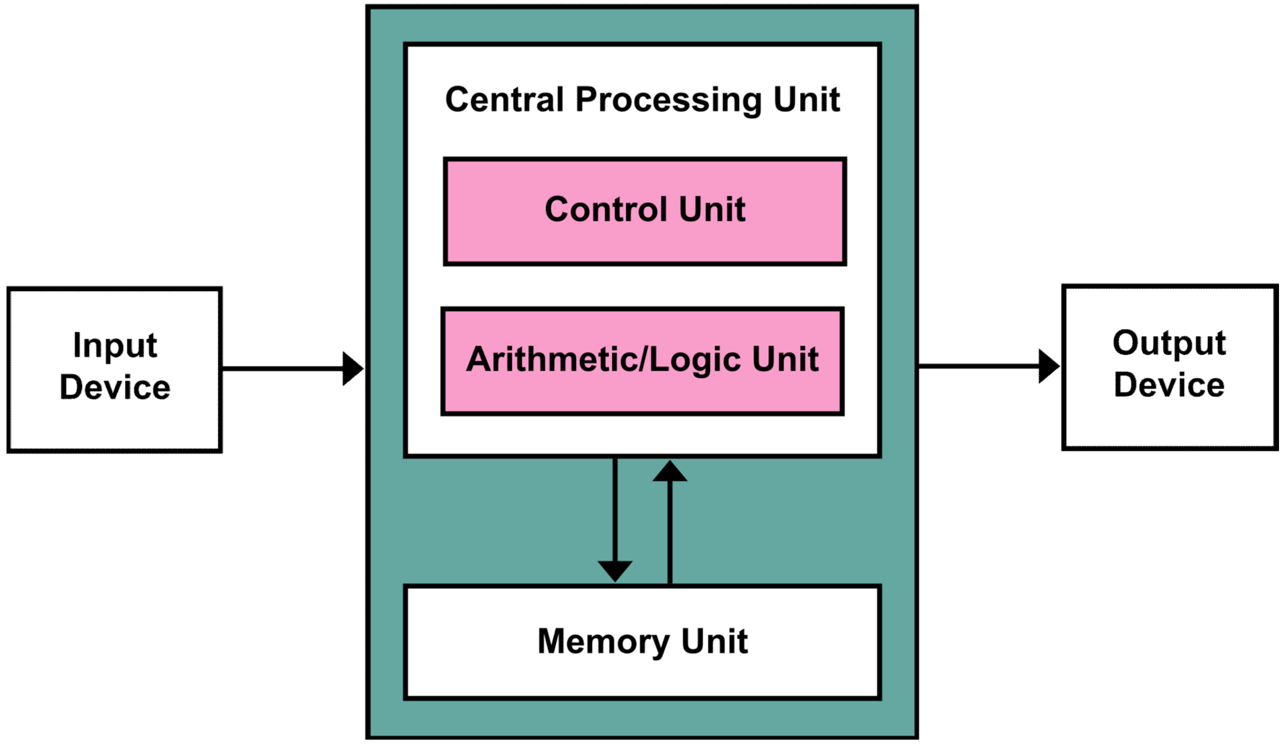
docker

Day 1

Computer Architecture

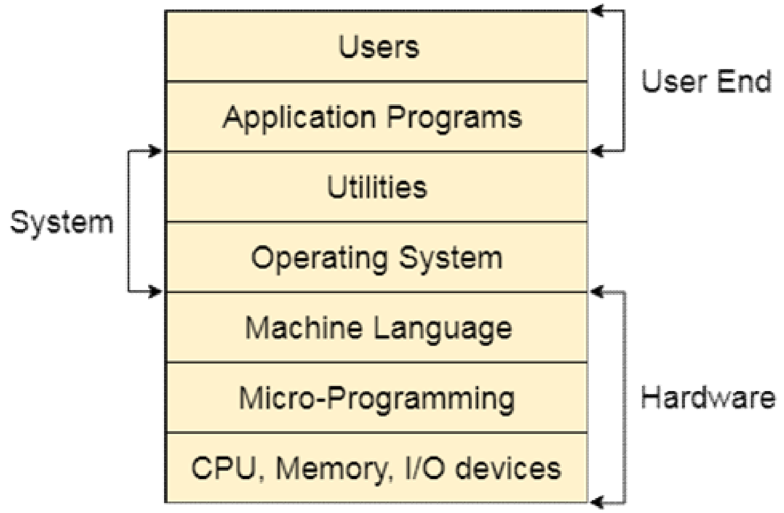
- In computer engineering, computer architecture is a set of rules and methods that describe the functionality, organization, and implementation of computer systems.
- The architecture of a system refers to its structure in terms of separately specified components of that system and their interrelationships.
- A program is a set of instructions that can help utilize the electronic hardware infrastructure for computation tasks

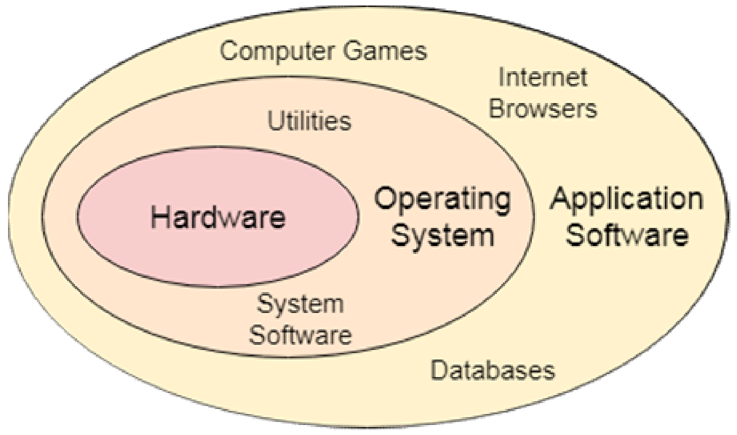
Computer Architecture



Overview of Operating System Concepts

- Operating System can be defined as an interface between user and the hardware. It provides an environment to the user so that, the user can perform its task in convenient and efficient way.





What does an OS do?

- Process Management
- Process Synchronization
- Memory Management
- CPU Scheduling
- File Management
- Security

Virtualization

- Virtualization uses software to create an abstraction layer over computer hardware that allows the hardware elements of a single computer—processors, memory, storage and more—to be divided into multiple virtual computers, commonly called virtual machines (VMs).
- **Each VM runs its own operating system (OS) and behaves like an independent computer, even though it is running on just a portion of the actual underlying computer hardware.**
- Benefits include: resource efficiency, easier management, minimal downtime and faster provisioning

Virtual Machines

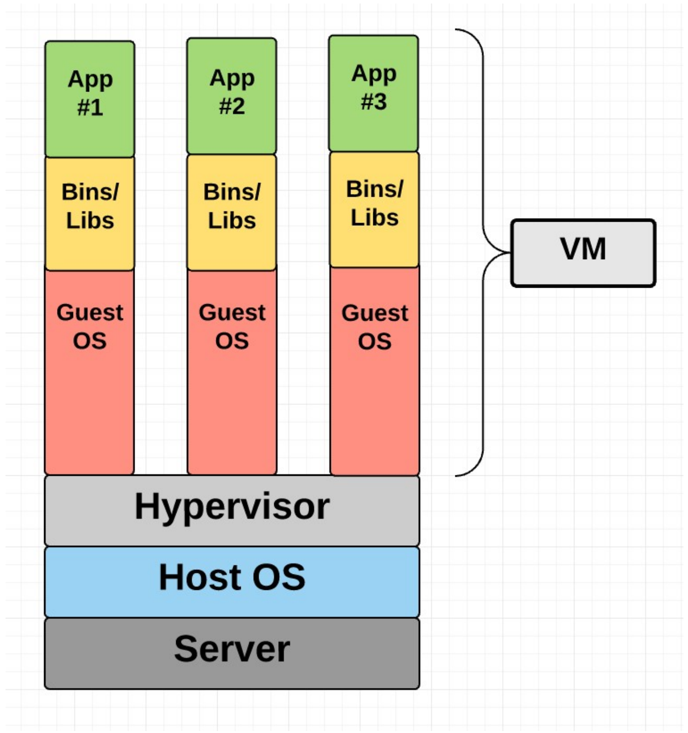
- A virtual machine is a virtual representation, or emulation, of a physical computer. They are often referred to as a guest while the physical machine they run on is referred to as the host.
- Virtualization makes it possible to create multiple virtual machines, each with their own operating system (OS) and applications, on a single physical machine.
- A VM cannot interact directly with a physical computer. Instead, it needs a lightweight software layer called a **hypervisor** to coordinate between it and the underlying physical hardware.
- The hypervisor allocates physical computing resources—such as processors, memory, and storage—to each VM. It keeps each VM separate from others so they don’t interfere with each other.

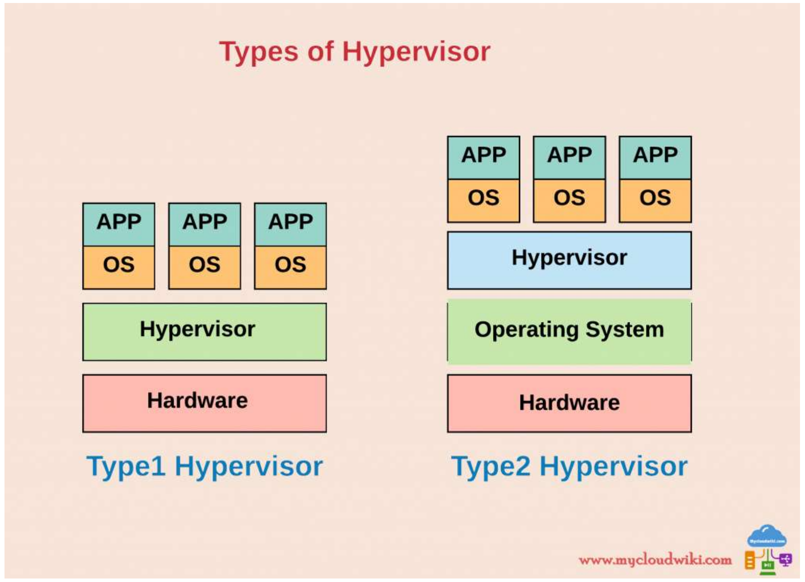
Hypervisors

- A hypervisor is the software layer that coordinates VMs.
- It serves as an interface between the VM and the underlying physical hardware, ensuring that each has access to the physical resources it needs to execute.
- It also ensures that the VMs don’t interfere with each other by impinging on each other’s memory space or compute cycles.
- There are two types of hypervisors:
    - Type 1 or “bare-metal” hypervisors interact with the underlying physical resources, replacing the traditional operating system altogether. They most commonly appear in virtual server scenarios.
    - Type 2 hypervisors run as an application on an existing OS. Most commonly used on endpoint devices to run alternative operating systems, they carry a performance overhead because they must use the host OS to access and coordinate the underlying hardware resources.

Hypervisors

- When a hypervisor is used on a physical computer or server, (also known as bare metal server), it allows the physical computer to separate its operating system and applications from its hardware. Then, it can divide itself into several independent “virtual machines.”
- Each of these new virtual machines can then run their own operating systems and applications independently while still sharing the original resources from the bare metal server, which the hypervisor manages. Those resources include memory, RAM, storage, etc.
- The hypervisor acts like a traffic cop of sorts, directing and allocating the bare metal’s resources to each of the various new virtual machines, ensuring they don’t disrupt each other.


App #1
App #2
App #3
Bins/ Libs
Bins/ Libs
Bins/ Libs
Guest OS
Guest OS
Guest OS
VM
Hypervisor
Host OS
Server


Types of Hypervisor
APP
OS
APP
OS
APP
OS
Hypervisor
Hardware
Type1 Hypervisor
APP
OS
APP
OS
APP
OS
Hypervisor
Operating System
Hardware
Type2 Hypervisor
www.mycloudwiki.com

Types of Virtualization

- Desktop virtualization
- Network virtualization
- Storage virtualization
- Data virtualization
- Application virtualization
- Data centre virtualization
- CPU virtualization
- GPU virtualization
- Linux virtualization
- Cloud virtualization

Container

- A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.
- Available for both Linux and Windows-based applications, containerized software will always run the same, regardless of the infrastructure.
- Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.
- Containers can live in a repository: public/private, official
  - Example: Docker Hub

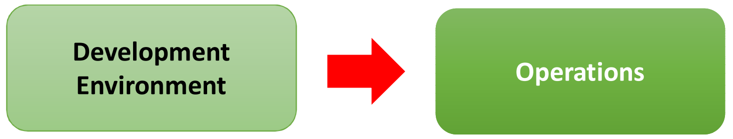
Virtualization vs. Containerization

- Server virtualization **reproduces an entire computer in hardware,** which then runs an entire OS. The OS runs one application. That’s more efficient than no virtualization at all, but it still duplicates unnecessary code and services for each application you want to run.
- Containers take an alternative approach. They share an underlying OS kernel, only running the application and the things it depends on, like software libraries and environment variables. This makes containers smaller and faster to deploy.

Docker

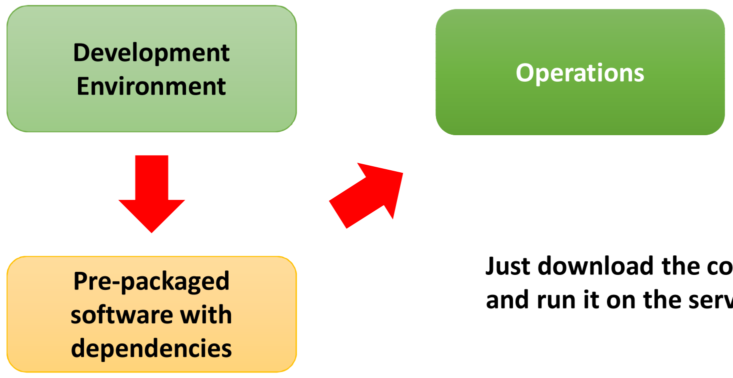
- Docker container technology was launched in 2013 as an open source Docker Engine.
- The problem it solves:

Regular Software Development Process

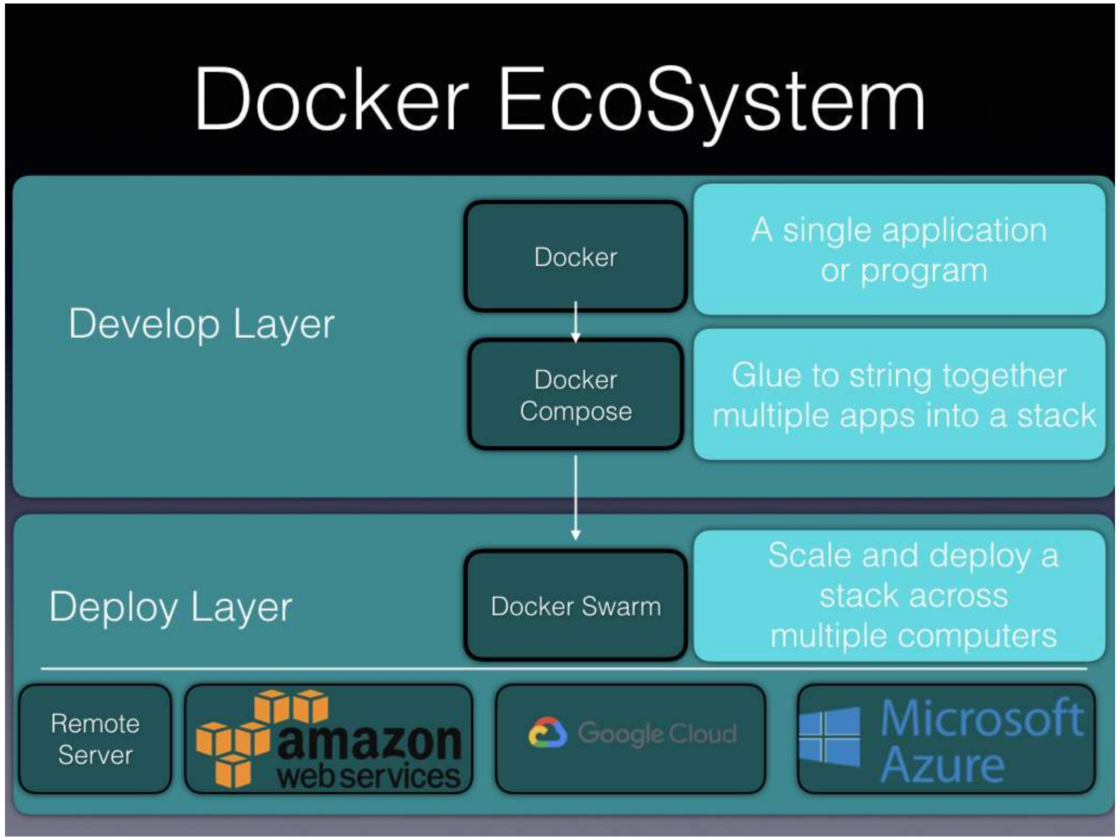


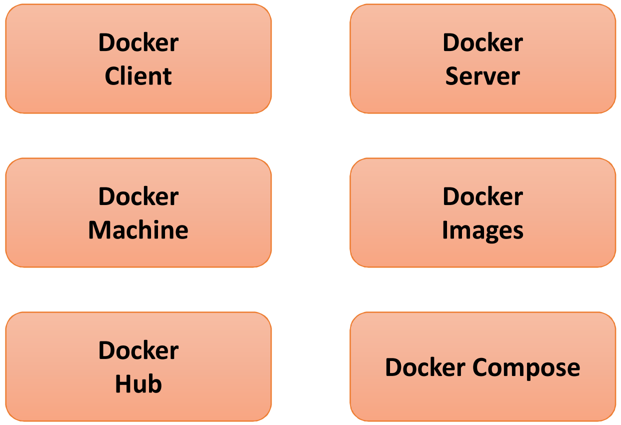
1. Read manuals to setup, misinterpretation
2. Use scripts written by development engineers
3. Many to and fro communication between teams until the software is deployed successfully

Software Development Process with Containers

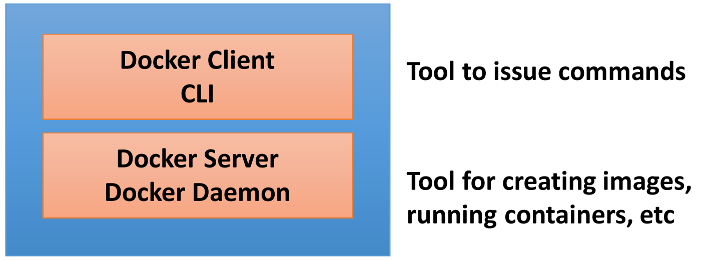


Just download the container and run it on the server





Docker for Windows/Mac


Docker Client CLI
Docker Server Docker Daemon
Tool to issue commands
Tool for creating images, running containers, etc

Docker Installation: Mac OS

- Download Docker Desktop for Mac OS

    https://docs.docker.com/desktop/install/mac-install/

Docker Installation: Windows

- Docker Desktop can be installed if the computer supports Windows Sub-system for Linux (WSL2)
- Docker Commands should be run from within WSL2 and not from the Windows file system
- Download Docker Desktop for Windows

https://docs.docker.com/desktop/install/windows-install/

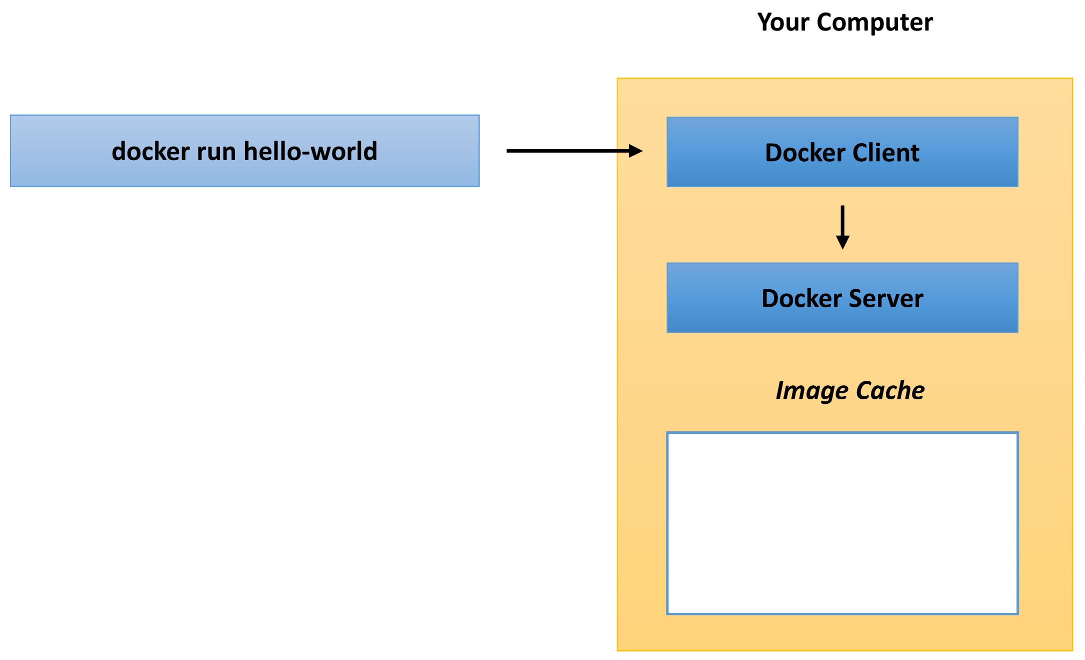
Docker Installation: Linux

- Download Docker Desktop on Linux
  - **https://docs.docker.com/desktop/install/linux-install/**

Using the Docker Client

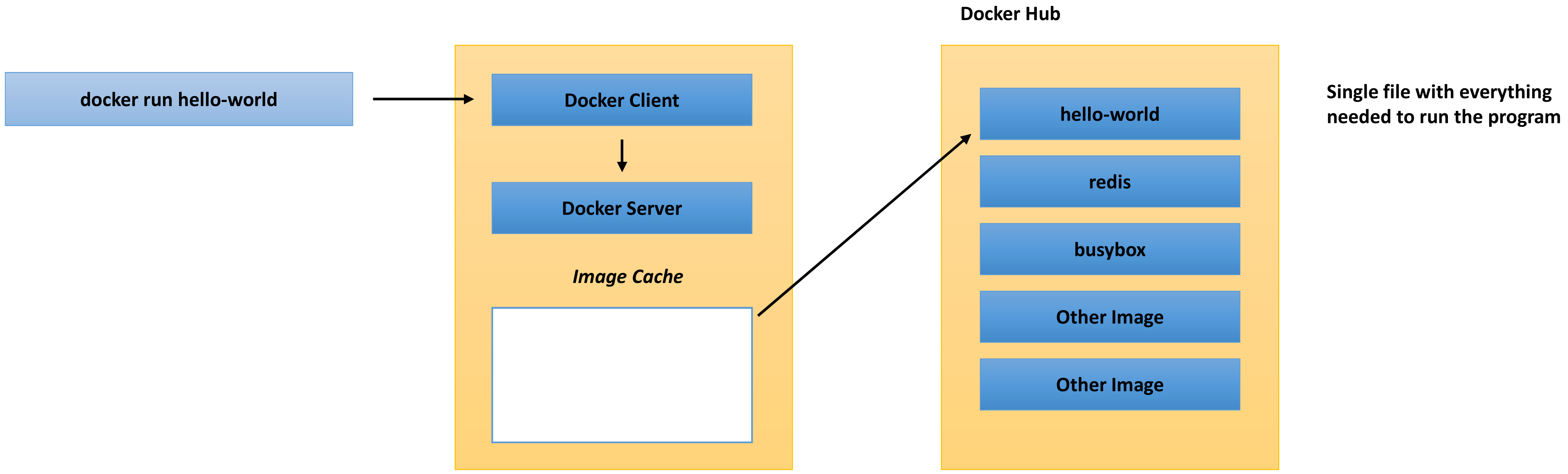
- Let’s run a command:

  docker run hello-world



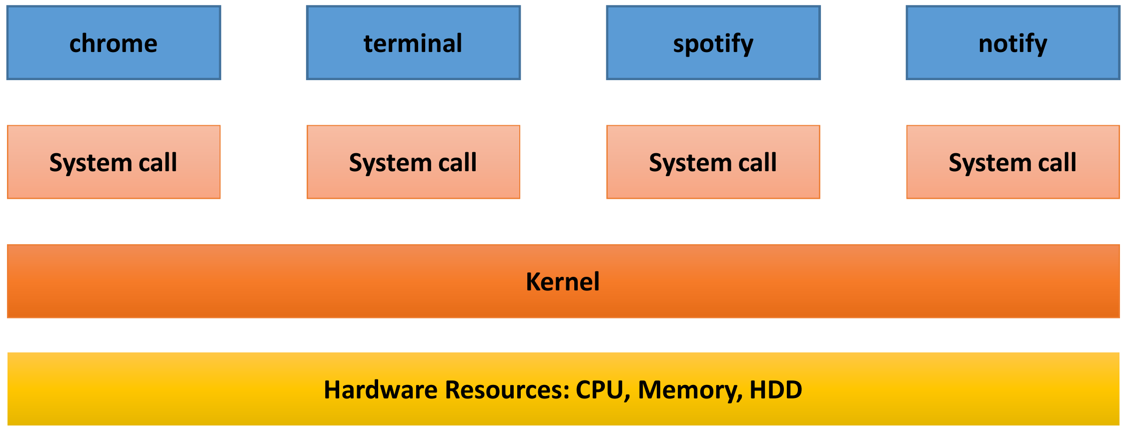
Docker Hub

- A free service which contains publically available images
- When the docker server doesn't find an image in the image cache it reaches out to dockerhub



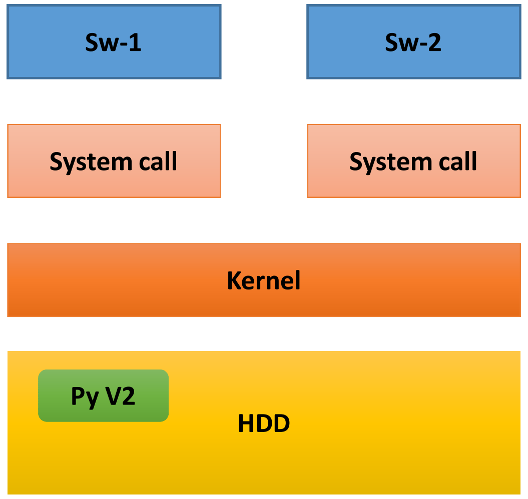
What really is a container?

- In a regular OS, the processes running on the system invokes a system call. The system call requests the Kernel to interact with a piece of hardware. Kernel governs the process and access to the hardware



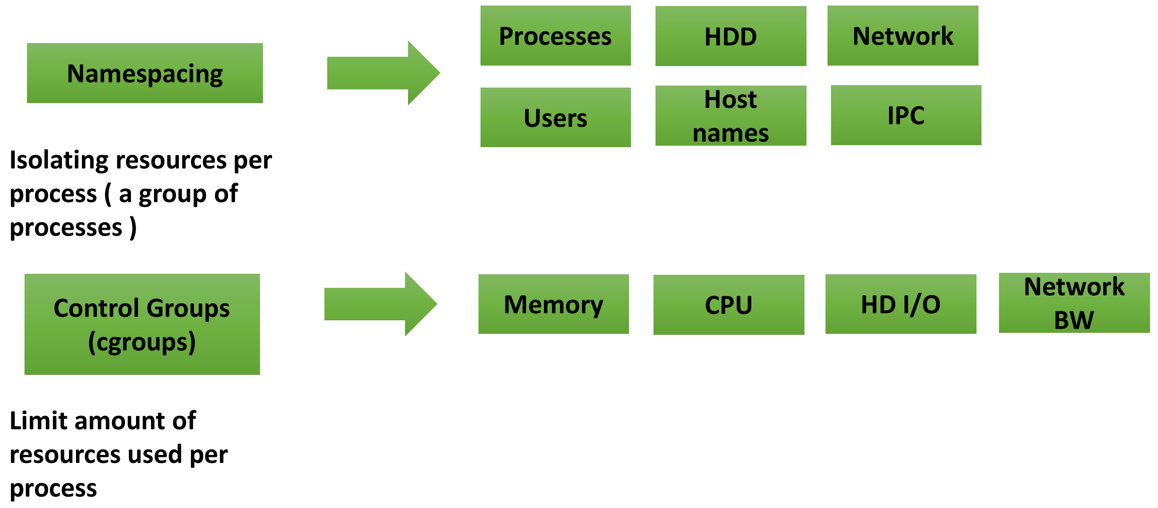
What really is a container?

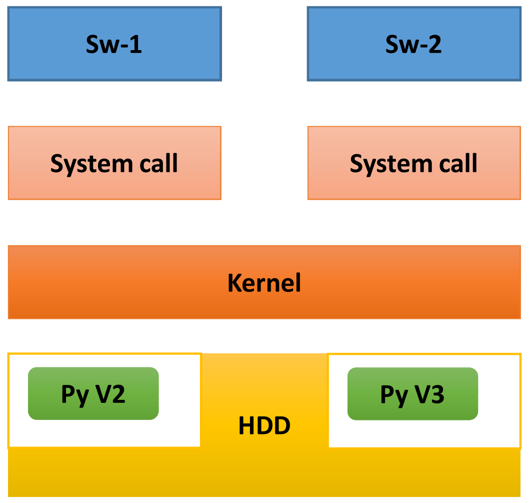
- Consider a hypothetical situation: Sw-1 depends on python v2 and Sw-2 depends on python v3
- In the given situation Sw-1 works properly but Sw-2 will not
- How do we solve this problem?



Namespacing

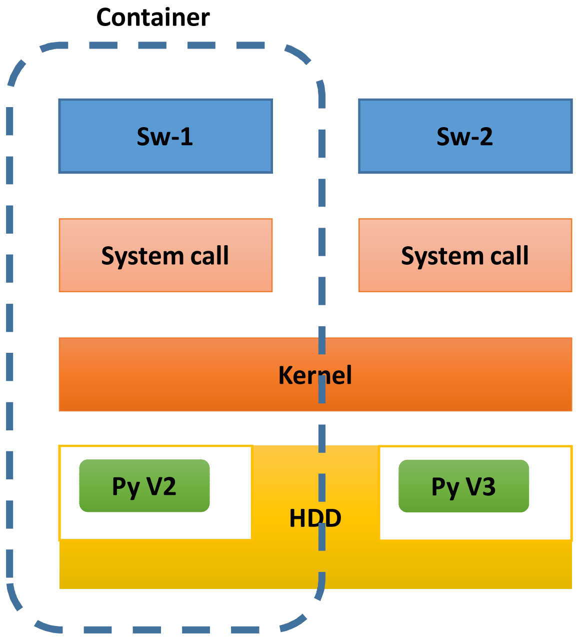
- Namespacing is an OS feature, we can segment the hardware resources. Using this feature, we can create a segment specifically to house each of the python versions.
- Whenever a process requests for a specific hardware, kernel should redirect it to the specific segment. Kernel should know which process is making the call.

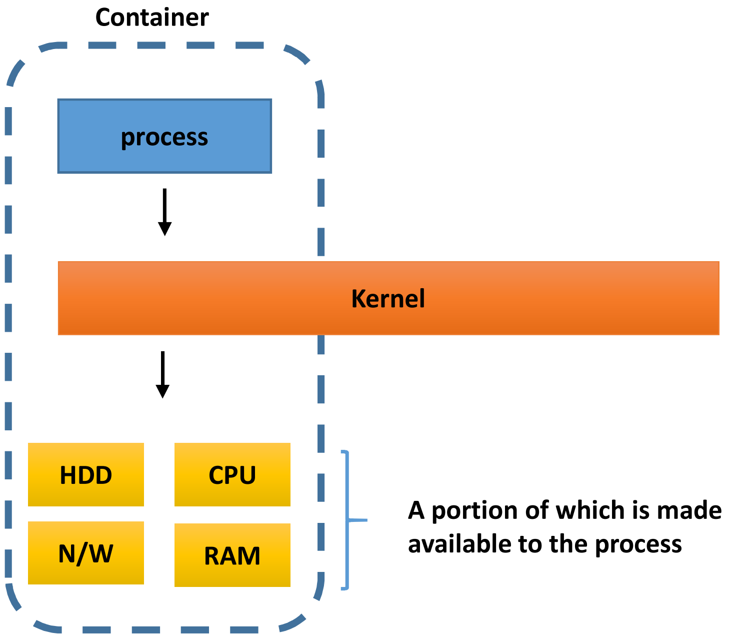

Namespacing
Processes
HDD
Network
Users
Host names
IPC
Isolating resources per process (a group of processes)
Control Groups (cgroups)
Memory
CPU
HD I/O
Network BW
Limit amount of resources used per process



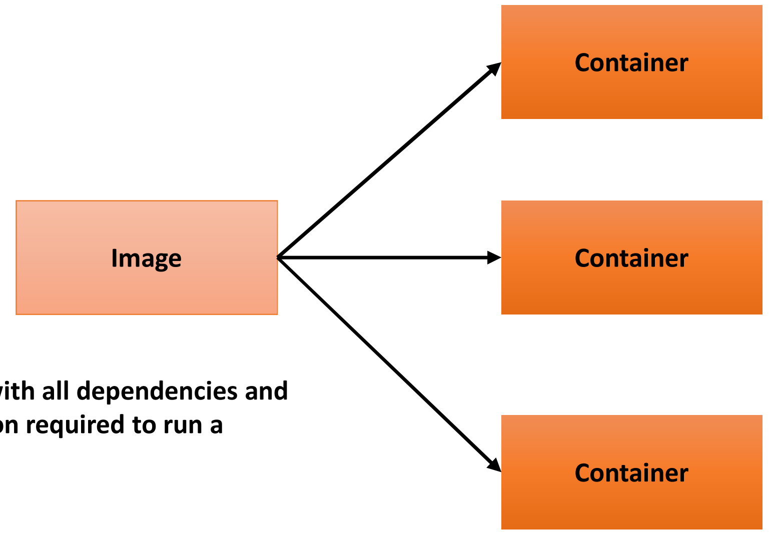
What really is a Container?

- A container is a process or a set of processes which has grouping of resources specifically assigned to it
- Please not that it is not a physical construct





Image



Own network, HDD, other resources

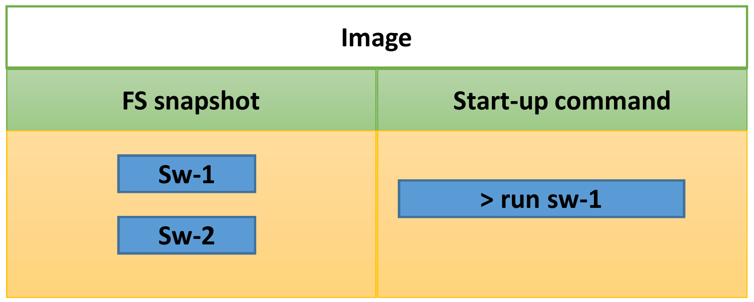
Instance of an image

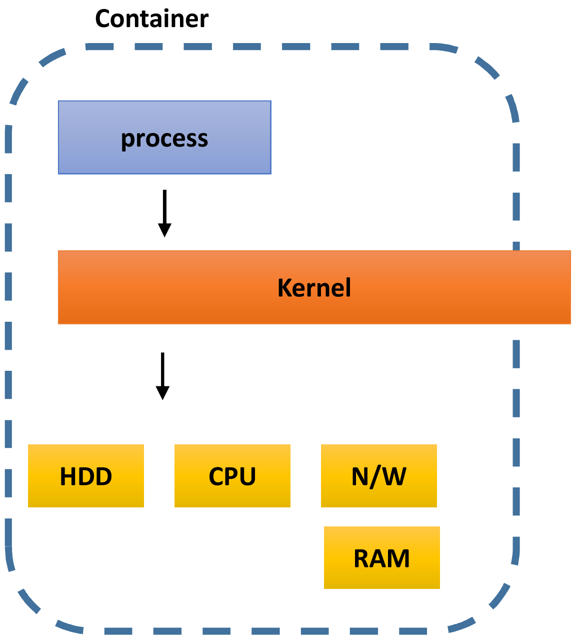
Single file with all dependencies and configuration required to run a program

Images and Containers

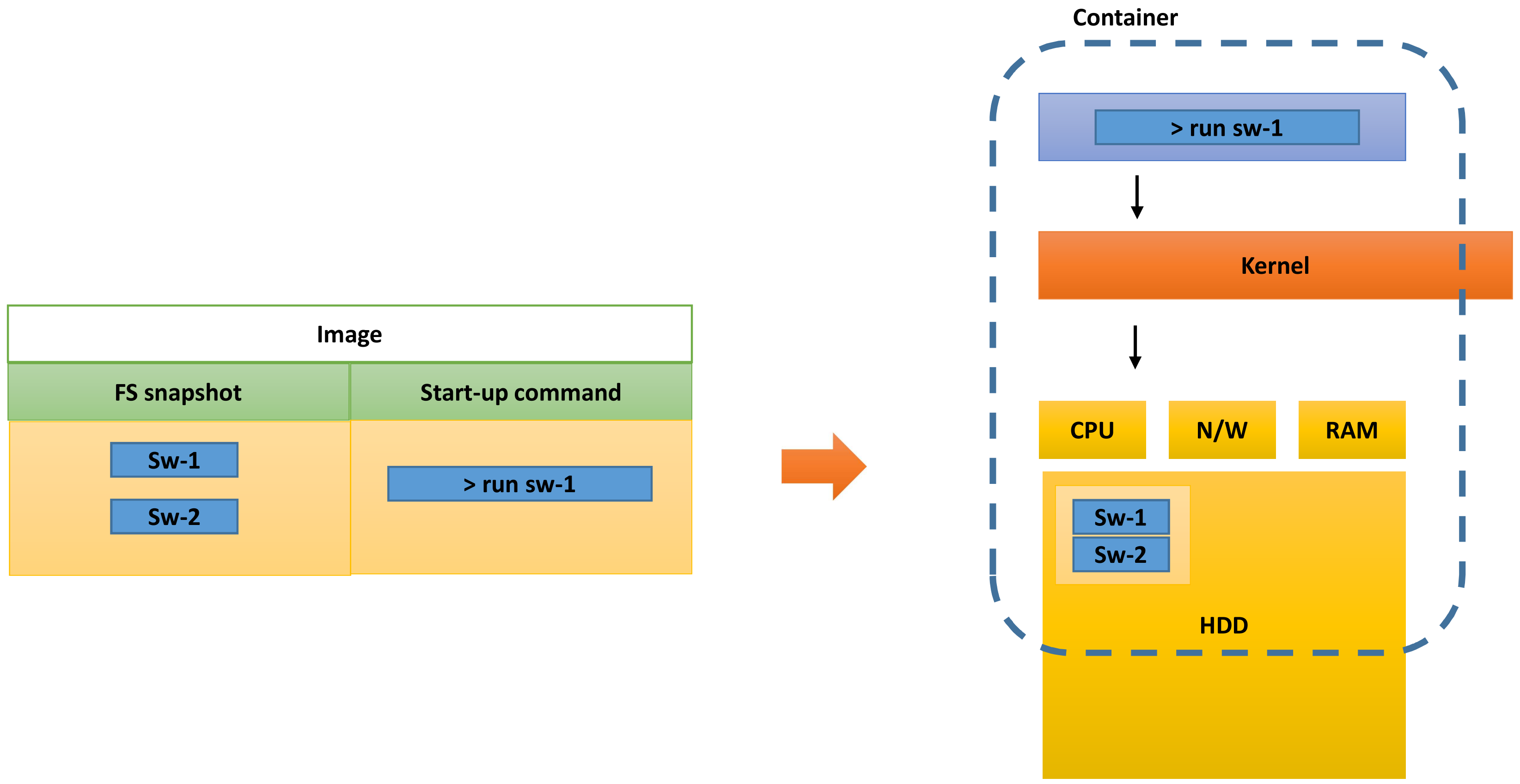
- How does the image which is a single file that would eventually create a container which is an instance of an image running a program?

Structure of an image:
Consists of a file system snapshot



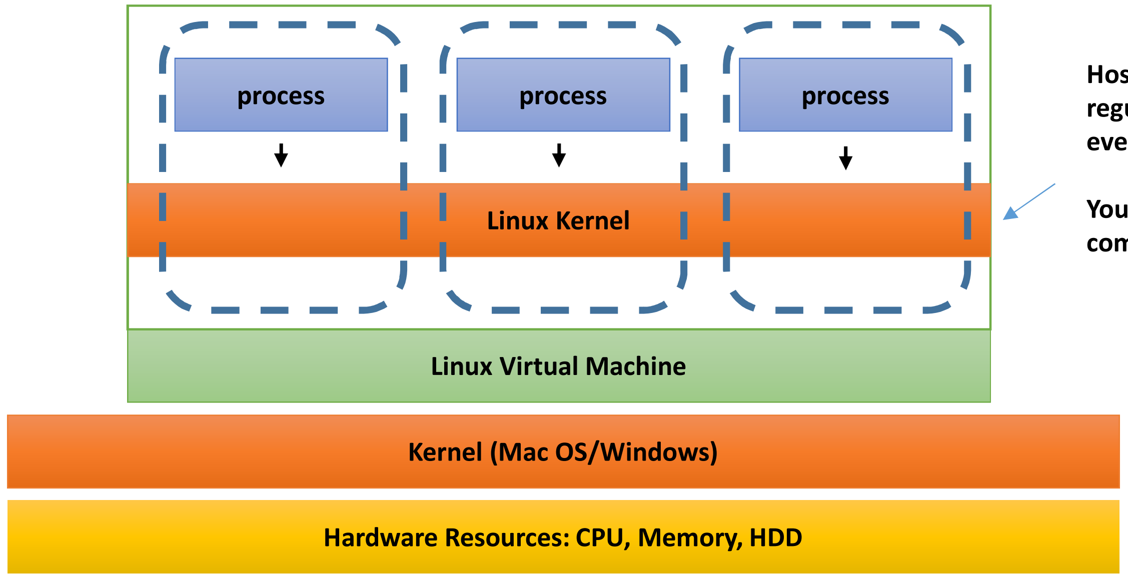


Images and Containers



How does Docker run in your computer?

- Namespacing and Control Groups are a concept of Linux – not Windows or Mac OS
- When you installed the Docker, a virtual machine was installed

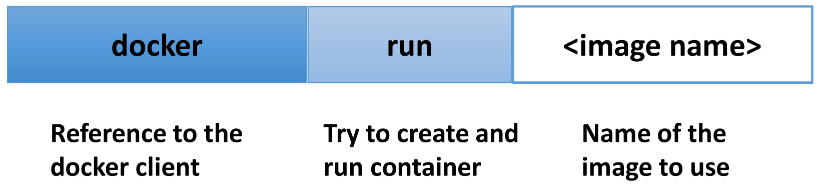


Hosts all processes, isolates and regulates access to hardware for every running process

You can use the docker version command and check the Linux VM

The docker run Command

- What happens when you run the docker run command?


docker
run
<image name>
Reference to the docker client
Try to create and run container
Name of the image to use

- Somewhere in the HDD we will have the FS snapshot, otherwise it needs to be downloaded to the image cache
- The FS snapshot will be copied to the specific location in the HDD, the running process is activated using the specified run command in the image. It is the default command to execute in the container.

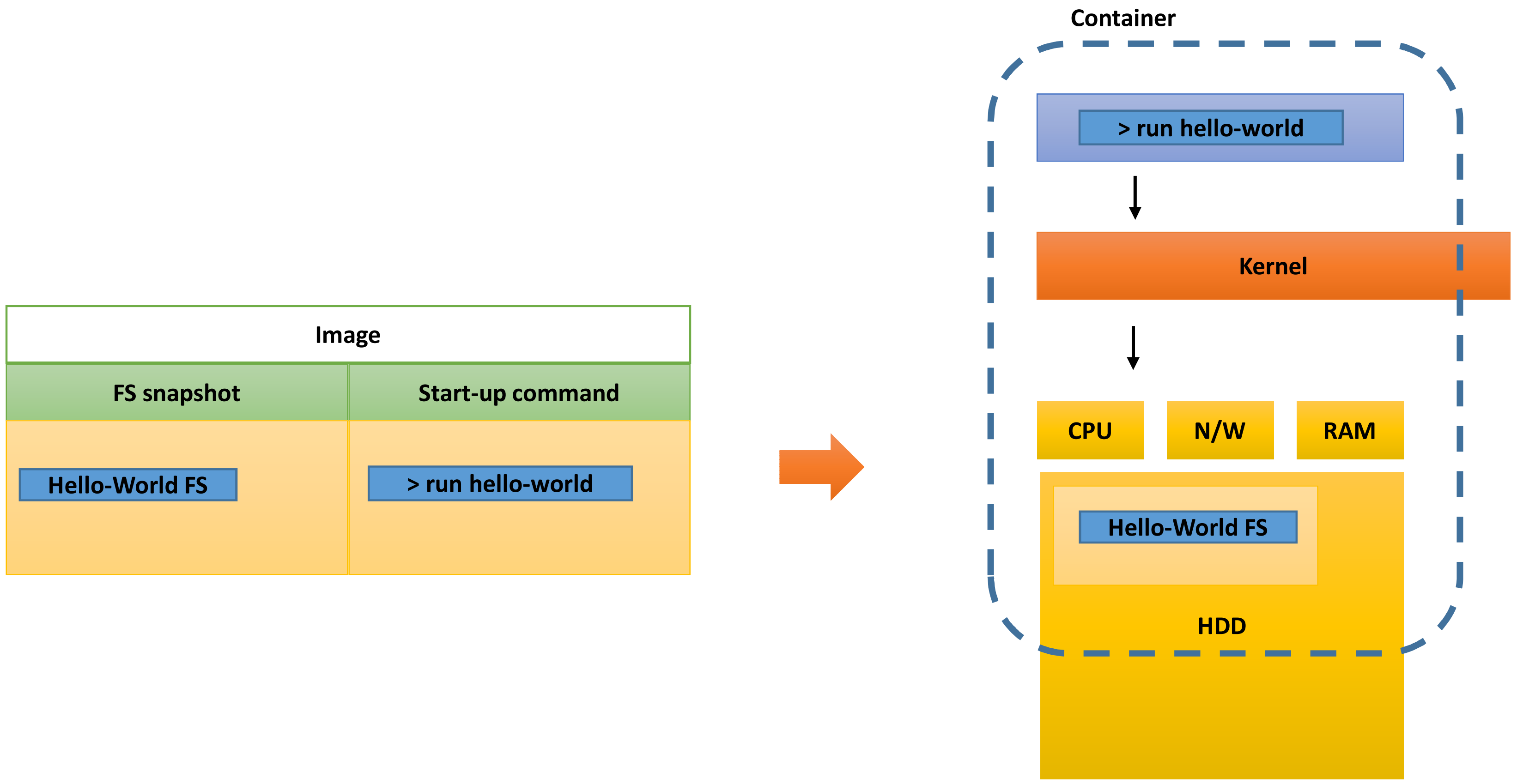
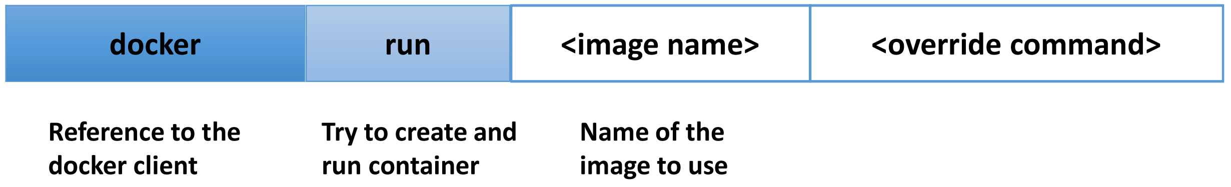


Image
FS snapshot
Start-up command
Hello-World FS
> run hello-world
Container
> run hello-world
Kernel
CPU
N/W
RAM
Hello-World FS
HDD

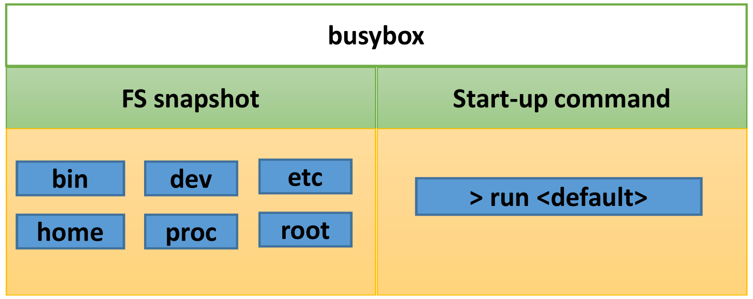
Overriding Options

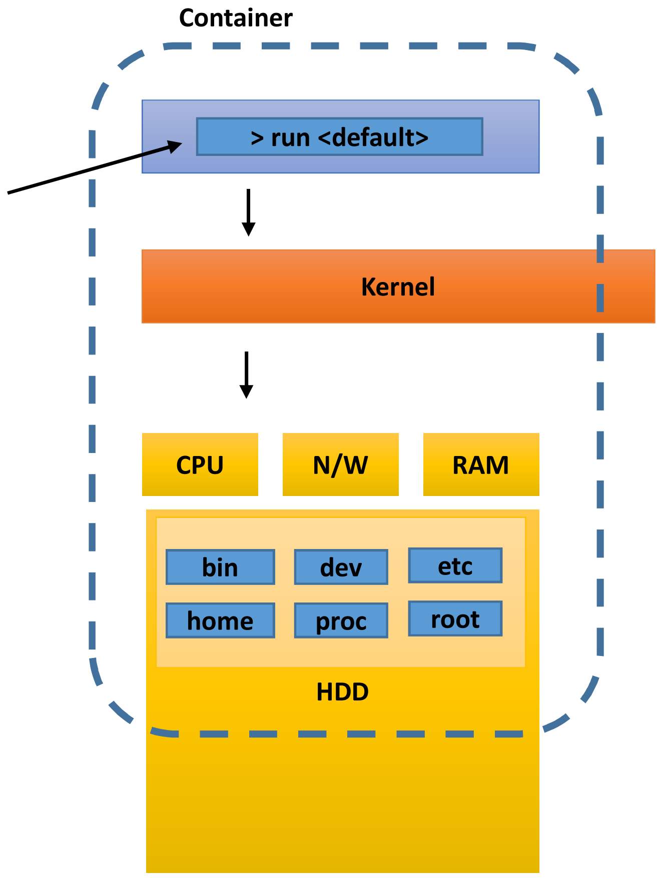
The echo or ls command is a program in the file system loaded, hence it will execute

- To override default command:



Example: **docker run busybox echo hi there**
docker run busybox ls





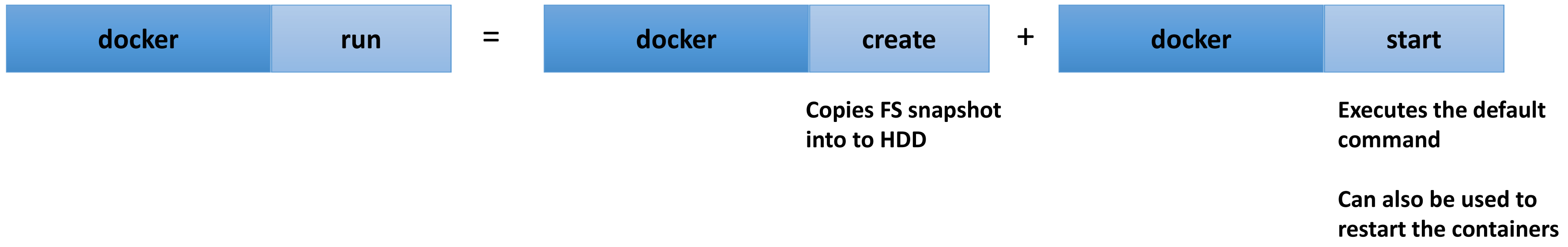
List Running Containers

- To list all the running containers:

```
docker ps
```

Container Lifecycle

- Do containers shutdown automatically?
- Why do they shutdown? When? What happens?


docker
run
=
docker
create
+
docker
start
Copies FS snapshot into to HDD
Executes the default command
Can also be used to restart the containers

Stopping and Removing Containers

- Sometimes we cannot leave the containers in the stopped state
- They eat up space. Therefore it is essential to prune them. After pruning you will have to re-download

docker	**system**	**prune**

Re-starting stopped containers

- To restart the container that was stopped:

```
docker start -a <id>
```

Retrieving Log Outputs

docker	**logs**	**<id>**

- It doesn’t run/create any container
- Best used in debugging the container

Stopping Running Containers

docker stop

Message to stop the process
Gives a little time for the process to clean-up and stop. If not stopped in 10 seconds, it is automatically killed

$$\begin{tikzcd}\text{stop} \arrow[r] & \text{SIGTERM} \arrow[r] & \text{process}\end{tikzcd}$$

docker kill

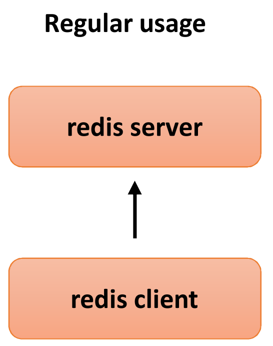
Stop immediately

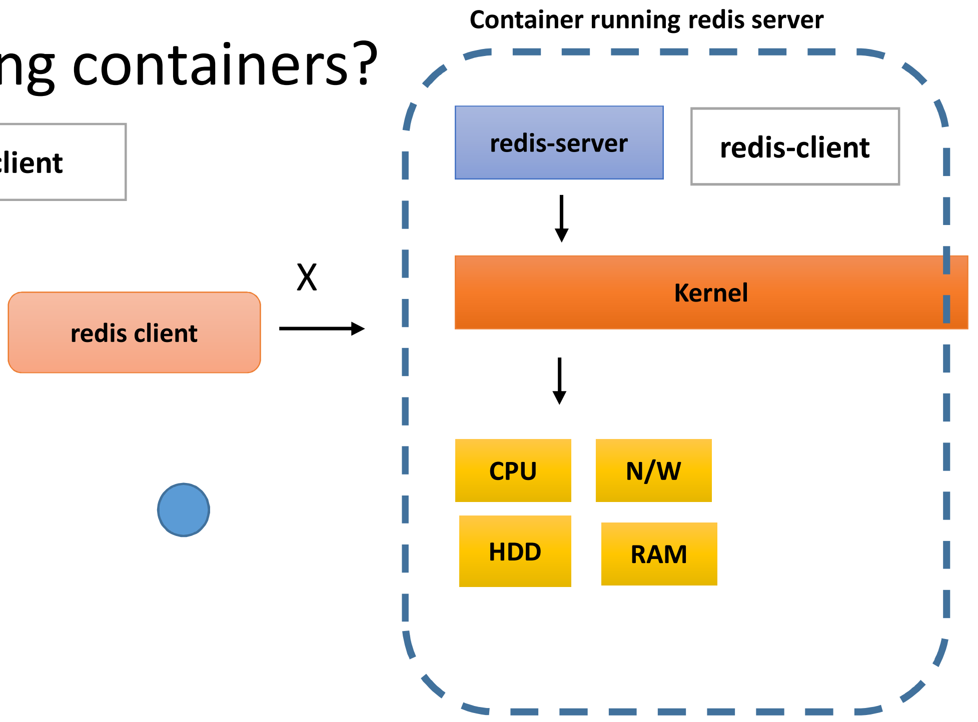
$$\begin{tikzcd}\text{kill} \arrow[r] & \text{SIGKILL} \arrow[r] & \text{process}\end{tikzcd}$$

Multi-command containers

- How to execute commands in running containers?

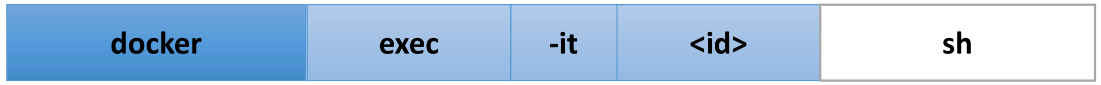
docker	exec	-it	<id>	redis-client





Importance of –it flag

- The –it flag allows for input from the console
  - If the flag is not specified, the docker will think that there is no way inputs can be received, so it will kick back to the prompt
- You will need terminal access to work with containers
  - Every single container is a virtual machine, it’s in a Linux world even if it is working on Mac/Windows



```
docker exec -it <id> sh
```

All shell commands including creating and managing directories can be issued in the shell

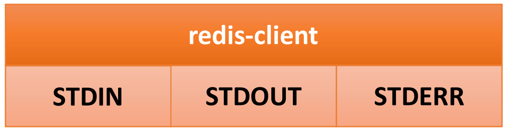
Importance of –it flag

- In Linux every process has STDIN, STDOUT and STDERR that helps communicate information in and out of the process

-it = -i & -t

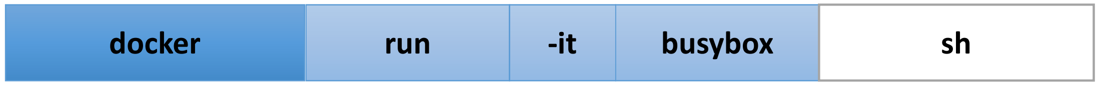
-i: **Stuff you type in terminal needs to be directed to the STDIN of process**

-t: **Shows up in a nicely formatted manner**



Starting with a shell

- We could use a docker run command to start a container and poke around inside a container
- In this case, on the downside chances are the process may not be running



```
docker run -it busybox sh
```

Container Isolation

- Each container is isolated and **do not share the file system**, unless there is a setup to connect the two
- Namespacing feature allows the isolation of resources