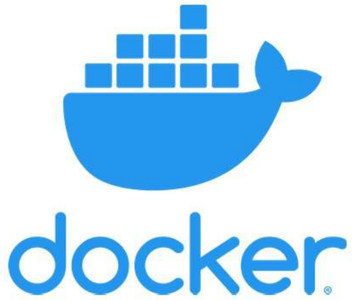
# Docker Deep Dive

# Chapter #3




## Docker & DevOps

# Docker

## DEVOPS PERSPECTIVE

### OPS PERSPECTIVE

Docker

### DEV PERSPECTIVE

Docker

## DOCKER

When you install docker you get two components

## DOCKER CLIENT

## DOCKER DAEMON

Also known as docker server or docker engine.

# Ops Perspective

You can give docker info command to check the two components.

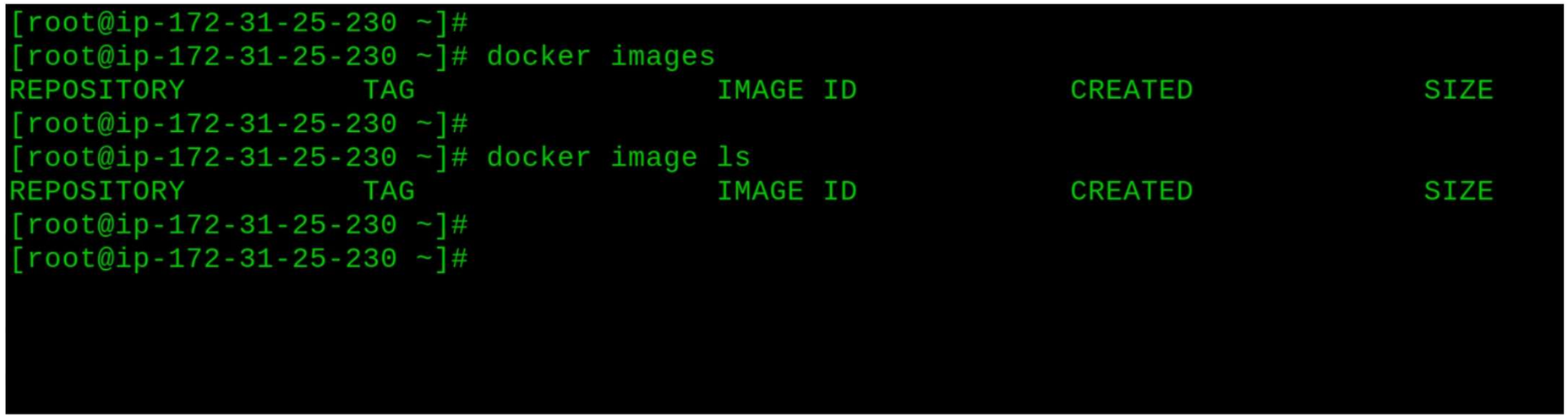
```
[root@ip-172-31-25-230 ~]#
[root@ip-172-31-25-230 ~]# docker version
Client:
 Version:           19.03.6-ce
 API version:       1.40
 Go version:        go1.13.4
 Git commit:        369ce74
 Built:             Fri Apr 24 18:30:51 2020
 OS/Arch:           linux/amd64
 Experimental:      false

Server:
 Engine:
  Version:          19.03.6-ce
  API version:      1.40 (minimum version 1.12)
  Go version:       go1.13.4
  Git commit:       369ce74
  Built:            Fri Apr 24 18:33:00 2020
  OS/Arch:          linux/amd64
  Experimental:     false
```

## Docker Images

It’s like a virtual machine template. A virtual machine template is essentially a stopped virtual machine. In "ops" world, image is a esentially a stopped container. In "dev" world, think of image as a class. Check images on your docker host:

# docker images



Getting images onto your Docker host is called “pulling”. Pull a image in your docker host using:

# docker pull centos:latest

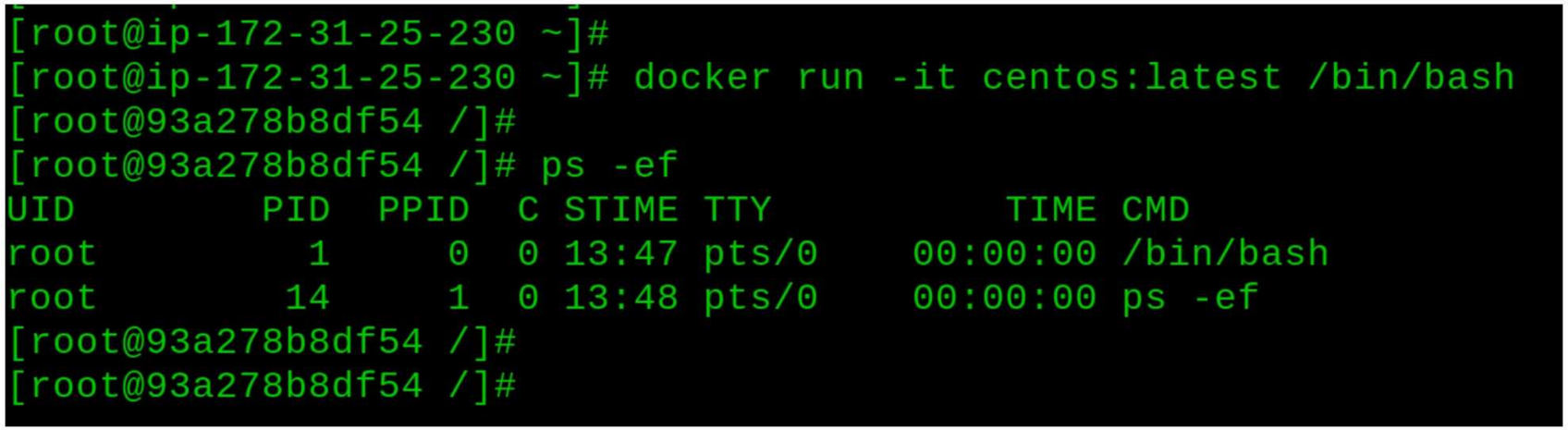
```
[root@ip-172-31-25-230 ~]#
[root@ip-172-31-25-230 ~]# docker pull centos:latest
latest: Pulling from library/centos
8a29a15cefae: Pull complete
Digest: sha256:fe8d824220415eed5477b63addf40fb06c3b049404242b31982106ac204f6700
Status: Downloaded newer image for centos:latest
docker.io/library/centos:latest
[root@ip-172-31-25-230 ~]#
[root@ip-172-31-25-230 ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
centos              latest              470671670cac        4 months ago        237MB
[root@ip-172-31-25-230 ~]#
[root@ip-172-31-25-230 ~]#
```

Each image gets its own unique ID. When working with the images you can refer to them using either IDs or names.

## Docker Containers

As we have an image pulled locally on our Docker host, we can use the docker container run command to launch a container from it.

# docker run -it centos:latest /bin/bash



Lets understand docker run command. docker run tells the Docker daemon to start a new container.

The -it flags tell the daemon to make the container interactive and to attach our current terminal to the shell of the container.

Next, the command tells Docker that we want the container to be based on the centos:latest image.

Finally, we tell Docker which process we want to run inside of the container. In this example we’re running a Bash shell.

Run a ps command from inside of the container to list all running processes.

Inside the Linux container there are only two processes running:

PID 1. This is the /bin/bash process that we told the container to run with the docker container run command.

PID 14. This is the ps -ef command/process that we ran to list the running processes.

The presence of the ps -ef process in the Linux output above is slightly confusing as it is a short-lived process that dies as soon as the ps command exits.

This means that the only long-running process inside of the container is the /bin/bash process.

Now if we exit from the container's bash shell, the container will die. Why?

```
[root@93a278b8df54 /]#
[root@93a278b8df54 /]# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 13:47 pts/0    00:00:00 /bin/bash
root        14     1  0 13:48 pts/0    00:00:00 ps -ef
[root@93a278b8df54 /]#
[root@93a278b8df54 /]# exit
exit
[root@ip-172-31-25-230 ~]#
```

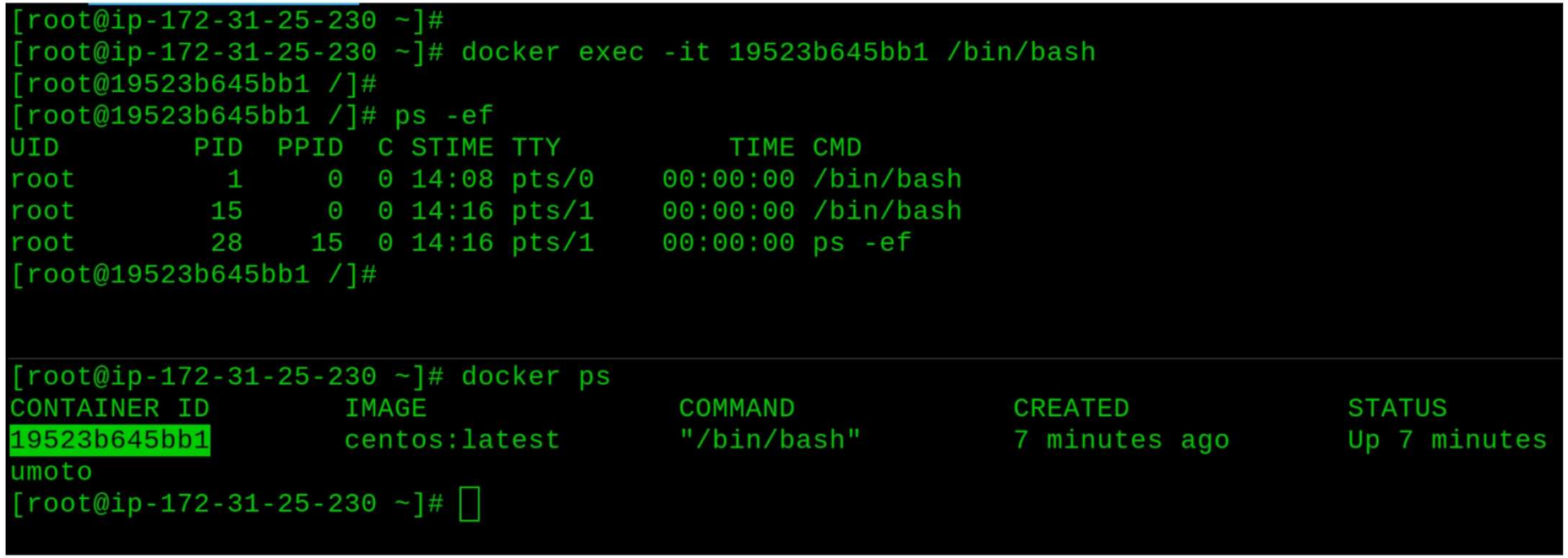
```
[root@ip-172-31-25-230 ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
93a278b8df54        centos:latest       "/bin/bash"         16 minutes ago
ckley
[root@ip-172-31-25-230 ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
[root@ip-172-31-25-230 ~]#
[root@ip-172-31-25-230 ~]#
```

Run another new container again using centos:latest image. Go inside the container and then press ctrl p ctrl q to exit from the container without terminating it. You can confirm using docker ps command from another shell that container is still running.

```
[root@ip-172-31-25-230 ~]# docker run -it centos:latest /bin/bash
[root@19523b645bb1 /]#
[root@19523b645bb1 /]# ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 14:08 pts/0    00:00:00 /bin/bash
root        14     1  0 14:08 pts/0    00:00:00 ps -ef
[root@19523b645bb1 /]# #press ctrl p ctrl q to exit container without
[root@19523b645bb1 /]# #terminating it
[root@19523b645bb1 /]# [root@ip-172-31-25-230 ~]#
```

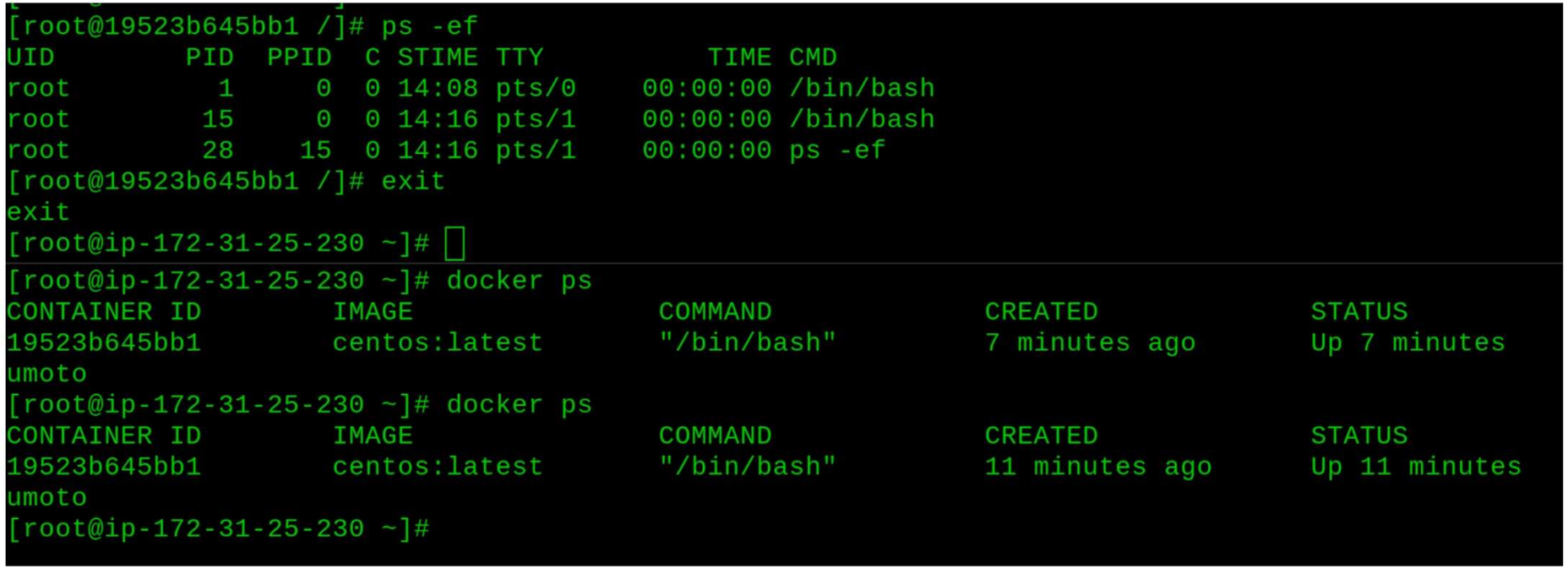
```
[root@ip-172-31-25-230 ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
19523b645bb1        centos:latest       "/bin/bash"         58 seconds ago      Up 57 seconds
umoto
[root@ip-172-31-25-230 ~]#
```

You can attach your shell to running containers with the docker container exec command. As the container from the previous steps is still running, let’s connect back to it.




Now we see there are two /bin/bash processes running inside the container. Why?

And if you exit from the container now, the container will not be terminated. Why?



Stop the container and kill it using the docker container stop and docker container rm commands.

```
[root@ip-172-31-25-230 ~]#
[root@ip-172-31-25-230 ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
19523b645bb1        centos:latest       "/bin/bash"         14 minutes ago      Up 14 minutes
umoto
[root@ip-172-31-25-230 ~]# docker stop 19523b645bb1
19523b645bb1
[root@ip-172-31-25-230 ~]# docker rm 19523b645bb1
19523b645bb1
[root@ip-172-31-25-230 ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
[root@ip-172-31-25-230 ~]#
[root@ip-172-31-25-230 ~]#
```

## CONTAINERS ARE APPS

Docker containers are all about apps

## DOCKERFILE

Customize DockerFile. Containerize it. Run it as a container.

# Dev Perspective

A sample linux app can be downloaded from:

Use the git clone command to download the application into your docker host. Install "git" if required.

```
[root@ip-172-31-25-230 ~]#
[root@ip-172-31-25-230 ~]# git clone https://github.com/networknuts/dockerdeepone.git
Cloning into 'dockerdeepone'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 9 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (9/9), done.
[root@ip-172-31-25-230 ~]#
[root@ip-172-31-25-230 ~]# cd dockerdeepone/
[root@ip-172-31-25-230 dockerdeepone]#
[root@ip-172-31-25-230 dockerdeepone]# ls
Dockerfile  index.html  README.md
[root@ip-172-31-25-230 dockerdeepone]#
```

Move inside the dockerdeepone directory and check the contents of Dockerfile. We will be discussing Dockerfile in detail as the course proceed.

```
[root@ip-172-31-25-230 dockerdeepone]# cat Dockerfile
FROM centos:latest
MAINTAINER networknuts <info@networknuts.net>
RUN yum install httpd -y
COPY index.html /var/www/html
CMD ["/usr/sbin/httpd", "-D", "FOREGROUND"]
EXPOSE 80
[root@ip-172-31-25-230 dockerdeepone]#
[root@ip-172-31-25-230 dockerdeepone]#
```

Also check the contents of index.html file.

```
[root@ip-172-31-25-230 dockerdeepone]# pwd
/root/dockerdeepone
[root@ip-172-31-25-230 dockerdeepone]# ls
Dockerfile  index.html  README.md
[root@ip-172-31-25-230 dockerdeepone]# cat index.html
<html>
<title>Network Nuts Docker Deep Dive</title>
<body>
<h1>Docker Deep Dive Training</h1>
<h2>A containerized webserver</h2>
<h3>- #networknuts
</body>
</html>
[root@ip-172-31-25-230 dockerdeepone]#
```

Currently, we have pulled some application code from networknuts git repo. We also have a Dockerfile containing instructions that describe how to create a new Docker image with the application inside.

Use the docker image build command to create a new image using the instructions contained in the Dockerfile. We will create a new docker image called deepone:latest.

Be sure to perform this command from within the directory containing the app code and Dockerfile.

```
[root@ip-172-31-25-230 dockerdeepone]# docker image build -t deepone:latest .
Sending build context to Docker daemon  61.95kB
Step 1/6 : FROM centos:latest
 ---> 470671670cac
Step 2/6 : MAINTAINER networknuts <info@networknuts.net>
 ---> Using cache
 ---> baa0c1672c77
Step 3/6 : RUN yum install httpd -y
 ---> Using cache
 ---> 693789c9030e
Step 4/6 : COPY index.html /var/www/html
 ---> 92628c07e3da
Step 5/6 : CMD ["/usr/sbin/httpd", "-D", "FOREGROUND"]
 ---> Running in 5b7914ea2c09
```

The process may take some time. Once its done, you can use docker images command to see your new application image available on your docker host.

```
[root@ip-172-31-25-230 ~]#
[root@ip-172-31-25-230 ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED              SIZE
deepone             latest              c64afc27d295        About a minute ago   279MB
centos              latest              470671670cac        4 months ago         237MB
[root@ip-172-31-25-230 ~]#
[root@ip-172-31-25-230 ~]#
```