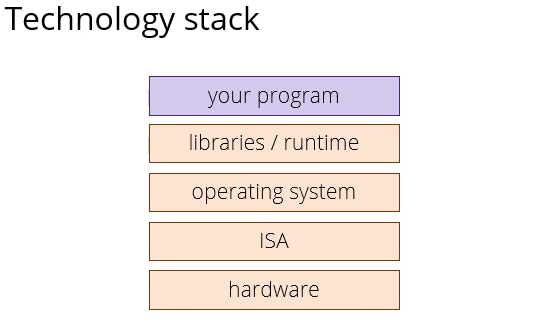
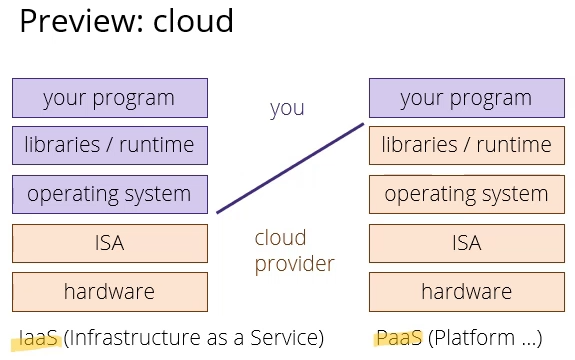
**Vagrant**

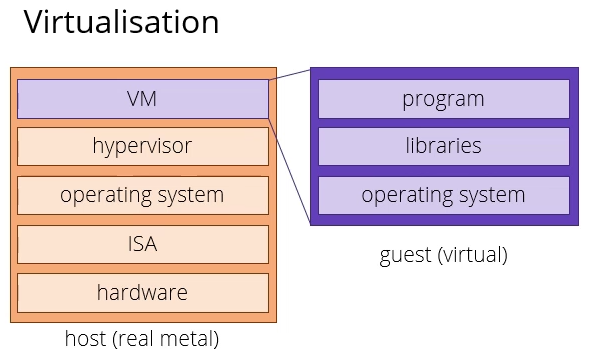
* A vagrant is an open-source tool that allows you to create, configure and manage boxes of VMs through an easy to use command interface
* A layer of software installed between a virtualization tool (e.g. VirtualBox) and a VM
* Here talking about VMs and the vagrants we’re going to be using to manage them
* To write a program, you (probably) need to use libraries (e.g. stdio.h)
* Below the libraries there will be an OS which deals with things like making sure you have access to a file system in the first place, network, etc.
* Below that have ISAs (Instruction Set Architecture) – interface between instructions and hardware
* Below that is the actual hardware
* Parts of what virtualisation is for is to make it easier to manage this stack



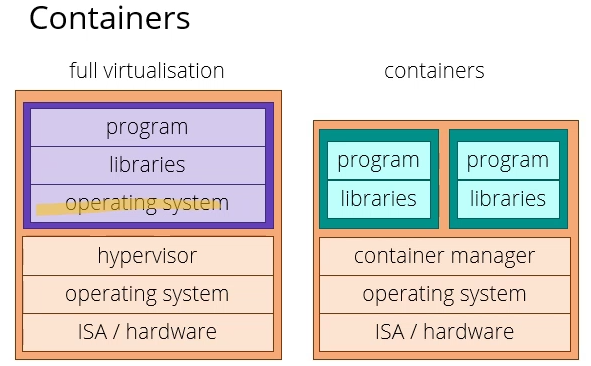
* Different kinds of cloud offerings
  + IaaS (Infrastructure as a Service) cloud provider gives you a bare machine (has hardware and some way to connect to it), you provide your own libraries, operating system, and programs
    - Being given a bare machine where only difference is that machine is sitting somewhere in a data centre in the cloud
  + PaaS (Platform as a Service) cloud provider provides everything you need, whole platform except your program
    - Just install your program and run



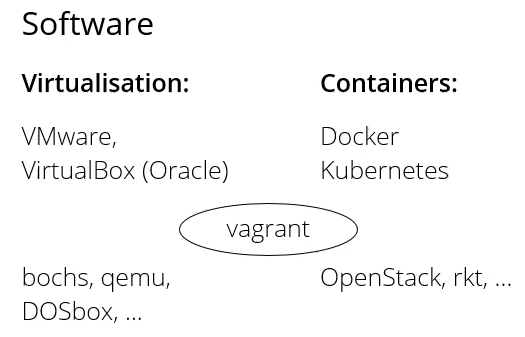
* Virtualisation is one of many techniques cloud providers use to provide this service to you and to provide security so users can’t access each other’s programs
  + Virtualisation is at the very top of a software stack on a real (host) machine
  + Hypervisor = emulates a different machine
  + VM is a program linked to hypervisor
  + Inside VM (guest) machine you could be running a completely different set of libraries, OS, and programs



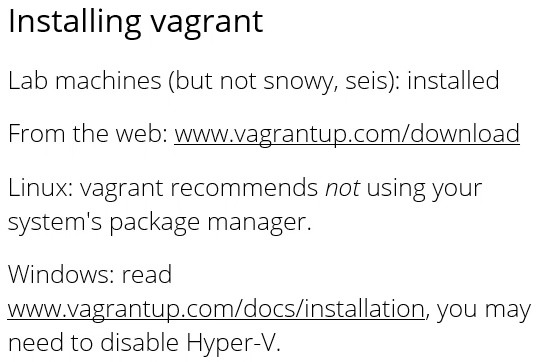
* Reasons to virtualise:
  + To emulate a different software stack (e.g. to run a different OS)
  + VMs can help you produce a different build environment
    - If you’re building some software, you want to make sure that it can run in the actual environment you want it to run in
    - Can have a process called build integration that builds and tests and runs your code in a particular environment
  + Cost/scalability – can deploy as many different versions of your software on as many VMs in the cloud as you fancy paying for
    - Can scale up really quickly
    - Virtualisation means no-one has to run around a data centre installing machines, can just do virtually
* Containers are a form of virtualisation provided by certain OS



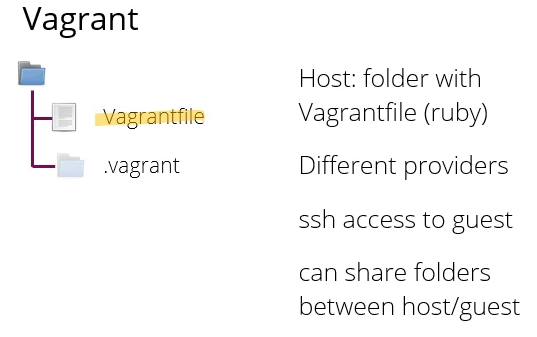
* In full virtualisation in the guest you have a full OS – a full software stack with everything simulated except hardware (which is simulated by the hypervisor)
* Modern OS and chip sets have features to make virtualisation a lot faster than it would be if you had to simulate a full machine
  + Hypervisor will pretend you’re running at almost the same speed as on the native machine
* If you want something even more lightweight, a container is basically a Linux kernel feature
  + Allows you to run a program with a set of libraries as if it were the only program on the machine
  + You have isolation between programs
  + You could be running several different programs, each of which needs a slightly different version of Python or some library, and each one is living in a container, so you don’t have to manage dependencies
  + If you weren’t using a container and you had programs that wanted different versions of Python it would be more complicated, but a container can just see each program and libraries as the only ones running on your machine
* A third, even more lightweight level called virtual environments
  + In e.g. Python
  + Copies a set of programs and libraries into a folder
  + You can run Python and say take this folder as if it were the whole system
* In this unit using full virtualisation



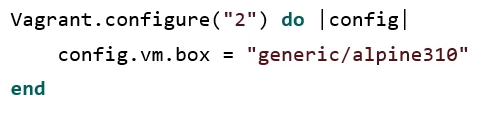
* For VMware and VirtualBox offer a hypervisor, can download a free version
* We’re using a tool called vagrant, which is really a meta-tool to manage these other tools
  + Helps you fetch and run boxes
  + Boxes can be run both as VMs and containers
  + We’re going to be using VM version



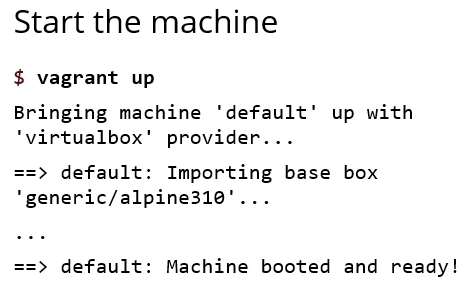
* To make a machine after installing vagrant, make a folder somewhere within which you make a file that has to be called Vagrantfile
* Vagrantfile is actually a script in Ruby, which is a configuration file that tells Vagrant what kind of machine you want
* Vagrant can use different providers, so you can configure that as well if you want



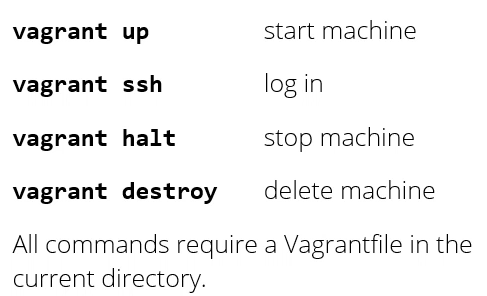
* When you run the machine, Vagrant can build and install everything for you, then from within that folder there’s a command that gives you access via SSH into the guest machine
* You can also share folders between the host and the guest
* Once you’ve run the machine for the first time, a hidden folder called .vagrant will appear in the folder where the Vagrantfile is
* That’s where vagrant stores local configuration options
* If you want to make a standard vagrant machine you can write these three lines in the Vagrantfile:



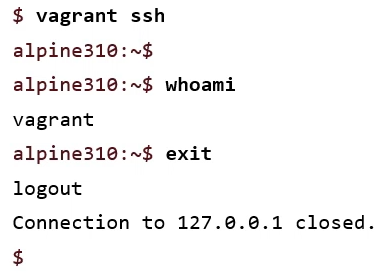
* **Vagrant.configure(“2”)** – goes into the configuration module and tells it that this is a configuration using version 2
* **Do** – takes a function as a parameter; calling a function and giving that function one parameter called config (Ruby uses || instead of () )
  + That function will be called when vagrant reads this file
  + The config parameter will be passed the vagrant configuration object
* **Config.vm.box** – for the VM, I am requesting this particular box  **“generic/alpine310”**
* **End** – will then go to online repositories (have to be connected to the internet), find the box called “generic/alpine310”, download, and then use that to build your VM
* So to get started, you make the folder, you make the Vagrantfile, then in the shell you type vagrant up



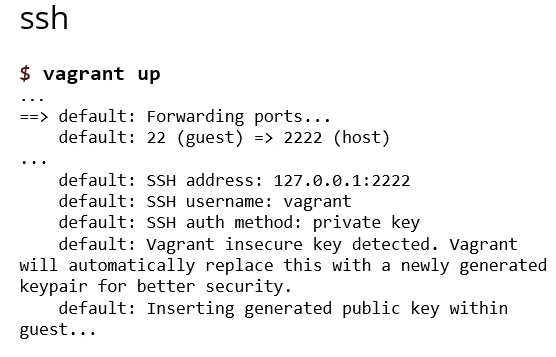
* Basic shell commands for vagrant:



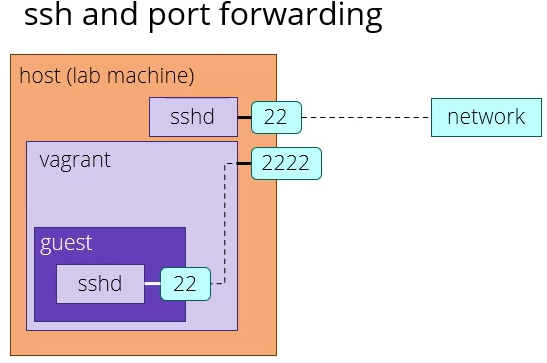
* Vagrant up – if machine isn’t running it will set it up correctly, it will download it if you haven’t done so already, if the machine’s already running it will do nothing
* Vagrant SSH will log in to the running machine
* Vagrant halt will stop the running machine
* Vagrant destroy will delete the machine (frees up disk space)
* All commands require a Vagrantfile in the current directory
* When the machine’s running, vagrant SSH logs you into the machine



* Whoami = tells you who you’re logged in as



* When you connect to something over a network, over the TCB protocol, each machine has different numbered ports which allows different services to run on the same machine
* Directory on the internet that explains which port is allocated to which service
* Port forwarding means connecting one port to another on the same or a different machine
* So in the example above, port 22 on the guest is connecting to port 2222 on the host
* When you log into a lab machine on the host, you are connecting to the SSH daemon on the lab machine, which is at port 22
* Daemon means something that is running in the background like a server
* When you set up the VM, it is also running SSH internally on its port 22
* Vagrant port forwarding is connecting the port 2222 on the host to port 22 on the guest
* If you SSH to port 2222 on the host machine, what you actually connect to is vagrant, and that will connect you to the guest machine
* “Vagrant insecure key detected” which should alarm, but “Vagrant will automatically replace this with a newly generated keypair…”
  + When you download vagrant machine, it comes with a default key
  + Same key for everyone in the world that downloads the vagrant machine
  + Not very secure, so when vagrant starts when you do vagrant up, if it detects the machine is using the default, it will create a new keypair particular to your account
* When you SSH into the lab machine, it will connect you by default to port 22 on the lab machine



* Port 22 is connected to a piece of software called SSHD
* The SSHD daemon (server here) lets you log into the lab machine
* When you’re on the lab machine, and you vagrant up, the host machine called vagrant will claim port 2222 for itself
* If you were to connect to that with the right permissions, vagrant will connect it to the guest machine
* Inside the guest machine, there’s another copy of SSHD running with a different key
* That one’s port 22 is connected to the host machine
* When you type vagrant SSH to connect, that will cause a connection to port 2222 on the host machine, which vagrant will forward to port 22 on the guest machine
* From the point of view of the guest machine, it’s a computer running an OS with an SSH daemon connect to its port 22, and a connection has just come in on that port
* Keys: basic principle is that keys come in pairs (a public key and a private key)
  + If you, on a machine, create a pair of keys and you keep the public key on that machine and tell SSH, then on any other machine that keeps the secret key, you can use that to log into this machine
* So when you download a vagrant box and you do vagrant up, it will create a custom key/pair to replace the default ones
* It will keep the public key stored in the guest (VM) and export the private key to the host
  + It’s one of the things that lives in the vagrant folder .host underneath where your Vagrantfile is
* When you type vagrant SSH, it will look in the vagrant folder to see if there’s a secret key stored in there
* If there is, it will use that key to connect to port 2222 which in turn connects to the guest
* Other thing you need to know is where your VM is stored
* On Linux, the VM itself (the files that contain the disk and all the OS software) will live inside your home directory in a folder called ~/.vagrant.d
  + If you go into that, you’ll see a subfolder called boxes, which is where the actual boxes are
* In Windows, it’s also in a folder called vagrant.d usually under your profile directory C:\Users\NAME\.vagrant.d
* Some configuration (e.g. keys) that are particular to a copy of the VM will then live in the .vagrant folder inside the folder with your Vagrantfile
* On the lab machines, VMs are stored in /tmp and might not survive host reboots (so they don’t run out of space)
* Also, they aren’t on NFS (network file storage), which means (unlike your usual files), if you log into a different lab machine the tmp directory will be different
* Treat VMs on lab machines as disposable and back up data elsewhere
* Alpine Linux chosen because it is minimal installation
* When you log into generic alpine box, there is nothing installed there that doesn’t absolutely have to be like SSH (otherwise you couldn’t log in in the first place)
* No gcc, git etc. by default
* Reason for this is so we can install and manage these things by ourselves
* Alpine Linux is a distribution that’s meant for machines in production
* If you’re running your own web servers, or your own machines on the cloud to run your applications, because Alpine is so minimal and security focussed, it’s a good choice in production
* If you were to run alpine in a container, it can fit in 8-10MB without storing too much extra software