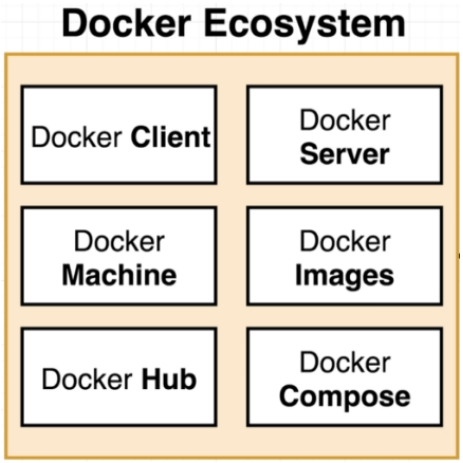
Docker introduction:

What is Docker?

Docker is an open source platform for developing, shipping, and running applications
or we can say docker is a platform or ecosystem about creating and running containers.



Why do we use docker?

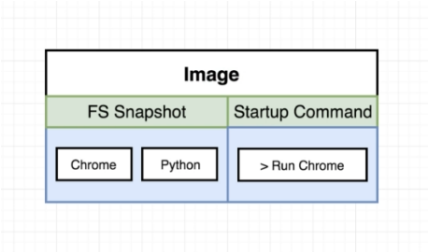
So the main reason behind using docker is that it makes it very easy to run any software by just running a single command without worrying about installing any additional components.

Components of Docker :

- Docker Container
- Docker Images
- Docker Client
- Docker Server
- Docker Engine
- Docker daemon

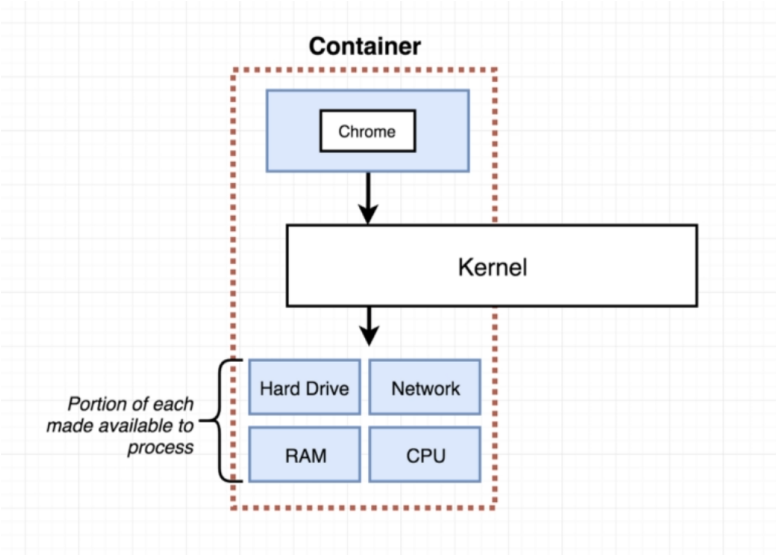
Image:

It is a collection of resources, deployments and configurations within a single file required to run a specific program.



Container:

Enables you to run a program with the help of images. It is a set of resources that enables users to use and run a program.



Docker Client:

It is a way to interact with docker. It allows you to perform operations like create, run and stop applications to a docker daemon. It sends the user requests to the docker server.

Docker daemon:

It is a background process to manage docker images, volumes, containers etc. It listens for the API requests and processes them.

Docker server:

It proceeds the requests of Docker clients and actually performs the operations.

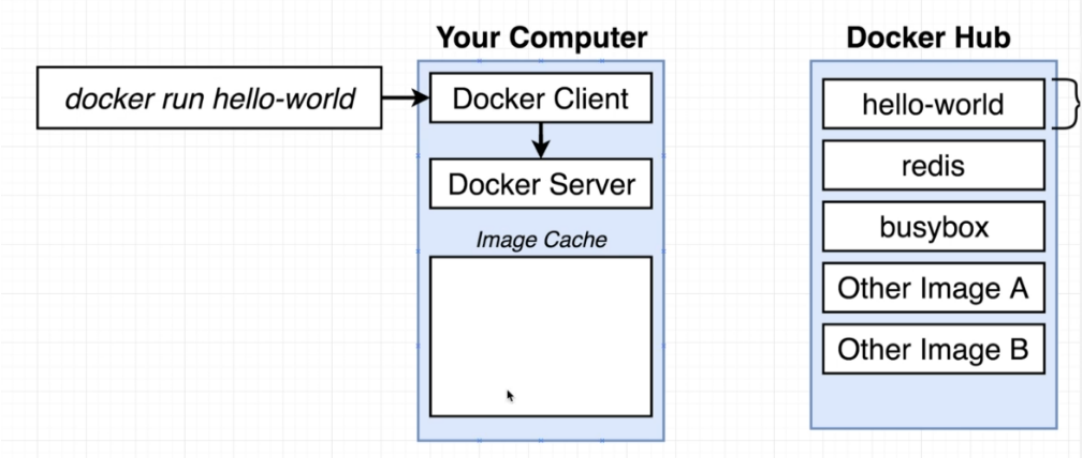
How to install Docker :

https://docs.docker.com/engine/install/

Basic docker command :

~ ***docker run hello-world***

So the above command is a basic command in Docker that will be executed in bunch of steps as follows:

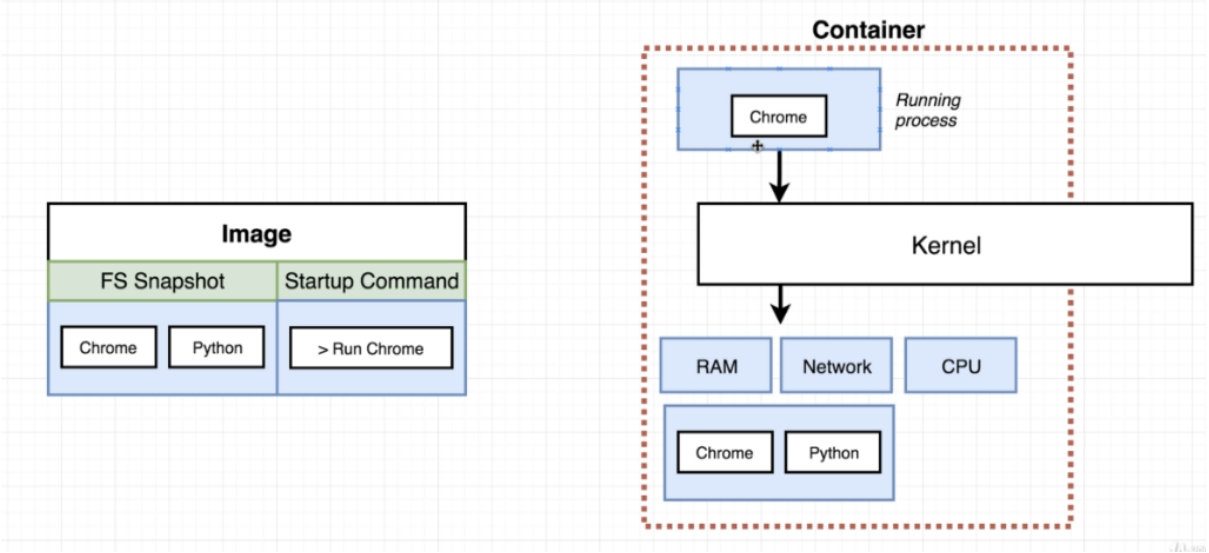


So in the above process we run a command docker run hello-world.
Now this command goes to the docker client and requests to pull the image hello-world. Then the client sends the request to the docker server where the server will check for the image in the memory of your computer. If server got the images of hello-world it will return to the command and run it but if the server is unable to find it locally on the system the it will go to a place i.e. Docker Hub to search for the same image and pull it from Docker Hub and then return to the main command and run it.
It should also be noted that after running this command if you try to run it again then it is not going to pull the image from docker hub rather the image will be available on your system locally.

Docker hub:

It is the Hub of all the Docker images to pull and run them.

How does a container actually work?

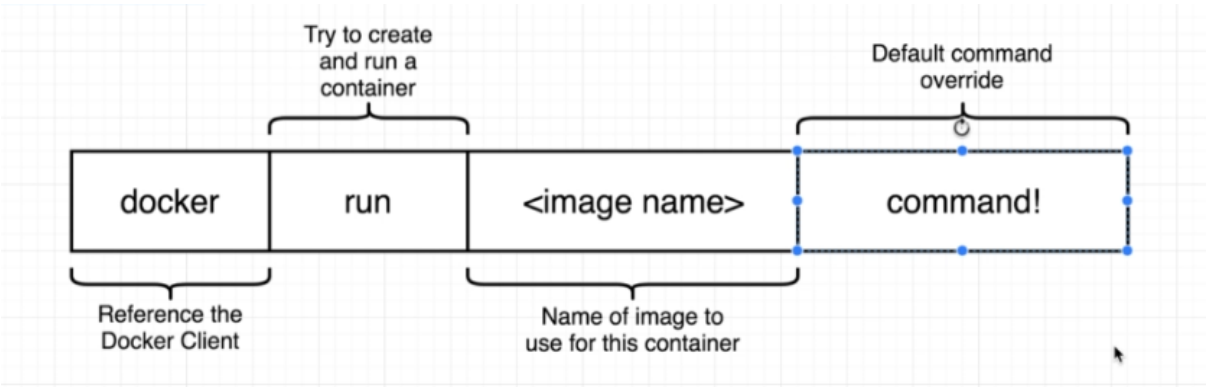


Look at the above diagram carefully
Here you notice that in image section the image is containing two portione i.e.

1.) **FS snapshot :** File System snapshot to create a snapshot in container.
2.) **Startup command :** to run a program.

Now when you try to pull any image and run it, then in the container section a snapshot of the image will be created as shown and along that the startup command will execute to run the mentioned image.

Docker basic command layout:



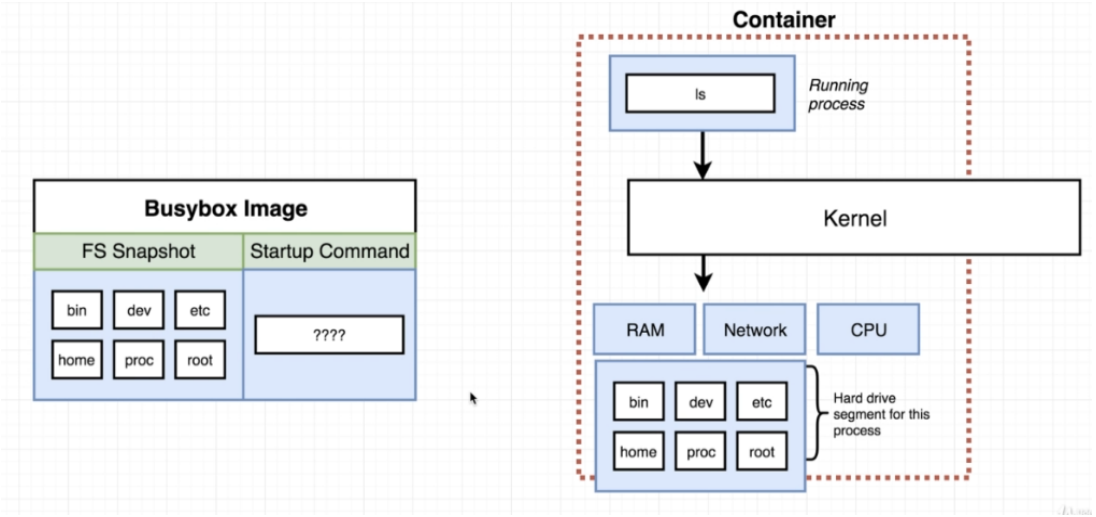
Docker basic commands:

```
➜ prod git:(master) docker run busybox ls
bin
dev
etc
home
proc
root
sys
tmp
usr
var
```

If we add a command after the image then the default commands in that image are not going to run, rather the command after the image will be executed as you can see in the above img.

Similarly,

```
➜ prod git:(master) docker run busybox echo hi there
hi there
➜ prod git:(master) docker run busybox echo bye there
bye there
➜ prod git:(master) docker run busybox echo how are you
how are you
```



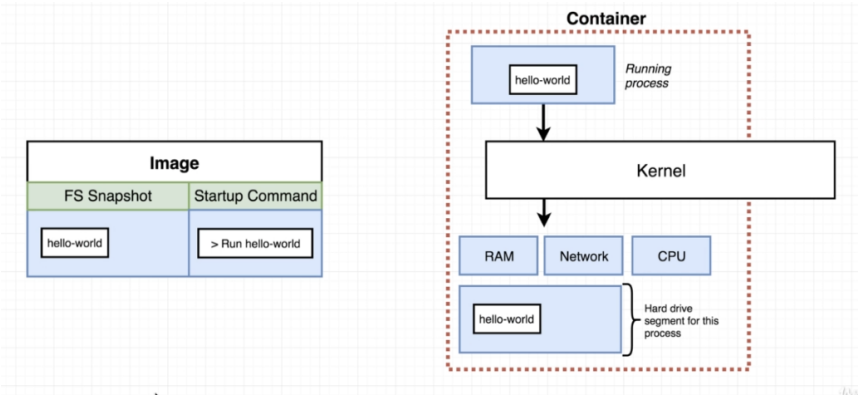
From this you can get a better idea, just focus on the running process command in the container i.e. ls command.

But,

NOTE: Only those commands can be executed after the image which have their source in that image FS snapshot.

```
➜ prod git:(master) docker run hello-world ls
docker: Error response from daemon: OCI runtime create failed: container_li
nux.go:348: starting container process caused "exec: \"ls\": executable fil
e not found in $PATH": unknown.
➜ prod git:(master) docker run hello-world echo hi there
docker: Error response from daemon: OCI runtime create failed: container_li
nux.go:348: starting container process caused "exec: \"echo\": executable f
ile not found in $PATH": unknown.
➜ prod git:(master)
```

Here you got an error because the FS snapshot of the hello-world image doesn't include ls or echo commands in it.



Here the hello-world image only contains a single file system.

Docker commands:

docker ps: It will list out the running containers.
docker ps –a : it will list out all the containers either running or not.
docker ps - -all : it will list out the history of the container ever created by the user i.e. which container was run at which time and at what time it stopped and all.

If you want to see a container running you can simply run a code as:

```
➜ prod git:(master) docker run busybox ping google.com
PING google.com (172.217.11.238): 56 data bytes
64 bytes from 172.217.11.238: seq=0 ttl=37 time=3.653 ms
64 bytes from 172.217.11.238: seq=1 ttl=37 time=2.854 ms
64 bytes from 172.217.11.238: seq=2 ttl=37 time=3.228 ms
64 bytes from 172.217.11.238: seq=3 ttl=37 time=2.877 ms
```

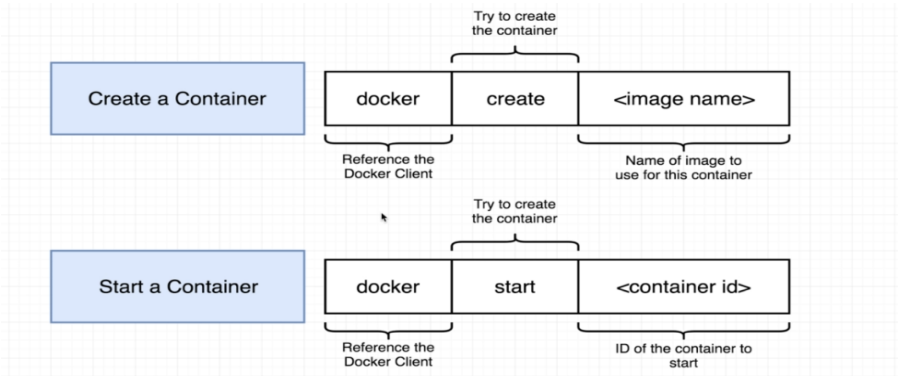
This command is going to run continuously until you terminate it manually. So without terminating it open a new tab in terminal and type: docker ps
And it will list out the running container i.e.

```
➜ prod git:(master) docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
14aded86a0cd        busybox             "ping google.com"   25 seconds ago      Up 24 seconds                           epic_cori
➜ prod git:(master)
```

And now to terminate the previous command just go to the tab onto which that command is running and press ctrl+C. And your running container will be terminated.

Docker run vs docker create vs docker start:

docker run = docker create + docker start



Let's take an example to make it clear:
First let's say you want to create a docker image to run a container:
~ ***docker create <image>***
It will give you a container id:

```
➜ prod git:(master) docker create hello-world
9271d26374ff54f715ce137ab71b5d86a7e9af3638b51d0a470fd8765c55821e
```

Then if you want to run the container using this image so you will type:
~ ***docker start -a <container id>***

```
➜ prod git:(master) docker start -a 9271d26374ff54f715ce137ab71b5d86a7e9af3638b51d0
a470fd8765c55821e

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.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash
```

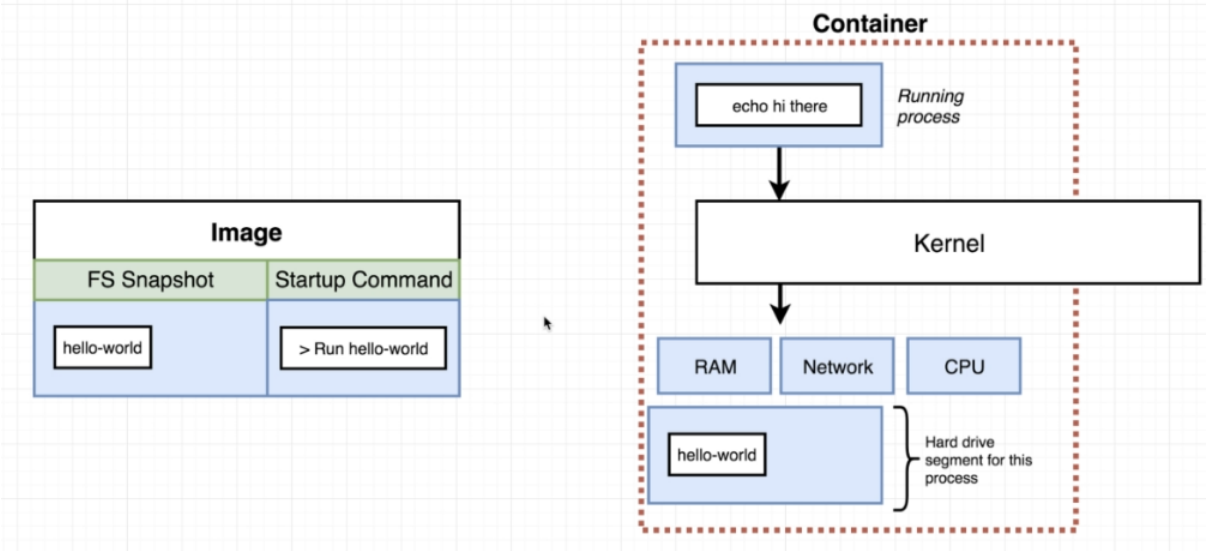
But if you want to do the two steps in just a single cmd you have to type :

~ ***docker run <image>***

```
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pulling fs layer
1b930d010525: Verifying Checksum
1b930d010525: Download complete
1b930d010525: Pull complete
Digest: sha256:9572f7cdcee8591948c2963463447a53466950b3fc15a247fcad1917ca215a2f
Status: Downloaded newer image for hello-world:latest

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.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

So, it will first pull the image and then run it using a single cmd.

- docker create = copying FS snapshot of image to the container.
- docker start = initialising the running process by passing setup command of image to container.
- docker run = copying FS snapshot + running the container.



Removing stopped containers:

If you want to remove stopped containers in your system just type:
~ ***docker system prune***

```
➜  prod git:(master) docker system prune
WARNING! This will remove:
        - all stopped containers
        - all networks not used by at least one container
        - all dangling images
        - all build cache
Are you sure you want to continue? [y/N] y
Deleted Containers:
37a8c45a23e15afd71684ad0c8e26e328661c5dc0855609bbd66b861b0fb0d8e
8849c62dd8d512ab0485b174db5f4ca13f3fcc03fbe1de35324e77b494b6ed69

Total reclaimed space: 0B
```

And here you will get a warning, just type y and you are good to go.

Docker logs :
You can get the output of a previously run container by using logs and it is pretty much simple jus type :
~ ***docker logs <container id>***

Stopping running container in docker :
If you want to stop a running container just type:
~ ***docker stop <container-id>***
Of if the container is not terminating form above command you can use kill command:
~ ***docker kill <container-id>***

Container command outside container:

If you want to run an container command outside the container then you have to use :
~ ***docker exec -it <container-id> <command>***

```
➜ prod git:(master) ✗ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
          STATUS              PORTS               NAMES
093b6e72ff2b        redis               "docker-entrypoint.s…"   3 minutes
ago       Up 3 minutes        6379/tcp            jovial_leakey
➜ prod git:(master) ✗ docker exec -it 093b6e72ff2b redis-cli
127.0.0.1:6379>
```

To exit from this new terminal type : ***exit or ctrl+C***

Use of -it : Here -it is used to provide an interactive terminal for the running container.

Command prompt in a container:

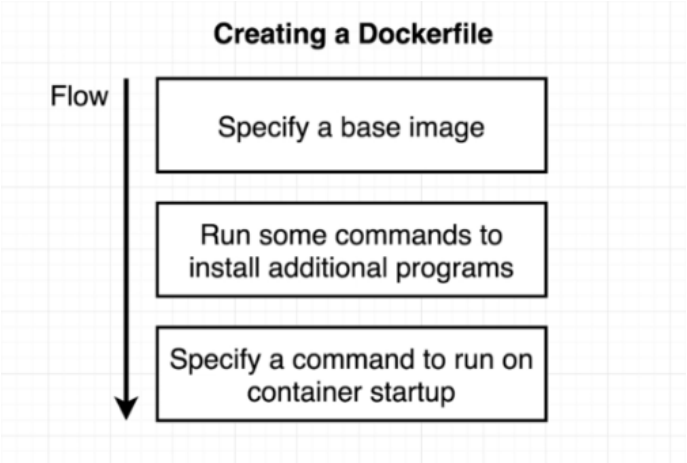
To open a shell in docker use :
~ ***docker exec -it <container-id> sh***

It will open a shell command prompt.

```
➜ prod git:(master) docker exec -it 4e3d15293585 sh
# cd ~/
```

Creating Docker images using Dockerfile :

Dockerfile: structured format that defines how the container should behave.



Basic Dockerfile:

```
# use an existing docker image as a base
FROM alpine

# Download and install a dependency
RUN apk add --update redis

# Tell the image what to do when it
# starts as a container
CMD ["redis-server"]
```

Base Image:

A base image is something that gives us an initial starting point for further process.

Why do we use alpine?

So Apline is a set of programs that are suitable to fulfil our needs of running that particular image.

RUN command:

Here the run command is nothing to do with docker rather it is used to install/update packages used for the images. Here akp stands for **Alpine Linux package keeper.**

CMD command:

Command that will run after the execution of Dockerfile.

Docker build process:

Now to build a container from the above dockerfile we just need to run a simple command :
~ docker build .
Here don't forget to put a dot(.) at the end of the command.

After running above command you will get an output like this:

```
(1/1) Installing redis (4.0.10-r1)
Executing redis-4.0.10-r1.pre-install
Executing redis-4.0.10-r1.post-install
Executing busybox-1.28.4-r0.trigger
OK: 6 MiB in 14 packages
Removing intermediate container 30c5aa616f98
 ---> 38ec9aea7e10
Step 3/3 : CMD ["redis-server"]
 ---> Running in e73a22decb35
Removing intermediate container e73a22decb35
 ---> fc60771eaa08
Successfully built fc60771eaa08
```

Here note the container id at the last line of output. This one is the container id of the container that we have built using the above dockerfile.

Now to run this container just type:
~ ***docker run <container id>***

```
➜  redis-image git:(master) ✗ docker run fc60771eaa08
1:C 08 Aug 17:58:36.873 # oO0OoO0OoO0Oo Redis is starting o
0Oo
```

Building a small Dockerfile to print hello-world:

1.) Make the Docker file using the same syntax as mentioned above:

```
# use an existing docker image as a base
FROM alpine

# Tell the image what to do when it
# starts as a container
CMD ["echo","hello-world"]
```

2.) Now run build the image using :
~ docker build .

```
 ---> Using cache
 ---> 6f335cbc4604
Successfully built 6f335cbc4604
```

3.) Now run the container using the above image:

```
> docker run 6f335cbc4604
hi there
```

Specifying the name of image while building it:

If you want to add a name to your image during building it, just type :
~ ***docker build <image-name> .***
