CLOUDADMIN

Nick Veitch has a whale of a time with containers

Docker

Virtualisation, fixed. That's the premise of containers in general and Docker in particular.

To get a better idea of what Docker is and how it can be of use to you, it is easier to start with the problems that Docker was created to solve. The project's logo of a whale as a container ship lays it out somewhat – Docker is containerisation, which as far as possible tries to be at once as generic as possible (so you can put whatever you want in your container) and as efficient as possible (so you don't need to put absolutely everything in your container).

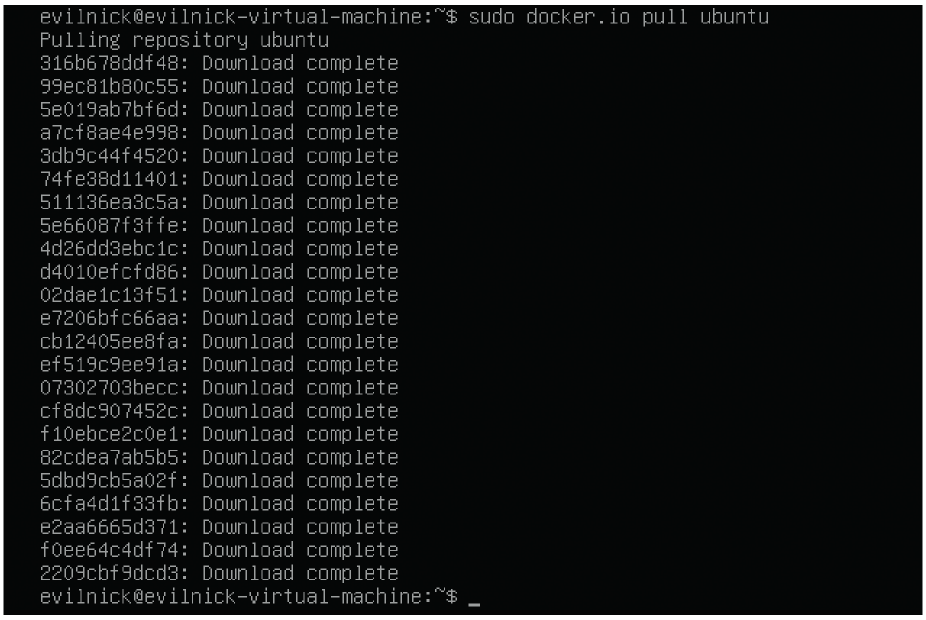
As we saw in LV003 (you missed it? really? You can subscribe you know, so we never have to have this conversation again) there are great benefits to containers over VMs (see the boxout, right). LXC, or Linux Containers, is far and away the best implementation of such a system on Linux, and recently reached a stable 1.0 milestone. Aside from being wildly useful for development work in general, LXC is also the technology underlying Docker.

If LXC is so great at containers, you may be wondering why you need Docker. Well, of course, you don't need it just to use containerisation. However, for the specific use case of developing and deploying applications, Docker adds several very helpful features on top of LXC:

- **Versioning** Docker includes version-control capabilities for tracking different revisions of a container, and doing all the usual cool stuff such as generating diffs between revisions, rolling back to previous versions and such. It also means you can suck down new versions of a container from upstream (which includes only the deltas, so no huge files) if you are building on top of someone else's work.
- **N Stacking** Docker intelligently 'stacks' components, and re-uses them where possible. For example, you could build an application on top of Apache and Ubuntu, which could contain three separate components, but if you changed only the application, the other two stacked components would remain the same. This makes for smaller bits to transfer around, and also saves resources if you are creating/deploying similar things.
- **N Sharing** There is a public registry (**http://index.docker.io**) of containers already created by others, so you don't have to do everything from scratch all the time.

N Integration Docker provides an API, which means that other tools, particularly devops tools, can easily interact with your containers. You may want to imagine Docker as a sort of portable, version-controlled build system.

But let's not waste time – let's see how we can get it to work for us.



```
evilnick@evilnick-virtual-machine:~$ sudo docker.io pull ubuntu
Pulling repository ubuntu
316b678ddf48: Download complete
99ec81b80c55: Download complete
5e019ab7bf6d: Download complete
a7cf8ae4e998: Download complete
3db9c44f4520: Download complete
74fe38d11401: Download complete
511136ea3c5a: Download complete
5e66087f3ffe: Download complete
4d26dd3ebc1c: Download complete
d4010efcfd86: Download complete
02dae1c13f51: Download complete
e7206bfc66aa: Download complete
cb12405ee8fa: Download complete
ef519c9ee91a: Download complete
07302703becc: Download complete
cf8dc907452c: Download complete
f10ebce2c0e1: Download complete
82cdea7ab5b5: Download complete
5dbd9cb5a02f: Download complete
6cfa4d1f33fb: Download complete
e2aa6665d371: Download complete
f0ee64c4df74: Download complete
2209cbf9dcd3: Download complete
evilnick@evilnick-virtual-machine:~$ _
```

Hurrah for Ubuntu! Other versions of Linux are available, honest

Sudo or not sudo

Newer versions of Docker use sockets to handle networking, which means on many Linux variants, having to use **sudo** to run Docker commands. You can get around this by creating a group for Docker and giving it the necessary privileges – check out the Docker page for more info

Docker

1 Install docker

Find out if packages are available for your distro or grab the source code (this is a bit messy, as it's written in Go) from the website. For Ubuntu (Trusty) you can just:

```
apt-get install docker.io
```

On some distros (like Ubuntu) the command and package are known as **docker.io** to avoid confusion with the existing KDE docker applet (in which case, use **docker.io** instead of **docker** in the following steps). You may also need to use **sudo** to run the **docker** command (see box)

2 Fetch some images

As mentioned before, there is a collaborative index of images, and also some official images for things you might want to use. You can 'pull' these images to download them locally for use:

```
docker pull ubuntu
```

This will fetch a bunch of Ubuntu images. You can see what you have by running:

```
docker images
```

This will return a list of image tags, their ID and some other useful info. The tags make it easier to tell what images you have, but the ID number is important. This is a generated UUID that you will be able to use to identify versions (like in GitHub).

3 Run an image

To run something in a container, we use the syntax:

```
docker run <image id> <command>
```

the image ID you give can usually just be the first five characters – enough to distinguish it from the others. For example, the ID for the 'trusty' image we downloaded might be **99ec81b80c55**, but I could run:

```
docker run 99ec cat /etc/lsb-release
```

Which would fire up the container and run the command. The output in this case would tell me what version of Ubuntu I was running.

You will notice that containers are process driven. The command ran, there was output and then it returned, so the container stopped running. We will see how to keep them running in a bit, but let's talk about versioning first. Try this:

```
docker run 99ec apt-get install -y vim
```

This installs Vim (unreasonably absent from the minimal image) onto the container. the **-y** switch, by the way, stops it from asking silly questions. So, the persistent storage is now different. If I were to run **docker ps -l** now, it would give me a different ID for my image, as well as telling me the last command I ran. If I want to permanently keep this image, I would commit these changes to a local repository like so:

```
docker commit 079e13  evilnick:vim
```

Now the Vim image is stored in my repository and I can run that any time I like.

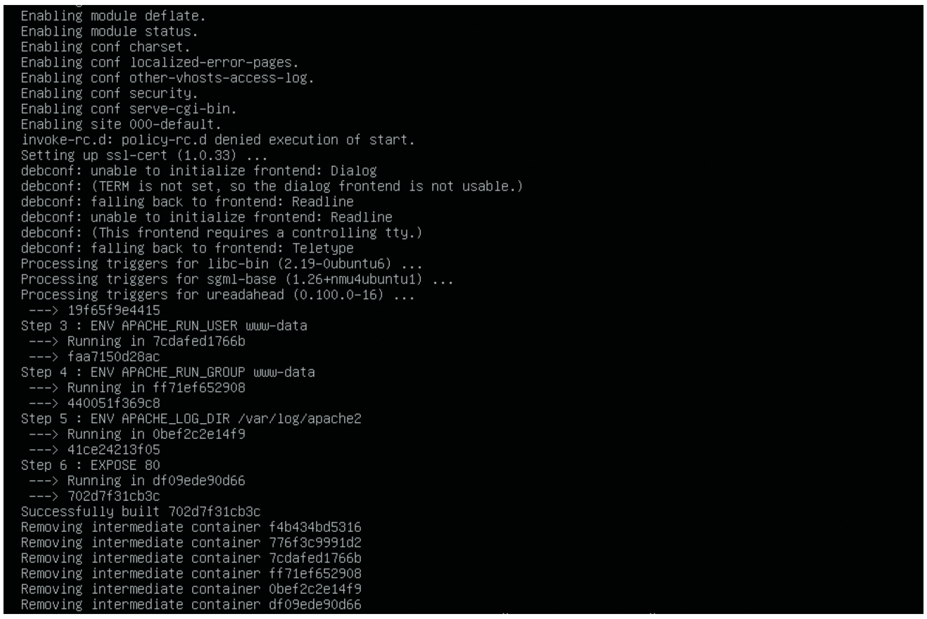
4 Build a docker file

As well as running commands as instances, you can make a "Docker file" which is simply a recipe for things for the container to do. If I wanted to install the Apache 2 server on the image I just created, for example, I might create a Docker file like this

```
FROM evilnick/vim

RUN apt-get update
RUN apt-get install -y apache2

ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data

EXPOSE 80
```

These commands are pretty self-explanatory. The **FROM** specifies a source image. The lines that begin with **RUN** execute those commands. The **ENV** directive causes environmental variables to be set in the container. **EXPOSE** tells docker that port 80 should be opened on the container (this will be mapped to a random port on the host). To build this I run:

```
sudo docker build -t 'evilnick:vim-apache' .
```



```
Enabling module deflate.
Enabling module status.
Enabling conf charset.
Enabling conf localized-error-pages.
Enabling conf other-vhosts-access-log.
Enabling conf security.
Enabling conf serve-cgi-bin.
Enabling site 000-default.
invoke-rc.d: policy-rc.d denied execution of start.
Setting up ssl-cert (1.0.33) ...
debconf: unable to initialize frontend: Dialog
debconf: (TERM is not set, so the dialog frontend is not usable.)
debconf: falling back to frontend: Readline
debconf: unable to initialize frontend: Readline
debconf: (This frontend requires a controlling tty.)
debconf: falling back to frontend: Teletype
Processing triggers for libc-bin (2.19-0ubuntu6) ...
Processing triggers for sgml-base (1.26+nmu4ubuntu1) ...
Processing triggers for ureadahead (0.100.0-16) ...
 ---> 19f65f9e4415
Step 3 : ENV APACHE_RUN_USER www-data
 ---> Running in 7cdafed1766b
 ---> faa7150d28ac
Step 4 : ENV APACHE_RUN_GROUP www-data
 ---> Running in ff71ef652908
 ---> 440051f369c8
Step 5 : ENV APACHE_LOG_DIR /var/log/apache2
 ---> Running in 0bef2c2e14f9
 ---> 41ce24213f05
Step 6 : EXPOSE 80
 ---> Running in df09ede90d66
 ---> 702d7f31cb3c
Successfully built 702d7f31cb3c
Removing intermediate container f4b434bd5316
Removing intermediate container 776f3c9991d2
Removing intermediate container 7cdafed1766b
Removing intermediate container ff71ef652908
Removing intermediate container 0bef2c2e14f9
Removing intermediate container df09ede90d66
```

Building a Docker file doesn't take long, but the results can last a lifetime. Probably.

5 Run containers

To run a container interactively, you just need to pass a few extra flags to Docker:

```
docker run -t -i 992e7  /bin/bash
```

This will run Bash on the container, and stay connected until you exit. You can use this shell to make further changes to your container if you like.

Going further

There is of course much more you can do with containers – we have just outlined the basic mechanics here. Fortunately, there is a great deal of good documentation on the Docker website. I heartily recommend the interactive tutorial on building Docker files – **www.docker.com/tryit/**.

Containers vs VMs

For longer than you think, people have worried about consistency in platforms and about getting software to run reliably on an underlying platform that it wasn't written specifically to work on. In fact, that is how Virtual Machines came to be invented, not just because it was cool to fool your laptop into thinking it was a Macbook. IBM invented the idea of the modern VM back in the 60s as a way to test improvements to their hardware and OS, a legacy that lives on in their highly effective z/VM OS for mainframes.

A VM though requires overheads. Most implementations on Linux require processors designed specifically to support virtualisation. They also require dedicated resources – you have to allocate things like disk space and memory to a VM, and if your solution requires other specific hardware, you need to virtualise that too. Perhaps still the most pervasive annoyance of VMs is that you need a complete OS image for them to run – if you want to ship or store your software pre-loaded in a VM, you have to store everything else too.

Containers, on the other hand, view the problem differently. For shipping, testing and running software applications, the application is the only thing that needs to be in its own place (as long as the other parts can be standardised). Container systems such as LXC (which is used by Docker) separate out the bits you need to be separate (the application's view of where it is running) while retaining as much of the underlying system as possible (the kernel, resources and such).The disadvantage of containers? You can't run a completely different OS, and the shared resources mean that multiple containers can end up fighting over them.