04. Virtualisation

Virtualisation vs Emulation

- **Emulation:** a software simulation of another computer (think of game console emulations).
- **Virtualisation:** using CPU instructions that allow context switching between operating systems. You can think of it as hardware accelerated emulation. Other parts of the systems still have to be emulated, but this isn't very CPU intensive.

Virtualisation is a key enabling technology of DevOps.

Virtual Machine Terminology

- **Host:** your actual hardware and the OS that is running on top of it.
- **Guest:** the VM itself, a virtual computer running on the host.

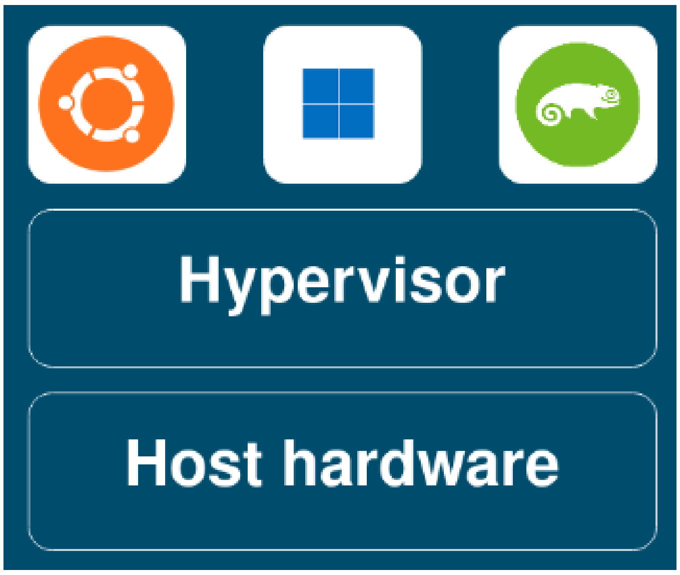
Context Switching refers to alternating between processes avoiding conflicts. This allows multiple processes to share a single CPU.

Hypervisors

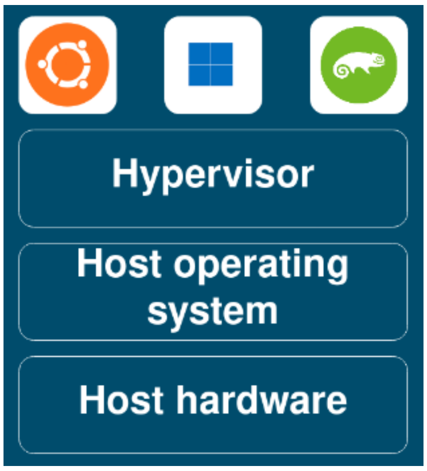
A hypervisor is a piece of software which manages a VM. There are two main types:

- **Type 1:** The hypervisor runs directly on the bare-metal instead of an operating system.
- **Type 2:** The hypervisor runs on top of the operating system, and is just an ordinary application programme.

> 💡 There are hypervisor APIs in the OS: Linux - KVM (Kernel Virtual Machine).



Type 1 hypervisor



Type 2 hypervisor

Examples:

- **Type 1:**
- **Type 2:**

Xen ORACLE VirtualBox vmware® QEMU

Paravirtualisation

Paravirtualisation refers to modifying VMs and guest operating systems so that they are aware they are a virtualised guest. Some major examples are Mouse pointer integration and VirtIO.

Vagrant

A tool to manage VMs running on top of a (type 2) hypervisor.

It can download premade VMs (called boxes) off the internet and install them on VirtuaBox for you and automatically run scripts to provision the VMs after creation.

Vagrant calls hypervisors like VirtualBox a provider.

Vagrant helps sharing development environments but it takes extra space reserved just for the VM. The Vagrantfile is code so it can be uploaded to a git repo. When a new employee arrives they just clone the repo and run the Vagrantfile to obtain the development environment.

The Vagrantfile is always up-to-date, sometimes documentation is not.

Set up a Vagrantfile

Each VM should be stored in a separate directory.

```
mkdir arch_vm
cd arch_vm
vagrant init archlinux/archlinux # Creates a Vagrantfile
# name before / is vendor name, name after / is the version of the VM
```

Vagrantfile contents (simplified)

The Vagrant file is written in Ruby, which makes Vagrant slow.

```
Vagrant.configure("2") do |config|
	config.vm.box = "archlinux/archlinux"
end
# the "2" is the config version
```

Starting the VM

```
vagrant up # Downloads the VM box from Vagrant Cloud and boots it up on VBox
```

Logging into the VM

```
vagrant ssh # Logs into the VM via SSH session
# Vagrant handles the encryption keys (no need to type a password)
```

Shutting down the VM

```
vagrant halt
```

File Sharing

```
ls /vagrant/
# this directory is shared between host and guest
```

Networking: port forwarding

```
Vagrant.configure("2") do |config|
	config.vm.box = "opensuse/Leap-15.6.x86_64"
	config.vm.network "forwarded_port", guest: 80, host: 8080
end
# Forward port 80 on the guest to localhost:8080 on the host
```

Networking: private network

```
Vagrant.configure("2") do |config|
	config.vm.box = "debian/bookworm64"
	config.vm.network "private_network", ip: "192.168.33.10"
end
# Create a private network between the host and guest. Give the guest the
IP 192.168.33.10 on this network.
```

Upgrade the CPUs and RAM of the VM

```
Vagrant.configure("2") do |config|
	config.vm.box = "freebsd/FreeBSD-14.1-STABLE"
	config.vm.provider "virtualbox" do |vb|
		vb.memory = 4096
		vb.cpus = 4
	end
end
```

Provisioners: run a script on the VM

```
# Embedded script in Vagrantfile
Vagrant.configure("2") do |config|
    config.vm.box = "debian/bookworm64"
    config.vm.provision "shell", inline <<-SHELL
        sudo apt install -y git build-essential
    SHELL
end
```

Provisioners: copy a file to the VM

```
Vagrant.configure("2") do |config|
    config.vm.box = "fedora/40-cloud-base"
    config.vm.provision "file", source: "file.c", destination: "~/file.c"
end
```

Delete a box/VM

```
vagrant destroy
# Complete delete a VM when you no longer need it. It CAN'T be undone!
```

Delete a box image

```
vagrant box remove archlinux/archlinux
# Completely delete a VM when you no longer need it. It CAN'T be undone!
but you can always download it again from the cloud.
```

Golden images

Pre-made VM images ready for deployment.

A **golden image** is a pre-made image (for either a VM or real hardware) that contains everything needed for a particular use-case already set up.

Think of disk images deployed by a company IT department to all of their machines.

Systems like Vagrant can be used to automatically create and manage golden images. These are the most primitive way to doing DevOps.