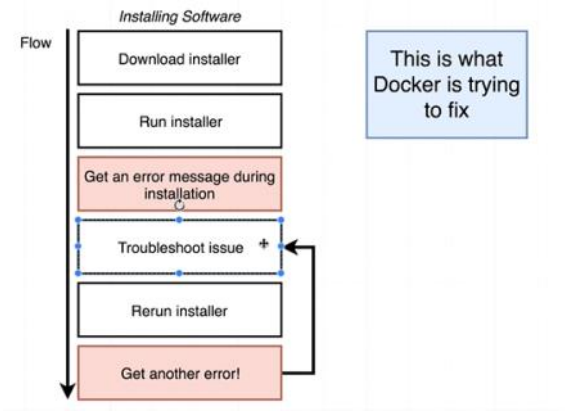
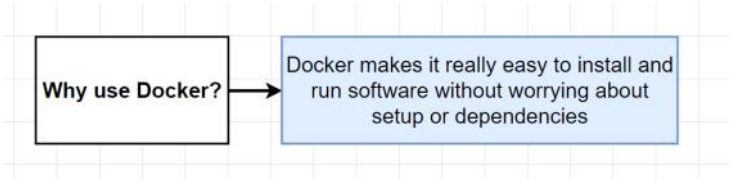
Docker - Intro

Wednesday, July 21, 2021 6:38 PM

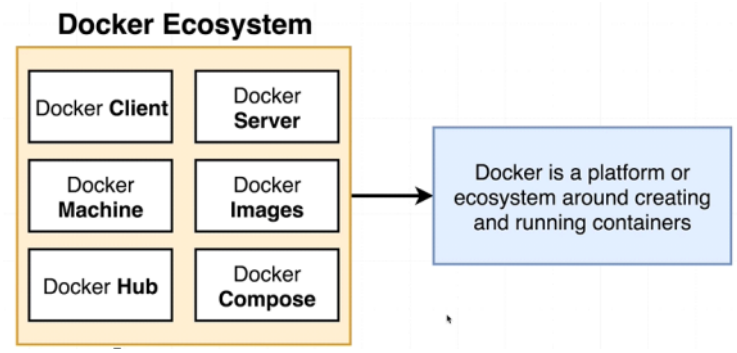
Why Docker?

- Install and run softwares on a container instead of using client machine installation individually

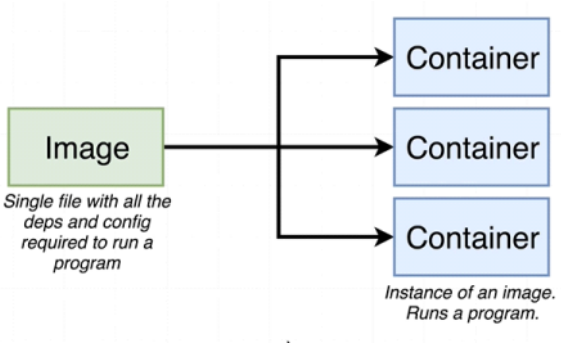




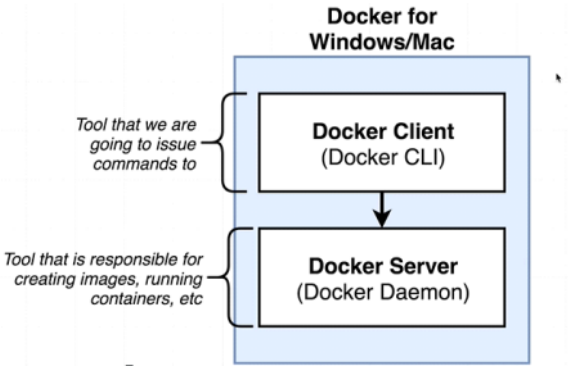
What is Docker?



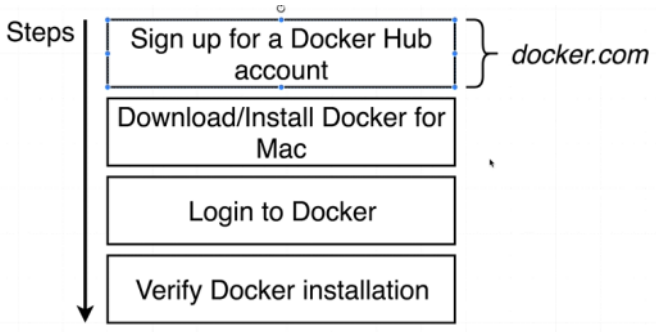
What is container?

- 

- when we ran "docker run -it redis"
  -> The docker-cli reaches out to docker-hub and it downloads a single file called image
  -> The image containing all the dependencies and all the configuration required to run a very specific program like Redis
  -> This image will be stored on your hard-drive and at some point of time we can use this image to create a container.
  -> A container is the instance of an image
  -> **A container is a program which its own isolated set of hardware resources, it has its own little space of memory, own little space of networking technology and own little space of hard drive as well.**

- 

Installation

- 

- WSL2
  - Some Windows 10 Home users may be able to install Docker Desktop if their computer supports the Windows Subsystem for Linux (WSL2).

    Please see this guide first which includes all of the steps to install and enable WSL2:

    https://docs.microsoft.com/en-us/windows/wsl/install-win10

- Once you have successfully installed and enabled WSL2 and then installed the Linux OS of your choice, continue to the Docker Desktop installation docs here:

  https://docs.docker.com/docker-for-windows/wsl/

- With Docker Desktop installed using WSL2 as a backend, you will now be able to follow along with the course. You will access the applications at localhost just like we do in the video lectures.

- **Important** - A significant difference when using WSL2 is that you will need to create and run your project files from within the Linux filesystem, not the Windows filesystem. This will be very important in later lectures when we cover volumes.

- You can access your Linux system by running wsl in the Cortana bar.

- Docker Toolbox
  - If you are using a Windows 10 machine that does not support WSL2, or, you are using even older versions such as Windows 7, you will need to install Docker Toolbox. It is important to note, however, that this software has been officially deprecated by the Docker engineers and may not continue to work into the future.

  - Release downloads are available here: https://github.com/docker/toolbox/releases

  - Download the .exe for the latest release and run the installer. Docker Toolbox will setup and install everything you need including VirtualBox.

  - You may also need to enable virtualization in your computer's BIOS settings. This will be different for each manufacturer, please refer to their documentation on which keys to use to access these settings on reboot.

- After Toolbox is finished installing, open the Docker Quickstart Terminal. This will complete the setup and provision your VirtualBox machine.

- Launch the Docker QuickStart terminal and type the command: docker run hello-world / docker version

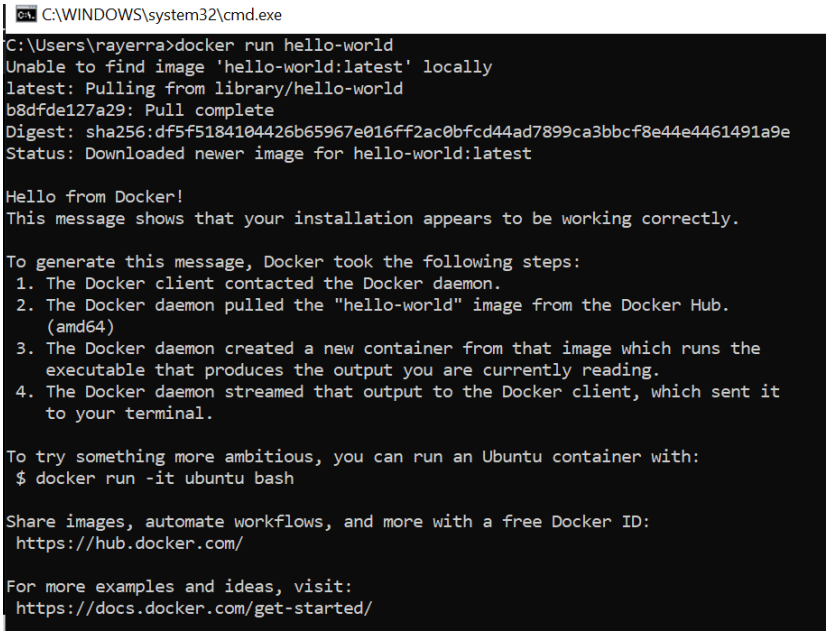
- This should pull down the test container and print hello-world to your screen.

- **Important** - A major difference between the course lectures using Docker Desktop vs. using Docker Toolbox is that you will not be able to use localhost anymore. Instead, you will need to use the IP address returned by running docker-machine ip

- Very commonly this is **192.168.99.100** but for you, it could be slightly different.

Running Docker

- *docker run hello-world*

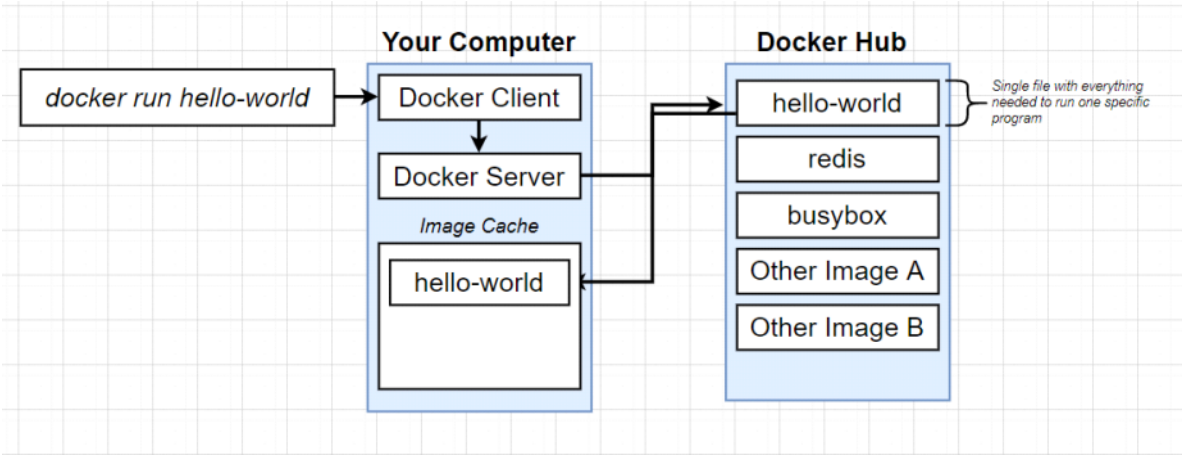
  - 

  - As soon as run this command in terminal, which starts up the **docker-client (docker-cli)**
  - **docker-cli** is in charge of taking commands from user and communicating them to the **docker-server**.
  - **docker-server** is in-charge of the heavy lifting .
  - When we ran the command "*docker run hello-world*" it means that we want to start up a new container using the **image** named "hello-world"
  - The **docker-server** saw that we are trying to start a new container using an **image** called "hello-world" and **docker-server** immediately checked for a local copy of "hello-world" in the personal machine. So to be precise the docker-server looked into something called "**Image Cache**"
  - As soon as **docker-server** figures out that the **Image Cache** is empty, it reaches out to a free service called **docker-hub**.
  - **docker-hub** is a repository of free public images that we can freely download and run on our personal machine.

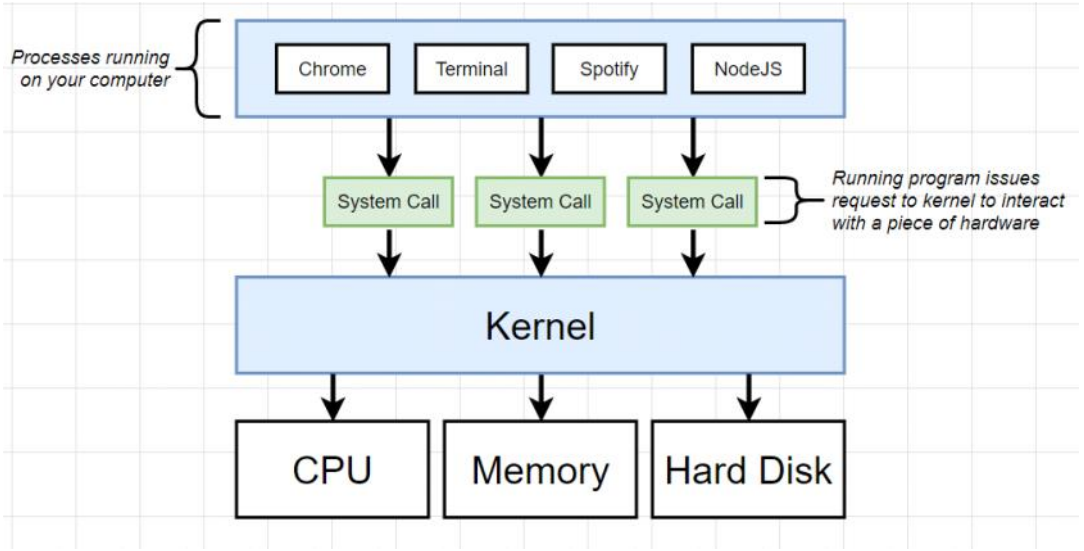
- As soon as **docker-server** finds the "hello-world" image in the **docker-hub**, **docker-server** downloads the image and places it in the **image cache**.
- Now that we got the image we need to use it to create the **instance** of an **image** through **container.**
- Docker-server takes that image file loads up into memory, created a container out of it and ran the program associated with it to generate the message with steps docker took.

```
Hello from Docker!
This message shows that your installation appears to be working correctly.

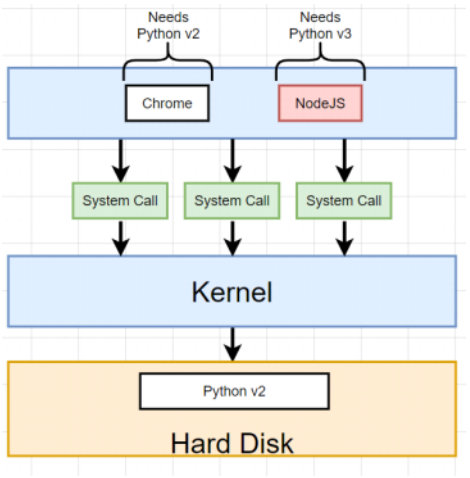
To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.
```



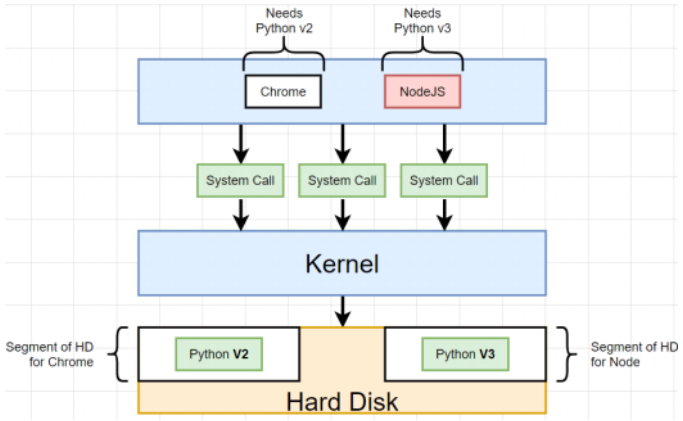
Container



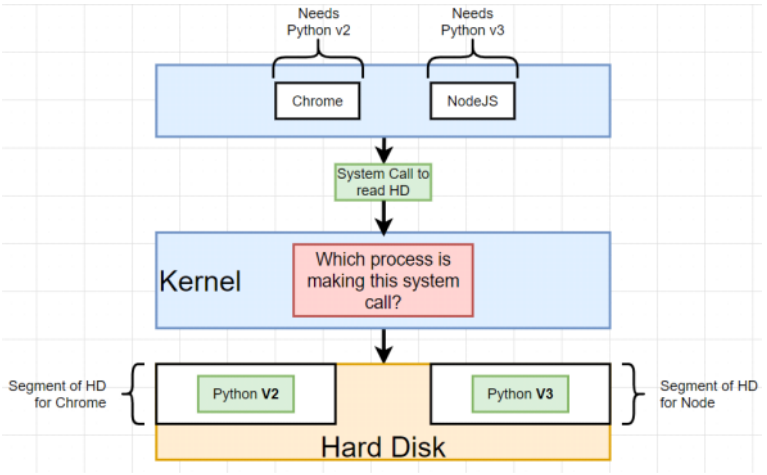
- To understand an container we can take how our OS runs on the computer as an example
- **Kernel** -> Running software process that governs access between all the programs running on the computer(Chrome, Terminal, NodeJs, ..) and all the physical hardware connected to your computer(CPU, Memory, Hard Disk)
- NodeJs does not directly speaks to the device, it communicates via Kernel that it wants to write a file to the hard disk. The information is passed over to the Kernel by NodeJs and the kernel eventually persists it to the hard disk.
- The running programs(Chrome, Terminal, NodeJs, ..) interact with the Kernel using System call.
- System calls are like function invocations, the kernel exposes different endpoints to process the information to the physical hardware.
- Ex. If we have two programs running on the same computer(Chrome and terminal) and both of them needs different versions of python to run it (v2 and v3)

- 

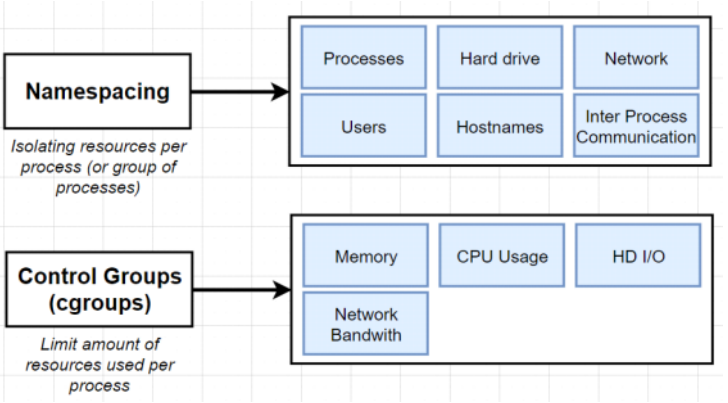
- To resolve these kind of issues, we can use something called namespacing (OS feature)
- With namespacing we can take a look at all the different hardware resources connected to the computer and we can segment out portions of those resources as shown below

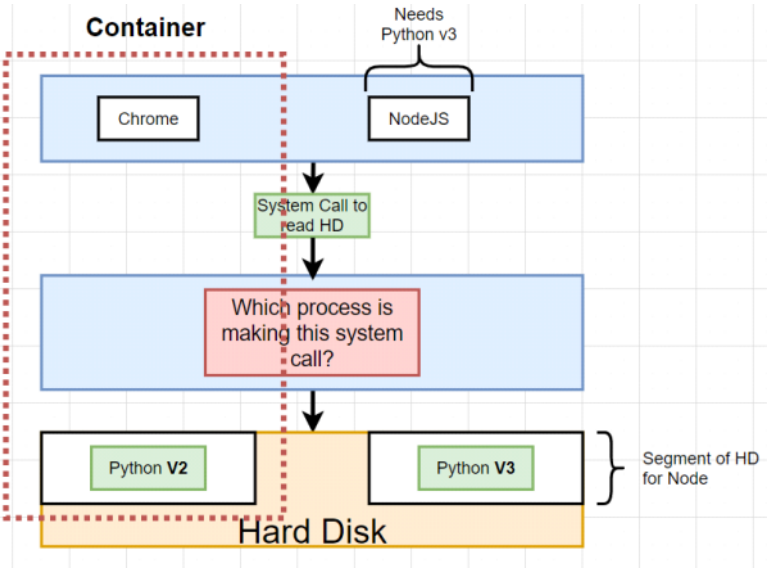
  - 

    - The Kernel will figure out which process is making the system call and reaches to the respective segment.

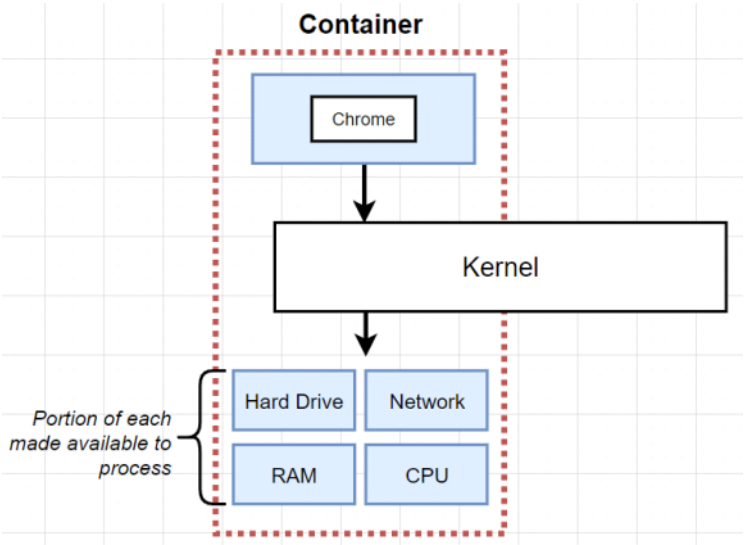
      - 

- Similarly we can use namespacing for software as well to redirect requests to an resource for a particular process
- Similar to namespacing we have a concept called Control Groups which can be used to limit the amount of resources used for a particular process
- **So namespacing is used to tell which area of hard disk can be used for a particular process and control group is to limit the amount of memory, CPU usage, network bandwidth, hard drive input/output that process can use.**

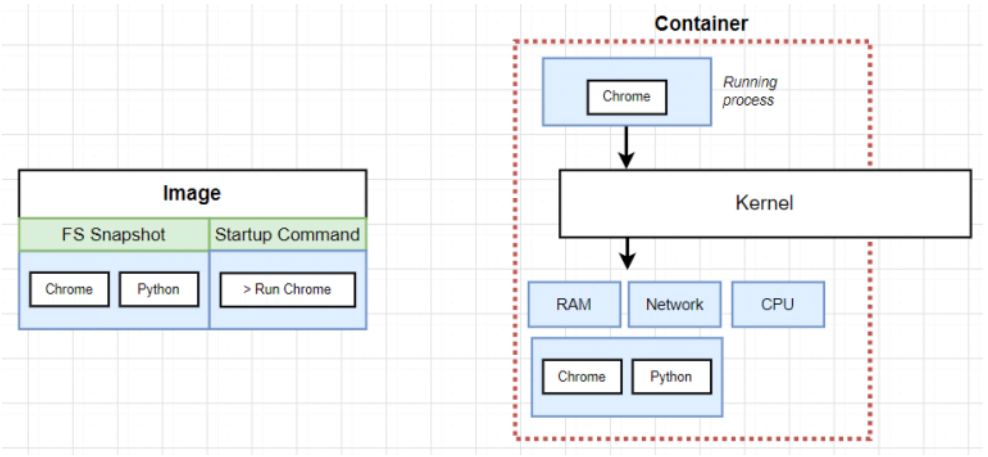
- 

- 

- **The entire vertical of a running process and the segment of the hard disk the process can talk to is referred as a Container.**
- **A container is a process or a group of processes that have a group of resources associated with it.**

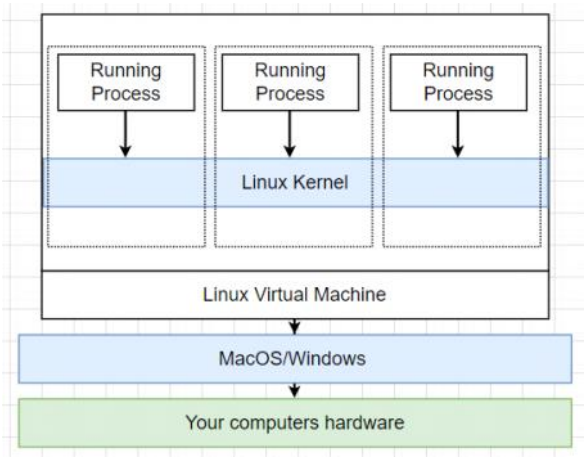
- 

- How is the above explanation for Container related to the image for which we are creating an instance?

- 

    - Image is a FileSystem Snapshot, it is like a copy paste of the very specific set of directories/files (For ex. image can contain only chrome and pythin)
    - Image also has a startup command
    - So, when we Container creates an instance of the image, the file system snapshot inside the image is taken and placed in the segment allocated in the hard drive.
    - Then the startup command is executed and chrome is opened.

- However the namespacing and control groups features are specific to Linux OS (Not windows and MacOS)
- But how are we running docker on windows/MacOS ?

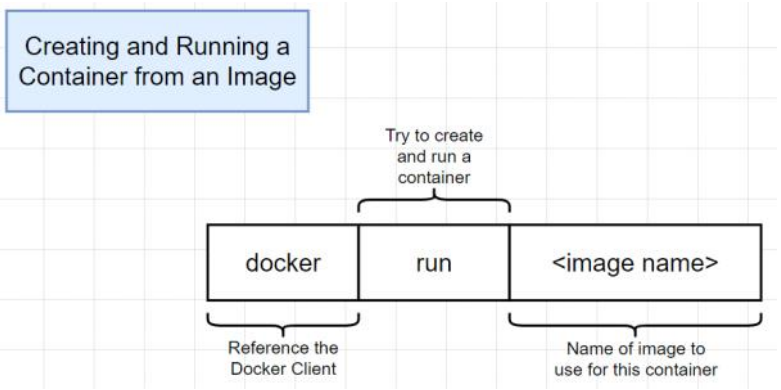
- 

    - When we installed Docker we installed a Linux virtual machine, where all the containers will be created
    - Inside the Linux VM, we have the linux kernel where it will be hosting running processes inside of containers, limiting/constraining/isolating access to different resources in the computer.

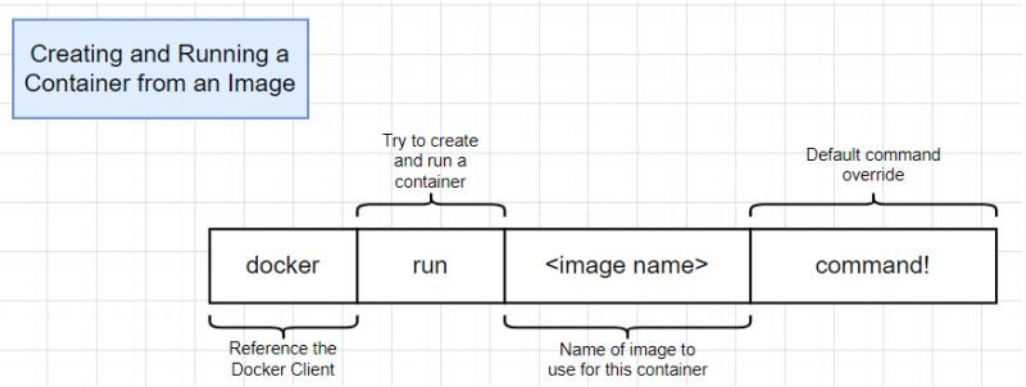
Manipulating Containers with Docker Client

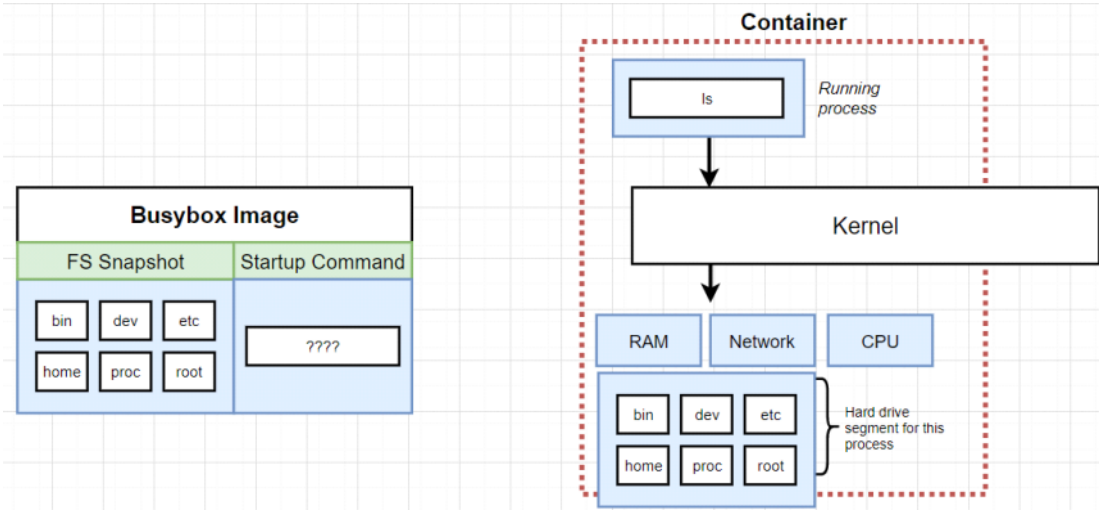
Monday, July 26, 2021 11:23 AM

Docker run

- 

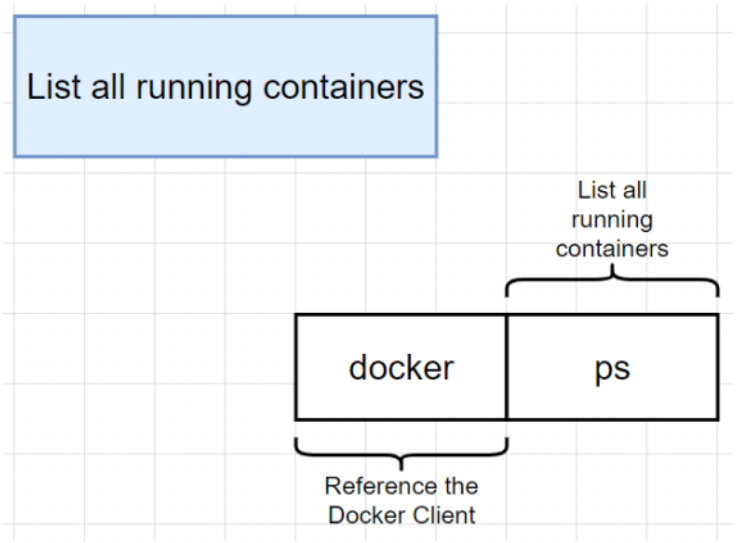
Overriding default commands

- 

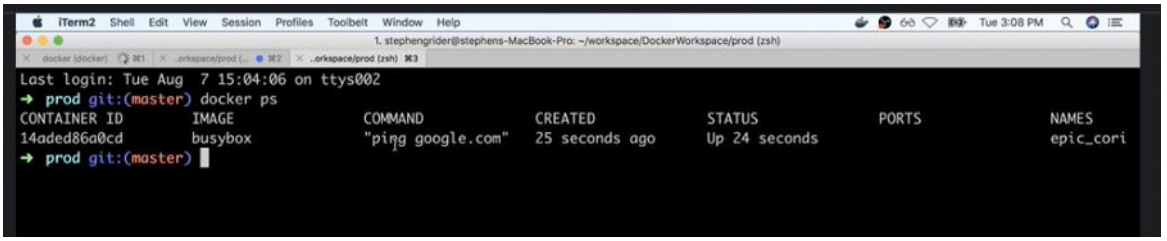
- *docker run busybox echo Hi There*
  - Displays message "Hi There"
- *docker run busybox ls*
  - Lists all the folders that exists in the container
  - 

Listing running containers

- *docker ps*
- Helps in getting the ids of the containers



- Ex.
  - *docker run busybox ping google.com*
  - *docker ps*
  - The above command will the list the containers that are running



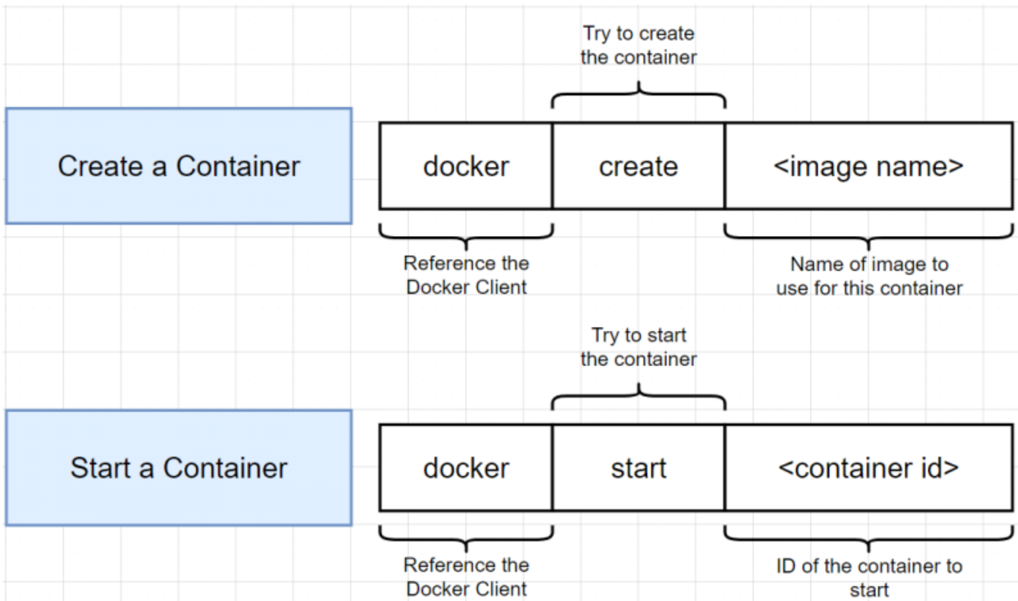
- Ex.
  - *docker ps*

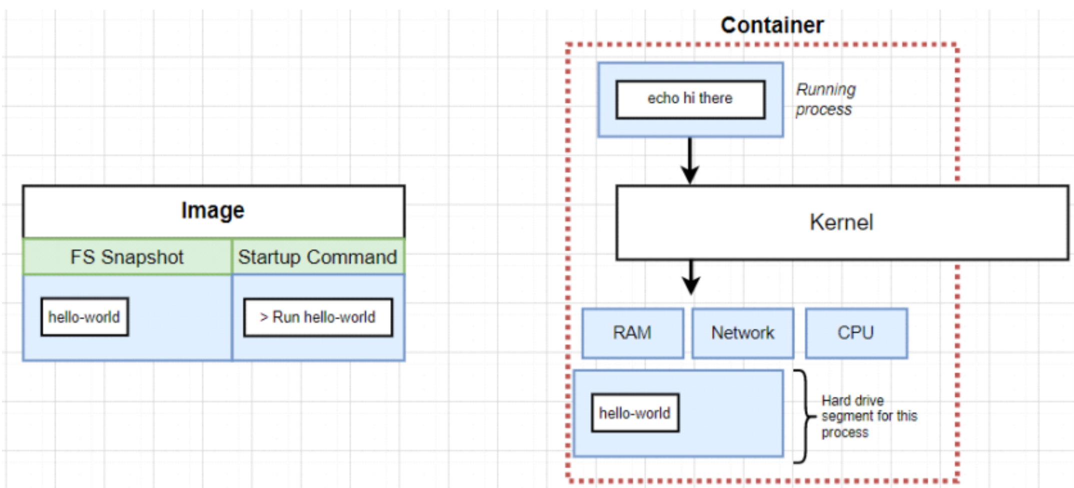
- To list all containers that were ever created in Docker
  - *docker ps --all*

- ```
  C:\Users\rayerra>docker ps --all
  CONTAINER ID   IMAGE                   COMMAND                   CREATED        STATUS                    PORTS
                                                                                                                                 NAMES
  f937542b058c   hello-world             "/hello"                  4 days ago     Exited (0) 4 days ago
                                                                                                                                 sweet_mo
  ser
  493da5b61094   hello-world             "/hello"                  4 days ago     Exited (0) 4 days ago
                                                                                                                                 hungry_k
  eldysh
  6a40da75174a   redis:4-alpine          "docker-entrypoint.s…"    2 weeks ago    Up 36 minutes             127.0.0.1:6380->6379/tcp
                                                                                                                                 cortex_r
  edis_1
  707202a04413   rabbitmq:3-management   "docker-entrypoint.s…"    3 weeks ago    Up About an hour          4369/tcp, 5671/tcp, 0.0.0.0:
  5672->5672/tcp, :::5672->5672/tcp, 15671/tcp, 15691-15692/tcp, 25672/tcp, 0.0.0.0:15672->15672/tcp, :::15672->15672/tcp    rabbitmq
  7e5d7b30ba8e   mongo                   "docker-entrypoint.s…"    3 weeks ago    Up About an hour          0.0.0.0:27017->27017/tcp, ::
  :27017->27017/tcp                                                                                                              mongodb
  ```

Container Lifecycle

- 

- creating a docker container and running it are two different processes.
- 

- The process of creating a container is taking out the file system from the image and setting it up in the container hard drive segment used for this process
- The process of starting a container is when we start running the process the container should execute
   - 
  ```

- Ex.
    - *docker create hello-world*
        - return the id of the container created
    - *docker start -a <container_id>*
        - *-a* -> used to watch for the output in the container and print it out on the screen(terminal)
        - starts the process to be executed in the container

Restarting stopped containers

- *docker start -a <container_id>*
- Note: We cannot replace the default command Ex. We cannot replace the ping google.com command which was used when the container was created to *"docker start -a 14aded86a0cd ping yahoo.com"*

Removing stopped containers

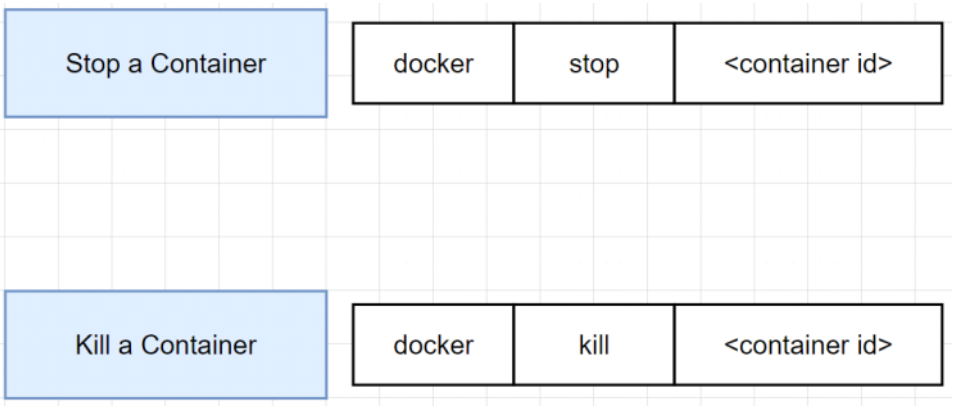
- *docker system prune*
- To delete all the stopped containers, all networks which are not used by atleast 1 container, all dangling images, all build cache (i.e all images in the image cache)

Retrieving log outputs

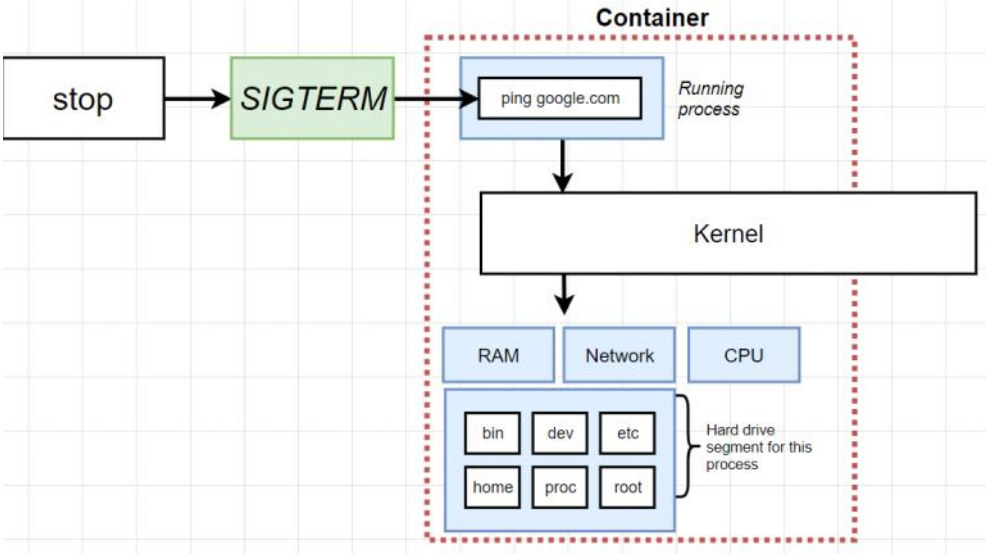
- Using docker logs does not re-start/re-run the container it only logs the messages logged in the container
- Ex.
    - *docker create busybox echo Hello Again!* // Return container id
    - *docker start <container_id>* // Starts the container but does not log any messages
    - *docker logs <container_id>* // Logs all the messages displayed by the container

Stopping Containers

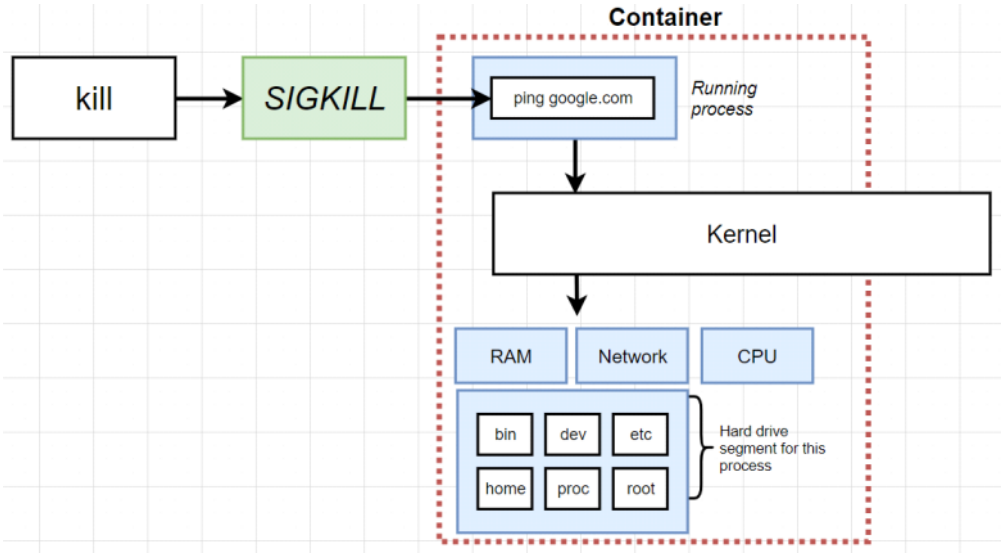
- *docker stop* is preferred over *docker kill*, since it has some time to do the clean-up, but if the *docker stop* does not stop in about 10seconds it issues the *docker kill* command.
- Ex.
    - *docker create busybox ping google.com* // pings google.com continuously until stopped
    - *docker start <container_id>* // Starts the container but does not log any messages
    - *docker logs <container_id>* // Logs all the messages displayed by the container
    - *docker stop <container_id>* / *docker kill <container_id>* // We can use any of them



- We can use *docker stop <container_id>* / *docker kill <container_id>*
- *docker stop <container_id>*
    - When we issue a stop command a hardware signal is sent to the primary process in the container.
    - We send a SIGTERM message which is short term for terminate signal.
    - SIGTERM is a message that will be received by the process in the container telling it to shutdown at its own time after a cleanup.

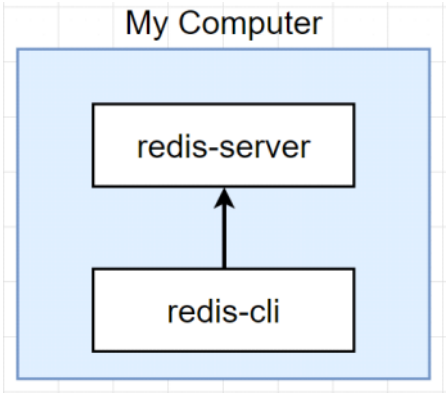


- *docker kill <container_id>*
  - When we issue a kill command SIGKILL is sent to the primary process in the container which tells it to stop the process right now.



Multi-command containers

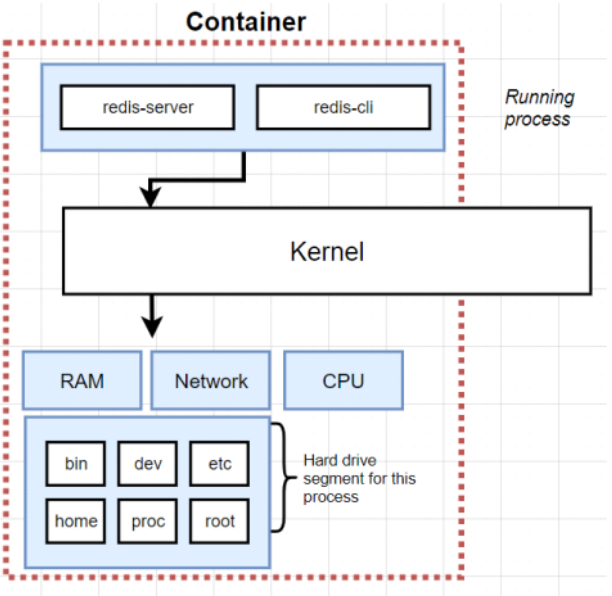
- Redis - In-memory data store
- Ex.
  - redis-server
  - redis-cli



- Use docker to startup redis-server and redis-cli
- *docker run redis*

```
C:\Users\rayerra>docker run redis
Unable to find image 'redis:latest' locally
latest: Pulling from library/redis
33847f680f63: Pull complete
26a746039521: Pull complete
18d87da94363: Pull complete
5e118a708802: Pull complete
ecf0dbe7c357: Pull complete
46f280ba52da: Pull complete
Digest: sha256:cd0c68c5479f2db4b9e2c5fbfdb7a8acb77625322dd5b474578515422d3ddb59
Status: Downloaded newer image for redis:latest
1:C 27 Jul 2021 12:53:02.540 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
1:C 27 Jul 2021 12:53:02.540 # Redis version=6.2.5, bits=64, commit=00000000, modified=0, pid=1, just started
1:C 27 Jul 2021 12:53:02.540 # Warning: no config file specified, using the default config. In order to specify a config
 file use redis-server /path/to/redis.conf
1:M 27 Jul 2021 12:53:02.541 * monotonic clock: POSIX clock_gettime
1:M 27 Jul 2021 12:53:02.543 * Running mode=standalone, port=6379.
1:M 27 Jul 2021 12:53:02.543 # Server initialized
1:M 27 Jul 2021 12:53:02.543 # WARNING overcommit_memory is set to 0! Background save may fail under low memory conditio
n. To fix this issue add 'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.ov
ercommit_memory=1' for this to take effect.
1:M 27 Jul 2021 12:53:02.543 * Ready to accept connections
```

- Now we have to run the redis-cli as well, and we need to figure out on how to run the related commands in the container where redis-server is running since any command executed outside the container will not reflect in the container



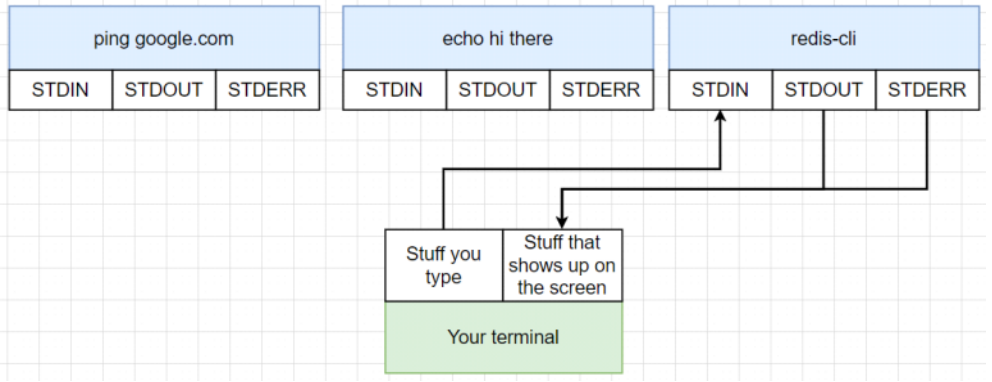
- *docker exec -it <container_id> redis-cli*

```
C:\Users\rayerra>docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS          PORTS

      NAMES
9c00ee1a420d   redis                  "docker-entrypoint.s…"   23 minutes ago   Up 23 minutes   6379/tcp

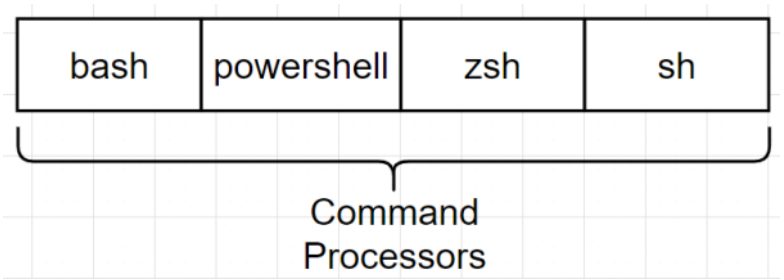
      vibrant_elion
6a40da75174a   redis:4-alpine         "docker-entrypoint.s…"   3 weeks ago      Up 32 hours     127.0.0.1:6380->6379/tc
p
      cortex_redis_1
707202a04413   rabbitmq:3-management  "docker-entrypoint.s…"   3 weeks ago      Up 32 hours     4369/tcp, 5671/tcp, 0.0
.0.0:5672->5672/tcp, :::5672->5672/tcp, 15671/tcp, 15691-15692/tcp, 25672/tcp, 0.0.0.0:15672->15672/tcp, :::15672->15672
/tcp   rabbitmq
7e5d7b30ba8e   mongo                  "docker-entrypoint.s…"   3 weeks ago      Up 32 hours     0.0.0.0:27017->27017/tc
p, :::27017->27017/tcp
      mongodb

C:\Users\rayerra>docker exec -it 9c00ee1a420d redis-cli
127.0.0.1:6379>
```

- *-it* flag
  - Every process we run in a container has 3 channels attached to it
    - STDIN - To communicate into the process
    - STDOUT - To convey information coming from the process
    - STDERR - To convey information which is error in nature
    - 

    - - it are 2 separate flags: -i and -t
      - docker exec -i -t
      - -i -> To redirect all text we type to STDIN
      - -t -> To show all text we type and information that needs is coming out in a formatted way

Docker Shell

- Below are the list of command processors
  - 

- Ex.
  - *docker ps* // to retrieve id of the container where the shell needs to be opened to run required commands
  - *docker exec -it <container_id> sh* // to open the shell for the full terminal access of the specific containers

- To start the docker shell
  - *docker run -it busybox sh* // To startup a container with the busybox image, run the sh(shell) program
    - Cons: Assumption with *docker run -it* is that you will not be running any other process
    - Note: Ctrl+D to exit

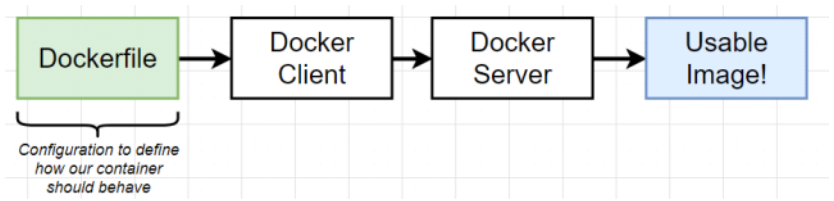
Container Isolation

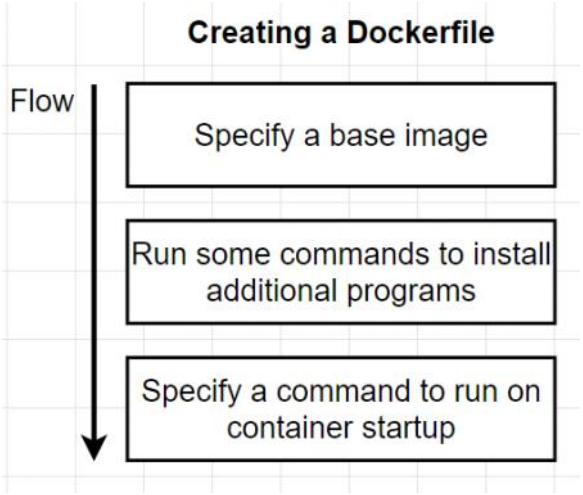
- Ex.
  - *docker run -it busybox sh* // Terminal 1
  - *docker run -it busybox sh* // Terminal 2
  - *docker ps* // To check the running containers
  - Both the containers running are independent of each other, data will not be shared between them

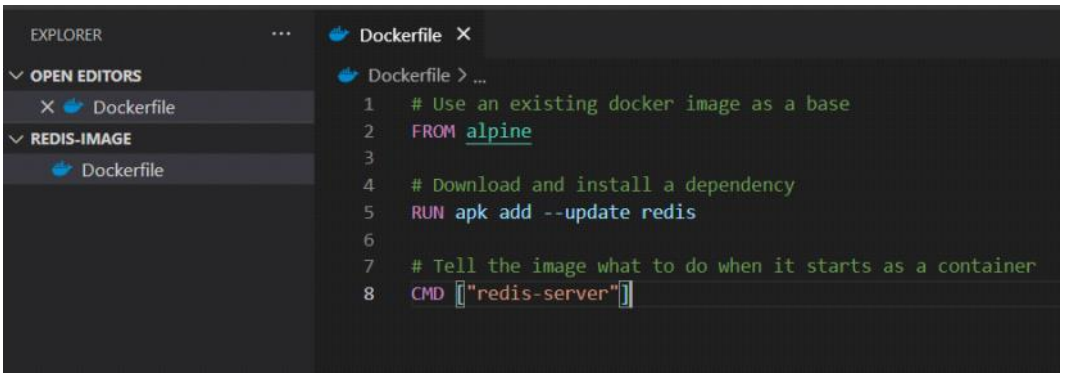
Building custom images through docker server

Wednesday, July 28, 2021 7:13 PM

Creating Docker images

- 

- Dockerfile
  - Contains the configuration that defines the container behavior.
  - 

  - Ex.
    - Create a image that runs redis-server using docker file
    - Create a folder "redis-image" and create Dockerfile as below
    - 

    - Run "*docker build* ." command in the command prompt "redis-image"

```
C:\Users\rayerra\Documents\Docker Workspace\redis-image>docker build .
[+] Building 17.3s (6/6) FINISHED
 => [internal] load build definition from Dockerfile                                                    0.2s
 => => transferring dockerfile: 242B                                                                    0.0s
 => [internal] load .dockerignore                                                                       0.2s
 => => transferring context: 2B                                                                         0.0s
 => [internal] load metadata for docker.io/library/alpine:latest                                       11.3s
 => [1/2] FROM docker.io/library/alpine@sha256:234cb88d3020898631af0ccbbcca9a66ae7306ecd30c9720690858c1b007d2a0    2.3s
 => => resolve docker.io/library/alpine@sha256:234cb88d3020898631af0ccbbcca9a66ae7306ecd30c9720690858c1b007d2a0    0.0s
 => => sha256:234cb88d3020898631af0ccbbcca9a66ae7306ecd30c9720690858c1b007d2a0 1.64kB / 1.64kB          0.0s
 => => sha256:1775bebec23e1f3ce486989bfc9ff3c4e951690df84aa9f926497d82f2ffca9d 528B / 528B              0.0s
 => => sha256:d4ff818577bc193b309b355b02ebc9220427090057b54a59e73b79bdfe139b83 1.47kB / 1.47kB          0.0s
 => => sha256:5843afab387455b37944e709ee8c78d7520df80f8d01cf7f861aae63beeddb6b 2.81MB / 2.81MB          1.5s
 => => extracting sha256:5843afab387455b37944e709ee8c78d7520df80f8d01cf7f861aae63beeddb6b               0.7s
 => [2/2] RUN apk add --update redis                                                                    3.1s
 => exporting to image                                                                                  0.1s
 => => exporting layers                                                                                 0.1s
 => => writing image sha256:995ba196fd71ff7efce8cf8017c59e3f0ce65f0753919344d3442826bfb41f97            0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them

C:\Users\rayerra\Documents\Docker Workspace\redis-image>
```

- *docker run <container_id>* // To start the redis-server

```
C:\Users\rayerra\Documents\Docker Workspace\redis-image>docker run 995ba196fd71ff7efce8cf8017c59e3f0ce65f0753919344d3442
826bfb41f97
1:C 28 Jul 2021 14:10:06.253 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
1:C 28 Jul 2021 14:10:06.253 # Redis version=6.2.5, bits=64, commit=af329dfc, modified=0, pid=1, just started
1:C 28 Jul 2021 14:10:06.253 # Warning: no config file specified, using the default config. In order to specify a config
 file use redis-server /path/to/redis.conf
1:M 28 Jul 2021 14:10:06.254 * monotonic clock: POSIX clock_gettime
1:M 28 Jul 2021 14:10:06.256 * Running mode=standalone, port=6379.
1:M 28 Jul 2021 14:10:06.256 # Server initialized
1:M 28 Jul 2021 14:10:06.256 # WARNING overcommit_memory is set to 0! Background save may fail under low memory conditio
n. To fix this issue add 'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.ov
ercommit_memory=1' for this to take effect.
1:M 28 Jul 2021 14:10:06.256 * Ready to accept connections
```