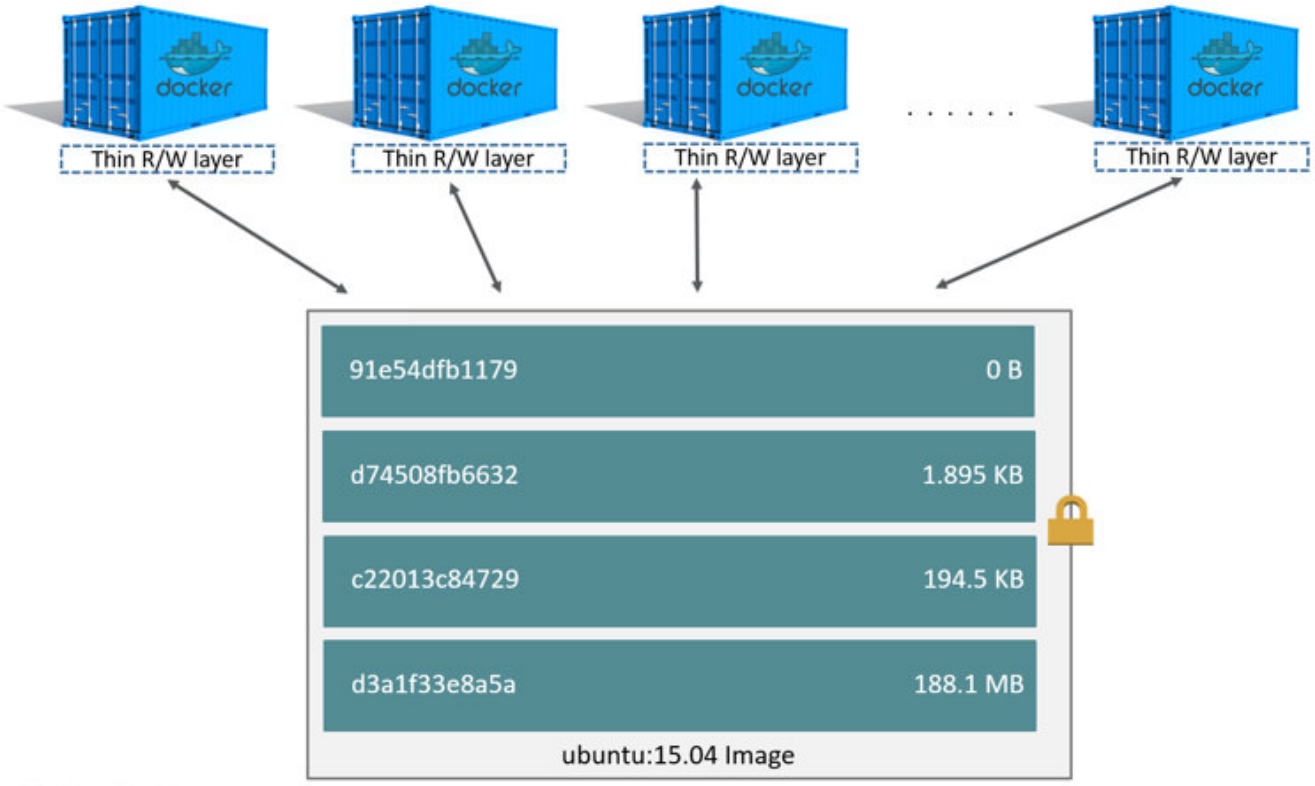
Chapter 4 - Layers

How Docker Images Work

Docker images work by stacking layers.

```
d1725b59e92d: Pull complete
Digest:
sha256:0add3ace90ecb4adbf7777e9aacf18357296e799f81cabc9fde470971e499788
Status: Downloaded newer image for hello-world:latest
```

The `hello-world` image we pulled down earlier had only one layer, but generally more useful images are made from 10 layers or more.



What Layers Get You

In the last chapter we discussed how we store images locally, but if we store two images that share a layer, we only store the layer once.

The main benefit is if we already have a layer, we don't have to rebuild it. That means, as we develop Docker images we only build the layers that have changed.

Getting Into A Running Container

Last time we ran a container, it outputted some text. But what if we want to go into one?

Start a container from the `alpine` image:

```
docker run -it alpine
```

You should be present with a prompt, you should be able to run commands like `ls` and `cd`

Drop out of the container:

```
exit
```

This will drop you out of the container back into the host shell.

What The 'i' And 't' Flags Do

The `-i` flag makes the container `interactive`. Effectively this means the `stdout` and `stdin` from your terminal are passed through to the container.

The `-t` flag allocates a `pseudo-tty` to the container, which is similar to how `ssh` works.

When you combine them, it's functionally similar to having `ssh`-ed onto a server.

Making Our Own Layers and Images

Start the container:

```
docker run -it alpine
```

Create a file:

```
touch myfile
```

Check the directory contents:

```
ls
```

Check there's now a file called `myfile` listed

Exit the container:

```
exit
```

List all the containers:

```
docker container ps -a
```

The top container will be the one you just exited, it is in a 'Stopped' state which we will cover in more detail later.

Copy the CONTAINER ID

Create a new Docker image:

```
docker commit <container id> myimage
```

Start a new container from the image:

```
docker run myimage ls
```

You should be able to see you file

Isn't There A Better Way?

Generally you won't use `docker commit`, but the command underlies Dockerfiles which are coming in Chapter 7. But understanding this mechanism is important for understanding how Dockerfiles work.