CLOUDADMIN

Running make-believe boxes is virtually compulsory, suggests Nick Veitch.

Virtualisation: the options

There are many shades of hypervisor.

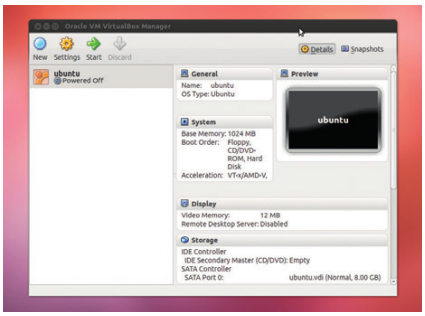
Virtualisation is a key technology that helped give birth to the modern cloud as we understand it. It helps run the services on the cloud and often helps build clouds too. But virtualisation is also important to developing tools to run clouds. As our foray into the 'dev' part of devops has already led us to look at how continuous integration is used (see LV002) we should also take a look at the virtualisation technologies commonly in use, their alternatives, and how they may differ from your experience on the desktop.

KVM

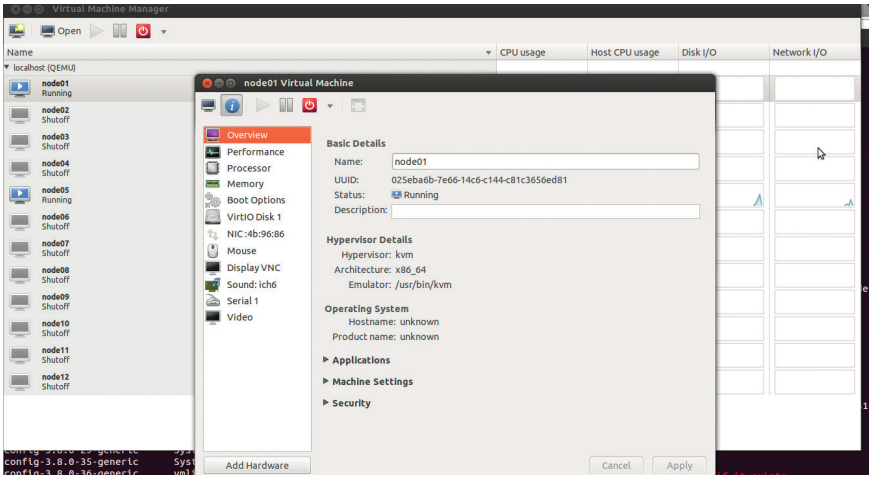
KVM works on top of Qemu, so for the purists, when we talk about KVM here we mean KVM/Qemu. KVM is a Linux-only virtualisation technology, parts of which are included in the mainline kernel. The software relies on kernel modules to interface with the host CPU's virtualisation extensions – as such it will only run on CPUs that support (for example) Intel VT or AMD-V extensions (there is also an ARM port).

Popularity of KVM has not been driven by the desktop – it still lacks a lot of the snazzy configuration tools of VirtualBox – but it is very very popular for 'serious' use due to factors such as kernel integration and the unambiguous open source nature of the code (not to mention that it works very well).

Although it lacks somewhat in terms of graphical tools, VMs are controllable from the command line (and therefore, also easily by scripts and other software which uses the libvirt API – see our tutorial on page 94) to a greater degree than pretty much anyone could ever want. As well as being a great tool for development, it is also widely used for spinning up VMs within clouds (eg OpenStack).



A popular choice for desktop users, Oracle's VirtualBox has some good things going for it in the developer space too.



Virt-manager is a useful graphical front-end for KVM, but you should really familiarise yourself with the virsh commandline tools, especially if you need tricky network setups.

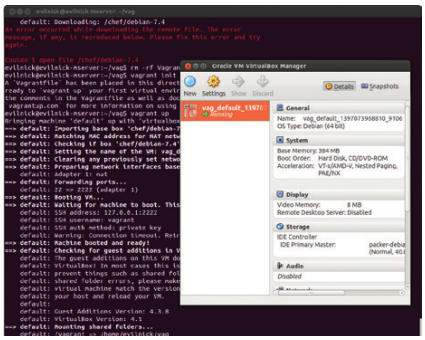
VirtualBox

VirtualBox came to prominence by virtue of it being a very featureful, well performing VM hypervisor that worked cross-platform and had an easy to understand management interface. The software has a colourful history – the original company that created it, Innotek, was acquired by Sun Microsystems, before many parts of the disintegrating Sun empire were snapped up by Oracle. As a largely open source project (there is a non-open source version, which makes use of proprietary device drivers for

"Vagrant was originally developed to work with VirtualBox, but a system of plugins enable it to work with numerous hypervisors."

VMWare

No, we didn't forget about VMWare. Although it has been in the vanguard of virtualisation technologies for some time, VMWare is not open source. Although that doesn't exclude it from consideration in the world at large, it does tend to make it less relevant to the emerging cloud platforms, and certainly a little out of the scope of this FLOSS-loving publication.



Vagrant is an effective tool for provisioning VMs and working collaboratively.

graphics, and the open source version seems to be stuck on an LGPLv2 licence) it sits a little ill at ease in the Oracle stable.

Nevertheless it is a mature and competent environment for running VMs. It relies a lot on paravirtualisation – special drivers that allow a more efficient throughput of data to and from the host OS. These do bring performance benefits, but rely somewhat on the co-operation of the guest OS, so if you are running custom kernels on strange distros you may not reap the full benefits.

It does have the huge advantage of also running on Mac OSX and Windows (and even Solaris), which can be beneficial in some collaborative environments.

Vagrant

Vagrant isn't a virtualisation engine, but it is most definitely worth talking about. The idea behind vagrant is that it becomes a sort of meta-manager for virtualised instances. Vagrant was originally developed to work with VirtualBox, but a system of plugins enable it to work with numerous hypervisors. Once you have installed Vagrant, you can fetch virtual machine images (which in Vagrant terms are known as 'boxes') and use them to bring up virtual machines.

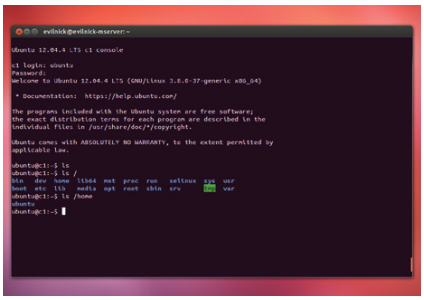
You may ask yourself "What on earth is the difference between this and just creating a machine in VirtualBox)?". The answer, at least at the system level, is "not much". Start up your box, and it behaves pretty much like any other VM you have initiated with VirtualBox (or Qemu/KVM if you use that as the back-end).

The real difference is in provisioning. If you spend your life testing software, bringing up a clean VM is only part of the day-to-day grind. You then have to prepare that system for use. This ranges from the mundane installation of dependency packages to the more annoying repeatedly setting up options like host configuration or adding SSH keys so you can access the VM you created.

Yes, you can do this once in something like VirtualBox and create a snapshot image. Before you know it though you have half a dozen different snapshots that all differ in subtle ways, and aside from taking up loads of disk space, it can get pretty confusing. By using Vagrant to provision systems from a common set of boxes, you can reduce changes to your install to just changing some options in the Vagrant file that the software uses to bring up the VM. Of course, you can still create your own boxes, and there is a new service specifically for sharing those images in the cloud, so collaboration is much easier than trying to shift gigabytes of VM filestorage around.

Linux containers (LXC)

LXC is not a hypervisor for virtual machines. It is better than that; well, at a lot of things anyhow. LXC uses some very useful user-mode kernel features to containerise an implementation of a Linux OS – think chroot, but taken to extremes. Like a virtual machine, the container is able to carry out its business independently of the host OS, even to the point of running a different OS entirely. What you get is a self-contained running instance that is separate from the host OS, but which can dynamically share resources – there is no need to pre-allocate RAM and disk space for example, because the LXC container will simply consume what it needs just like any other process. LXC also plays nicely with libvirt, so you can use the same tools to control containers as you may use with VMs (though to be fair, there are some peculiarities of LXC that aren't adequately addressed by libvirt, but you can also use the comprehensive LXC command line tools).



Linux containers are not a VM, and that is the whole point!

As there is no CPU or hardware virtualisation, there is a much lower overhead to running containers than VMs – there are no virtualisation layers to go through, so things like file access are much faster, and the scalable resources also mean that more efficient use can be made of hardware.

There are of course, disadvantages to using containers. For a start, you can only run Linux-based containers. It can even sometimes be tricky to run completely different distributions without additional tinkering. Added to that, the lack of virtualisation also means no virtual hardware – which can be a pain when it comes to configuring networking. For simple networks, LXC makes use of a bridged driver, which means the container can access an external network through the host's network setup, but complicated VLAN topologies become more troublesome. There can also be some nagging suspicions that what may work in a container could behave differently on real hardware. Gosh, not that anyone has real hardware!

LXC arguably has the most mature support on Debian and derivative distros, and is well worth experimenting with.

The virtual future

It sometimes seem mad that we run an OS on virtual machines through cloud software, which itself can be running on virtual machines, which themselves can be running on the very same OS. Don't think about it too much, it hurts. The point is that VMs (and containerisation) provide the essential flexibility of cloud implementations, and as the overhead associated with them gets smaller, they become more and more important enablers of future technologies.

Docker

Missing from this VM get-together is Docker. Like LXC, Docker is a containerisation solution, and we have left it out because we will be having a very detailed look at it next issue!

Further reading

- Why Vagrant? **https://docs.vagrantup.com/v2/why-vagrant/index.html**
- Stephane Graber's LXC primer **https://www.stgraber.org/2013/12/20/lxc-1-0-blog-post-series**
- Using KVM with Ubuntu **https://help.ubuntu.com/community/KVM**
- VirtualBox homepage **https://www.virtualbox.org**