris Containers · Virtuozzo · Workload Partitions
	Application containers		Docker · lmctfy · rkt
	Virtual kernel architectures		User-mode Linux · vkernel
	Related kernel features		BrandZ · cgroups · chroot · namespaces · seccomp
	Orchestration		Amazon ECS · Kubernetes · OpenShift
Desktop virtualization			Citrix XenApp · Citrix XenDesktop · Remote Desktop Services · VMware Horizon View · Ulteo Open Virtual Desktop
Application virtualization			Ceedo · Citrix XenApp · Dalvik · InstallFree · Microsoft App-V · Remote Desktop Services · Symantec Workspace Virtualization · Turbo · VMware ThinApp · ZeroVM
Network virtualization			Distributed Overlay Virtual Ethernet (DOVE) · Ethernet VPN (EVPN) · NVGRE · Open vSwitch · Virtual security switch · Virtual Extensible LAN (VXLAN)

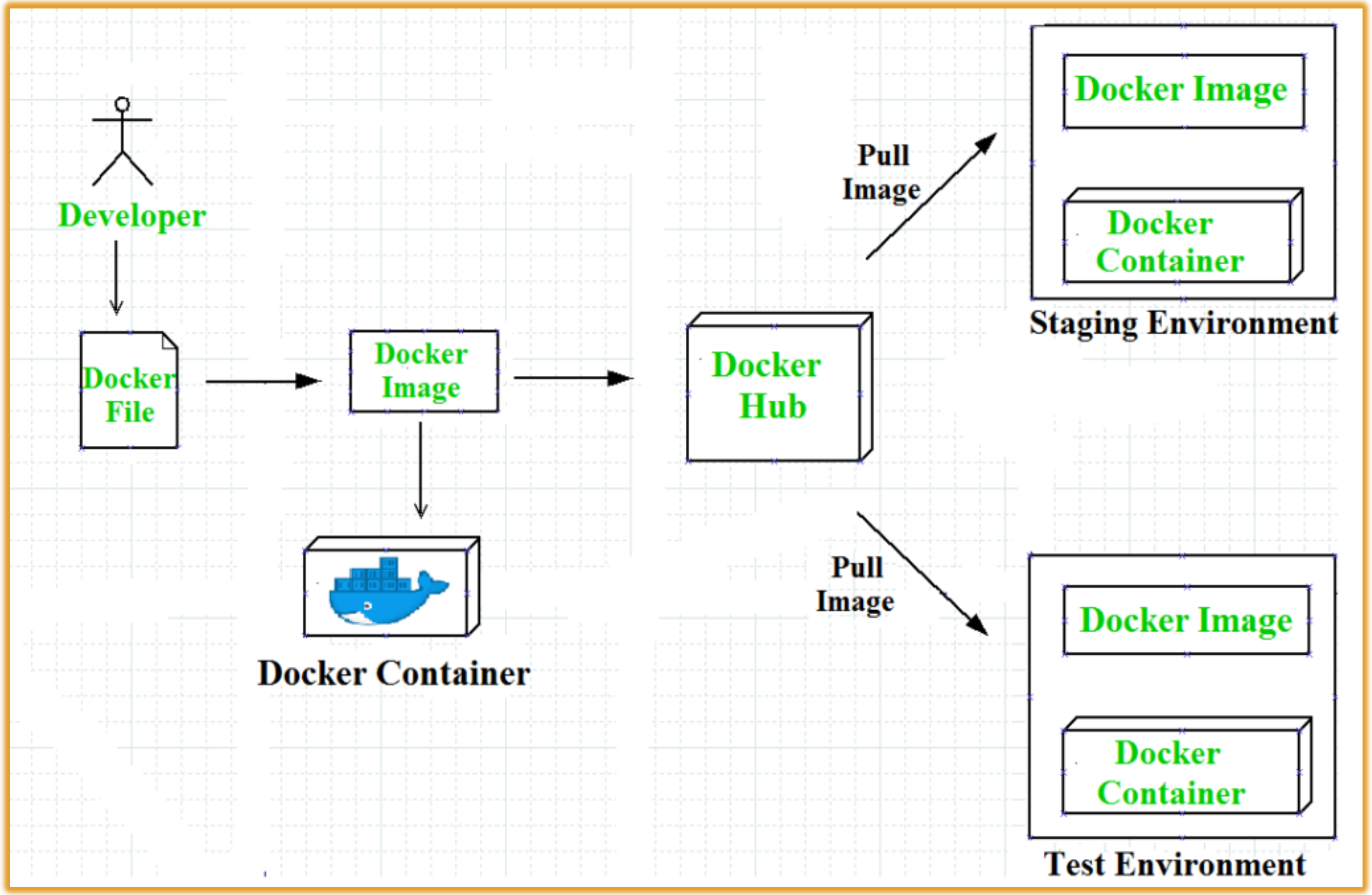
See also: List of emulators

Docker – Purpose

https://docs.docker.com/engine/docker-overview/#what-can-i-use-docker-for

Docker allows developers to work in standardized environments using ***containers*** which provide applications and services. ***Containers*** are great for *CI/CD* workflows.

1. Developers write code locally in a ***development environment*** and share their work using Docker ***containers***.
2. They use Docker to push their applications into a test environment to execute automated and manual tests.
3. Bugs can be fixed in the ***development environment*** and redeployed to the ***test environment*** for re-testing and validation.
4. When testing is complete, push the updated image to the ***production environment***.



Docker – Benefits

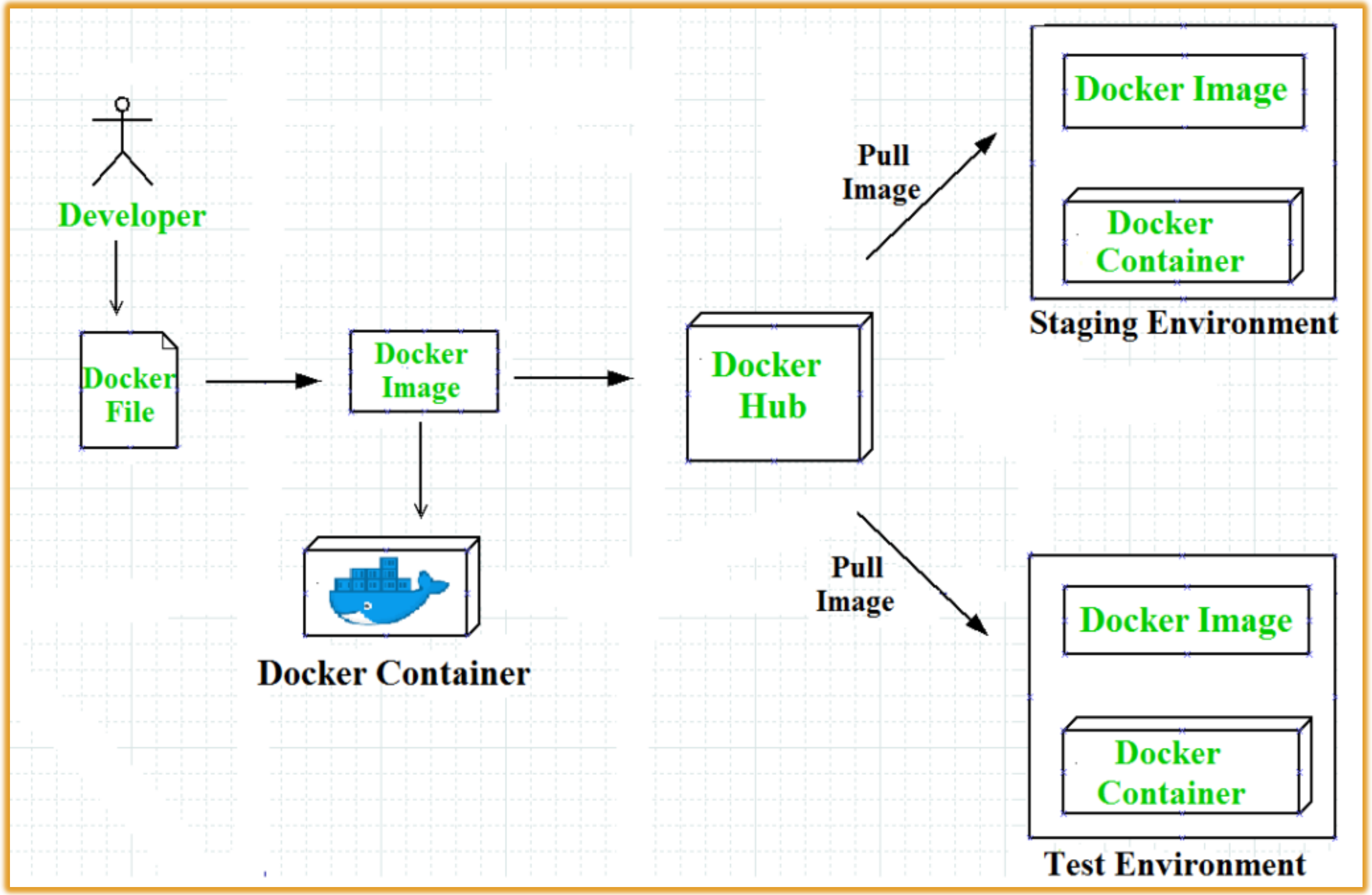
https://docs.docker.com/engine/docker-overview/#what-can-i-use-docker-for

Responsive deployment and scaling

- *Docker Containers* are portable and can run on a developer's local laptop, on physical and virtual machines, in a data center, or on cloud providers.
- You can scale up or tear down applications (and services) as business needs dictate.

Running more workloads on the same hardware

- Docker is lightweight and fast.
- Docker is NOT itself a Virtual Machine.
    - a *virtual machine* (VM) runs a full-blown "guest" operating system with *virtual* access to host resources through a hypervisor. VMs incur a lot of overhead.



The Docker Platform

https://www.docker.com/resources/what-container

Docker provides a platform to manage the entire lifecycle of your *containers*:

1. You develop an application and its supporting components using *containers*.
2. The *container* becomes the unit for distributing and testing your application.
3. Deploy your application into your production environment as a *container*. This works the same whether your production environment is a local data center, a cloud provider, or a hybrid of the two.

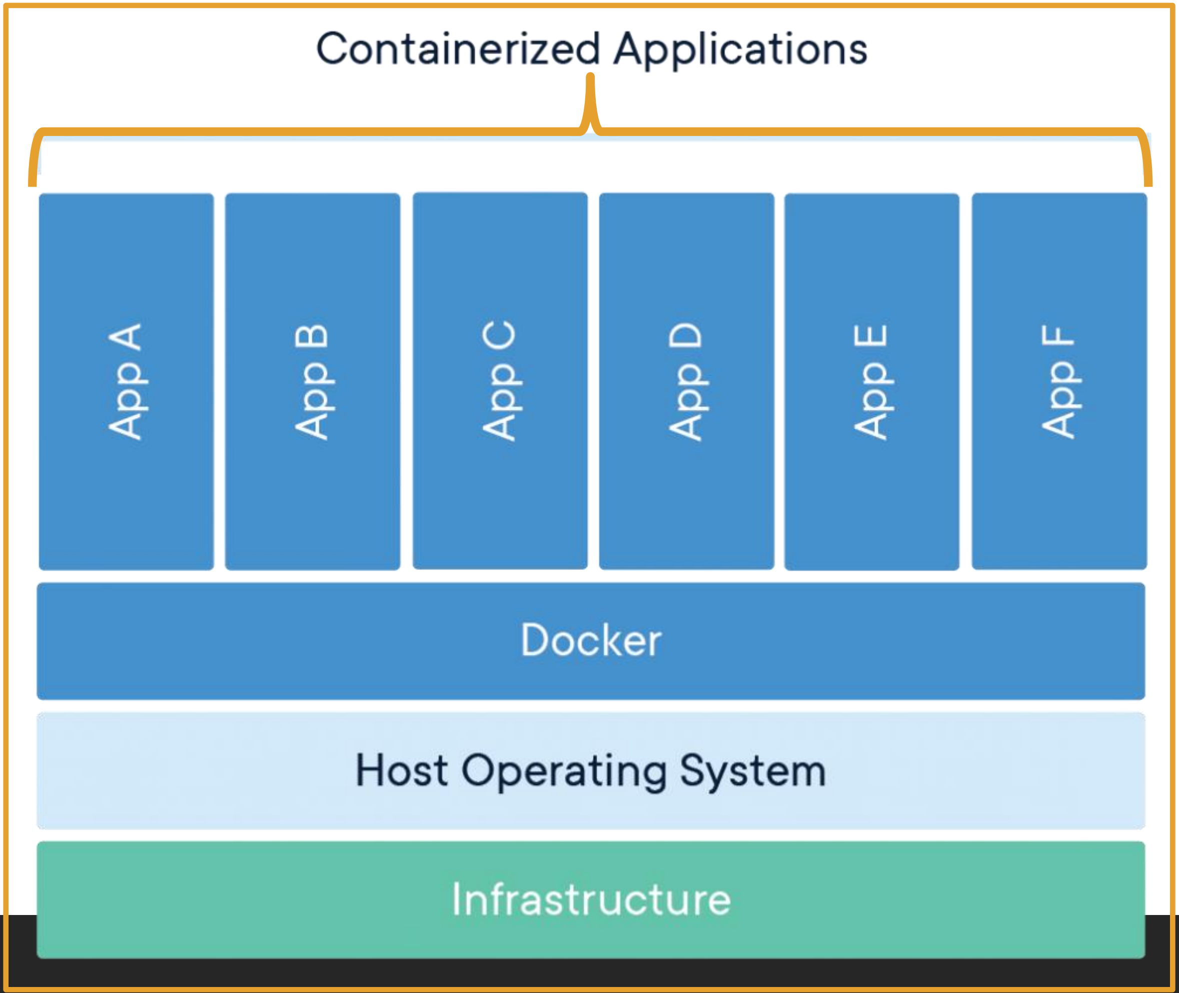
Docker Container

https://www.docker.com/resources/what-container
https://docs.docker.com/get-started/

A ***Docker Image*** is a standalone executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries, and settings.

Docker Images become ***Docker Containers*** at runtime when run on the ***Docker Engine***. ***Containers*** run identically, regardless of the infrastructure (Linux or PC).

A ***Docker Container*** isolates software from its environment. Each container interacts with its own private filesystem.

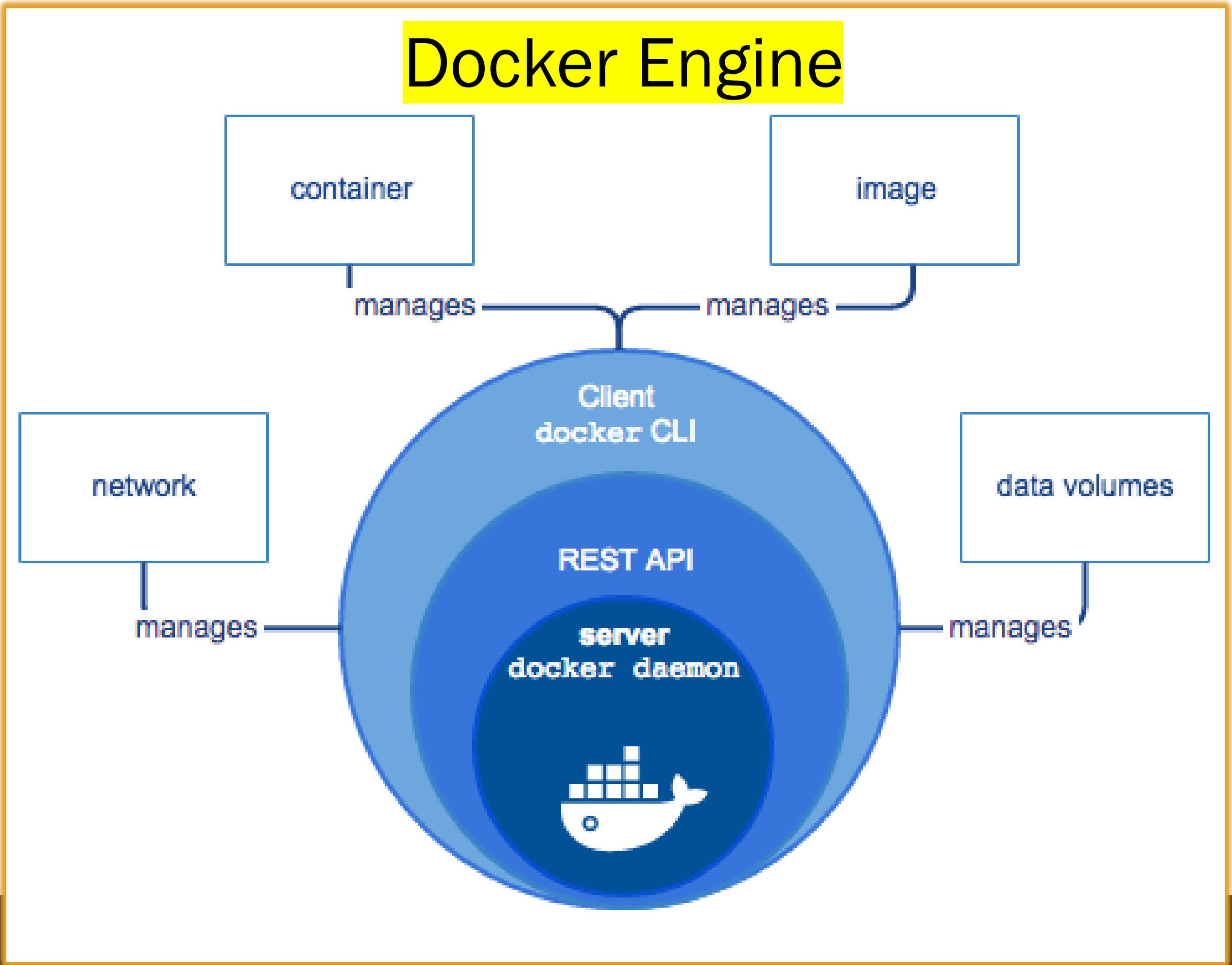


Docker Engine

https://docs.docker.com/engine/docker-overview/#docker-engine

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

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

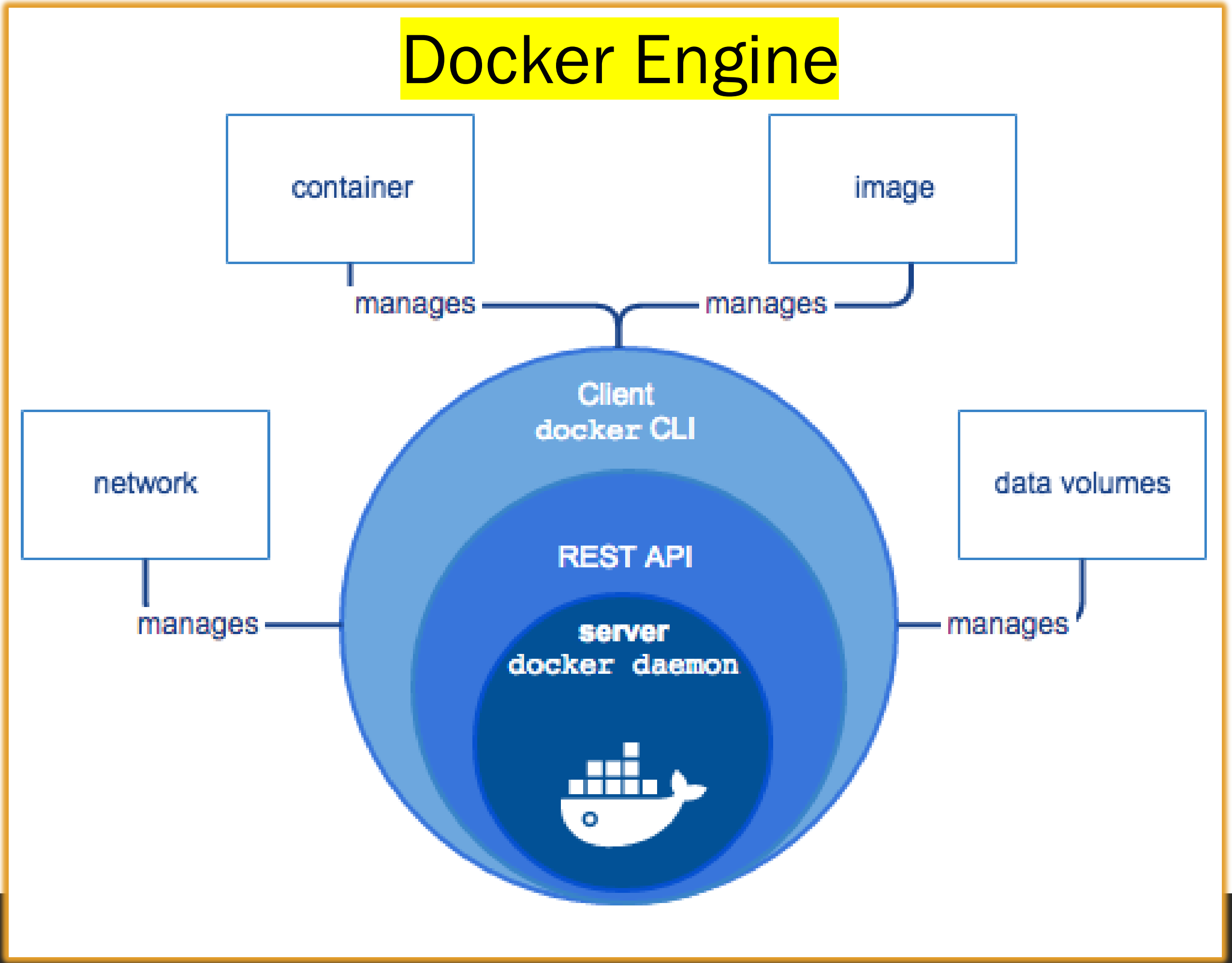


Docker Engine

https://docs.docker.com/engine/docker-overview/#docker-engine

The *CLI* uses the *Docker REST API* to control or interact with the *Docker daemon* through scripting or direct *CLI* commands. Many other Docker applications use the underlying *API* and *CLI*.

The ***daemon*** creates and manages Docker objects, such as ***images***, ***containers***, networks, and volumes.



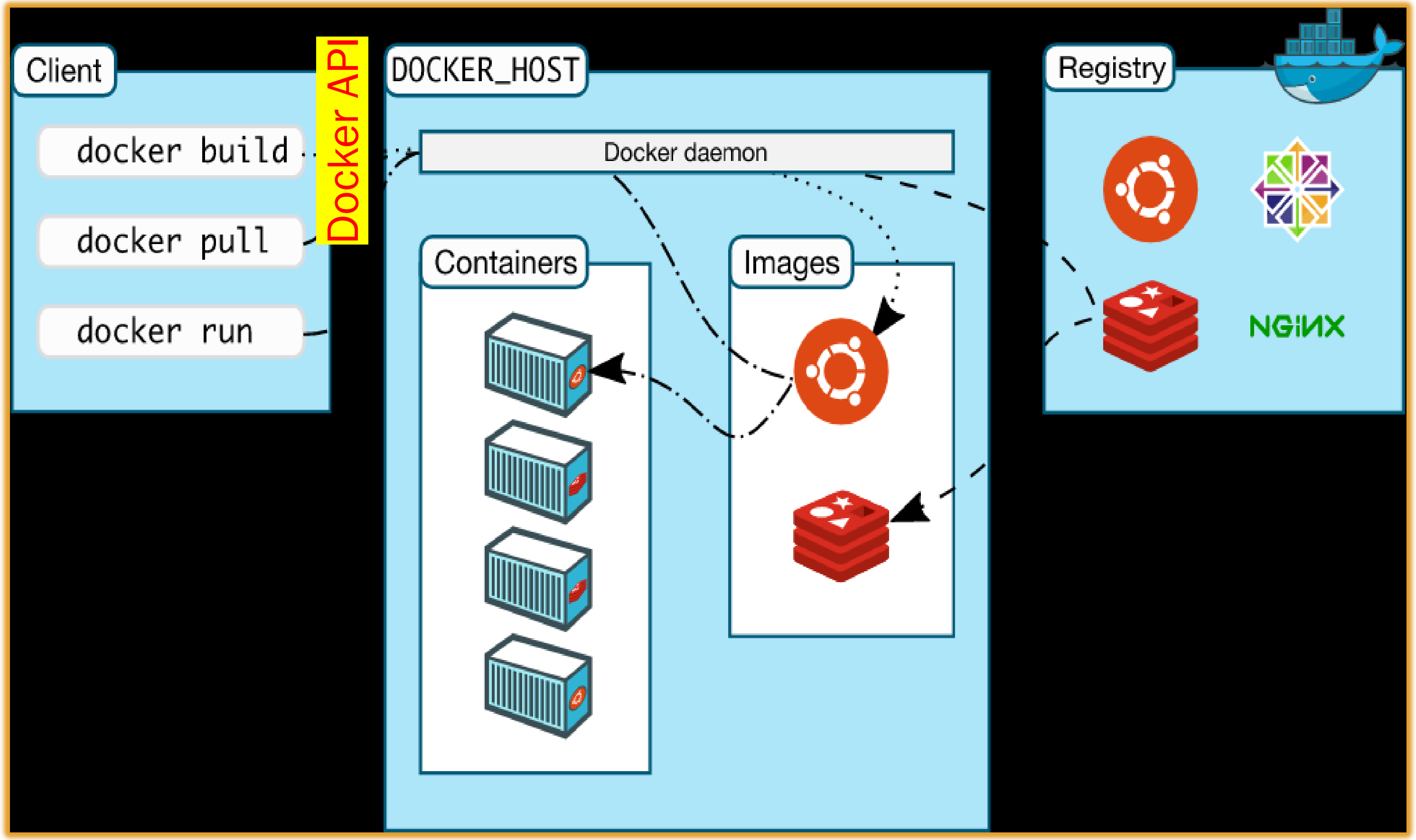
Docker Architecture

https://docs.docker.com/engine/docker-overview/#docker-architecture

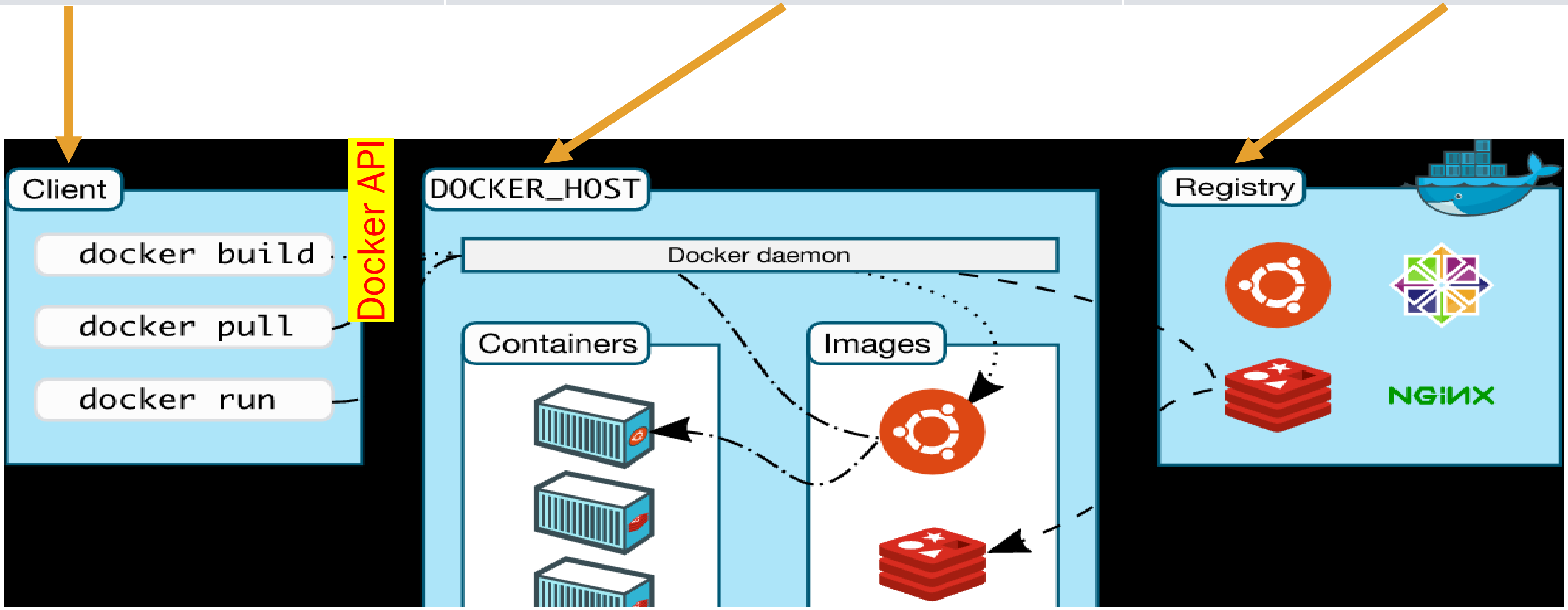
Docker uses a ***client-server architecture***. The ***Docker client*** talks to the ***Docker daemon (server)***, which does the heavy lifting of building, running, and distributing ***Docker containers***.

The ***Docker client*** and ***daemon*** can run on the same system, or you can connect a Docker client to a remote Docker daemon.

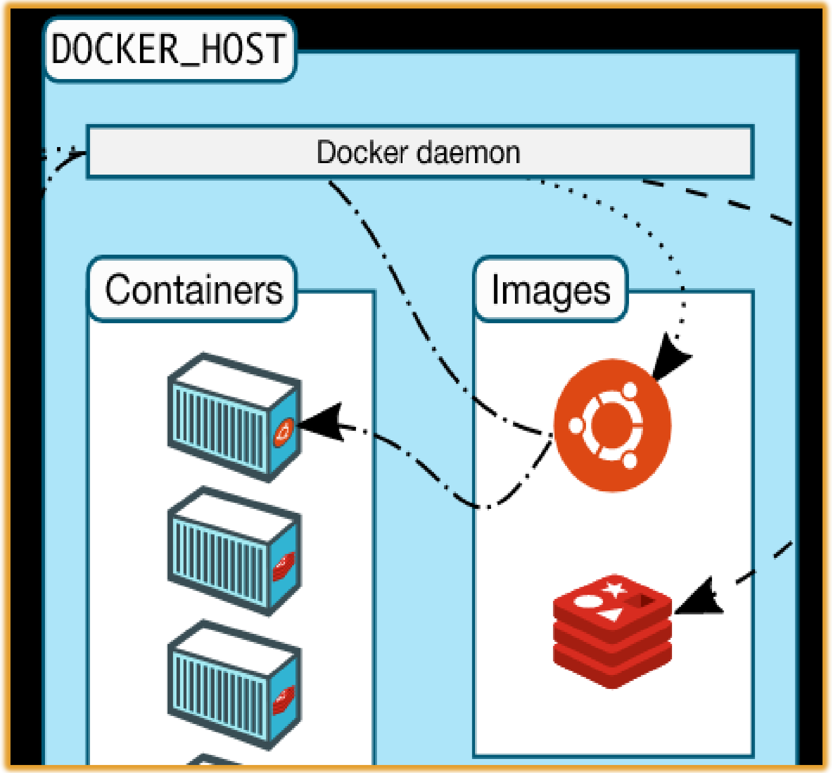
The ***Docker client*** and ***daemon*** communicate using a REST API.



Docker Client	Docker daemon	Docker registries
The *Docker client (docker)* is the primary way that most Docker users interact with Docker. With *docker run*, the client sends these commands to *dockerd*, which carries them out. The docker command uses the *Docker API*.	The *Docker daemon (dockerd)* listens for *Docker API* requests and manages *Docker objects* such as *images*, *containers*, *networks*, and *volumes*.	A **Docker registry** stores **Docker images**. **Docker Hub** is a public registry. With the **docker pull** or **docker run** commands, images are pulled from the configured registry. When you use the **docker push** command, the **image** is pushed to the configured registry.



A *container* is a runnable instance of an ***image***. You can create, start, stop, move, or delete a ***container*** 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.
A ***container*** is defined by its ***image*** as well as any configuration options you provide to it when you create or start it.



An *image* is a read-only template with instructions for creating a ***Docker container***. An ***image*** can be based on another ***image***, with some additional customization.

An ***image*** could be based on the ubuntu ***image***, but install the Apache web server and your application as well as the configuration details needed to make your application run. Images can be published in a registry.

To build an ***image***, create a ***Dockerfile*** defining the steps to create an ***image*** and run it.
When you change a ***Dockerfile*** and rebuild the ***image***, only those layers which have changed are rebuilt

List of Basic Docker commands

Command	Purpose
docker start [containername]	Start a container.
docker images	List images installed
docker container command	Manage containers
docker stop [containername]	Stop a running container
docker image ls	list the images downloaded to your machine.
docker ps --a	Lists all containers
docker run [containername]	
docker build –t myimage –f dockerfile .	Build an image called myimage from a Dockerfile
docker stop [containername]	Stop a running container
docker rm [containername]	Delete a container
docker push [imagename]	Push an image to your repo in the Docker Registry
docker create myimage	Create an unstarted container from an image
docker ps	Show running containers
docker attach [containername]	Connect to a running container

Docker – Setup and Test Container

https://docs.docker.com/get-started/

1. Download Docker Desktop.

2. Run ***docker –version*** in the Command Line.

3. Run ***docker run Hello-World*** to test that docker is running correctly.

4. Run ***docker image ls*** to list the downloaded hello-world image on your machine.

5. Run ***docker ps –all*** to see the container created from the ***Hello-World image***.

6. Do the Docker tutorial here.

7. Then complete the Getting Started Walk-through for Developers tutorial.

Docker in action

The following command runs an ubuntu container, attaches interactively to your local command-line session, and runs the /bin/bash script.

```
$ docker run -i -t ubuntu /bin/bash
```

The following happens (assuming default registry configuration):

1. If you do not have the ***ubuntu*** image locally, Docker pulls it from your configured registry, as though you had run ***docker pull ubuntu*** manually.
2. Docker creates a new container, as though you had run a ***docker container create*** command manually.
3. Docker allocates a read-write filesystem to the container, as its final layer.
   - This allows a running container to create or modify files and directories in its local filesystem.
4. Docker creates a network interface to connect the container to the default network,
   - because you did not specify any networking options.
   - This includes assigning an IP address to the container.
   - By default, containers can connect to external networks using the host machine's network connection.
5. Docker starts the container and executes */bin/bash*.
   - Because the container is running interactively and attached to your terminal (due to the *-i* and *-t* flags), you can provide input using your keyboard while the output is logged to your terminal.
6. When you type ***exit*** to terminate the */bin/bash* command, the container stops but is not removed.
   - You can start it again or remove it.

Next Steps

Now you can move on to creating your own Dockerfiles, Images, and Containers!