**Content**

[1. Introduction 2](#_Toc12784387)

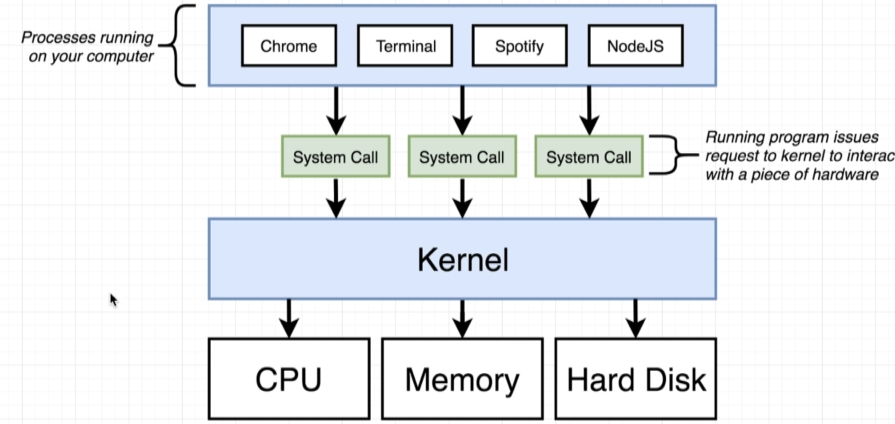
[1. Virtualization vs Containerization 5](#_Toc12784388)

[2. Dockerfile 6](#_Toc12784389)

[3. Docker commands 7](#_Toc12784390)

1. Introduction

**The Kernel** is a running software process that governs acces sbetween all the programs that are running on your computer and all the physical hardware that is connected to your computer as well. Intermediate layer that governs access between the programs and the actual hard drive.



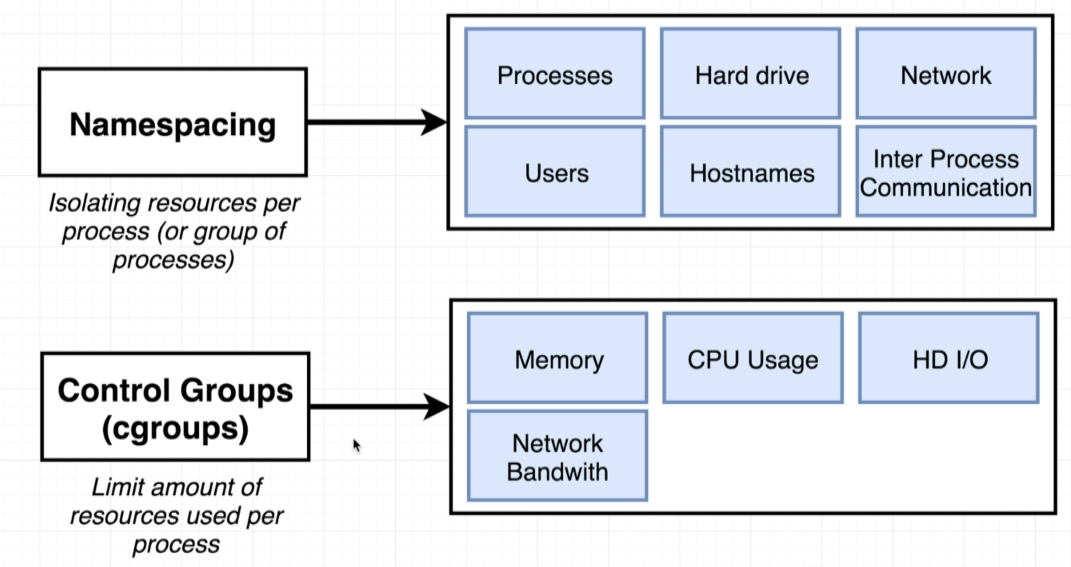
**CentOS** (Community Enterprise Operating System) is a Linux distribution that provides a free, enterprise-class, community-supported computing platform functionally compatible with its upstream source, Red Hat Enterprise Linux (RHEL).[5][6] In January 2014, CentOS announced the official joining with Red Hat while staying independent from RHEL,[7] under a new CentOS governing board.[8][9]

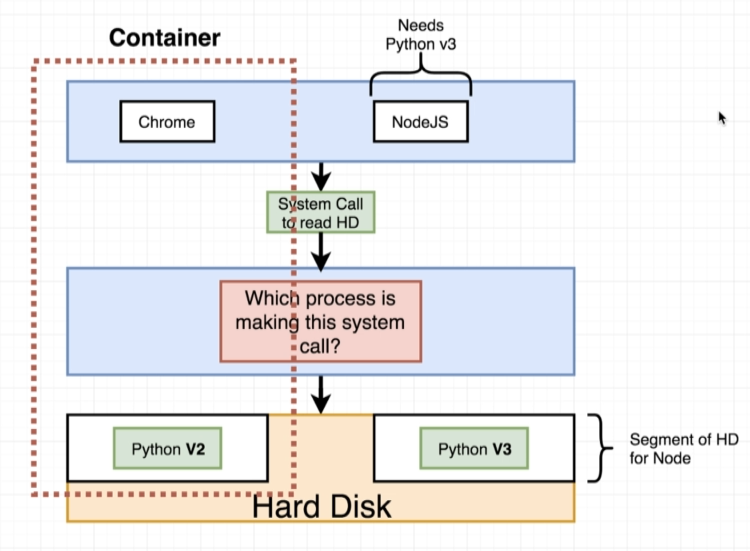
**Docker Hub** is a registry service on the cloud that allows you to download Docker images that are built by other communities. You can also upload your own Docker built images to Docker hub. In this chapter, we will see how to download and the use the Jenkins Docker image from Docker hub.

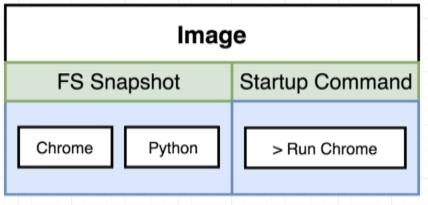
I**solate processes:**

**Namespaces** are named program regions used to limit the scope of variables inside the program. They are used in many programming languages to create a separate region for a group of variables, functions, classes, etc. With namespacing we are allowed to allocate resources as per a process or a group of processes and we essentially saying that anytime a particular process ask for a resource, we’re going to direct it to this one little specific area of the given piece of the hardware. But namespacing can be used even for software elements as well. We can namespace a process to restrict the area of a hard drive that is available for, or the network devices that are available or the ability to talk to other process. “redirect request for resource”

**Control Groups** – limit the amount of resources that a particular process can use. So namespacing is for saying: Hey, this area of the harddrive is for this proess, a control group can be used to limit the amount of memory that a process can use, the amount of CPU, the amount of harddrive I/O and network bandwidth.

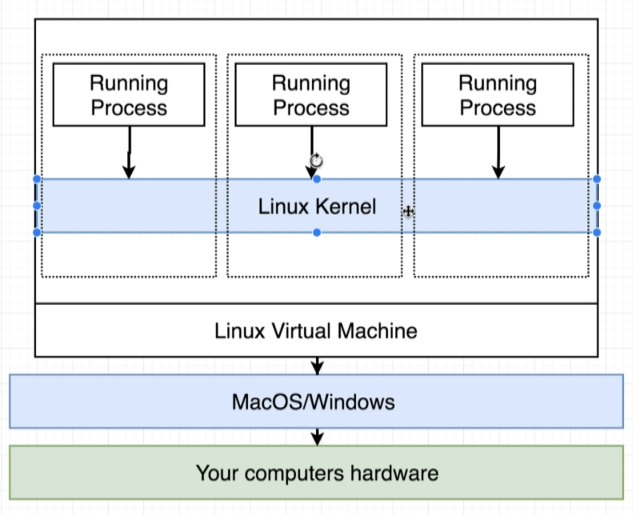




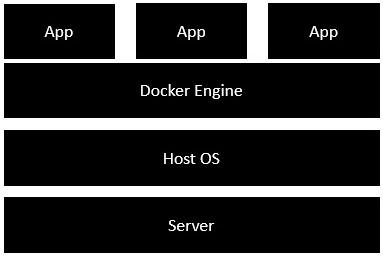
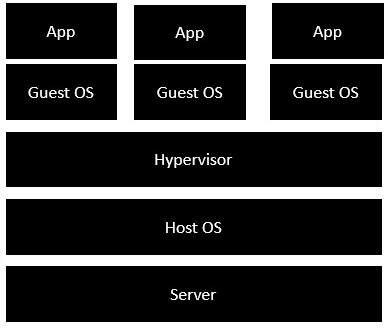


Container is a running process along with a subset of physical resources on your there are allocated to that process specifically.

An image is really kind of a snapshot of the file system along with a very specific start up command. Namespacing and Control Groups are not are not included by default with all operating systems – they are linux specific.



* 1. Virtualization vs Containerization



1. Dockerfile
2. FROM – specify the docker image what we want to use as base

FROM alpine

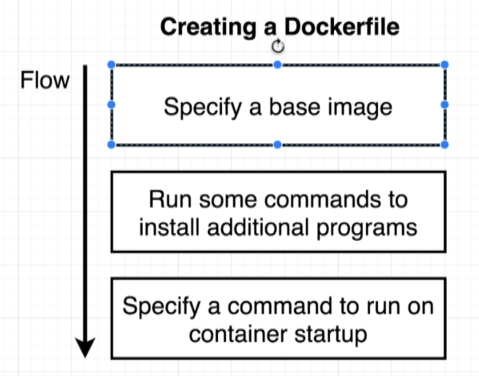
1. RUN – execute some commands while we are preparing our custom image

RUN apk add –update redis

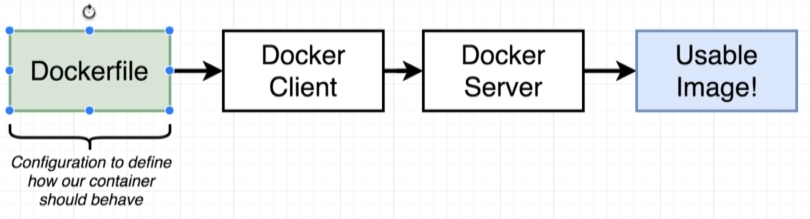
1. CMD – what shuld be executed when our image is used to start up a brand new container

CMD [“redis-server”]

1. LABEL
2. MAINTAINER (deprecated)
3. EXPOSE
4. ENV
5. ADD
6. COPY
7. ENTRYPOINT
8. VOLUME
9. USER
10. WORKDIR
11. ARG
12. ONBUILD
13. STOPSIGNAL
14. HEALTCHECK
15. SHELL



* 1. Creating a Docker Image

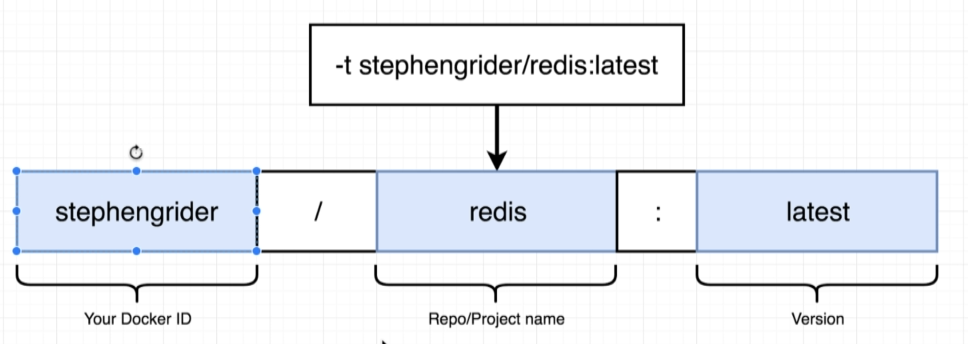


* Docker build .

(. – is the build context, set of files that belongs to our projects)

Docker Cache – if you want to change your Dockerfile you want the change to happen as far down as possible – the order matters when caching.

-t (tag the image, so we will not need to memorize / copy the image IDs )



Alpine and redis – these does not contain the docker ID because they are community images.

Whenever you want to create a custom image name it <docker id>/ …

* Docker commit –c ‘CMD [“redis-server”]’ <container id> (add a default command)

Pref Dockerfile because it is reusable. – fluid relation between containers and images.

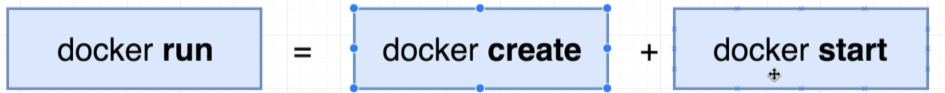
1. Docker commands

Bonus:

cat /etc/\*-release – get your linux distribution

1. Docker run <image name> <override initial command>

Create a container out of an image.



The process of creating the container is where we kind of take the file system in here and kind of prep it for use in this new container when we create the container. To start we execute the startup program.

1. Docker ps - to verify if docker is running? List containers (running)!
   1. Docker ps –a lists all of the images on the computer
2. Docker images
3. Docker rmi ImageID
4. Sudo service doker start / stop / status
5. Docker logs <container id> -> retrieve logs gen. by a given image
6. Docker system prune (deletes even the cache) – delete the containers that are sitting around and eating up the space.
7. Docker stop <container id> : a hardware signal is sent to the prim process of the container (SGITERM)
8. Docker kill <container id> : SIGKILL
9. Docker exec –it <container id> <command> (exec – run command in a running container)

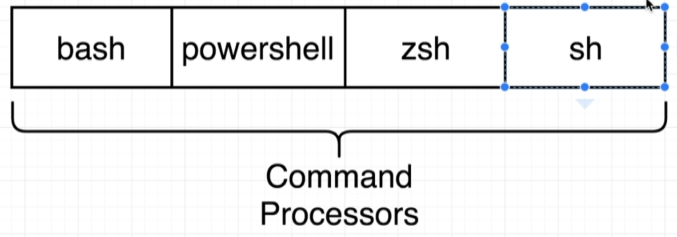
* -i standard in
* -t text formatting?

Open up a shell or a terminal in a context of your running container:

1. Docker exec –it <container id> sh

-a : attached? Give me any output coming from there

Sh is a command processor, something that allows us to type commands in and have them be executed inside that container.



1. Docker run –it <image name> sh

Start up a new container and run shell. Give me access to that shell. Typical is to start with an initial process like web server and then attach to it the shell w docker exec.