Get to Docker

Session 1

Agenda

- Docker overview
- Bash and basic commands
- Docker commands
- Docker run

Virtual Machines

- A VM is a software that emulates an entire computer, with its kernel, architecture and resurces.

Containers

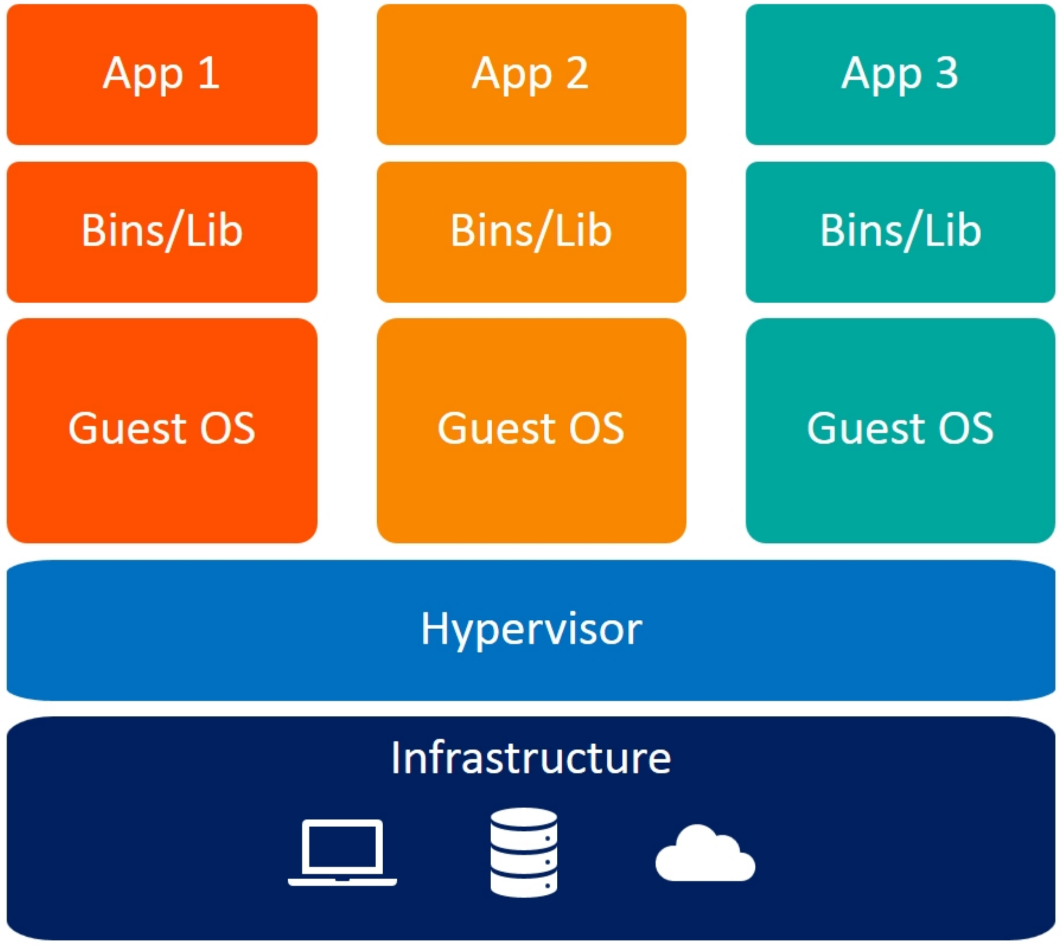
- Containers are virtual environments that keep applications and software isolated from the host system.
- Containers have been around for a long time but building them manually can be challenging, this is where Docker comes in.
- Docker uses existing container engines to provide consistent containers.

VM vs Containers

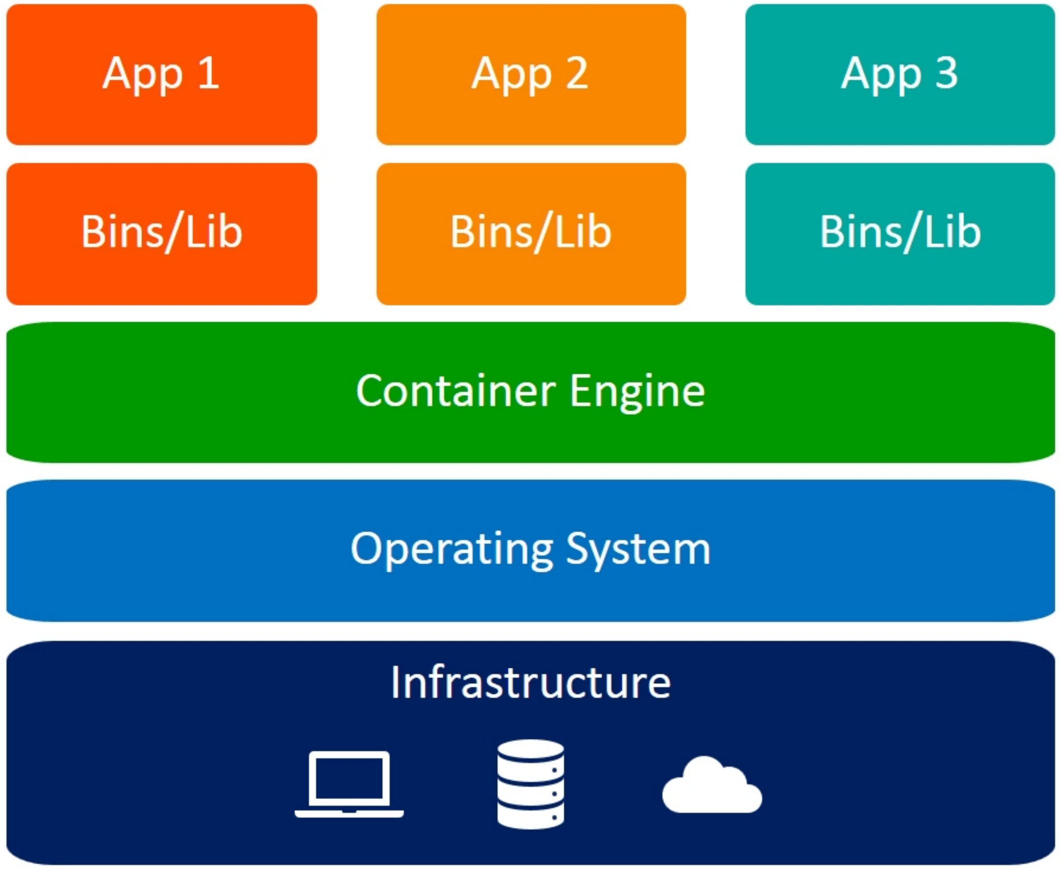
There is no "which is better" argument here as each one has its own use cases and benefits and disadvantges

But some differences to note:

- VMs are larger in size and resource consumption
- Containers require a Linux kernel to operate while VMs can be run on any OS
- Containers are usually faster and use resource needed only



Virtual Machines



Containers

Why Docker?

What is Docker?

- Docker is a command-line program, a background daemon and a set of services that helps in solving common software problems.
- It accomplishes this using containers.

WSL

- Windows Subsystem for Linux (WSL) allows you to run a Linux environment in your Windows machine.
- WSL's terminal is the same as a Linux terminal, so we need to look through some basic commands that will help us navigate through it

Bash

Bash is a shell and a command language, it's the most common shell to be found on Linux machines and WSL uses it too, let's look through some commands that is a must know in Bash.

- `cd` stands for Change Directory, use it to go to another directory (folder)
- `ls` stands for List, use it to list all files and directories in your working directory
- `touch` creates a file
- `cp` stands for Copy
- `mv` stands for Move, same thing as Cut (Ctrl + X)
- `rm` stands for Remove, deletes files
- `mkdir` stands for Make Directory, use it to create new directories (folders)
- `cat` prints contents of a file

Basic Docker commands

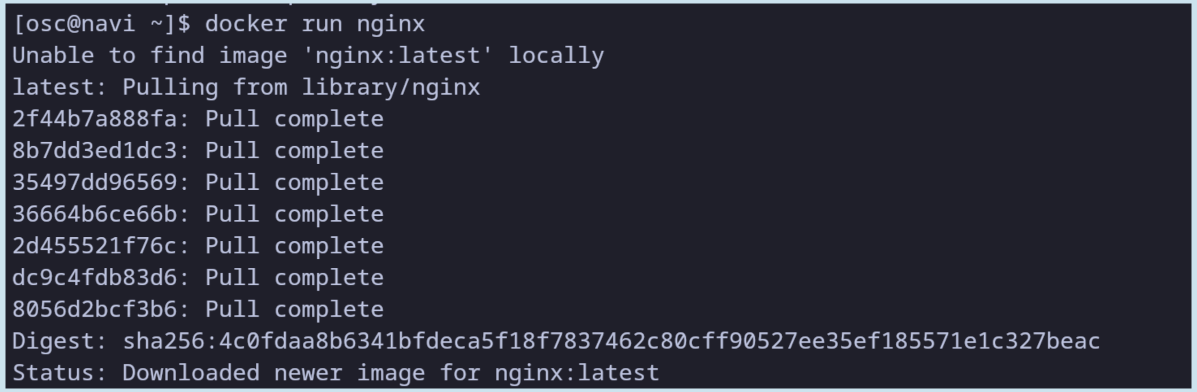
- Docker offers lots of functionalties and has lots of commands for specific uses, we will look now at the most common commands that are used with Docker.
- Before using any of these commands we must write `docker` before it.

Images

An image is a packaged application that contains all dependencies, source code and complete environment and configurations of that application, think of it as a blueprint that you use to create instances of the application, these instances are containers.

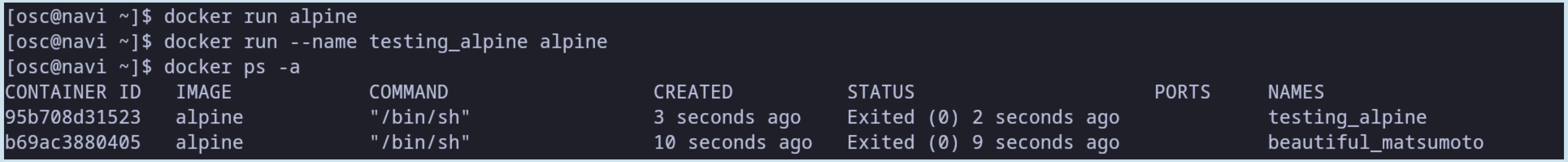
`run` start a container

- Syntax: `docker run <image_name`
- If the image is downloaded it will start, otherwise it is pulled and downloaded on your machine first.



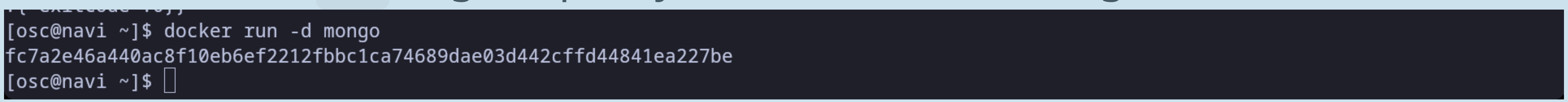
```
[osc@navi ~]$ docker run nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
2f44b7a888fa: Pull complete
8b7dd3ed1dc3: Pull complete
35497dd96569: Pull complete
36664b6ce66b: Pull complete
2d455521f76c: Pull complete
dc9c4fdb83d6: Pull complete
8056d2bcf3b6: Pull complete
Digest: sha256:4c0fdaa8b6341bfdeca5f18f7837462c80cff90527ee35ef185571e1c327beac
Status: Downloaded newer image for nginx:latest
```

- Docker gives containers random names if we don't specify one
- We can give our containers custom names by adding the `--name` to the `run` command



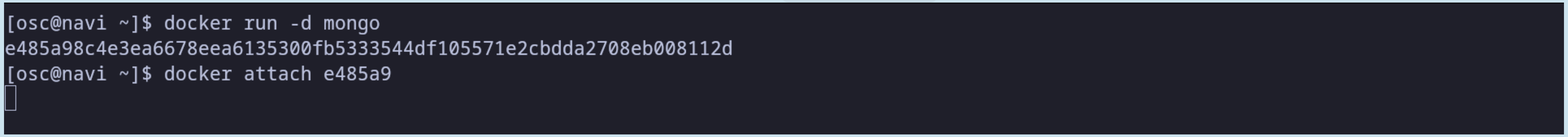
```
[osc@navi ~]$ docker run alpine
[osc@navi ~]$ docker run --name testing_alpine alpine
[osc@navi ~]$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS                      PORTS     NAMES
95b708d31523   alpine         "/bin/sh"                3 seconds ago    Exited (0) 2 seconds ago              testing_alpine
b69ac3880405   alpine         "/bin/sh"                10 seconds ago   Exited (0) 9 seconds ago              beautiful_matsumoto
```

- When we run a container it is run in attached mode, meaning it's attached to the terminal and we can't use the terminal until it exits.
- We can use the `-d` flag to specify the container running in detached mode.



```
[osc@navi ~]$ docker run -d mongo
fc7a2e46a440ac8f10eb6ef2212fbbc1ca74689dae03d442cffd44841ea227be
[osc@navi ~]$ 
```

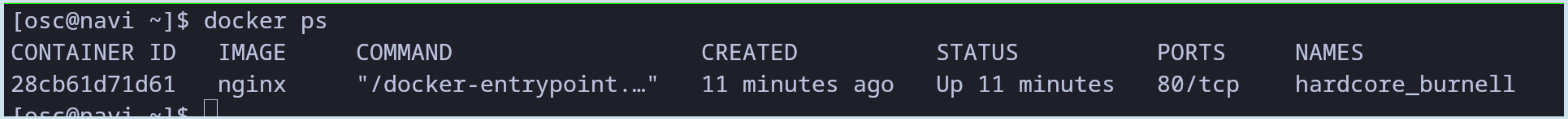
- To attach a detached container we use `attach` command



```
[osc@navi ~]$ docker run -d mongo
e485a98c4e3ea6678eea6135300fb5333544df105571e2cbdda2708eb008112d
[osc@navi ~]$ docker attach e485a9
```

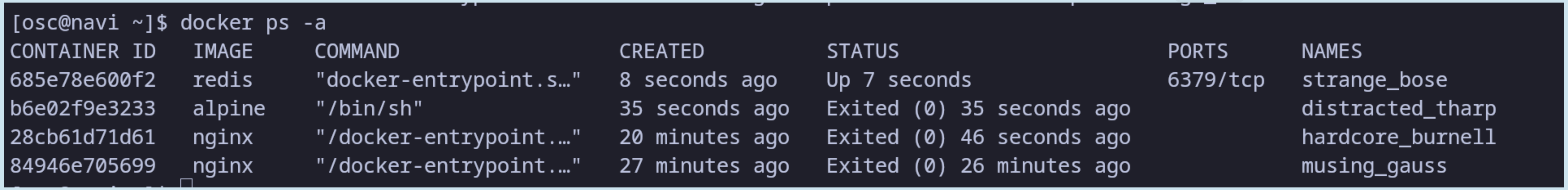
`ps` lists containers

- Syntax: `docker ps`
- Lists all running containers, and some basic info about them (image, id, name, etc…)



```
[osc@navi ~]$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
28cb61d71d61   nginx     "/docker-entrypoint.…"   11 minutes ago   Up 11 minutes   80/tcp    hardcore_burnell
[osc@navi ~]$
```

- If we want to list all containers, running or not, we use the `-a` option.

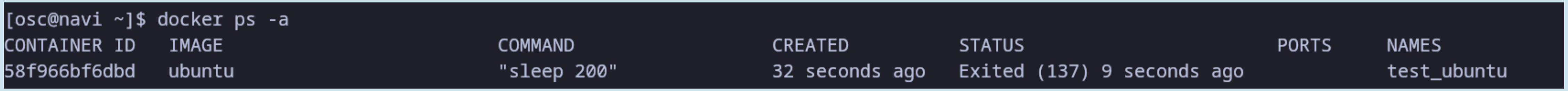


```
[osc@navi ~]$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS                      PORTS      NAMES
685e78e600f2   redis     "docker-entrypoint.s…"   8 seconds ago    Up 7 seconds                6379/tcp   strange_bose
b6e02f9e3233   alpine    "/bin/sh"                35 seconds ago   Exited (0) 35 seconds ago              distracted_tharp
28cb61d71d61   nginx     "/docker-entrypoint.…"   20 minutes ago   Exited (0) 46 seconds ago              hardcore_burnell
84946e705699   nginx     "/docker-entrypoint.…"   27 minutes ago   Exited (0) 26 minutes ago              musing_gauss
```

`start` starts containers

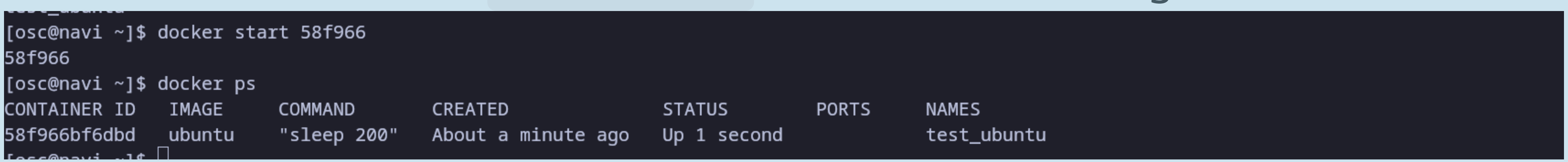
Syntax:

- `docker start <container_name>`
- `docker start <container_ID>`

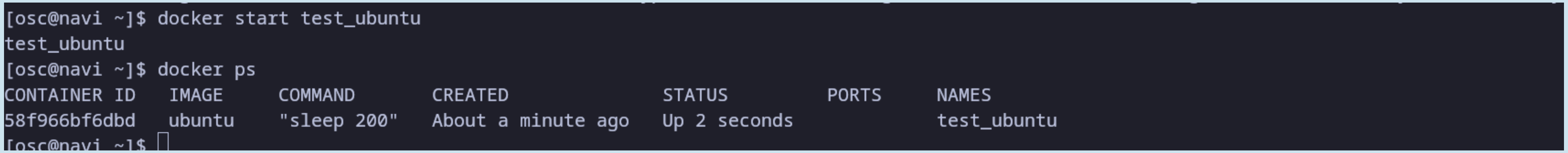


```
[osc@navi ~]$ docker ps -a
CONTAINER ID   IMAGE                          COMMAND                  CREATED          STATUS                          PORTS     NAMES
58f966bf6dbd   ubuntu                         "sleep 200"              32 seconds ago   Exited (137) 9 seconds ago                test_ubuntu
```

- Here we have container `test_ubuntu` which is not running



```
[osc@navi ~]$ docker start 58f966
58f966
[osc@navi ~]$ docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED              STATUS         PORTS     NAMES
58f966bf6dbd   ubuntu    "sleep 200"   About a minute ago   Up 1 second              test_ubuntu
[osc@navi ~]$ 
```



```
[osc@navi ~]$ docker start test_ubuntu
test_ubuntu
[osc@navi ~]$ docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED              STATUS          PORTS     NAMES
58f966bf6dbd   ubuntu    "sleep 200"   About a minute ago   Up 2 seconds              test_ubuntu
[osc@navi ~]$ 
```

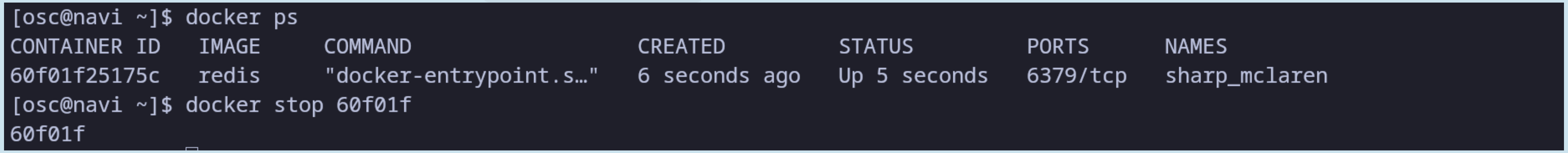
- Using the name or the ID we can start it.

`stop` stops containers

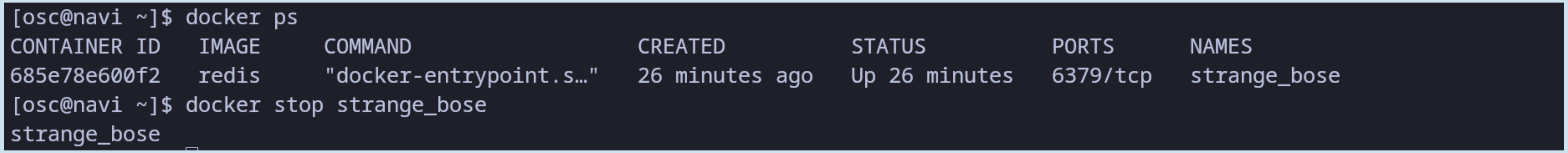
Syntax:

- `docker stop <container_name>`
- `docker stop <container_ID>`

- We need to pass the container ID or container name to stop a container, we can find those from the `docekr ps` command.



```
[osc@navi ~]$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS      NAMES
60f01f25175c   redis     "docker-entrypoint.s…"   6 seconds ago   Up 5 seconds   6379/tcp   sharp_mclaren
[osc@navi ~]$ docker stop 60f01f
60f01f
```

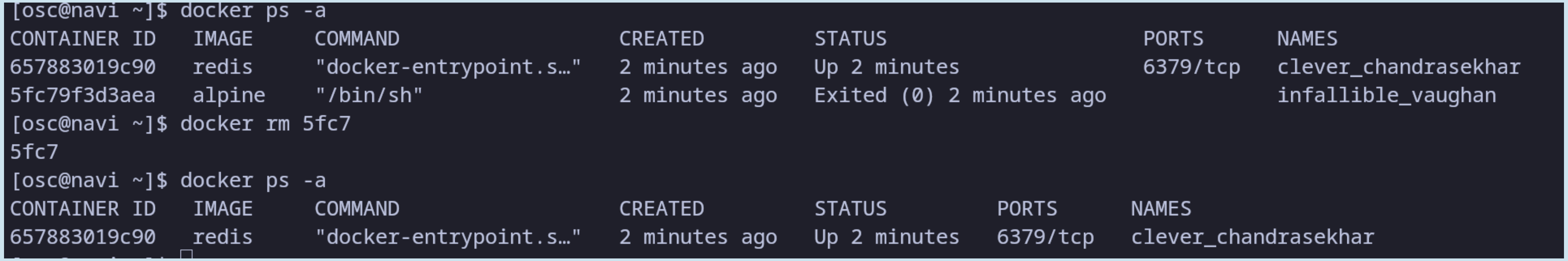


```
[osc@navi ~]$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS      NAMES
685e78e600f2   redis     "docker-entrypoint.s…"   26 minutes ago   Up 26 minutes   6379/tcp   strange_bose
[osc@navi ~]$ docker stop strange_bose
strange_bose
```

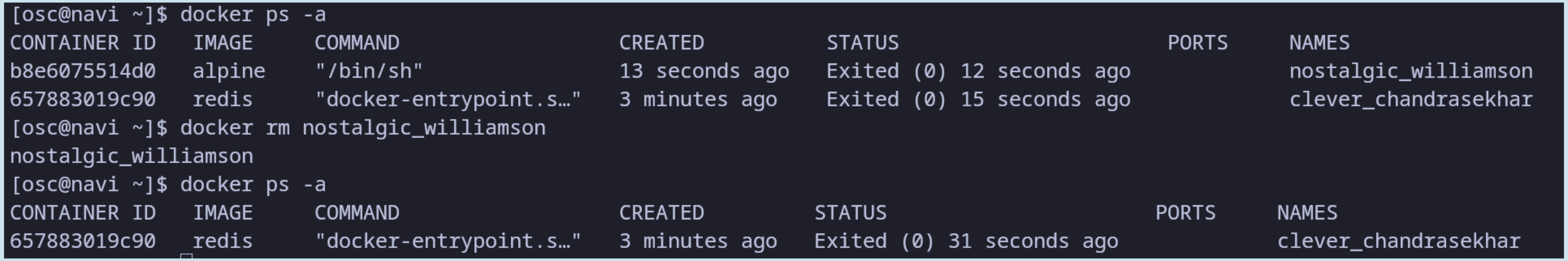
- After running `docker stop` it will print out the name or the ID that we passed.

`rm` removes containers

- Like `stop` we pass either the name or ID of the container, if removed succesfully, it will print out what we passed to it.

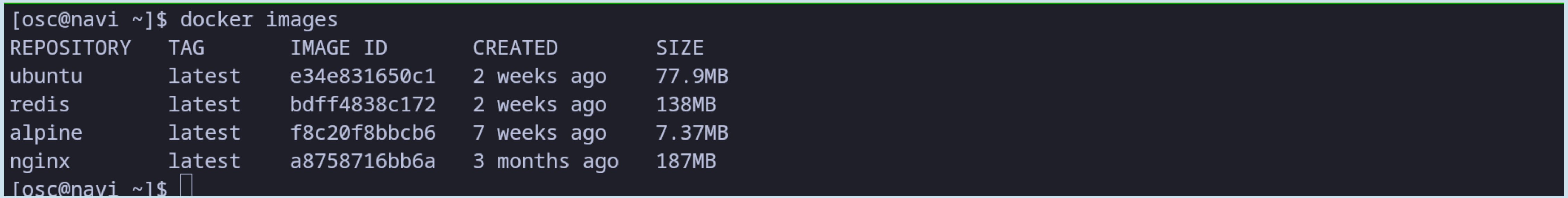


```
[osc@navi ~]$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS                          PORTS      NAMES
657883019c90   redis     "docker-entrypoint.s…"   2 minutes ago   Up 2 minutes                    6379/tcp   clever_chandrasekhar
5fc79f3d3aea   alpine    "/bin/sh"                2 minutes ago   Exited (0) 2 minutes ago                   infallible_vaughan
[osc@navi ~]$ docker rm 5fc7
5fc7
[osc@navi ~]$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS      NAMES
657883019c90   redis     "docker-entrypoint.s…"   2 minutes ago   Up 2 minutes   6379/tcp   clever_chandrasekhar
```



```
[osc@navi ~]$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS                       PORTS     NAMES
b8e6075514d0   alpine    "/bin/sh"                13 seconds ago   Exited (0) 12 seconds ago              nostalgic_williamson
657883019c90   redis     "docker-entrypoint.s…"   3 minutes ago    Exited (0) 15 seconds ago              clever_chandrasekhar
[osc@navi ~]$ docker rm nostalgic_williamson
nostalgic_williamson
[osc@navi ~]$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS                       PORTS     NAMES
657883019c90   redis     "docker-entrypoint.s…"   3 minutes ago   Exited (0) 31 seconds ago              clever_chandrasekhar
```

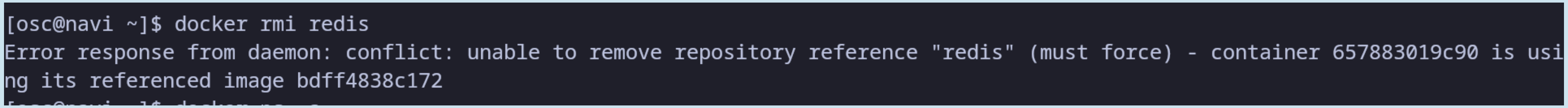
`images` lists images



```
[osc@navi ~]$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
ubuntu       latest    e34e831650c1   2 weeks ago    77.9MB
redis        latest    bdff4838c172   2 weeks ago    138MB
alpine       latest    f8c20f8bbcb6   7 weeks ago    7.37MB
nginx        latest    a8758716bb6a   3 months ago   187MB
[osc@navi ~]$
```

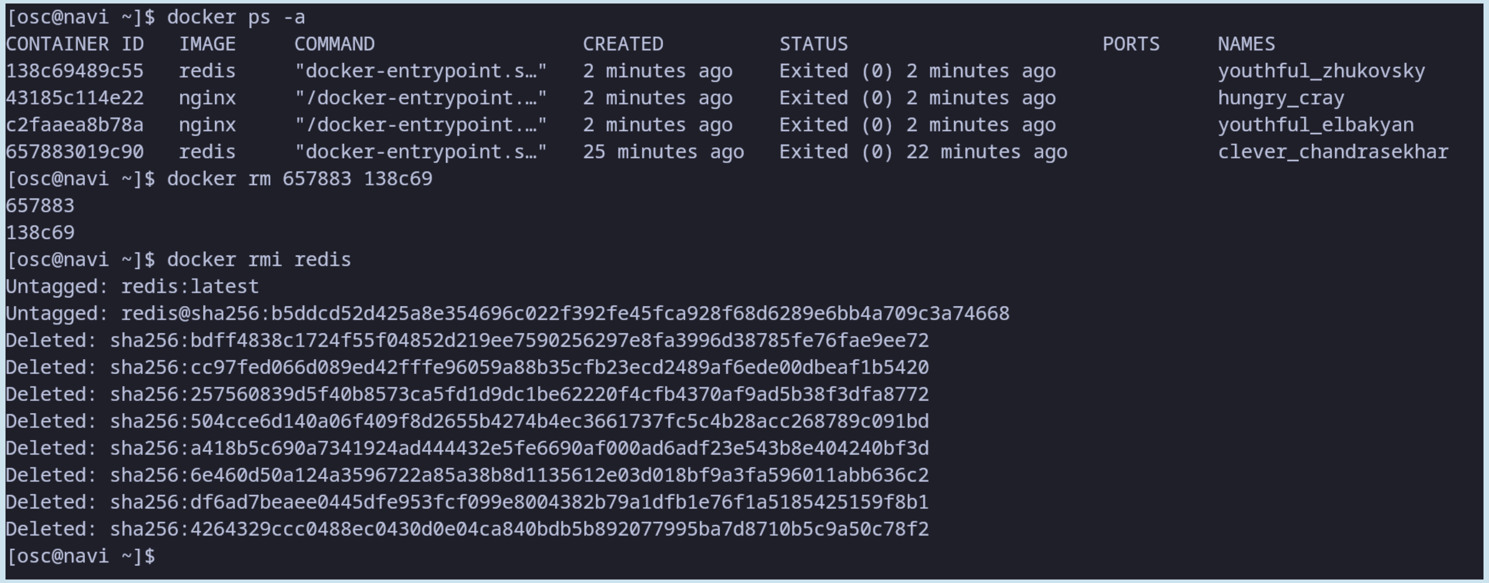
`rmi` removes images

- If we want to delete an image, we must remove all dependent containers first.



```
[osc@navi ~]$ docker rmi redis
Error response from daemon: conflict: unable to remove repository reference "redis" (must force) - container 657883019c90 is usi
ng its referenced image bdff4838c172
```

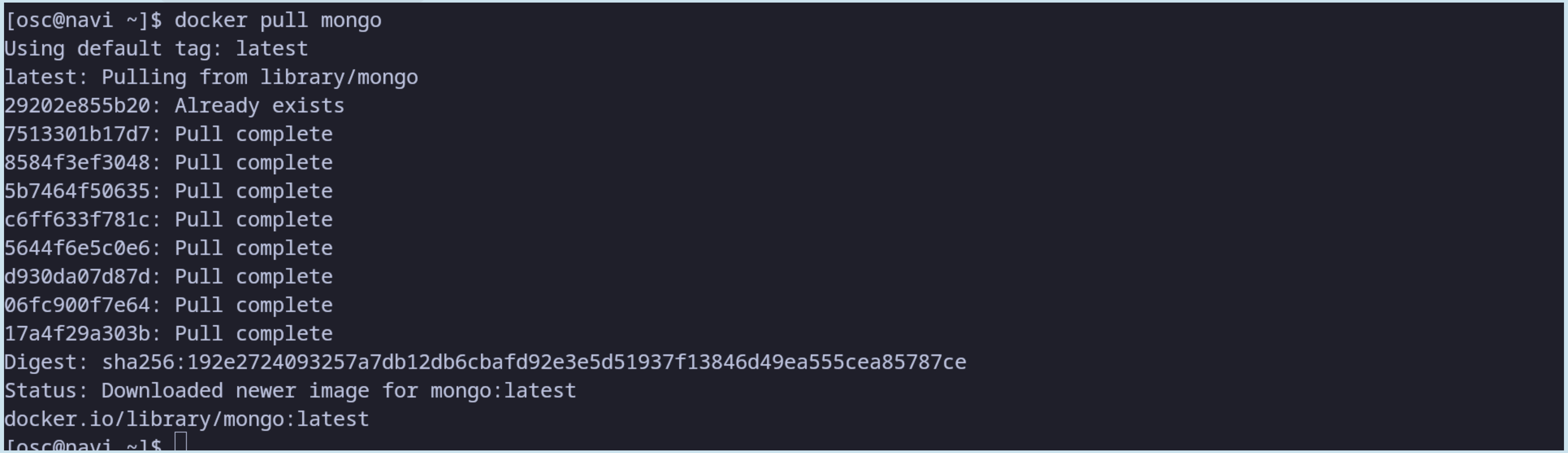
- Here we tried to delete the image `redis` but we had redis containers, even though they were stopped, it printed an error.



```
[osc@navi ~]$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS                       PORTS     NAMES
138c69489c55   redis     "docker-entrypoint.s…"   2 minutes ago    Exited (0) 2 minutes ago               youthful_zhukovsky
43185c114e22   nginx     "/docker-entrypoint.…"   2 minutes ago    Exited (0) 2 minutes ago               hungry_cray
c2faaea8b78a   nginx     "/docker-entrypoint.…"   2 minutes ago    Exited (0) 2 minutes ago               youthful_elbakyan
657883019c90   redis     "docker-entrypoint.s…"   25 minutes ago   Exited (0) 22 minutes ago              clever_chandrasekhar
[osc@navi ~]$ docker rm 657883 138c69
657883
138c69
[osc@navi ~]$ docker rmi redis
Untagged: redis:latest
Untagged: redis@sha256:b5ddcd52d425a8e354696c022f392fe45fca928f68d6289e6bb4a709c3a74668
Deleted: sha256:bdff4838c1724f55f04852d219ee7590256297e8fa3996d38785fe76fae9ee72
Deleted: sha256:cc97fed066d089ed42fffe96059a88b35cfb23ecd2489af6ede00dbeaf1b5420
Deleted: sha256:257560839d5f40b8573ca5fd1d9dc1be62220f4cfb4370af9ad5b38f3dfa8772
Deleted: sha256:504cce6d140a06f409f8d2655b4274b4ec3661737fc5c4b28acc268789c091bd
Deleted: sha256:a418b5c690a7341924ad444432e5fe6690af000ad6adf23e543b8e404240bf3d
Deleted: sha256:6e460d50a124a3596722a85a38b8d1135612e03d018bf9a3fa596011abb636c2
Deleted: sha256:df6ad7beaee0445dfe953fcf099e8004382b79a1dfb1e76f1a5185425159f8b1
Deleted: sha256:4264329ccc0488ec0430d0e04ca840bdb5b892077995ba7d8710b5c9a50c78f2
[osc@navi ~]$
```

`pull` downloads images

- We saw earlier that when running `docker run` it checks, if the image isn't available on the machine, it downloads it then runs it in a container directly.
- What if we want to only download an image without running a container?
- We use `docker pull` which "pulls" the image without creating a container



```
[osc@navi ~]$ docker pull mongo
Using default tag: latest
latest: Pulling from library/mongo
29202e855b20: Already exists
7513301b17d7: Pull complete
8584f3ef3048: Pull complete
5b7464f50635: Pull complete
c6ff633f781c: Pull complete
5644f6e5c0e6: Pull complete
d930da07d87d: Pull complete
06fc900f7e64: Pull complete
17a4f29a303b: Pull complete
Digest: sha256:192e2724093257a7db12db6cbafd92e3e5d51937f13846d49ea555cea85787ce
Status: Downloaded newer image for mongo:latest
docker.io/library/mongo:latest
[osc@navi ~]$
```

Practice 1

1. Create a new container from the `alpine` image
2. Create a new `alpine` container named "test_container"
3. Print out the images currently downloaded
4. Run a new `erseco/alpine-php-webserver` image in detached mode
5. Stop the `erseco/alpine-php-webserver` container
6. Remove the "test_container" container
7. Delete the `busybox` image

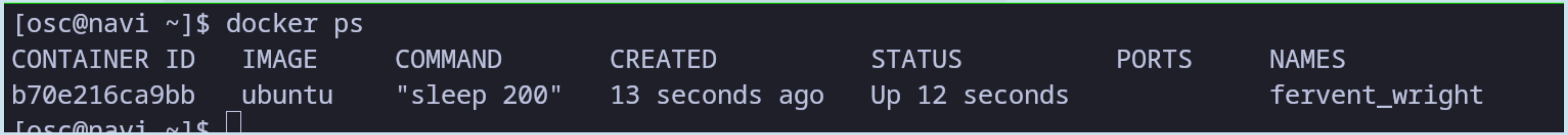
- When we run a linux container we will notice that it exits immediately, why is that?
- Containers are not meant to host an OS, it is meant to run a specific task or process, such as host an instance of a web server, database, or simply to carry a computation or analysis task, once this task is complete, it exits.
- This is why when we run an image of an OS, say Ubuntu, it exits immediately.

Appending commands

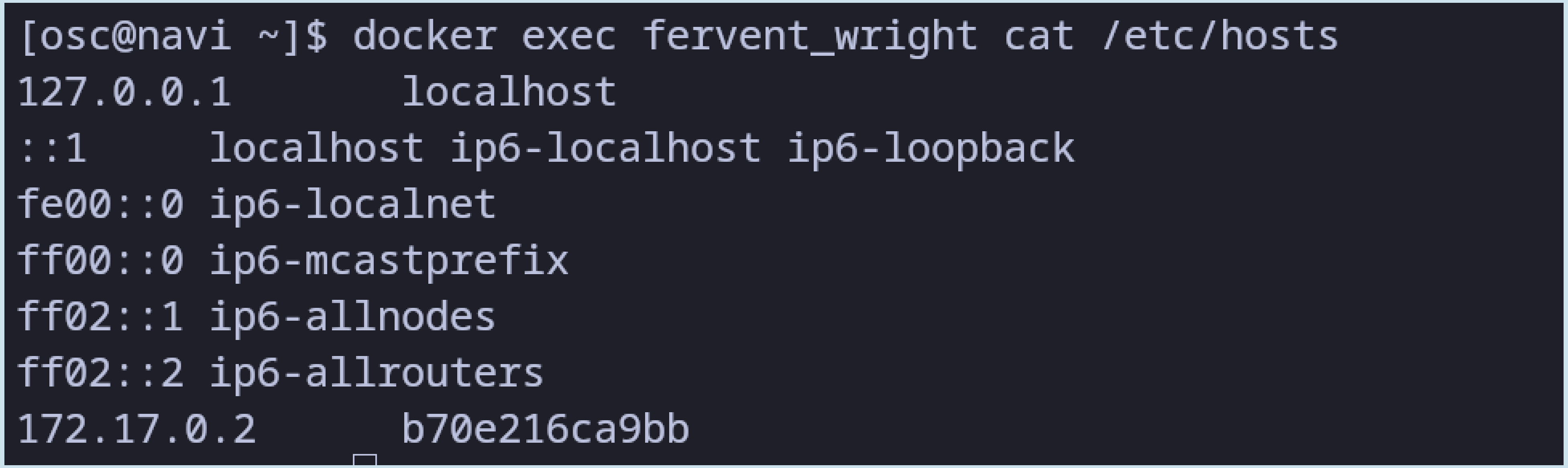
- We can append commands after `docker run` to run a specific command in the container when it starts.
- For example `docker run ubuntu sleep 10` will start an Ubuntu container and executes `sleep 10` and after the command exits the container exits too.

`exec` executes commands

- Okay so we saw how to run a command when we are creating the container, what if we have an already running container, how can we execute a command on it?
- We use `docker exec <container_name> <command>`



```
[osc@navi ~]$ docker ps
CONTAINER ID   IMAGE     COMMAND        CREATED          STATUS          PORTS     NAMES
b70e216ca9bb   ubuntu    "sleep 200"    13 seconds ago   Up 12 seconds             fervent_wright
[osc@navi ~]$
```



```
[osc@navi ~]$ docker exec fervent_wright cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2      b70e216ca9bb
```

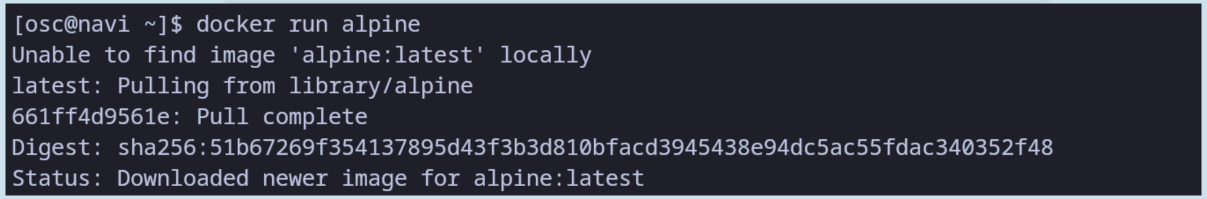
Break 🤩

Docker run

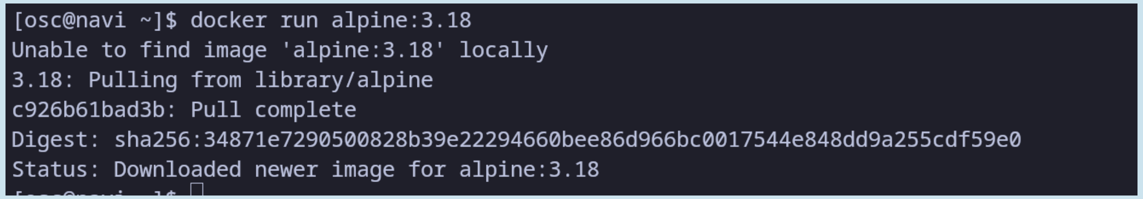
- One of the most commands that we use throught our usage of Docker is `run` and it has many options that we benefit from in configuring our containers, we will look at this command in some detail here.

Tag

- Tags are an identifier that is typically a version number or a variant of an image, if we don't specify a tag, the command uses latest by default.
- Syntax: `<docker run image_name:tag>`



```
[osc@navi ~]$ docker run alpine
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
661ff4d9561e: Pull complete
Digest: sha256:51b67269f354137895d43f3b3d810bfacd3945438e94dc5ac55fdac340352f48
Status: Downloaded newer image for alpine:latest
```



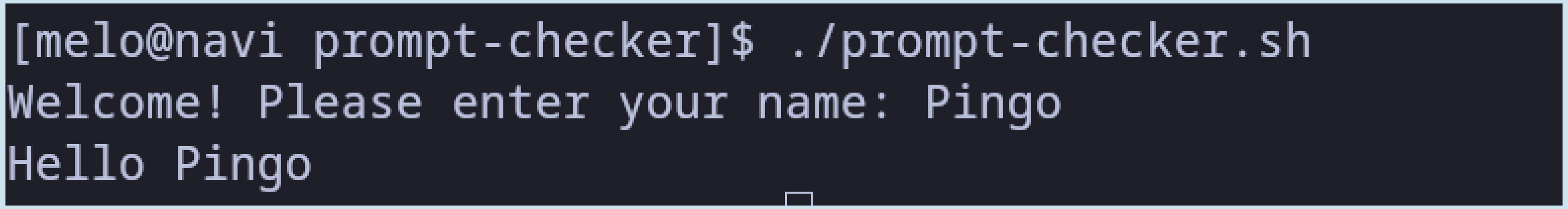
```
[osc@navi ~]$ docker run alpine:3.18
Unable to find image 'alpine:3.18' locally
3.18: Pulling from library/alpine
c926b61bad3b: Pull complete
Digest: sha256:34871e7290500828b39e22294660bee86d966bc0017544e848dd9a255cdf59e0
Status: Downloaded newer image for alpine:3.18
```

- In the first image we didn't specify a tag so it pulled latest, in the second we specified tag 3.18.
- To find all the tags relating to an image we look this image up on Docker Hub and we will see all tags there.

Standard input

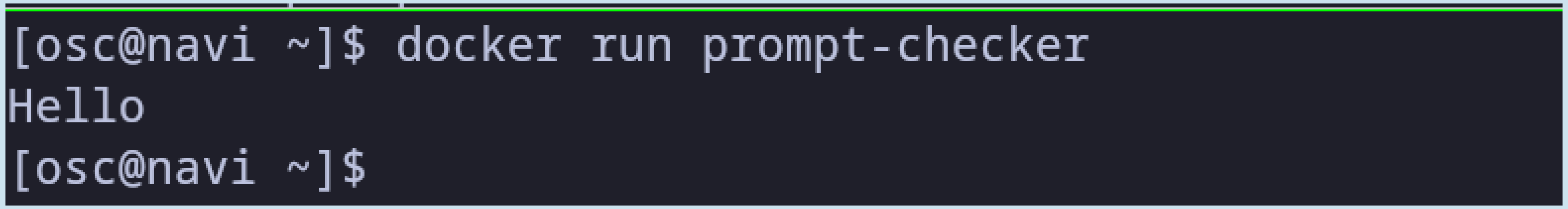
- Suppose we have a program that takes input, say a script that takes your name and prints `Hello <your_name>`.
- If we dockerize this program and run the container, it will not wait for the prompt, it will just print what it is supposed to on standard output.
- This is because Docker containers doesn't listen to stdin, even though we are attached to the container's console, it's not able to read any input, it doesn't have a terminal, in other terms the container runs in non-interactive mode.

- We have a program that takes a name and prints `Hello <name>`. This is it running normally



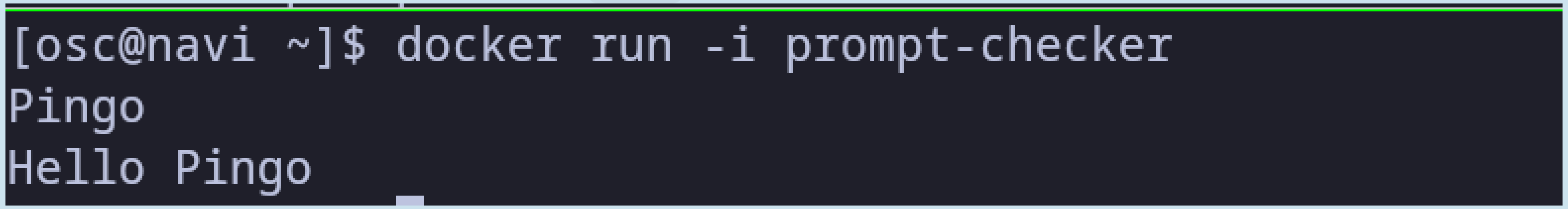
```
[melo@navi prompt-checker]$ ./prompt-checker.sh
Welcome! Please enter your name: Pingo
Hello Pingo
```

- If we dockerize it and run the container this will happen



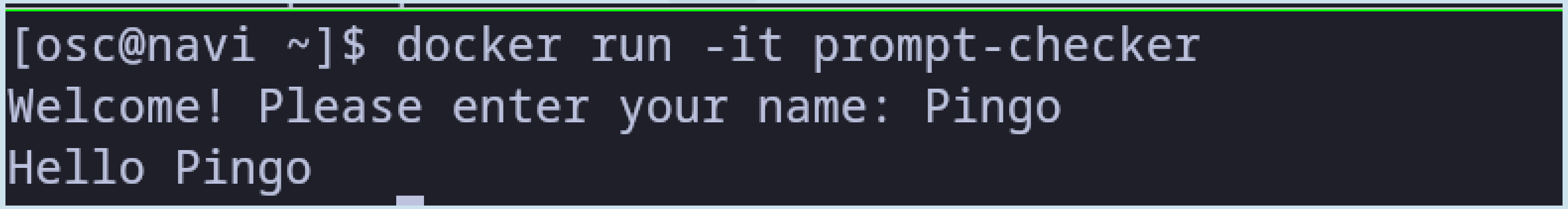
```
[osc@navi ~]$ docker run prompt-checker
Hello
[osc@navi ~]$
```

- We notice 2 things, first that it didn't take our input (our name in this case), to solve this we add the `-i` flag, which runs the container in interactive mode



```
[osc@navi ~]$ docker run -i prompt-checker
Pingo
Hello Pingo
```

- Now it takes our input and outputs correctly. But there is still the second problem, the prompt doesn't print, here it should ask first for the name but it doesn't, why? because the application prompts on the terminal and we haven't attached it to the container terminal.
- To solve this we add the `-t` flag which stands for psuedo-terminal.

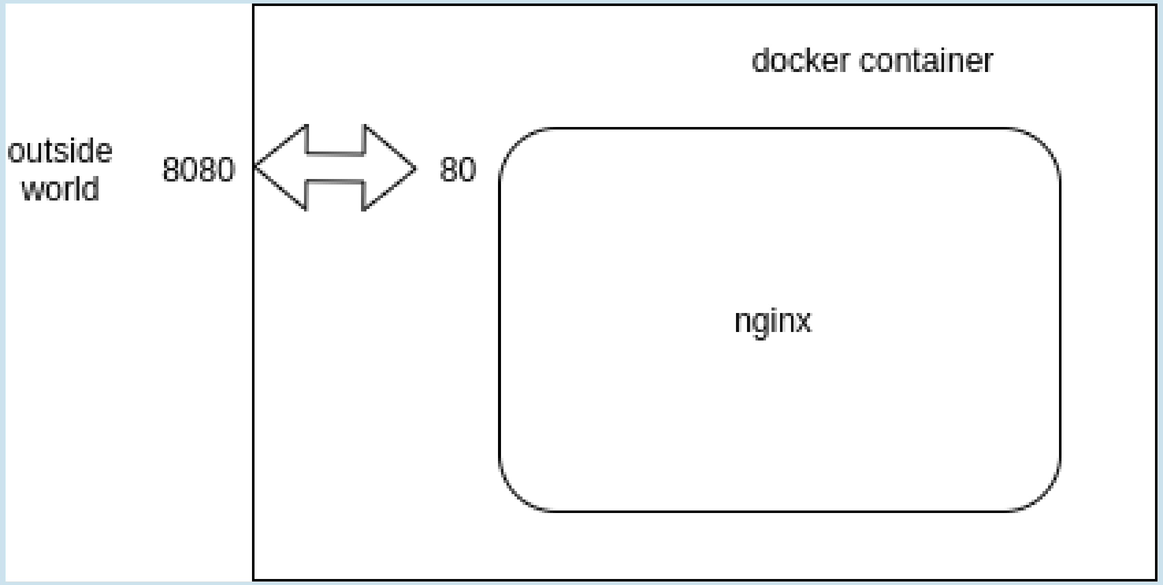


```
[osc@navi ~]$ docker run -it prompt-checker
Welcome! Please enter your name: Pingo
Hello Pingo
```

- So together, `-it` when used with docker run it basically takes us straight into the container's terminal.

Port Binding

- If we have a web app or server running in a container it will be given an IP on the closed Docker network, so anything outside the container can't access it, to solve this we need to use port binding (mapping) which is binding or mapping the container's port to the hosts's port to make the service available outside the container



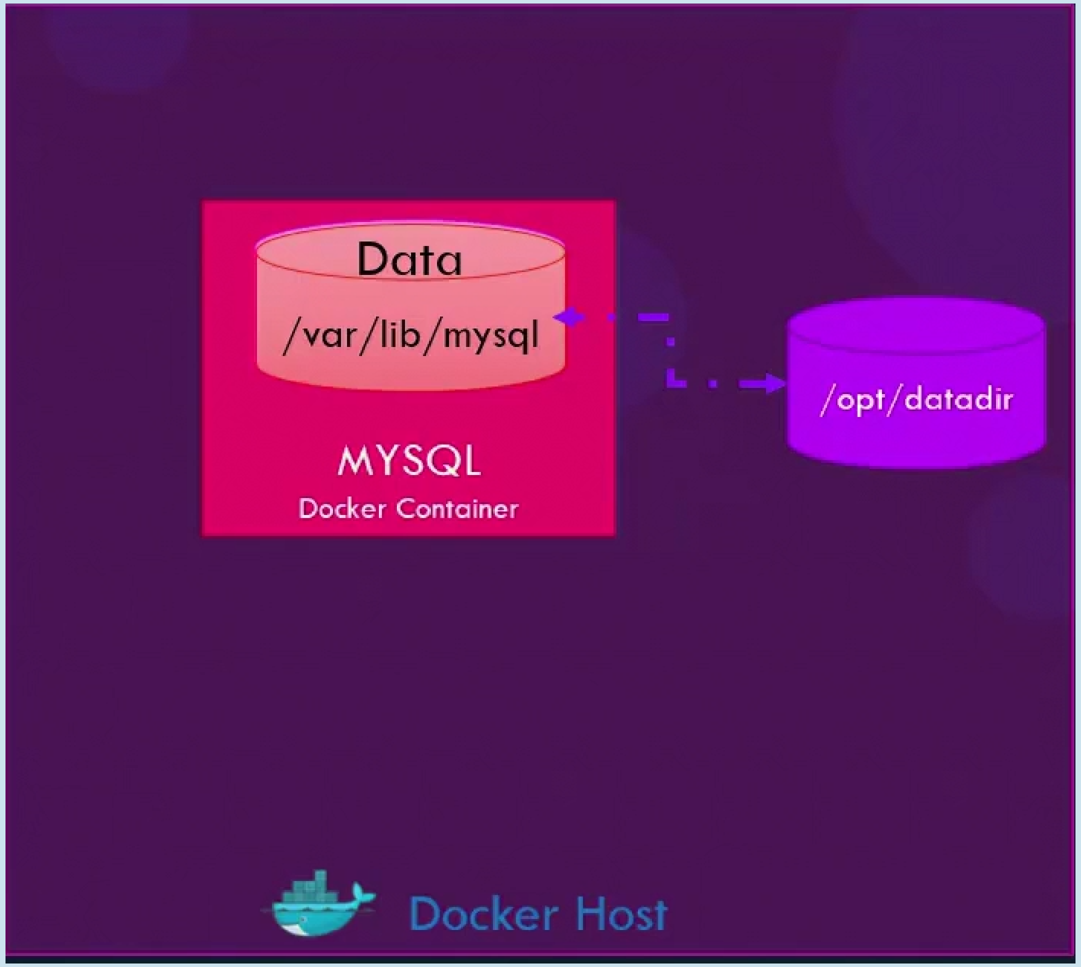
- nginx is a web server which runs on port 80, if we try to access the localhost at port 80 we will get nothing because port 80 has nothing on it now, we bind port 80 in the container to port 8080 on the host so now accessing the localhost on port 8080 we can access our server.

- Port mapping is done using the `-p` option with `docker run`
- `docker run -p <host_port>:<container_port> <image_name>`
- Note that only one service can run on this host port.

Volume Mapping

- Containers have their filesystem isolated from the host, so data stored and modified in the container aren't saved on the host.
- Let's say we have a database container, all data stored in this DB in stored in the container, if this container is removed, all data on the database is lost.

- To solve this issue we use volume mapping, where as in port mapping, we map a volume on our container to a volume on our host so data are saved on host and independent of the container in a way.



Any questions?

Thank You!