**DOCKER**

<https://www.udemy.com/course/docker-and-kubernetes-the-complete-guide/learn/lecture/11436636#overview>

This command will check Docker version.

***Docker run hello-world***

1. The Docker client contacted the Docker daemon.

2. The Docker daemon pulled the "hello-world" image from the Docker Hub.

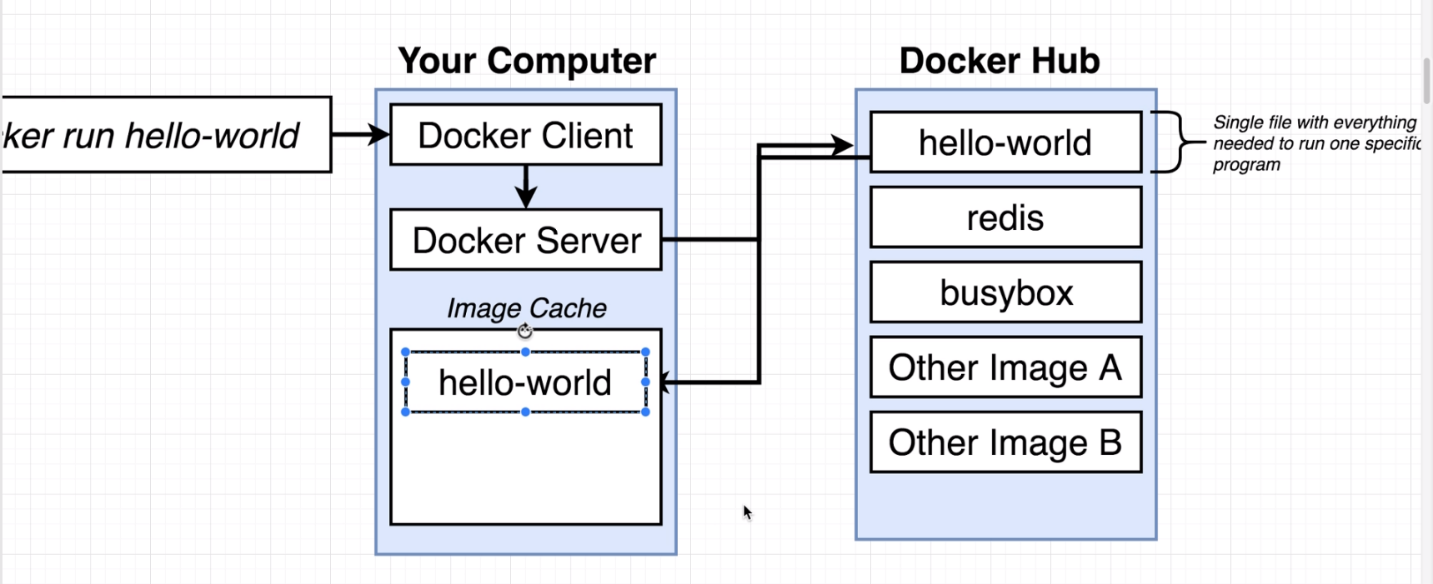
(amd64)

3. The Docker daemon/server created a new container from that image which runs the

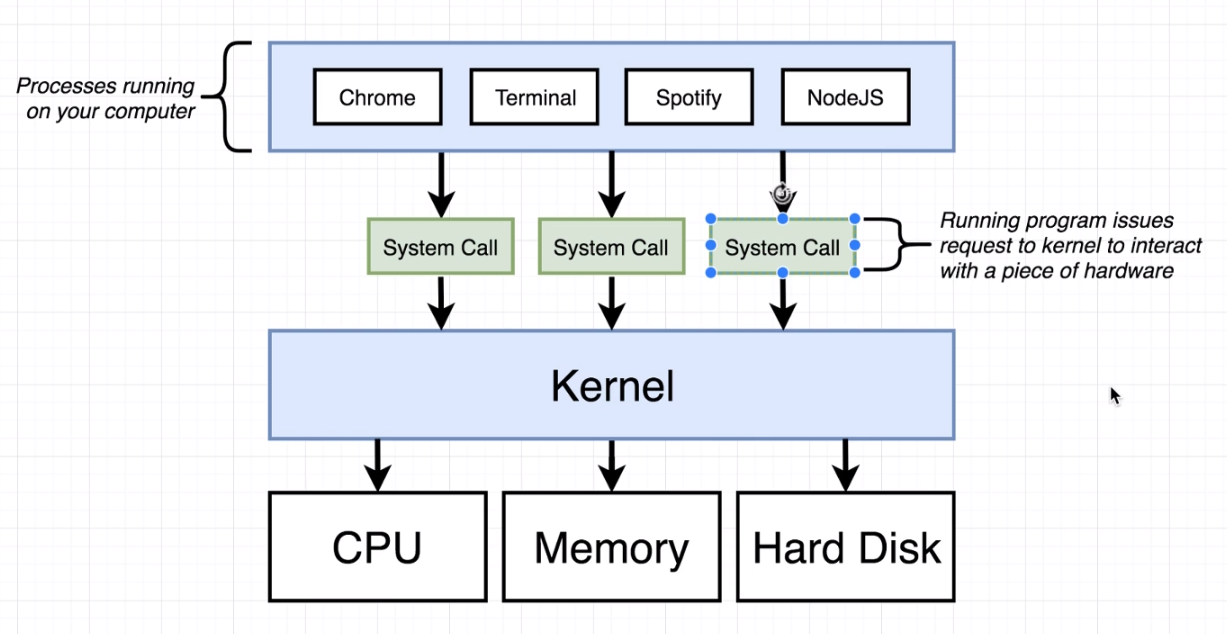
executable that produces the output you are currently reading.

4. The Docker daemon streamed that output to the Docker client, which sent it

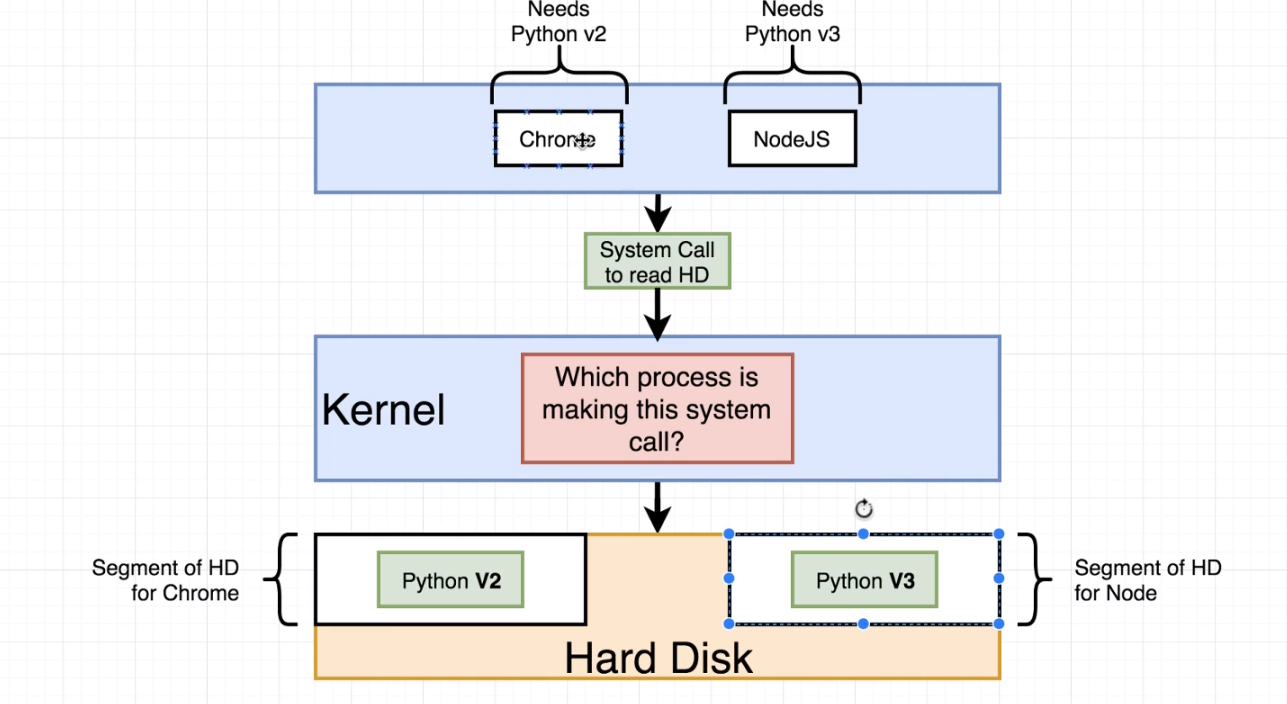
to your terminal [cached in computer locally].



Kernal layer is responsible to communicate between our program/application running on our computer and Hardware of computer like CPU/Memory etc.



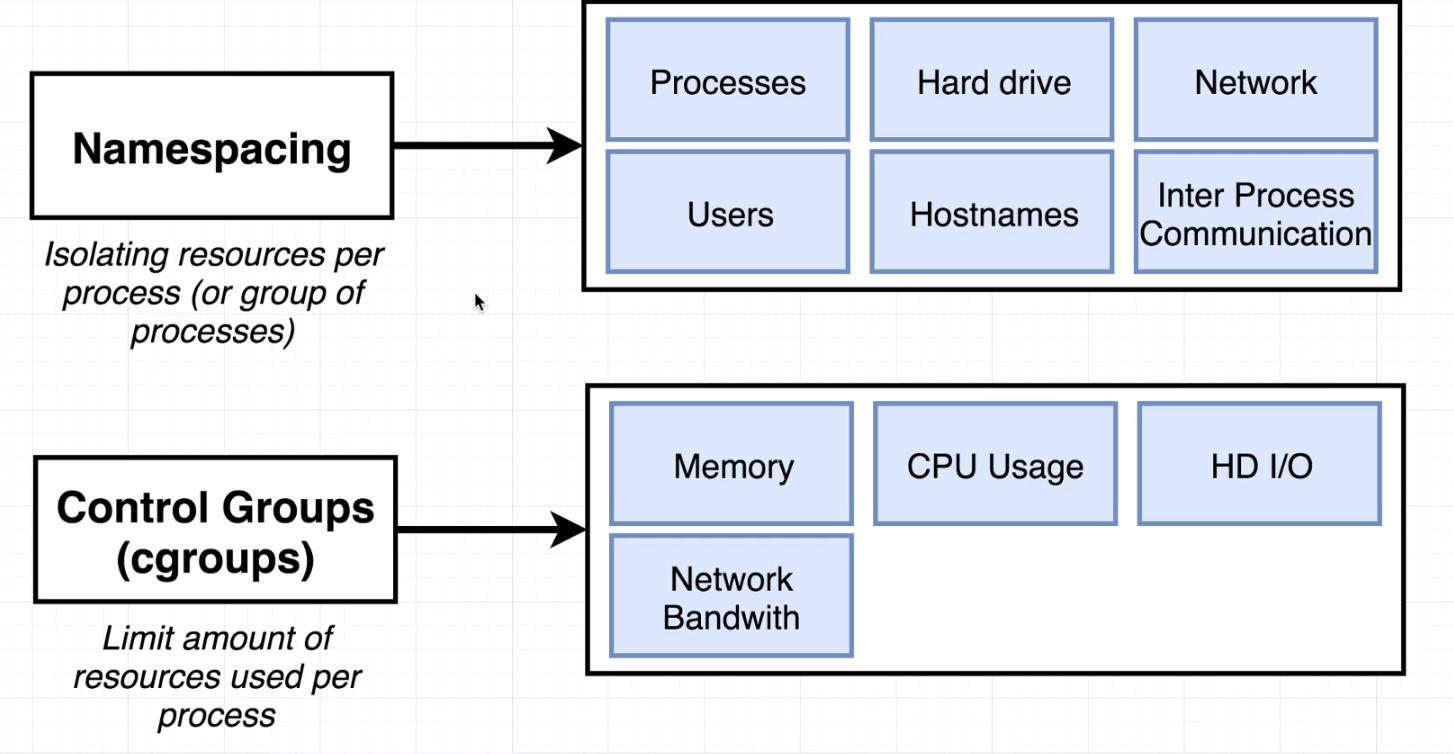
If we want, we can segmented our hardware resources according to requirement and can assign it to any application. This process it known as Namespacing.



In below diagram, Namespacing is saying that this particular area of resources [Hardware or software] is allocated for this particular process.

Control Group is saying this is the limit for resources allowed for a particular process.

Namespacing and Control Groups concept is only limited to Linux OS. Even I installed docker on local but I also installed virtual Linux to run docker.

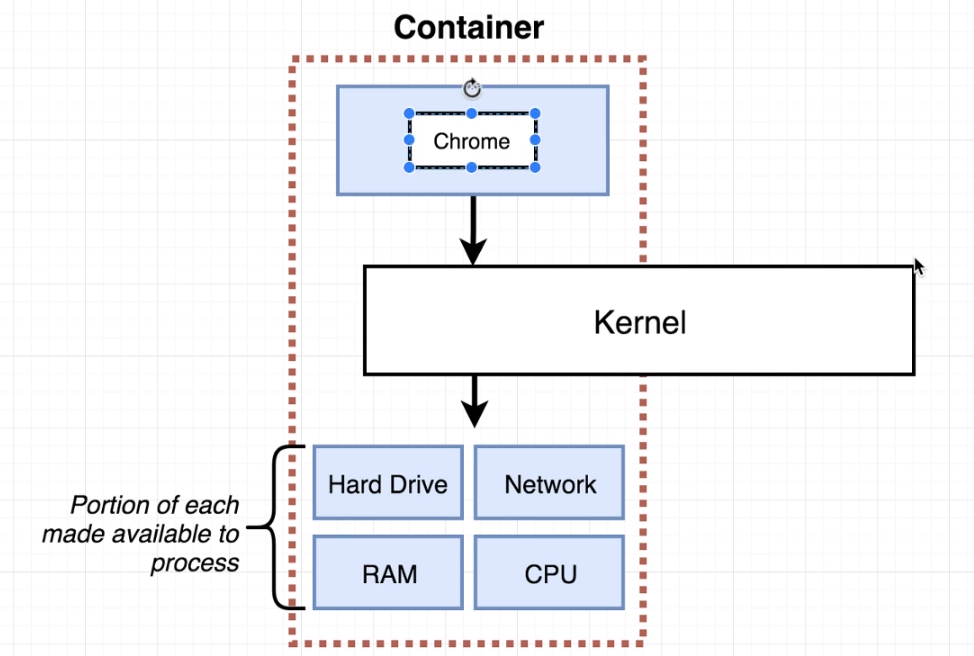


**Container:**

A process or set of processes that have group of resources specifically assigned to it. It has assigned very specific portion of computer resources like Hard Drive, RAM, N/W etc.

containers are dependent on images and use them to construct a run-time environment and run an application.

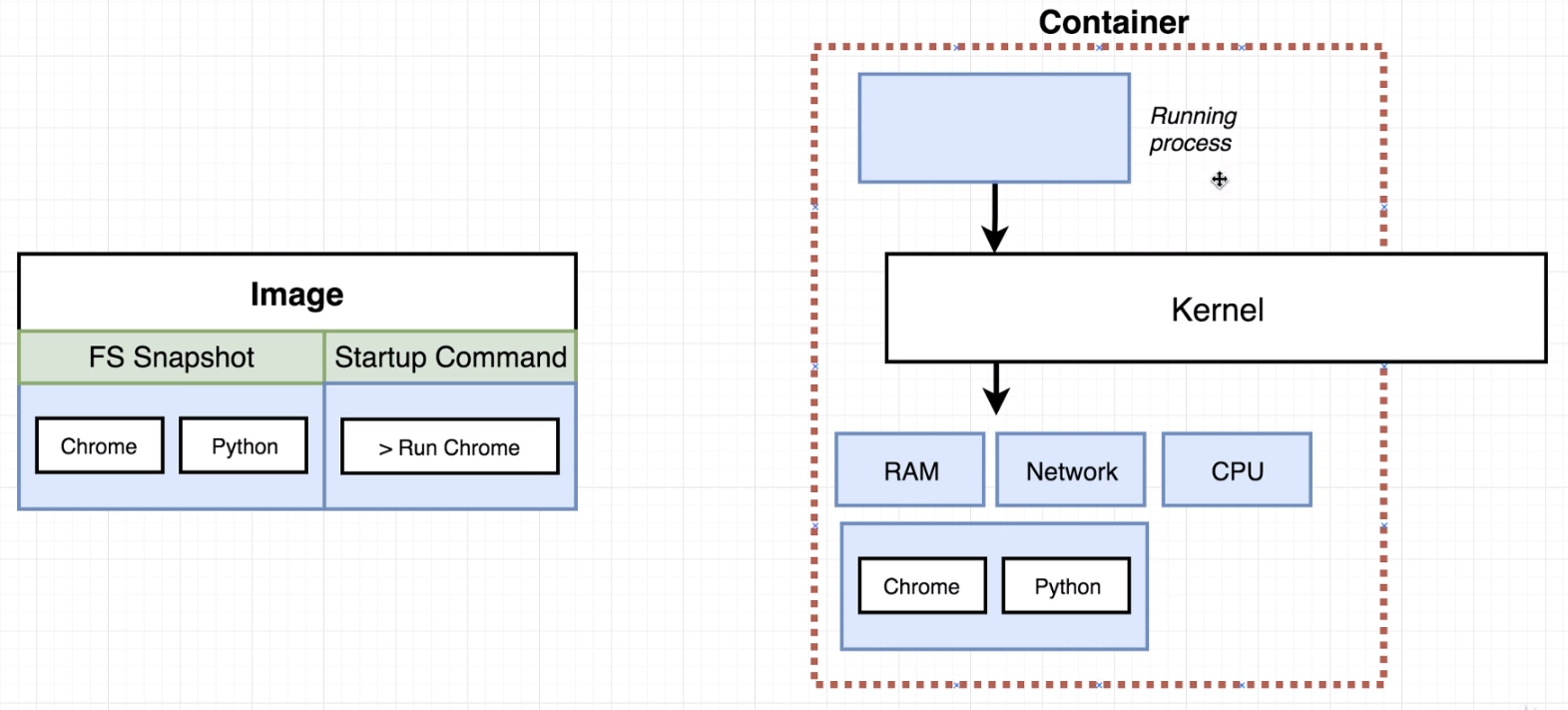
Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host.



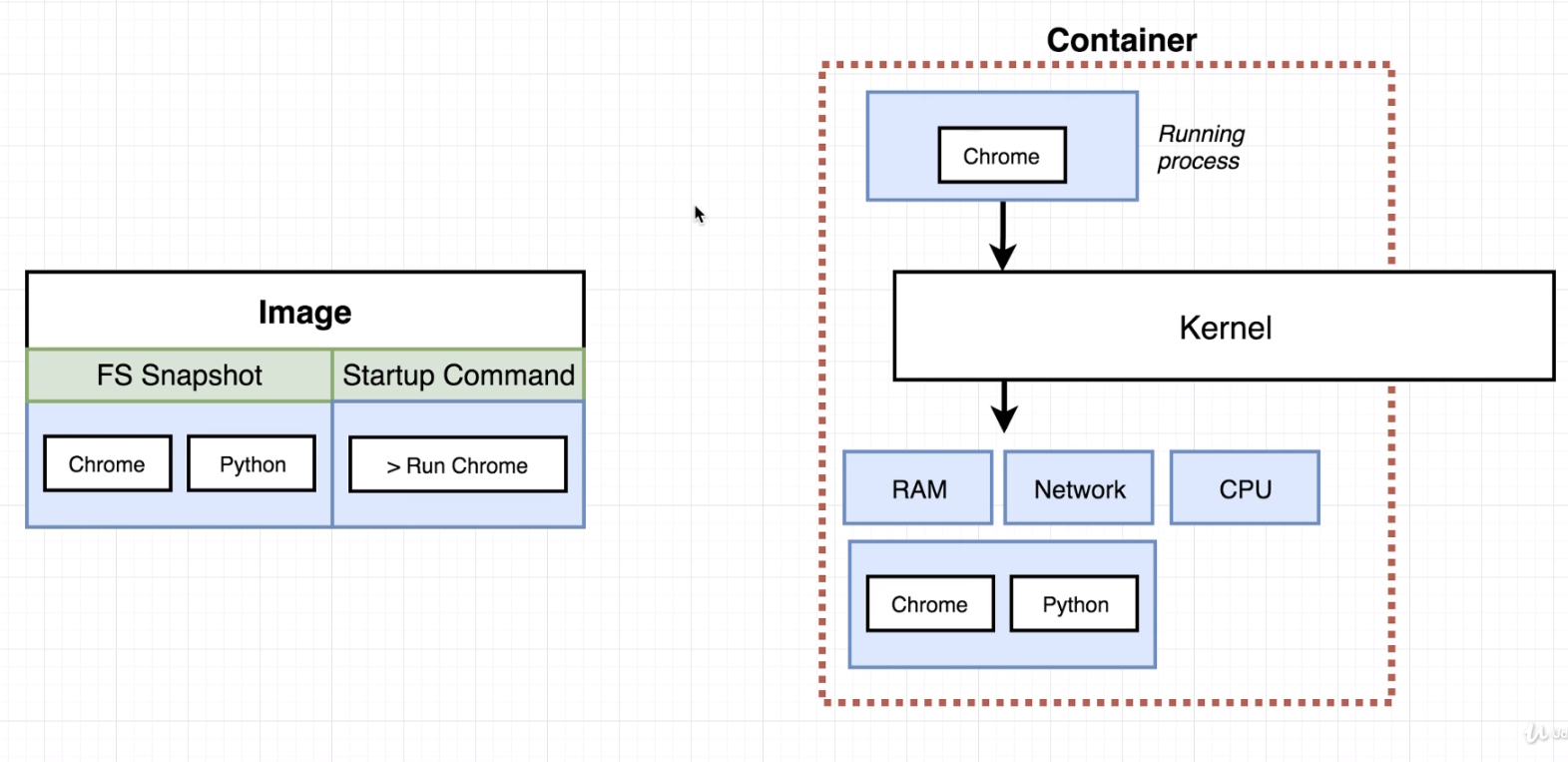
**Image:**

It’s a specific file system snapshot and startup command.

Once create container from Image, that time it will assign specific resources from our computer to this container.



Now startup command executed and it create new instance of this process [in our example we started chrome]. Once this instance started running, it is isolated from rest of resources like RAM, CPU etc as below:



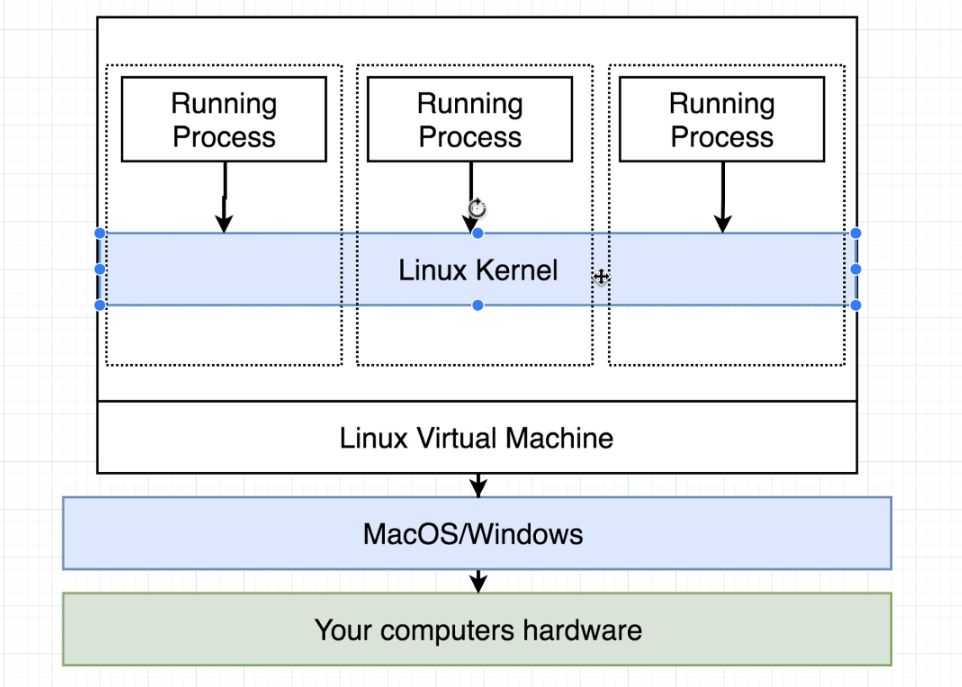
As we have virtual Linux up and running, docker containers are getting created inside this Linux virtual machine. Inside this virtual machine as shown below, we have a Linux Kernal, this Kernal will be hosting running process inside container and also responsible to restrict access of different hardware resources of our computer. We can see this virtual machine If we run **docker version** command, below details will come.

**Client:**

OS/Arch: windows/amd64

**Server**

OS/Arch: linux/amd64



**What is happening behind the scene once we run docker run command:**

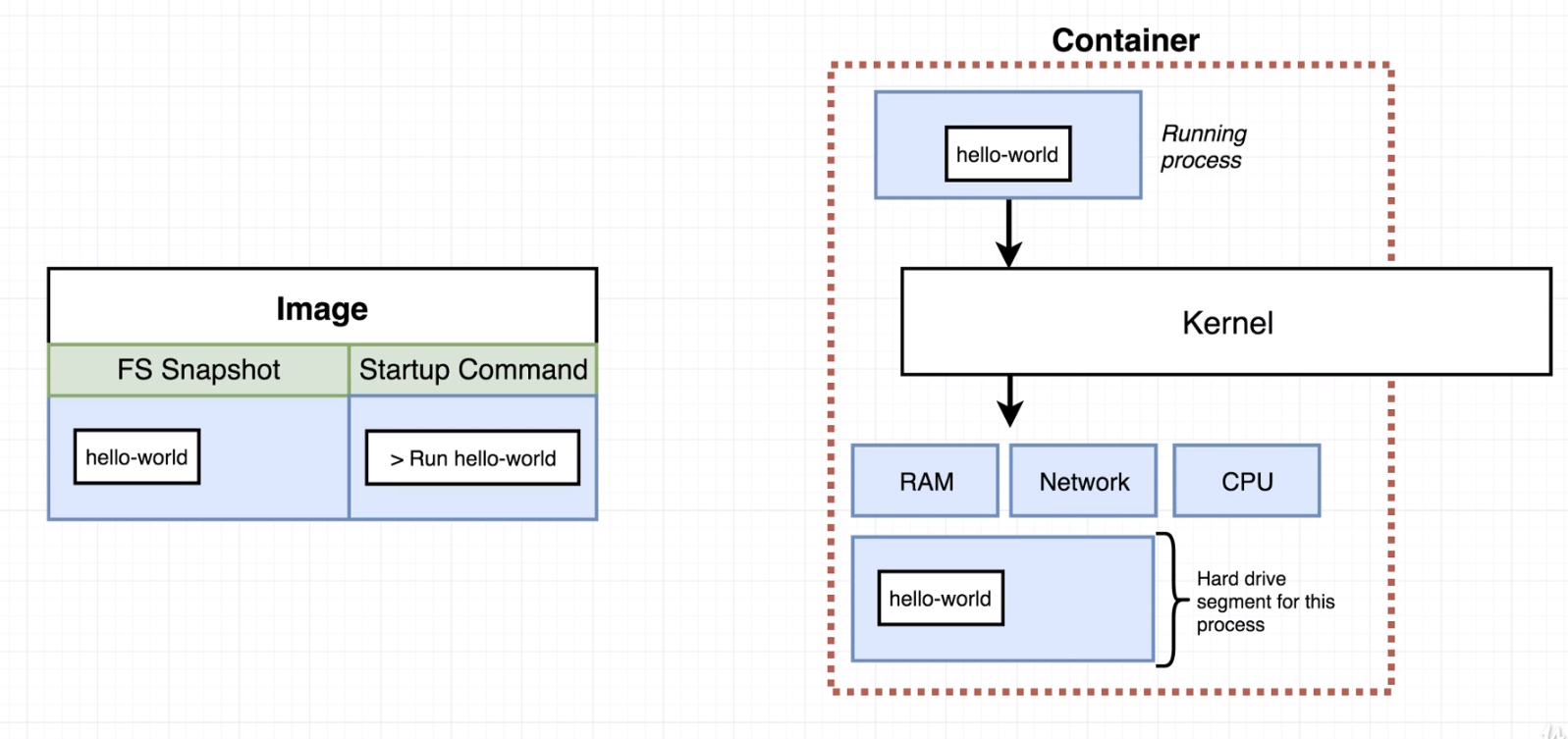


Image has code for hello-world along with Startup command. So, once we run it, a container will be created which is having running process of hello-world. Here Kernal has assign required hardware resources to it.

**docker run busybox** will pull image from docker hub and run it. We can override it’s functionality by giving one more command with it as below:

**docker run echo hi ash,** this command will return hi ash.

* **docker ps:** to list all running container
* **docker ps - -all:** to see all containers that ever been created on our machine
* **docker run = docker create + docker start**
* **docker create:** To create a container out of an image. We will get a container id post this command execution.
* **docker start -a <container id> :** To start a container which we have created out of an image. If we run docker start without -a then we will get only container id. So -a will be looking for output coming after running container and print on console.
* as **docker ps -all** gives us the list of all container that been ever created on our machine. We can take container Id from any of these containers and run it again. If we are running this container again. we can’t replace the default command for it. [Which was there while we created and ran this container first time] .
* **docker system prune:** To delete all container which were stopped and still available while we run docker ps –all. So, we can clean up our history using this command. It will delete:

- all stopped containers

- all networks not used by at least one container

- all dangling images

- all dangling build cache

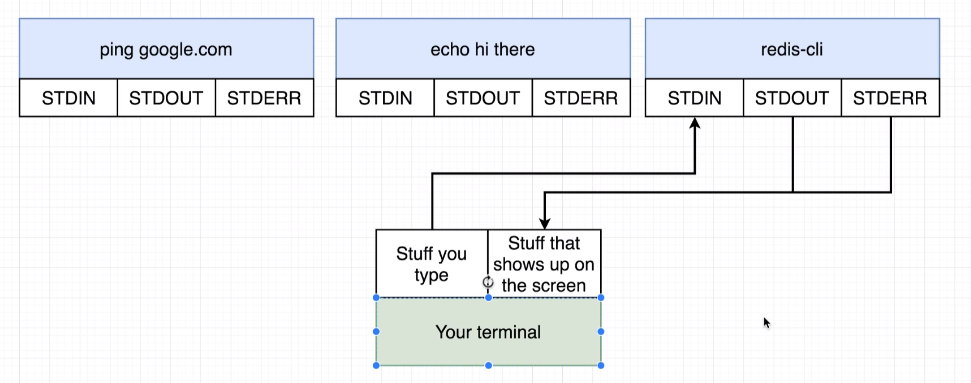
* **docker logs <container id>** : to retrieve all information for a container.
* **docker kill and docker stop**: docker stop will send SIGTERM (terminate signal) to the process and docker will have 10 seconds to clean up like saving files or emitting some messages. Use docker kill when container is locked up, if it is not responding.
* **docker exec:** this command is used when we want to start a second running program inside our container. For example: We have redis in-memory database in our container and we want to start redis-cli then we can use below command:

**docker exec -it <container id> redis-cli :** if we don’t use -it here, then we will directly come back to terminal without going to redis cli. Here in actual -it is separated, and it is -i and -t. Here -i is for attach our container terminal to STDIN [Standard Input] of new running process (which is redis cli here) and -t is just only for formatting the text.

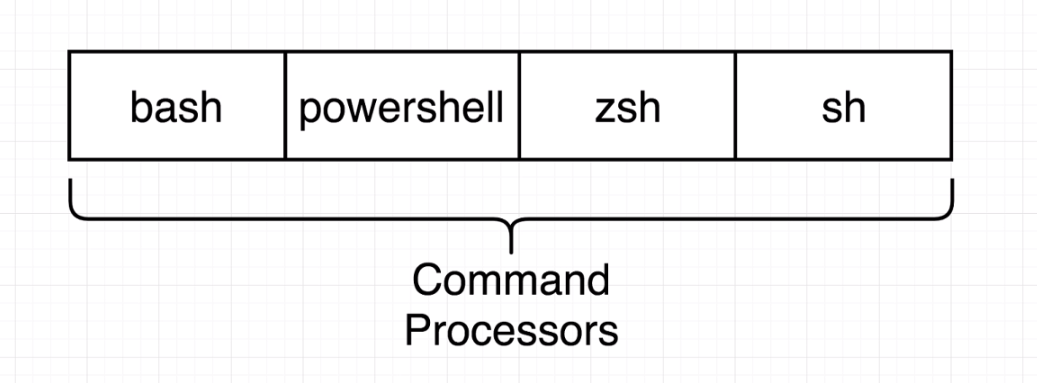
**set myValue 10**

**get mValue**

“10”



* **docker exec -it <container id> sh :** Here sh is a command processor/shell that is being executed inside container and allows us to type commands in and executed inside that container. Like git bash on window.



* **container isolation:** we can have more than 1 containers running for similar software/image. They will work independently without any clash.