Docker and Containers

Traditional Virtualization

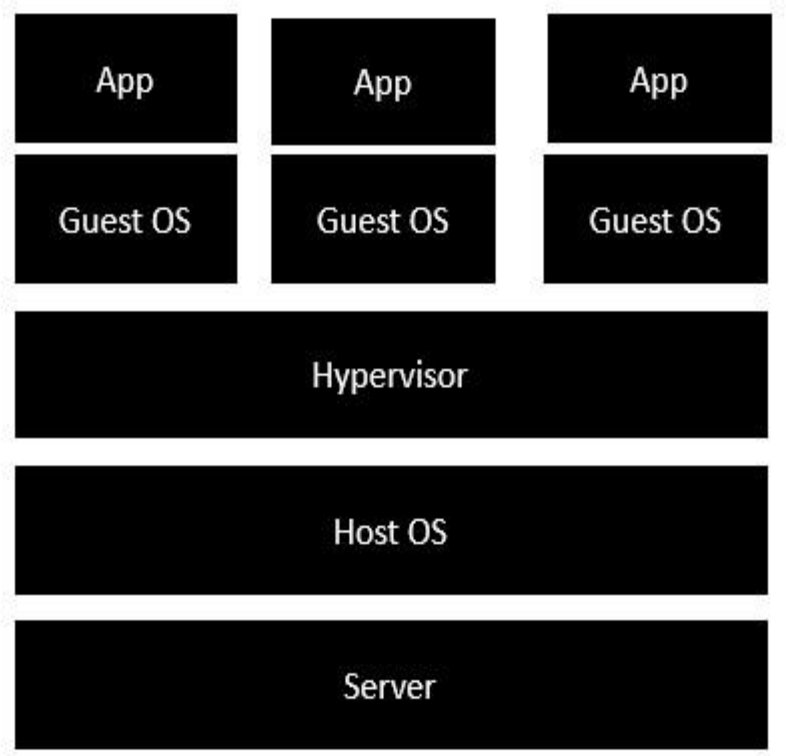
- Server is the physical server that is used to host multiple virtual machines.

- Host OS is the base machine such as Linux or Windows.

- Hypervisor is either VMWare or Windows Hyper V that is used to host virtual machines.

- One would then install multiple operating systems as virtual machines on top of the existing hypervisor as Guest OS.

- One would then host your applications on top of each Guest OS.



App App App

Guest OS Guest OS Guest OS

Hypervisor

Host OS

Server

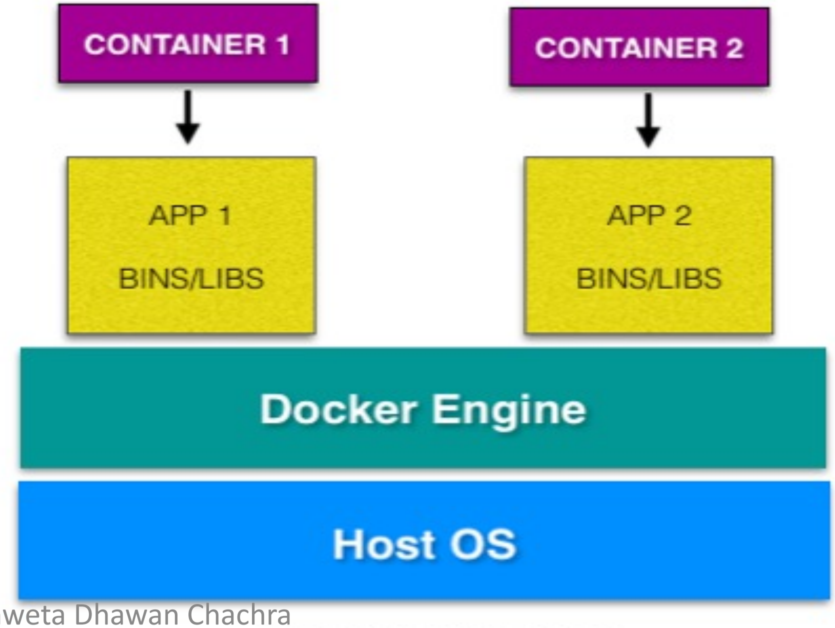
Dockers

- Dockers containers are **analogous to physical containers that you can use to store, package, and transport goods.**
- But instead of tangible goods, they're containers for software applications.

Dockers

- A container is an environment that **runs an application that is not dependent on the operating system.**

- A docker container is a portable unit of software—
- **that has the application**
- **along with the all of its associated dependency and configuration.**



DOCKER ARCHITECTURE

What is a Container?

- **The kernel of the host operating system**
- **serves the needs of running different functions of an app, separated into containers.**

What is a Container?

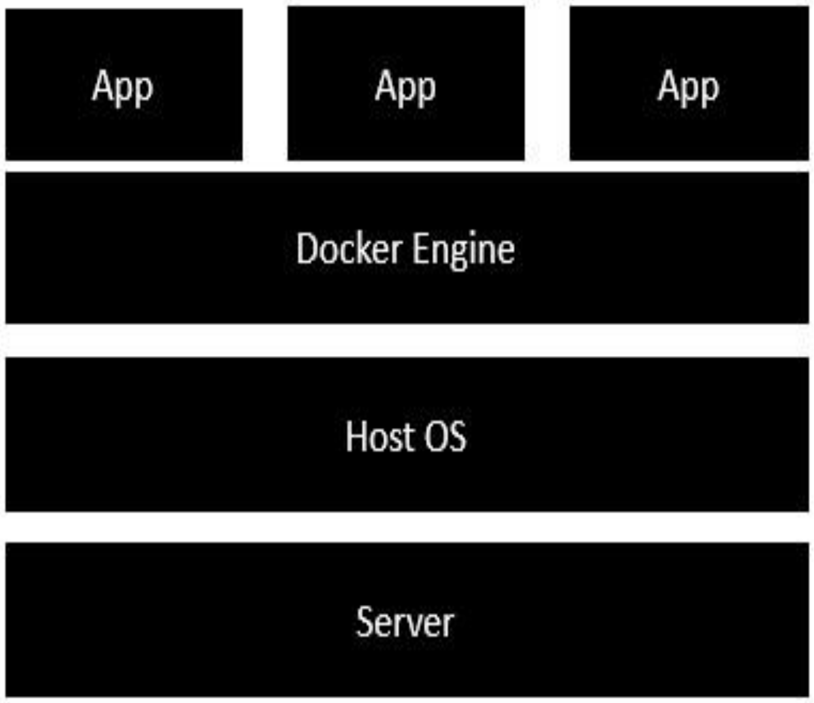
- **Each container runs isolated tasks.**
- **It cannot harm the host machine nor**
- **come in conflict with other apps running in separate containers.**

Docker – Architecture

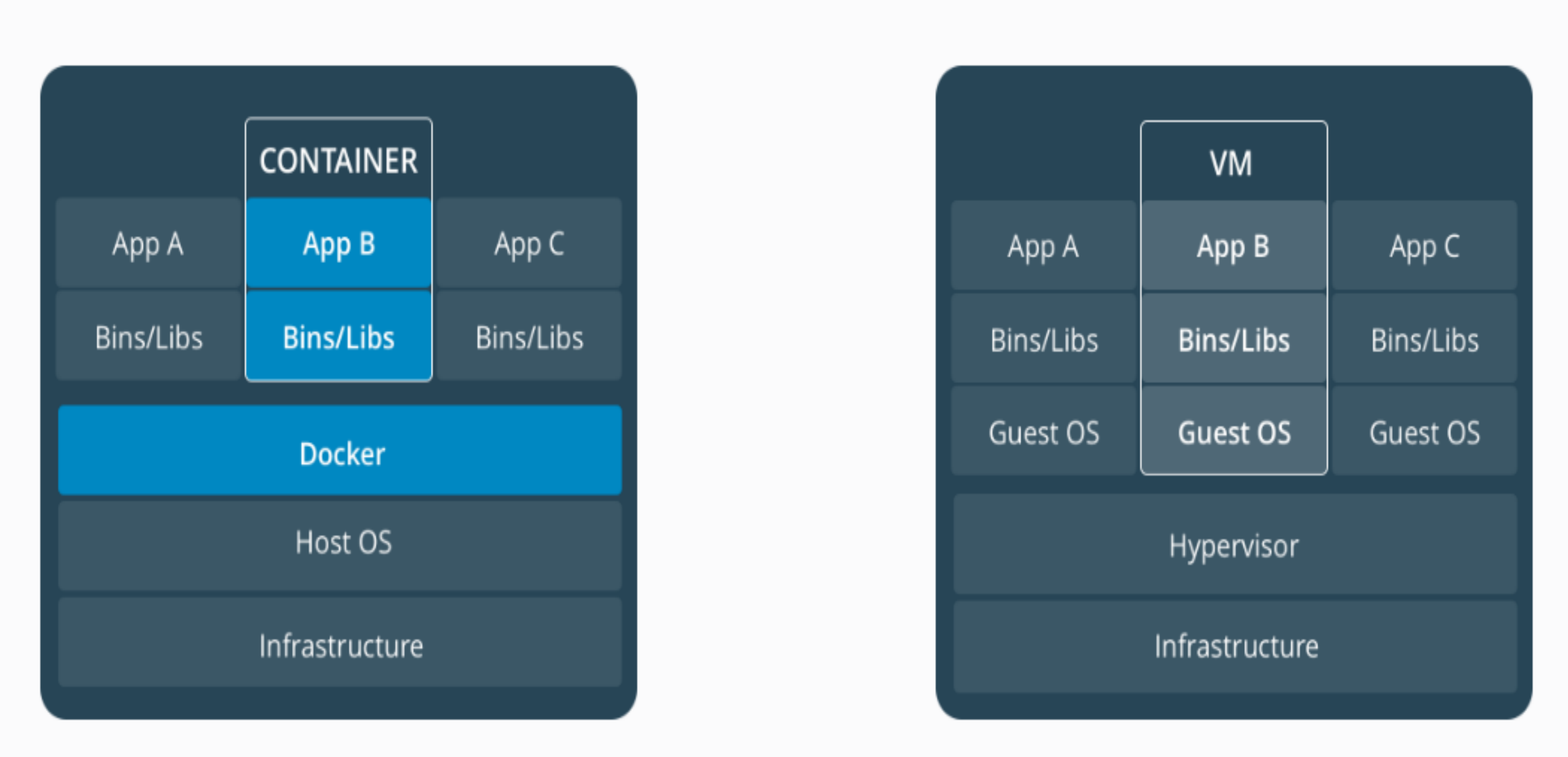
- **Docker engine-**

is used to virtualize the guest operating system which earlier used to run in virtual machines

- All of the Apps now run as Docker containers.

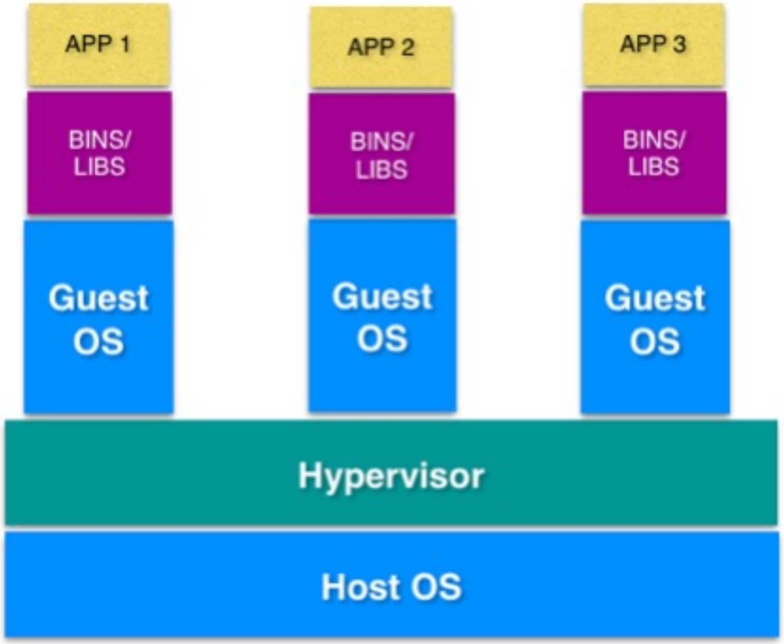


Containers vs Virtual Machines

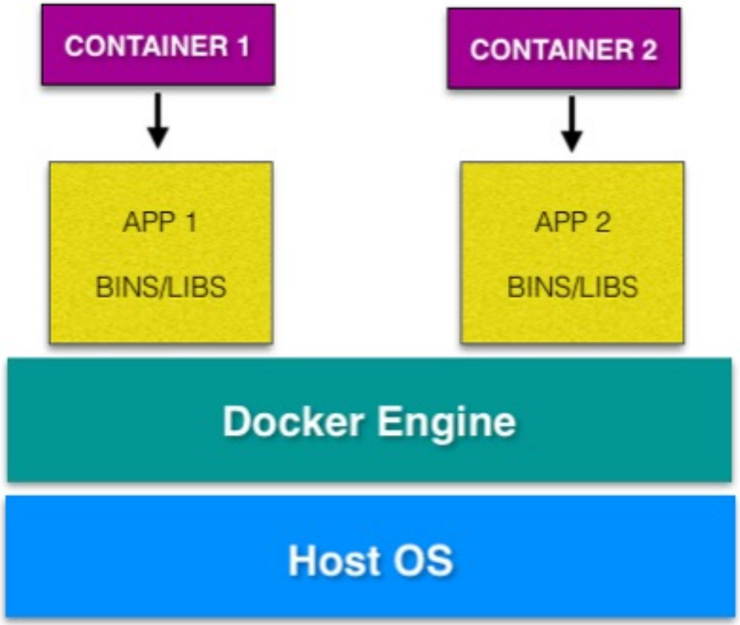


Container

VM



VIRTUAL MACHINE ARCHITECTURE



DOCKER ARCHITECTURE

Virtual Machines and Containers

- **Virtual machines** run guest operating systems - the OS layer in each box.
- Resulting disk image and **application state is**
- **an entanglement of OS settings,**
- **system-installed dependencies,**
- OS security patches

Virtual Machines and Containers

- **Containers** can share a single kernel **and the only information that needs to be in a container image is the executable and its package dependencies, which never need to be installed on the host system.**
- These processes run like native processes, and can be managed individually
- Because they contain all their dependencies, there is no configuration entanglement;
- A containerized app "runs anywhere"

Learn to build and deploy your distributed applications easily to the cloud with Docker

Dockers

- *An open-source project that automates the deployment of software applications inside* ***containers*** *by providing an additional layer of abstraction and automation of* ***OS-level virtualization*** *on Linux.*

- Wikipedia

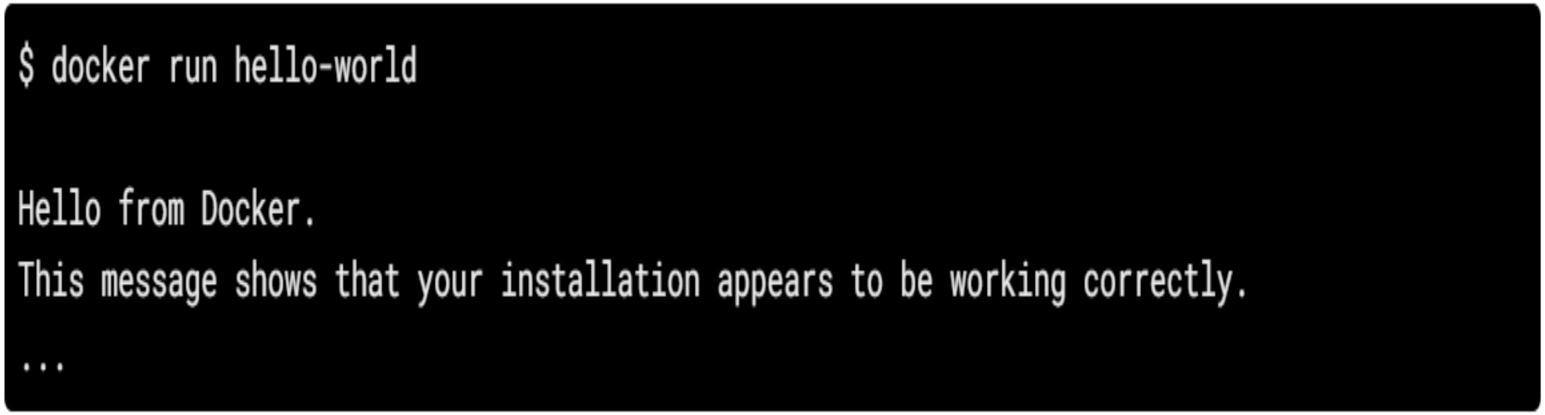
Dockers

- Containers offer a **logical packaging mechanism in which applications can be abstracted from the environment** in which they actually run.
- This decoupling allows container-based applications to be deployed easily and consistently, regardless of whether the target environment is **a private data center, the public cloud, or even a developer's personal laptop.**
- This gives developers the ability **to create predictable environments that are isolated from the rest of the applications and can be run anywhere.**

Docker commands

- Set up Docker on Mac, Linux and Windows.

- Once you are done installing Docker,
- Test your Docker installation by running the following:

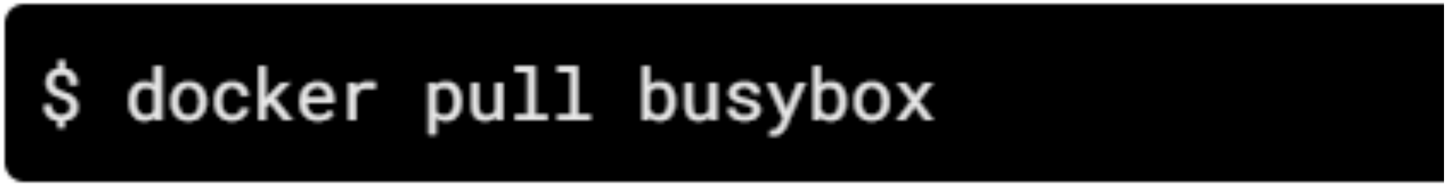


```
$ docker run hello-world


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

Docker commands

- Lets run a Busybox container on our system and get a taste of the docker run command.
- To get started, let's run the following in our terminal:



```
$ docker pull busybox
```

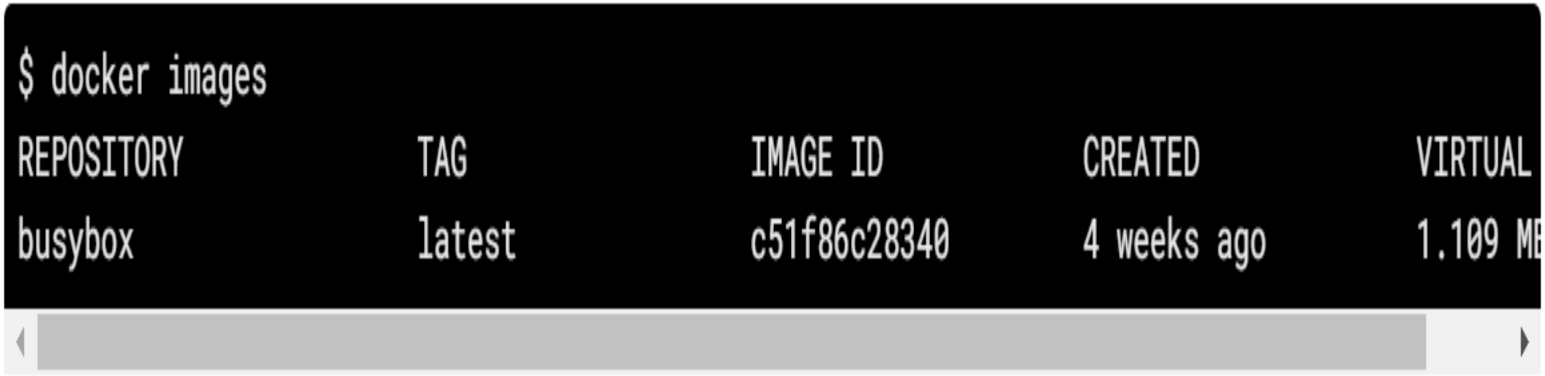
- **The pull command fetches the busybox image from the Docker registry and saves it to our system.**

Docker commands

```
$ docker pull busybox
```

- *Note: Depending on how you've installed docker on your system, you might see a* permission denied *error after running the above command.*
- ***If you're on a Mac, make sure the Docker engine is running.***
- ***If you're on Linux, then prefix your* docker *commands with* sudo.**

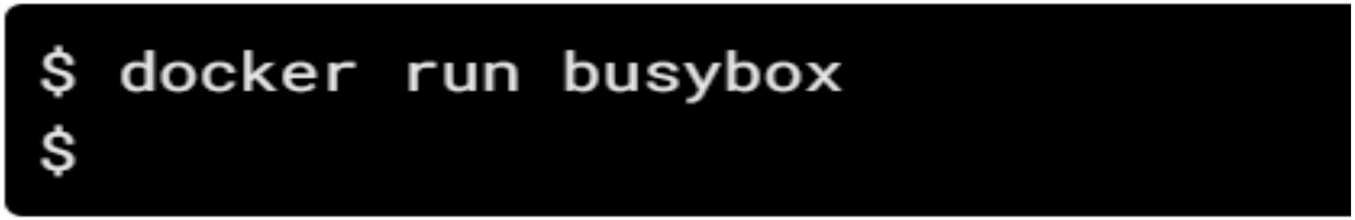
- You can use the docker images command to see a list of all images on your system.



```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL
busybox             latest              c51f86c28340        4 weeks ago         1.109 MB
```

Docker Run

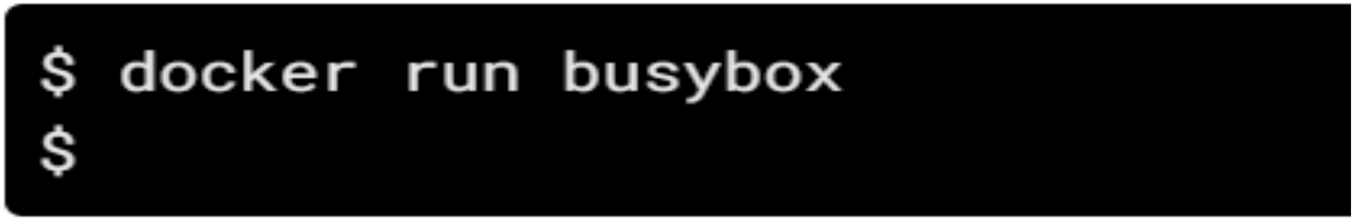
- Great! Let's now run a Docker **container** based on this image. To do that we are going to use the almighty docker



```
$ docker run busybox
$
```

- Wait, nothing happened! Is that a bug? Well, no. Behind the scenes, a lot of stuff happened.

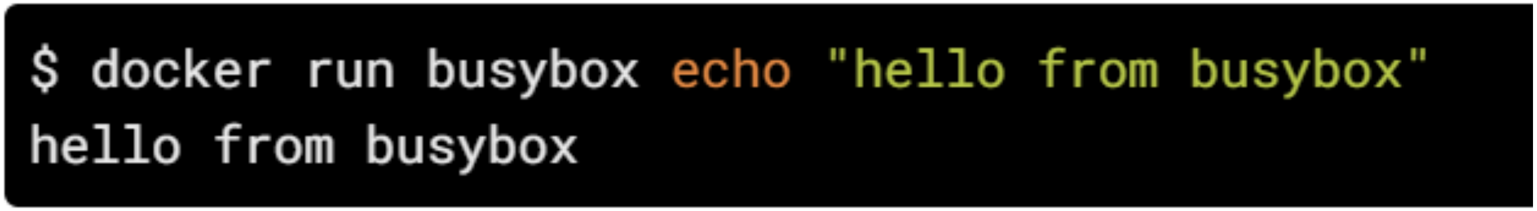
Docker Run



```
$ docker run busybox
$
```

- When you call run, the Docker client finds the image (busybox in this case), loads up the container and then runs a command in that container.
- When we executed docker run busybox, **we didn't provide a command, so the container booted up, ran an empty command and then exited.**

Docker Run

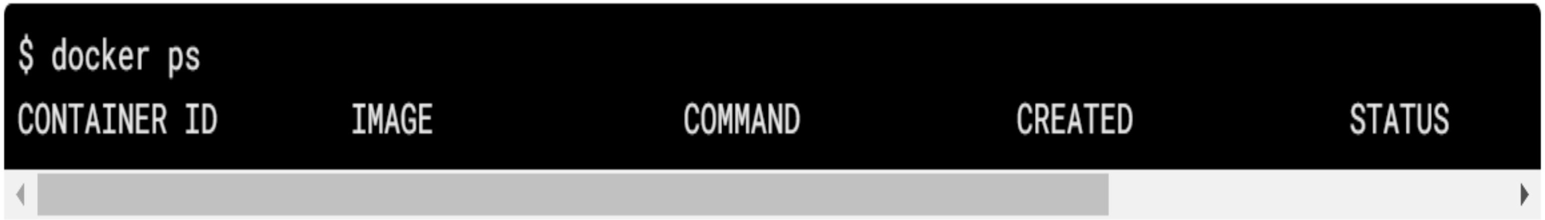


```
$ docker run busybox echo "hello from busybox"
hello from busybox
```

- Nice - finally we see some output.
- In this case, the **Docker client dutifully ran the echo command in our busybox container** and then exited it.
- If you've noticed, all of that happened pretty quickly. Imagine **booting up a virtual machine, running a command and then killing it. Now you know why they say containers are fast!**

Docker ps

- Ok, now it's time to see the docker ps command.
- The docker ps command shows you **all containers that are currently running.**



```
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
```

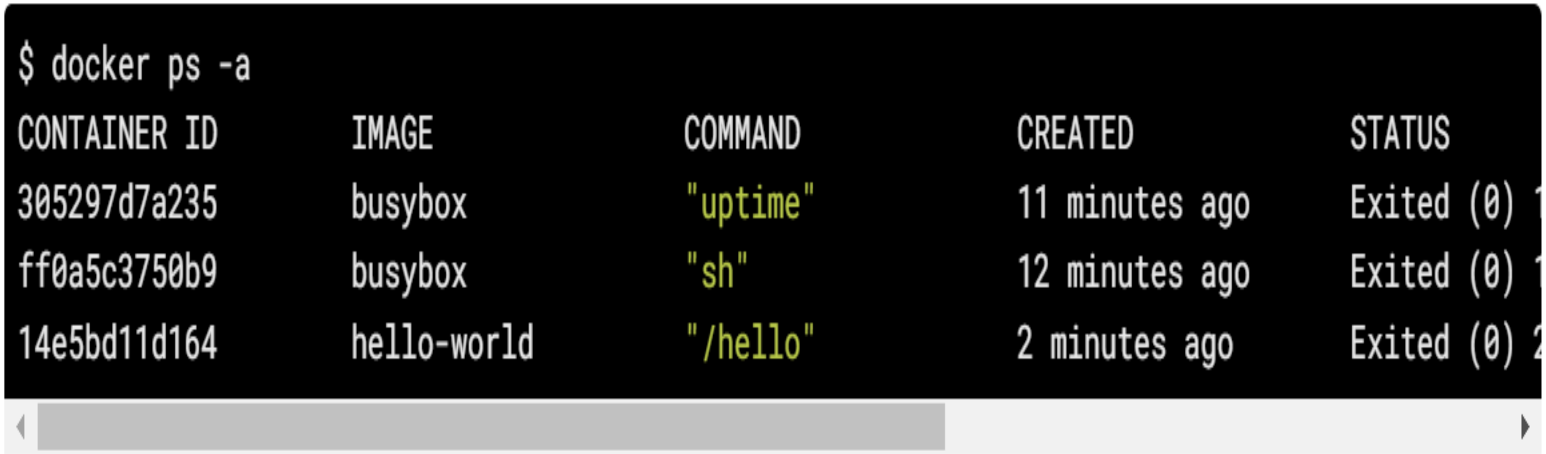
- Since no containers are running, we see a blank line.

- **A list of running containers can be seen using the docker ps command.**

Docker ps

- Let's try a more useful variant:

docker ps –a

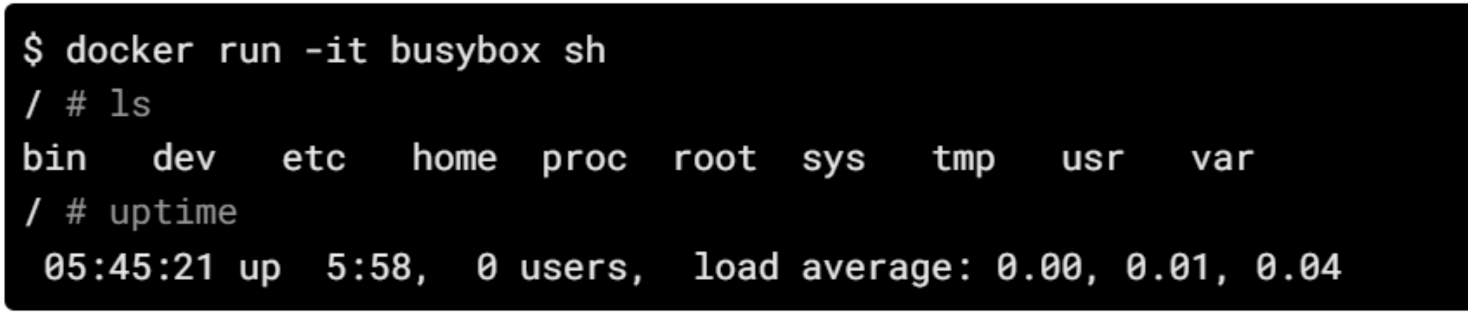


```
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
305297d7a235        busybox             "uptime"            11 minutes ago      Exited (0) 1
ff0a5c3750b9        busybox             "sh"                12 minutes ago      Exited (0) 1
14e5bd11d164        hello-world         "/hello"            2 minutes ago       Exited (0) 2
```

- So what we see above is a list of all containers that we ran.
- **Do notice that the STATUS column shows that these containers exited a few minutes ago.**

Docker Run

- You're probably wondering if there is a way to run more than just one command in a container. Let's try that now:



```
$ docker run -it busybox sh
/ # ls
bin   dev   etc   home  proc  root  sys   tmp   usr   var
/ # uptime
 05:45:21 up  5:58,  0 users,  load average: 0.00, 0.01, 0.04
```

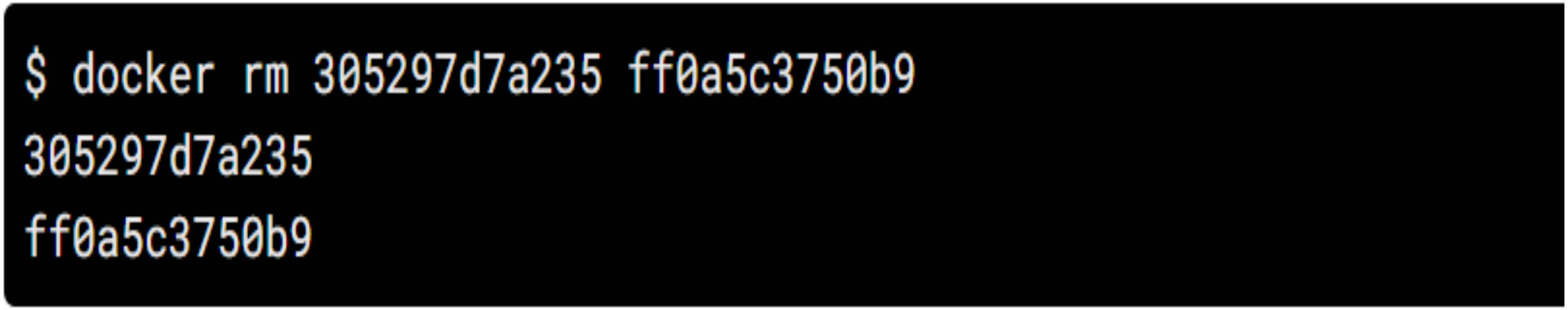
- Running the run command with the -it flags attaches us to an interactive tty in the container.
- **Now we can run as many commands in the container as we want.**

Docker Run

- let's quickly talk about deleting containers.
- We saw above that we can still see remnants of the container even after we've exited by running docker ps -a.
- You'll run docker run multiple times and leaving stray containers will eat up disk space.
- Hence, as a rule of thumb, I clean up containers once I'm done with them.

Docker rm

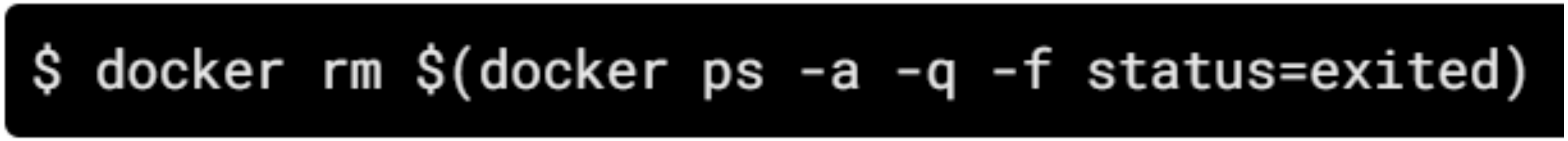
- To do that, you can run the docker rm command. Just copy the container IDs from above and paste them alongside the command.



```
$ docker rm 305297d7a235 ff0a5c3750b9
305297d7a235
ff0a5c3750b9
```

Docker rm

- On deletion, you should see the IDs echoed back to you.
- If you have a bunch of containers to delete in one go, copy-pasting IDs can be tedious.
- In that case, you can simply run -

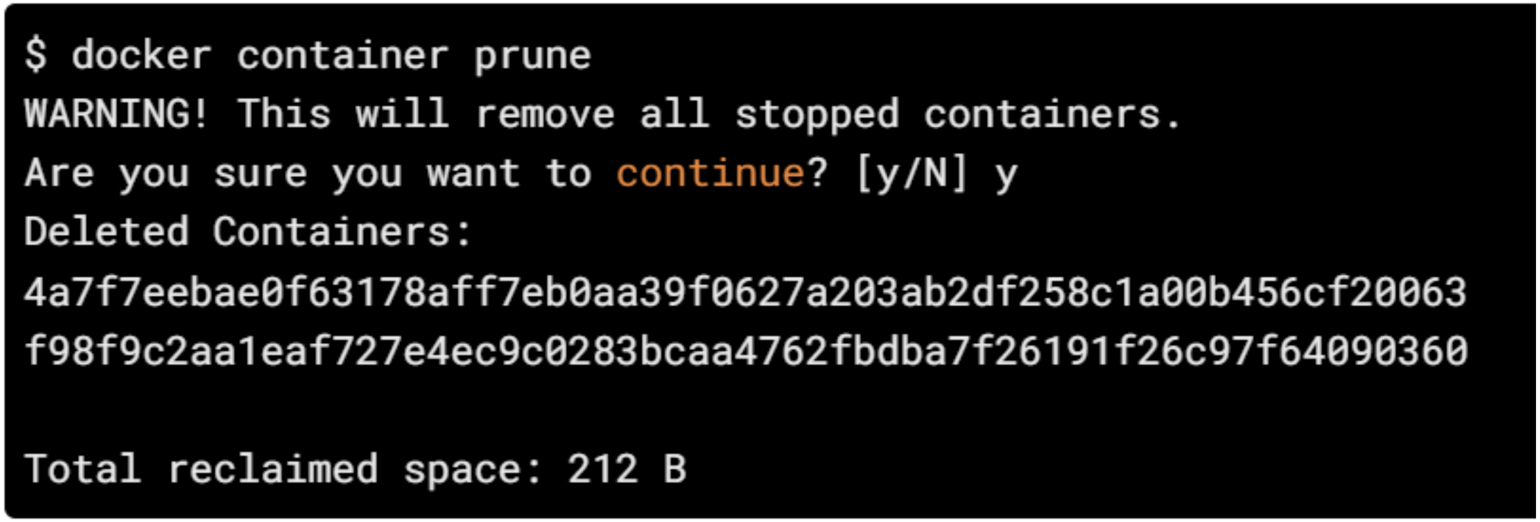


```
$ docker rm $(docker ps -a -q -f status=exited)
```

- This command deletes all containers that have a status of exited.
- the -q flag, only returns the numeric IDs and
- -f filters output based on conditions provided.

Docker prune

- In later versions of Docker, the docker container prune command can be used to achieve the same effect.



```
$ docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
4a7f7eebae0f63178aff7eb0aa39f0627a203ab2df258c1a00b456cf20063
f98f9c2aa1eaf727e4ec9c0283bcaa4762fbdba7f26191f26c97f64090360

Total reclaimed space: 212 B
```

Docker Terminology

- Images - The blueprints of our application which form the basis of containers. In the demo above, we used the docker pull command to download the busybox image.
- Containers - Created from Docker images and run the actual application. We create a container using docker run which we did using the busybox image that we downloaded.
- A list of running containers can be seen using the docker ps command.

Docker Terminology

- Docker Daemon - The background service running on the host that manages building, running and distributing Docker containers. The daemon is the process that runs in the operating system which clients talk to.
- Docker Client - The command line tool that allows the user to interact with the daemon. More generally, there can be other forms of clients too - such as Kitematic which provide a GUI to the users.

Docker Terminology

- Docker Hub - A registry of Docker images. You can think of the registry as a directory of all available Docker images. If required, one can host their own Docker registries and can use them for pulling images.