**DOCKER and KUBERNETES: The Complete Guide**

By: Stephen Grider

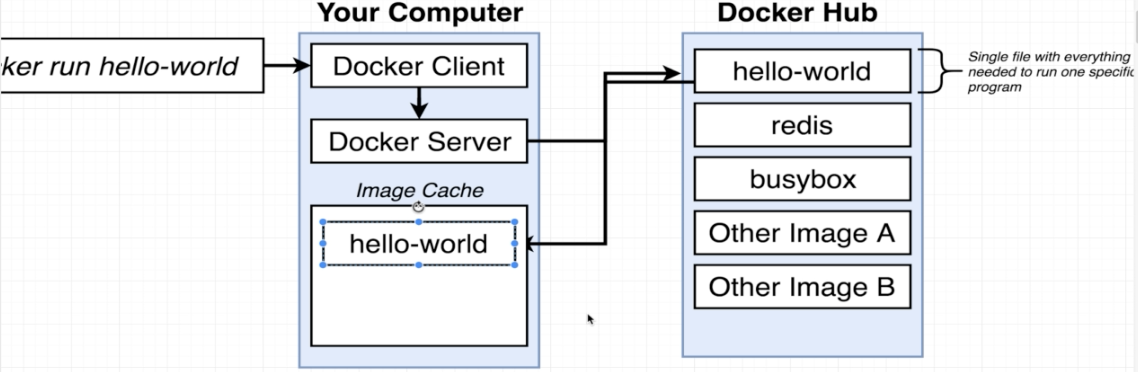
# What is Docker?

A container is a program with its own space of memory, its own space of networking technology and its own space of hard drive.

An image is a single file containing all dependencies and all the configuration required to run a very specific program.

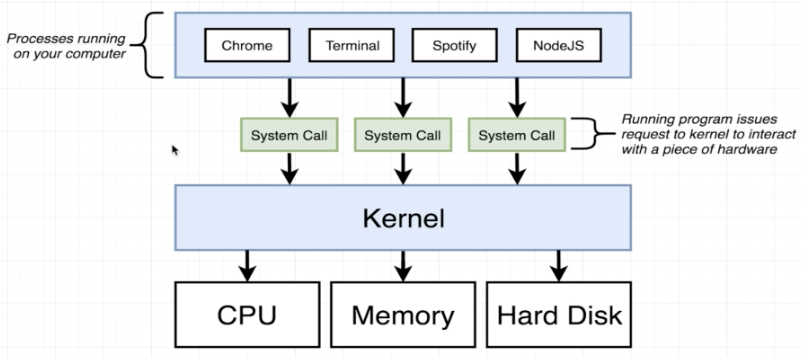
## Hello-world

* In iTerm, run these commands to:
  + Test docker image name hello-world: “docker hello-world”
  + This image print out a series of text
  + Docker client checks if there is a local image cache name “hello-world”, if there is none, docker client reaches out to docker server, which then connects to docker hub to download “hello-world” image.
  + Docker server then uses the image to create a container to run “hello-world” file.
  + Container runs a single program in it.



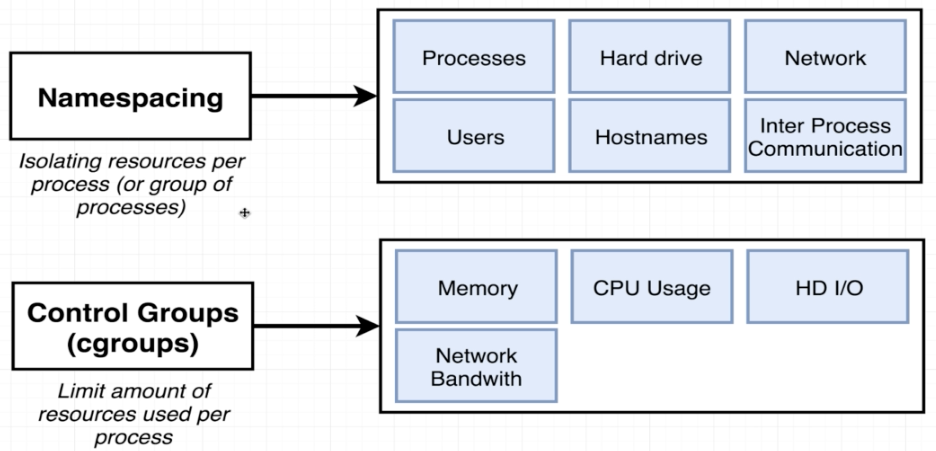
## What is a container?

- This is how an operating system works:



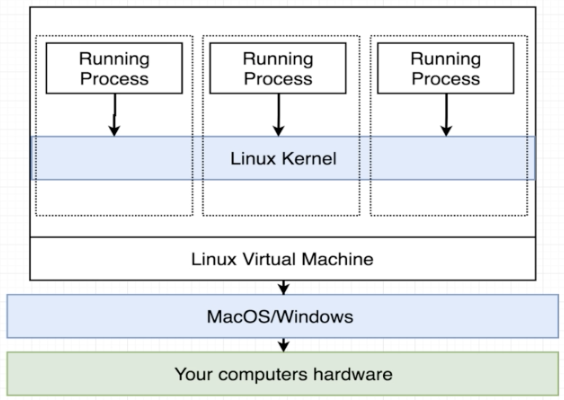
|  |  |
| --- | --- |
| **Specific Features for Linux Operating System** | |
| **Namespace** | **Control groups** |
| - Name space can be used for hardware and software.  - Can restrict the area of a hard drive that is **available**, or the **availability** of network devices. | - Limit the **amount** of resources |

* **Combination of Control Groups and Namespace** can be used to isolate a single process and limit the amount of bandwidth essentially that it can make use of.

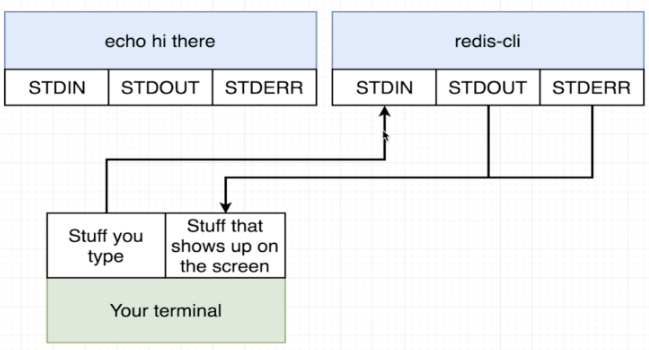


|  |  |
| --- | --- |
| **A container** | **An Image** |
| - Is a process or a set of processes that have a grouping of resources specifically assigned to it. | - Is a file system snap shot.  - Similar to a copy and paste of a very specific set of directories or files. |
| - A container is a running process (chrome) a long with a subset of physical resources on a computer.  - Kernel will isolate a hard drive and allocate that space for the container. | - An image can contain chrome any python.  - Another image can contain only start up command. |

* When running a docker, you really run a Linux virtual machine on your OS:



* Every process created inside a container has three communication channels attached to it: stdin, stdout, stderr:



## Container Life Cycle

### docker run = create + start

docker **run** = docker **create** + docker **start**

|  |  |  |
| --- | --- | --- |
| **Commands** | **Explain** | **Detail commands** |
| **Create** a container | The file system starting | docker create <image\_name> |
| **Start** a container | Executing the start up command | docker start <container id> |
| Docker start –a <id> | -a: attach to the container and watch for the out so that it can be printed onto the terminal. |  |

% docker create hello-world

61dd07632a20e4b53bc442486534d367f91f917dda470aa507ad23e8251e95b2

% docker start –a 61dd07632a20e4b53bc442486534d367f91f917dda470aa507ad23e8251e95b2

% …

### Restarting stopped containers

% docker run busybox echo hi there

hi there

% docker ps 🡺 show busybox naturally exited and its id

% docker start –a <container ID> 🡺 “-a”: attach to the container. Restart busybox

hi there

## ssdsds

-

# Docker commands

* Docker Command Format:

|  |  |  |  |
| --- | --- | --- | --- |
| **docker** | **run** | **<image\_name>** | **command!** |
| Reference the docker client | Try to create and run a container | Name of image to use for this container | Default command override |

|  |  |  |
| --- | --- | --- |
| **Docker commands** | **Notes** | **Result** |
| docker ps | Show running processes if there is any | Show headers: Container ID Image Command Created Status Ports Names |
| docker run busybox ping google.com | Run this command on another iterm. Now it will show info of the current container.  “ctrl + C” to stop the process. |  |
| docker ps |  | Info:  Container ID = 83c2ea82e13b  Image = busybox  Command = ping google.com  Created = 35 seconds ago  Status = Up 34 seconds  Ports =  Names = dreamy\_bells |
| docker ps --all | List all containers that were ever created on the current computer.  Use this command to get container ID | Show headers and info for each header: Container ID Image Command Created Status Ports Names |
| docker run hello-world | Run an image that print out a docker process of contacting docker daemon, etc | Print these out:  Hello from Docker!  This message shows that your installation appears to be working correctly   1. The docker clickent contacted the docker daemon. 2. The docker daemon pulled the “hello-world”. etc |
| docker pull Ubuntu | Install docker for Ubuntu |  |
| docker run –it ubuntu bash | “boot” into a virtual machine of linux.  “-it”: interactively  “bash”: shell command |  |
| docker version | Display client and Server versions |  |
| docker run busybox echo hi there | “busybox”: an image name.  “echo hi there”: alternate command to run inside the container after it is created. |  |
| docker run busybox | **Next time**, you can just type this command without “echo hi there”, it will print out the same output.  “echo hi there” is the default command. |  |
| docker run busybox ls | “ls”: print out all the files and folders inside a given directory. |  |
| docker run busybox ls / | Show files in root folder |  |
| docker system prune | Delete all the containers, and don’t leave them in stop state.  Warning will be displayed.  This command also deletes build cache and images; also show how much space has been reclaimed. |  |
| docker logs <container id> | Will collect the output of the given container and print those out on terminal. | Ex:  % docker create busybox echo hi there  % <containerId>  % docker start <containerId>  % docker logs <containerId>  % hi there |
| docker stop <container id> | Docker sends a “sigterm” which is short for terminal signal. It’s a message that tells the container (the running process) to shut down on its own time. Allow the process to clean up before shut down. |  |
| docker kill <container id> | Docker sends a “sigkill” which is short for kill signal. This message tells the container (the primary running process in the container) to shut down instantly.  Use this signal if there process seems to be hung. |  |
| docker run redis | Pull redis from docker hub, Install redis (in memory database) with docker, create and start redis. | …  \* Ready to accept connections |
| docker **exec** –it <container id> <command> | Execute an **additional command** in a container.  **-it**: interactively  Run “docker ps” to get container id of redis. redis-cli allows interaction with the database.  cli: command line interface.  Get shell access or terminal access to your running container. | Ex:  %docker ps  %docker exec –it 81087e9c6f21 redis-cli  % set myvalue 5  % OK  % get myvalue  % “5” |
| docker **exec** **–it** <container id> **sh** | Run a process with shell access.  Able to type unix commands at the prompt.  Full terminal access inside the container 🡺 good for debugging.  Enter : Ctrl+D to **get out of shell**. | Ex:  %docker exec –it 81087e9c6f21 sh  # cd ~/  # redis-cli  127.0.01:6342> set myvalue 5  127.0.01:6342> OK  127.0.01:6342> get myvalue 5  127.0.01:6342> Ctrl +C  # Ctrl + D  % |
| bash /powershell/ zsh/ sh | Command Processors |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Linux Commands

|  |  |  |
| --- | --- | --- |
| **Commands** | **Notes** | **Result** |
| top | View the busy processes |  |
| q | To quit top |  |
| ls / | Show files in root folder |  |
| exit | Close down Ubuntu shell |  |
| -i | Attach our terminal to the stand in channel of the running process. |  |
| -t | Kind of make the text show up in nicely format on screen. |  |
| -it | = -i -t |  |
| cd ~/ | Go to home directory |  |
| export b=5 | Export a variable b with value 5 |  |
| echo $b | Show value of variable b | #5 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |