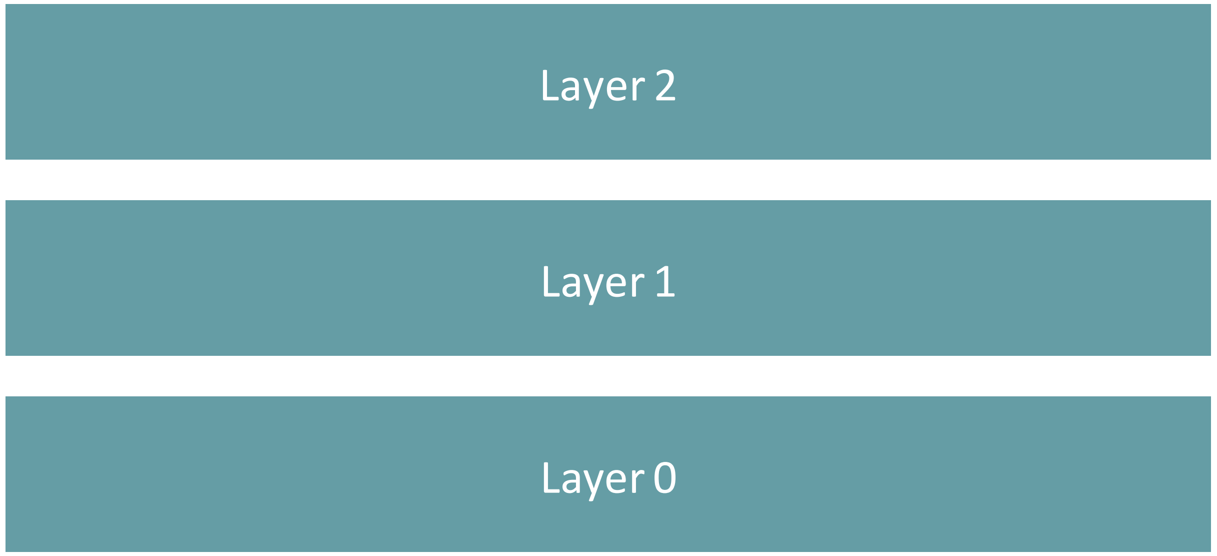
A Closer Look at Images and Containers

A Docker image is made up of filesystems layered over each other.



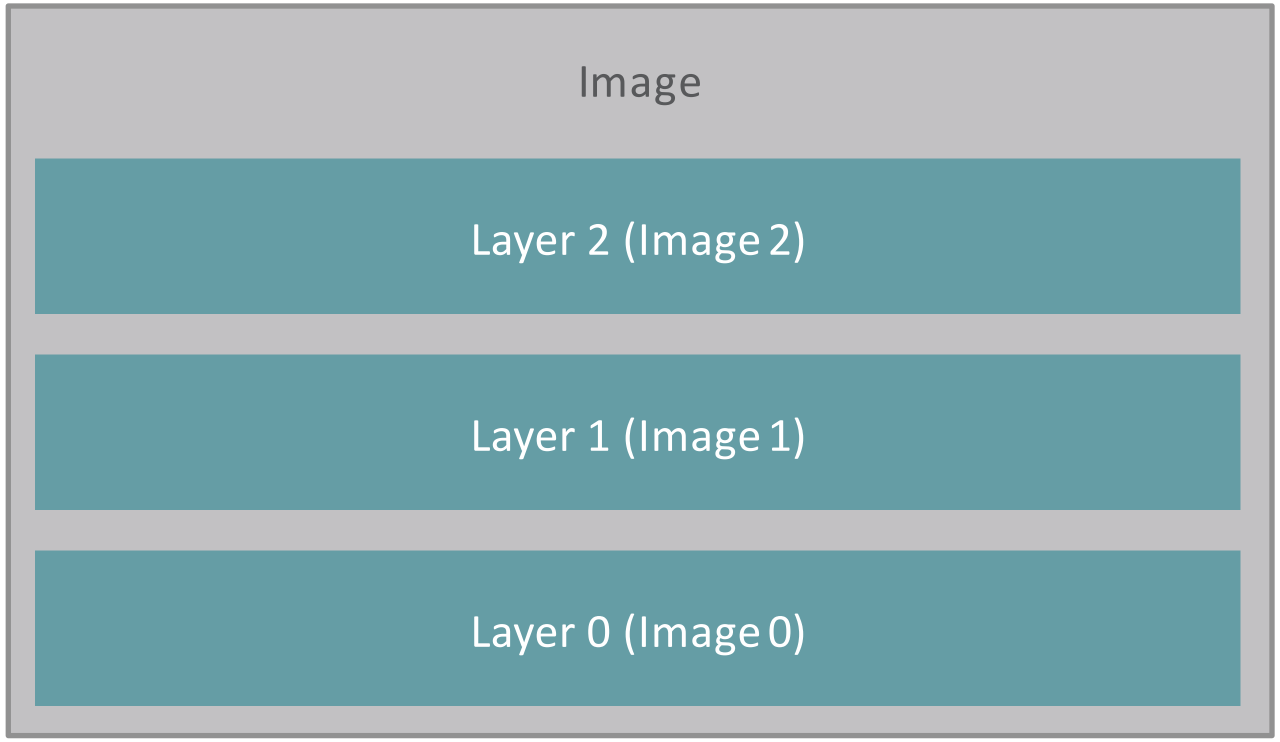
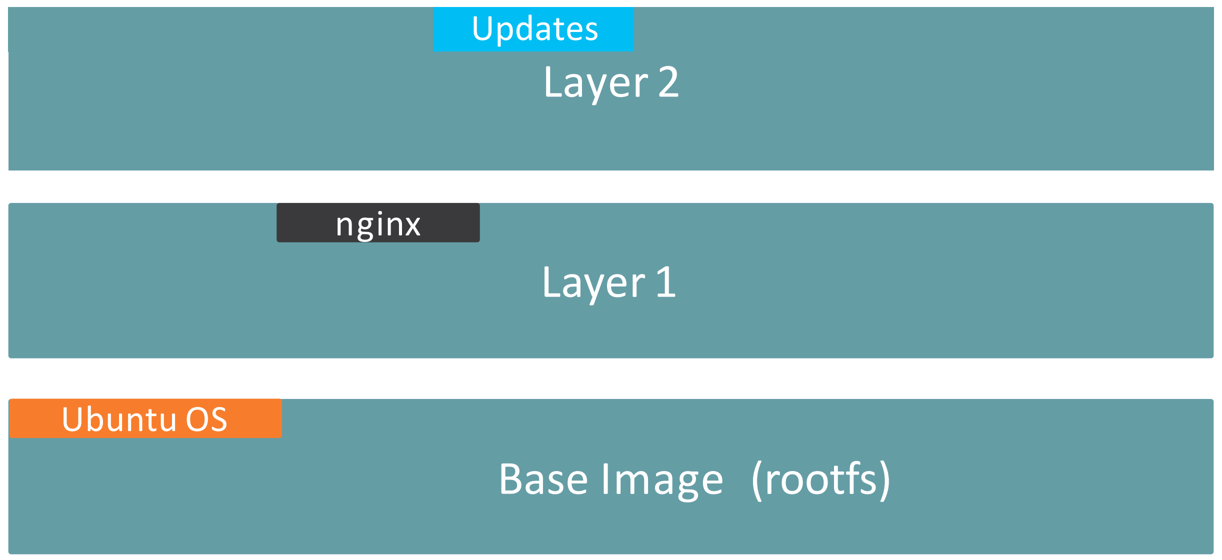
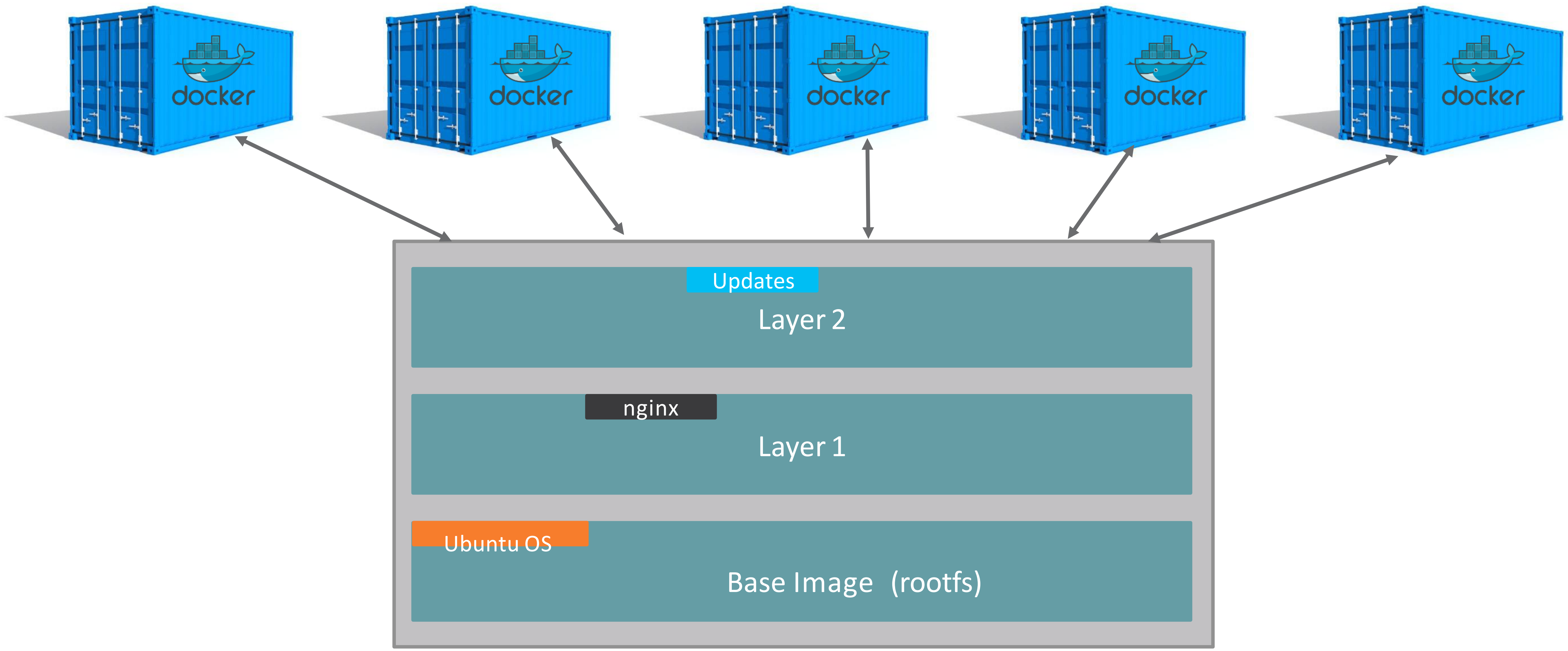
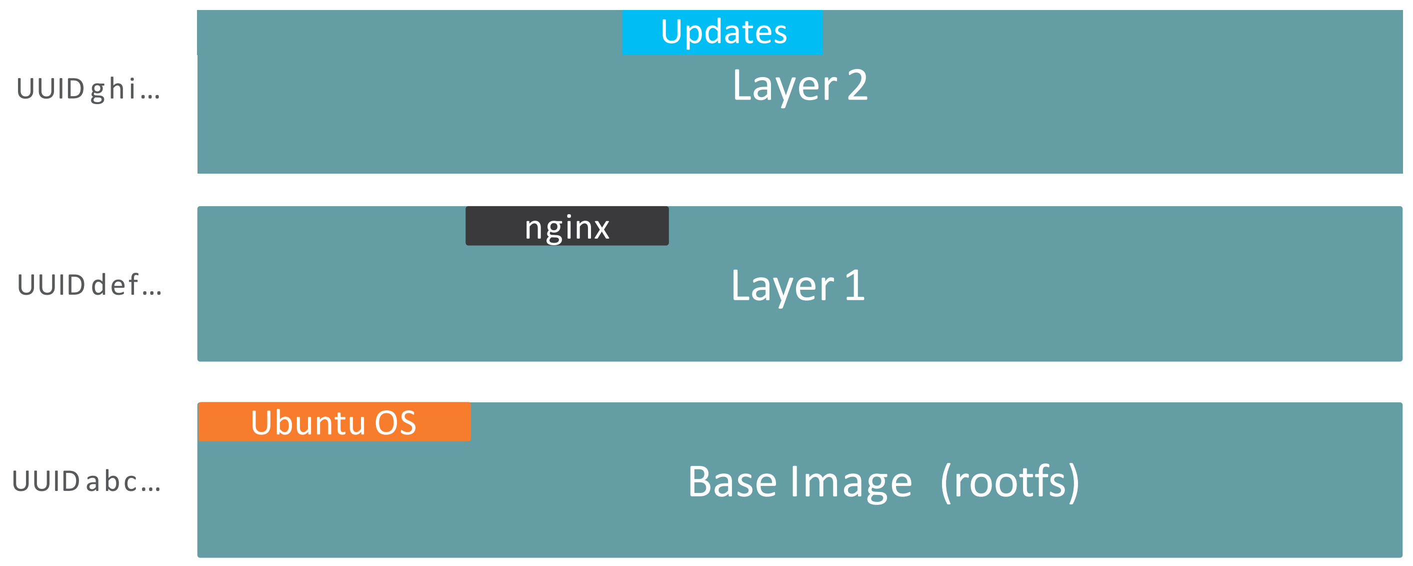


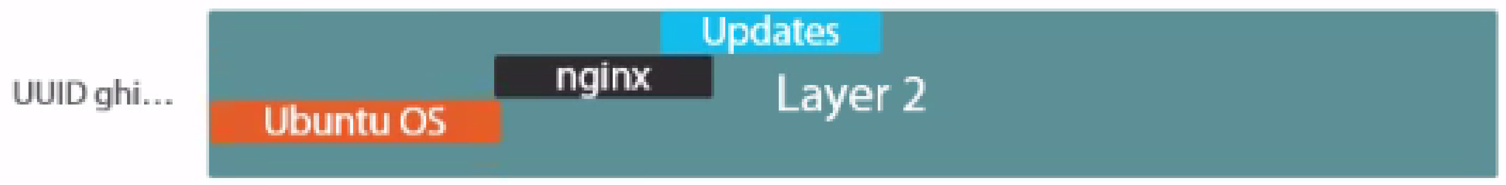
Image
Layer 2 (Image 2)
Layer 1 (Image 1)
Layer 0 (Image 0)


Updates
Layer 2
nginx
Layer 1
Ubuntu OS
Base Image (rootfs)



Single Image shared by multiple containers


Updates
UUID ghi…
Layer 2
nginx
UUID def…
Layer 1
Ubuntu OS
UUID abc…
Base Image (rootfs)

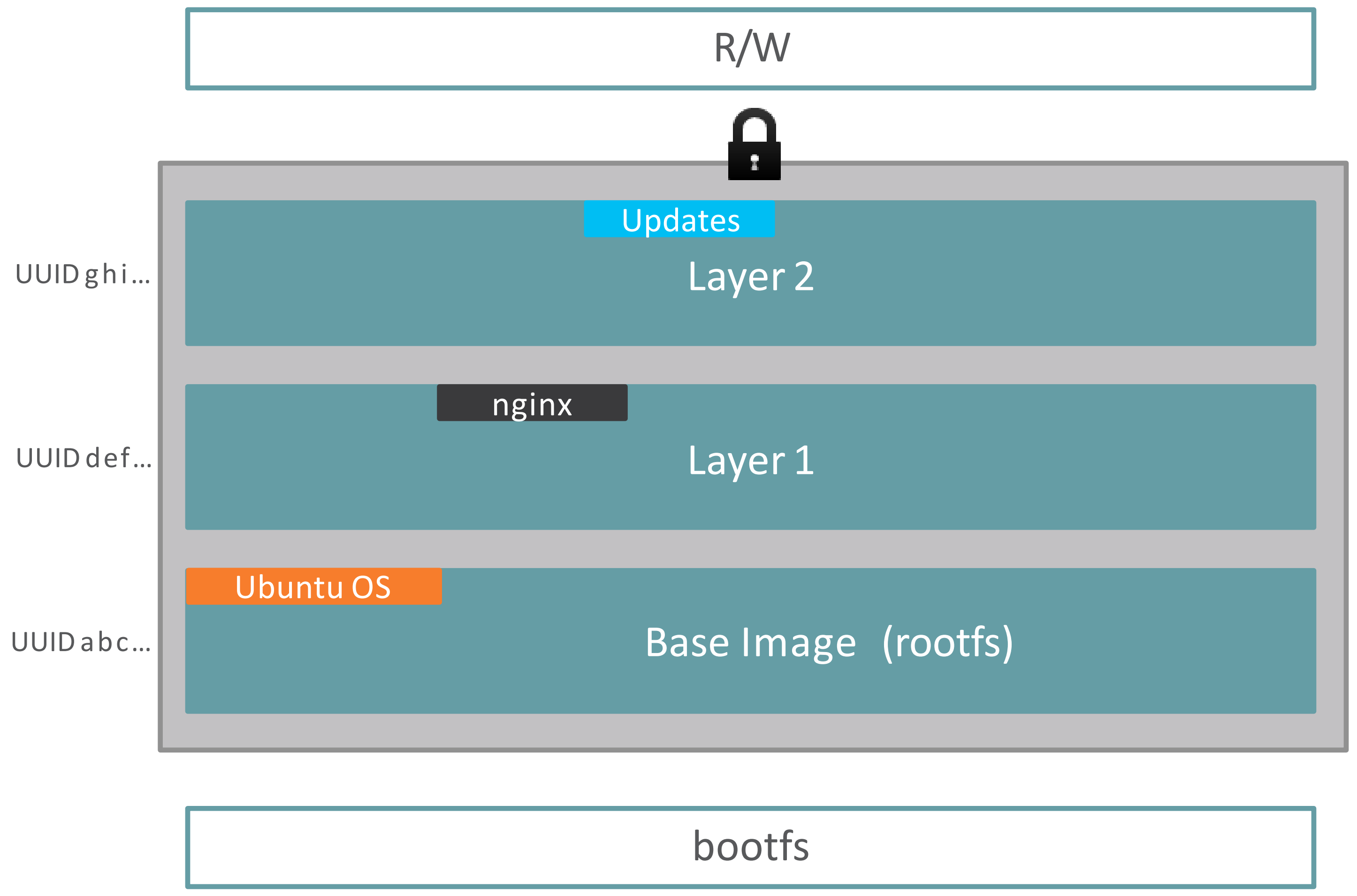


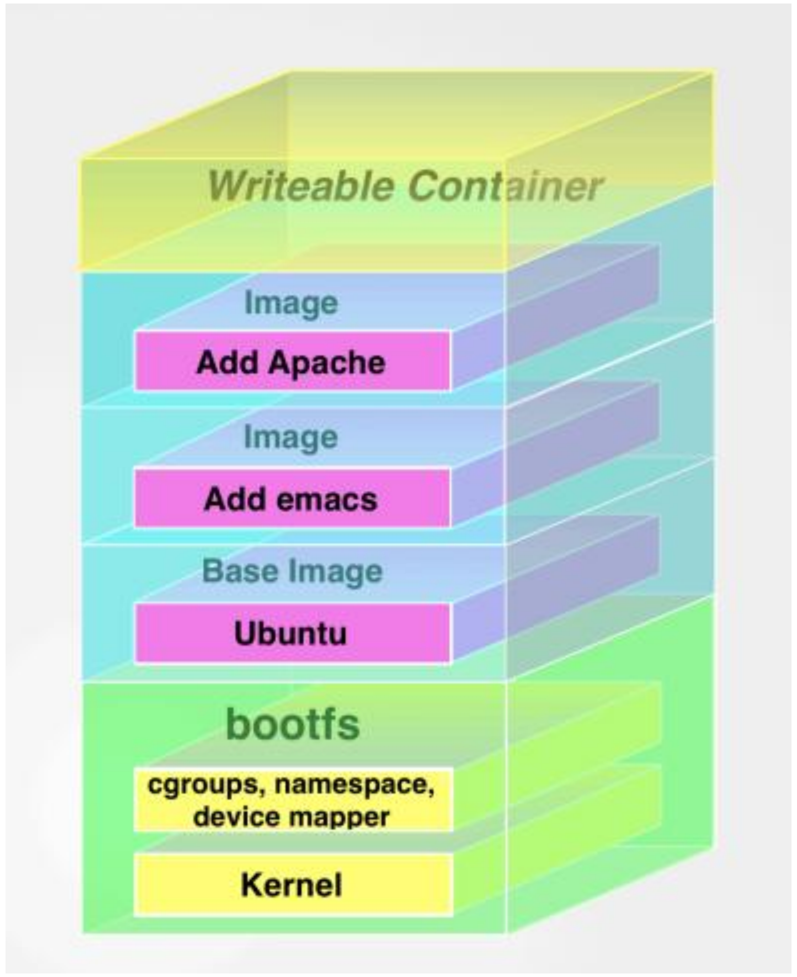
Higher layers always win when there is conflicts in Layers

Image Layering is accomplished through Union mounts

Union mounts Allows to mount multiple files systems over top of each other but combining all of they layers single view, giving the application/os single regular everyday filesystem.

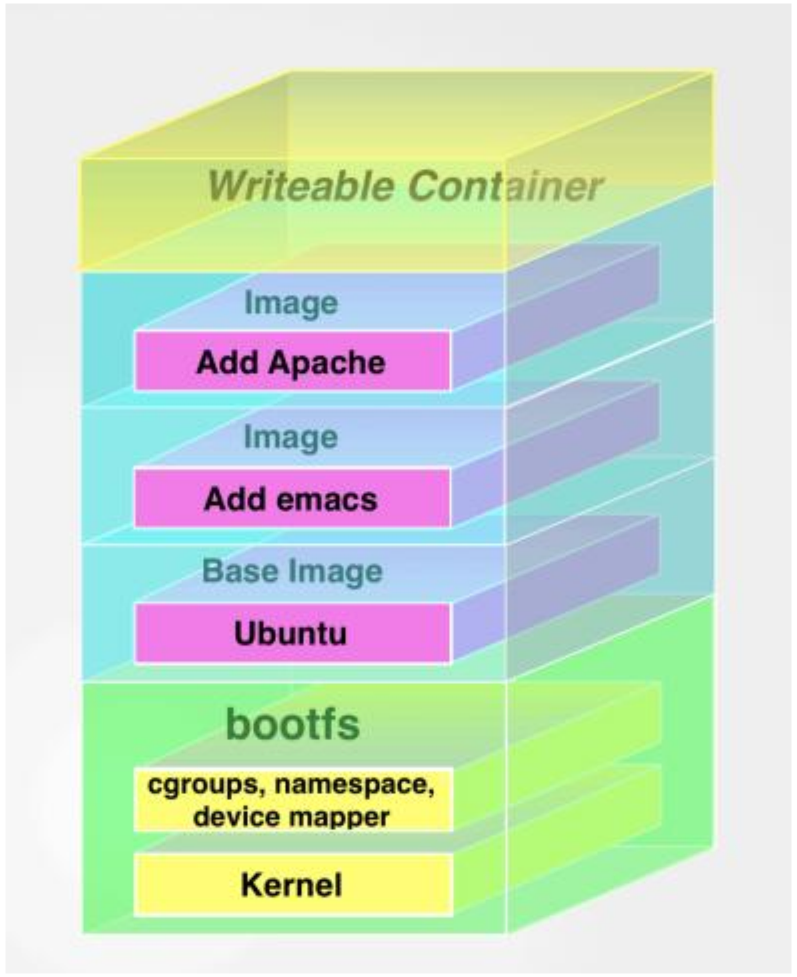
All layers are read only except top layers


R/W
UUID ghi…
Updates
Layer 2
UUID def…
nginx
Layer 1
UUID abc…
Ubuntu OS
Base Image (rootfs)
bootfs



bootfs - At the base is a boot filesystem, bootfs, which resembles the typical Linux/Unix boot filesystem. A Docker user will probably never interact with the boot filesystem. Indeed, when a container has booted, it is moved into memory, and the boot filesystem is unmounted to free up the RAM used by the initrd disk image.

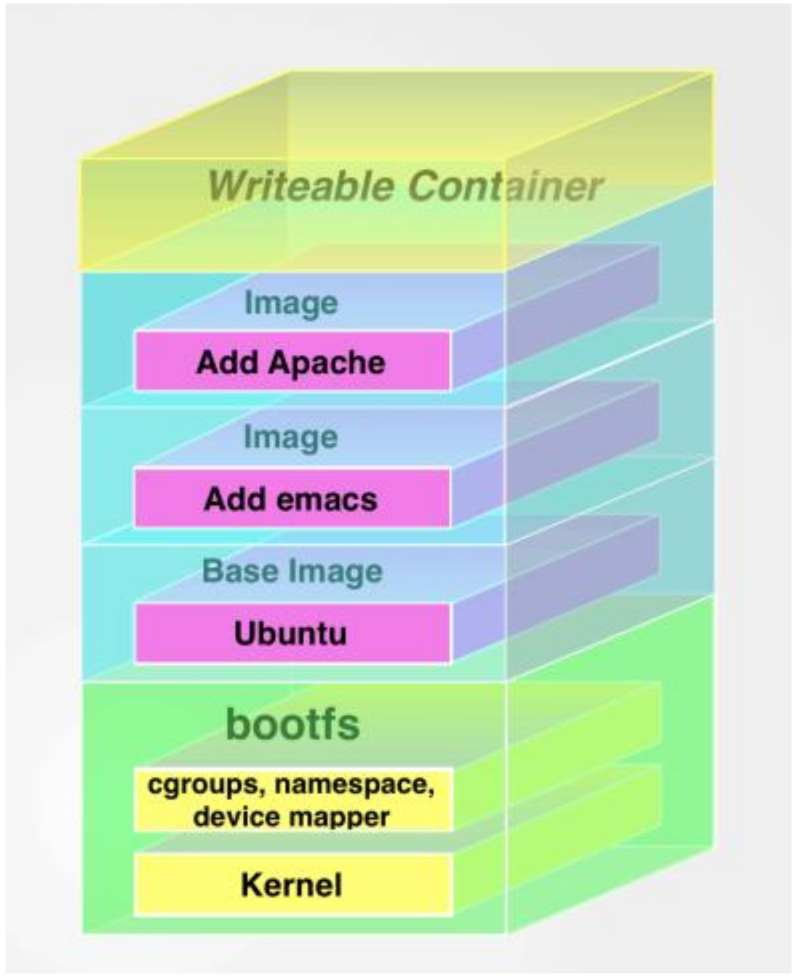
rootfs - Indeed, Docker next layers a root filesystem, rootfs, on top of the boot filesystem. This rootfs can be one or more operating systems (e.g., a Debian or Ubuntu filesystem).



In a more traditional Linux boot, the root filesystem is mounted read-only and then switched to read-write after boot and an integrity check is conducted.

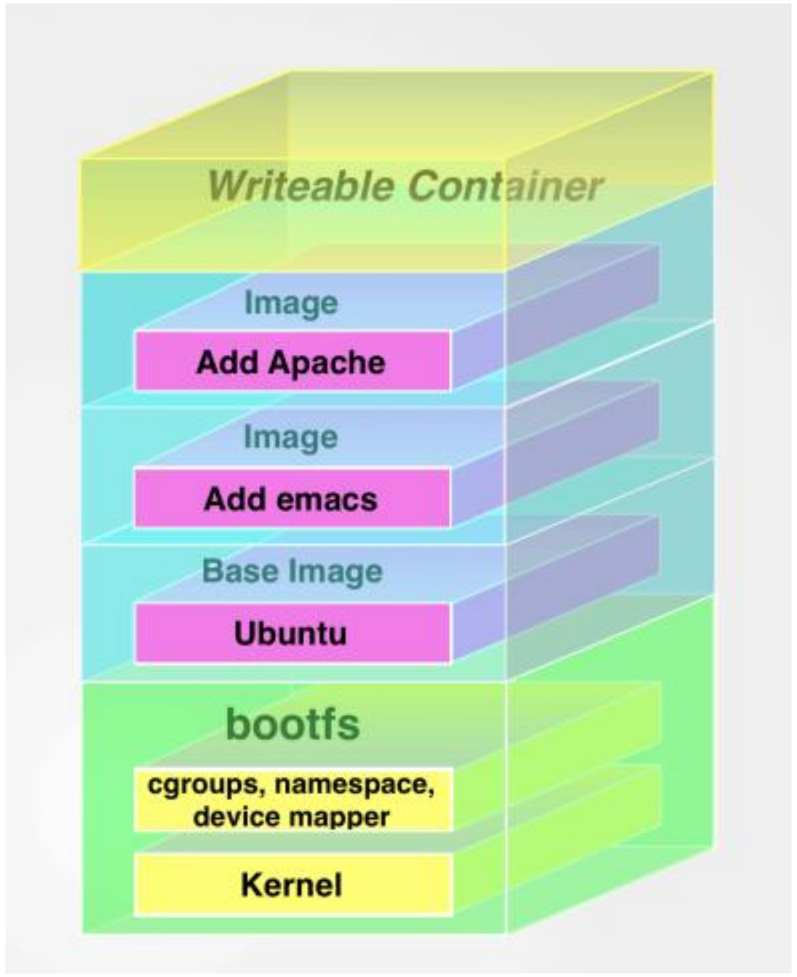
In the Docker world, however, the root filesystem stays in read-only mode, and Docker takes advantage of a union mount to add more read-only filesystems onto the root filesystem.

A union mount is a mount that allows several filesystems to be mounted at one time but appear to be one filesystem. The union mount overlays the filesystems on top of one another so that the resulting filesystem may contain files and subdirectories from any or all of the underlying filesystems. Docker calls each of these filesystems images.



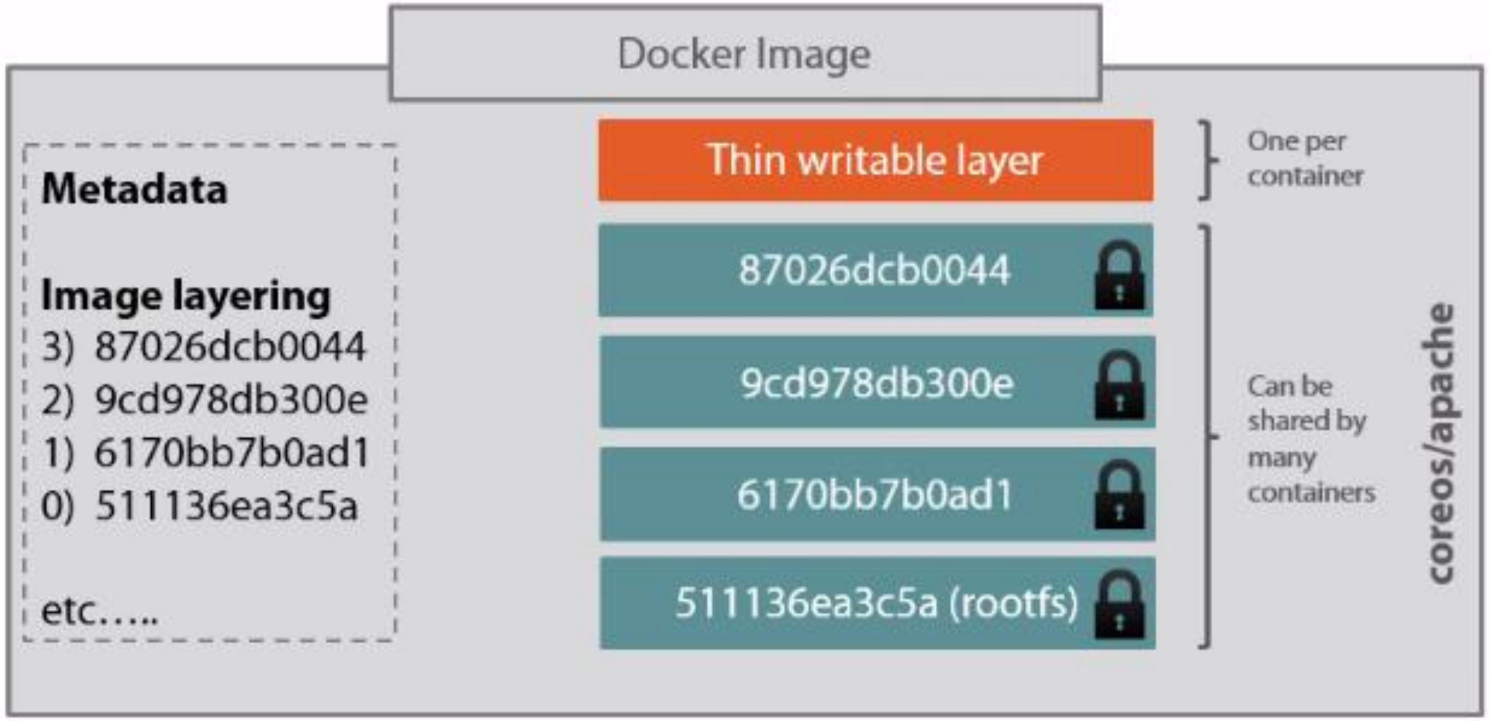
Images can be layered on top of one another. The image below is called the parent image and you can traverse each layer until you reach the bottom of the image stack where the final image is called the base image.

Finally, when a container is launched from an image, **Docker mounts a read-write filesystem on top of any layers below**. This is where whatever processes we want our Docker container to run will execute.



When Docker first starts a container, the initial read-write layer is empty.
As changes occur, they are applied to this layer; for example, if you want to change a file, then that file will be copied from the read-only layer below into the readwrite layer. The read-only version of the file will still exist but is now hidden underneath the copy.

This pattern is traditionally called “copy on write” and is one of the features that makes Docker so powerful.
Each read-only image layer is read-only; this image never changes. When a container is created, Docker builds from the stack of images and then adds the read-write layer on top.



Every **<u>container</u>** gets its own writable top layer

Some Commands

```
> docker images
```
List of the images downloaded locally

```
> docker pull coreos/apache
```
Downloading multiple layers of the images

```
> docker images --tree
```
How the images and layers are linked.

```
> ls -l /var/lib/docker/aufs/layers
```
List of layers stored locally

```
> cat /var/lib/docker/aufs/layers/layers-id
```
List out the lower layers listed

```
> ls -l /var/lib/docker/aufs/diff
```
Actual root files and directory core to base images.

Copy docker images from one host to another

https://goo.gl/4qmpXh

One Process per Container…

Usually…

One process per Container

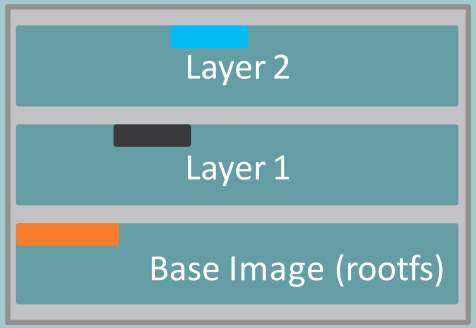
```
> docker run -d ubuntu /bin/bash -c "ping 8.8.8.8 -c 30"

> docker ps

> docker top cont-id

> docker ps -a
```

The container will "exit" when the process itself exits

Working with containers

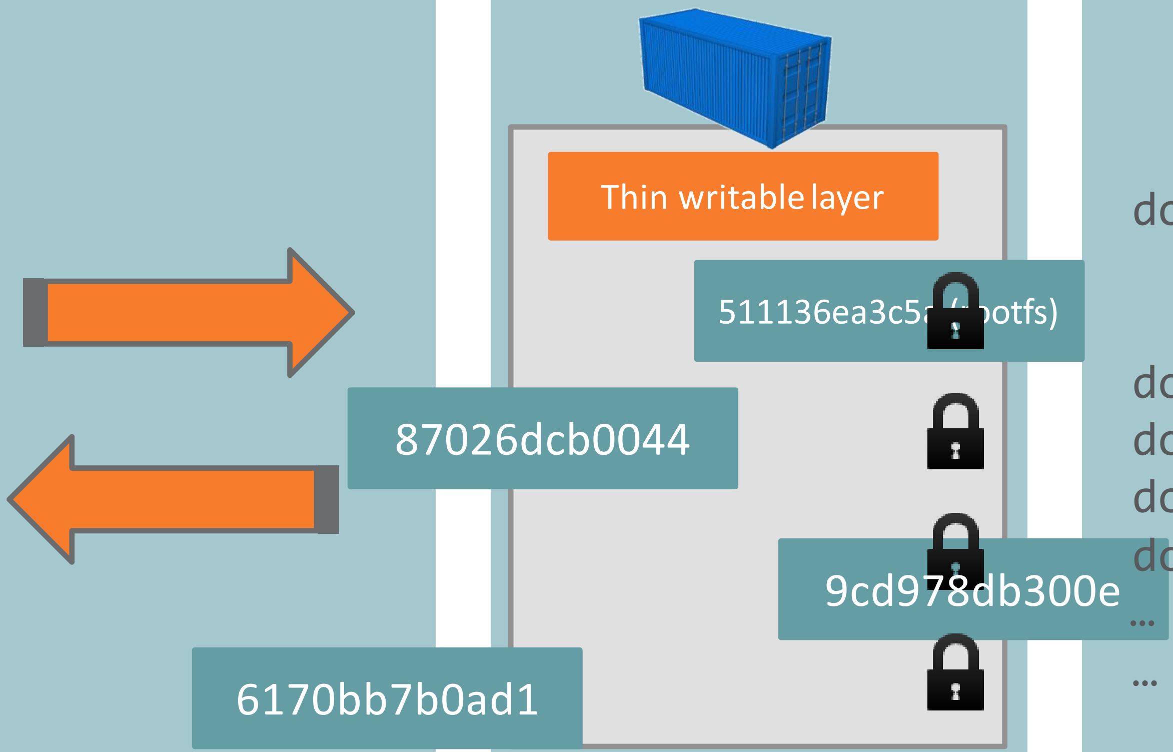
```
> docker run -d ubuntu:14.04.1 /bin/bash -c "ping 8.8.8.8"

> docker ps

> docker inspect cont-id

> docker attach name-of-container
& ctrl+c

> docker run --cpu-shares=256

> docker run memory=1g
```

Module Recap



(Union Mounts)

Image Layering



Importing and Exporting Images

Writable Layer

```
docker run
   ------cpu---shares=
   ------memory=
docker ps
docker top
docker inspect
docker attach
…
…
```

Containers and Processes