Virtual Machines and Vagrant
Part 01

References

1. The DevOPs 2.0 Toolkit, by Viktor Farcic, Published by Packt Publishing, 2016
2. Mastering KVM Virtualization, by Humble Devassy Chirammal; Prasad Mukhedkar; Anil Vettathu, Published by Packt Publishing, 2016
3. https://en.wikipedia.org/wiki/Operating-system-level_virtualization
4. https://en.wikipedia.org/wiki/Protection_ring
5. http://faculty.salina.k-state.edu/tim/ossg/Introduction/sys_calls.html
6. https://en.wikipedia.org/wiki/Hypervisor
7. https://en.wikipedia.org/wiki/Vagrant_(software)
8. Vagrant: Up and Running, by Mitchell Hashimoto. Publisher: O'Reilly Media, Inc.
9. https://www.vagrantup.com

Virtual Machines

- Our working definition of virtualization is (reference 2):
  - "a hardware environment that is not real"
  - hardware is duplicated/simulated and presented to an operating system.
  - this type of virtualization happen at a lower-level than the operating system

Virtual Machines

- Other types of virtualization include:
  - Software-Defined Networking (SDN)

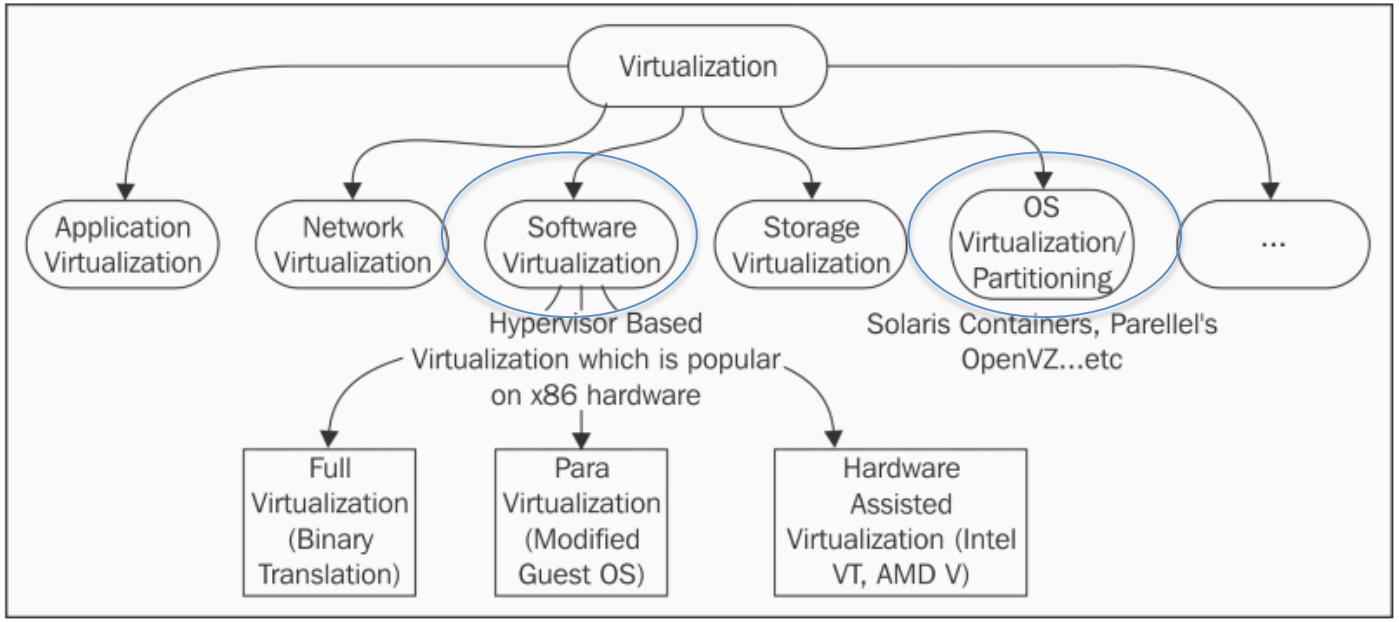
    https://en.wikipedia.org/wiki/Network_virtualization

  - Software Defined Storage (SDS)

    https://en.wikipedia.org/wiki/Software-defined_storage

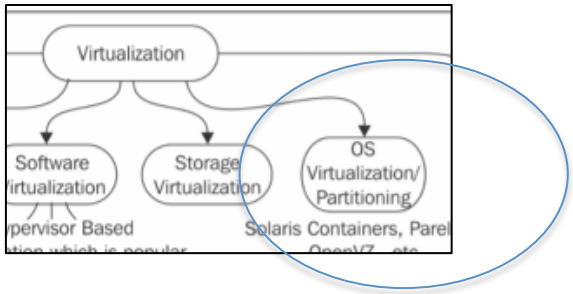
Virtual Machines

- The following slide from reference 2 illustrates different types of Virtualization



OS Virtualization

- From reference 3:
  https://en.wikipedia.org/wiki/Operating-system-level_virtualization



- Operating-system-level virtualization is a computer virtualization method in which the kernel of an operating system allows the existence of **multiple isolated user-space instances**

OS Virtualization

- From reference 3:
  https://en.wikipedia.org/wiki/Operating-system-level_virtualization

  Operating-system-level virtualization usually imposes little to no overhead, because programs in virtual partitions use the operating system's normal system call interface

  Operating-system-level virtualization is not as flexible as other virtualization approaches since it cannot host a guest operating system different from the host one

OS Virtualization

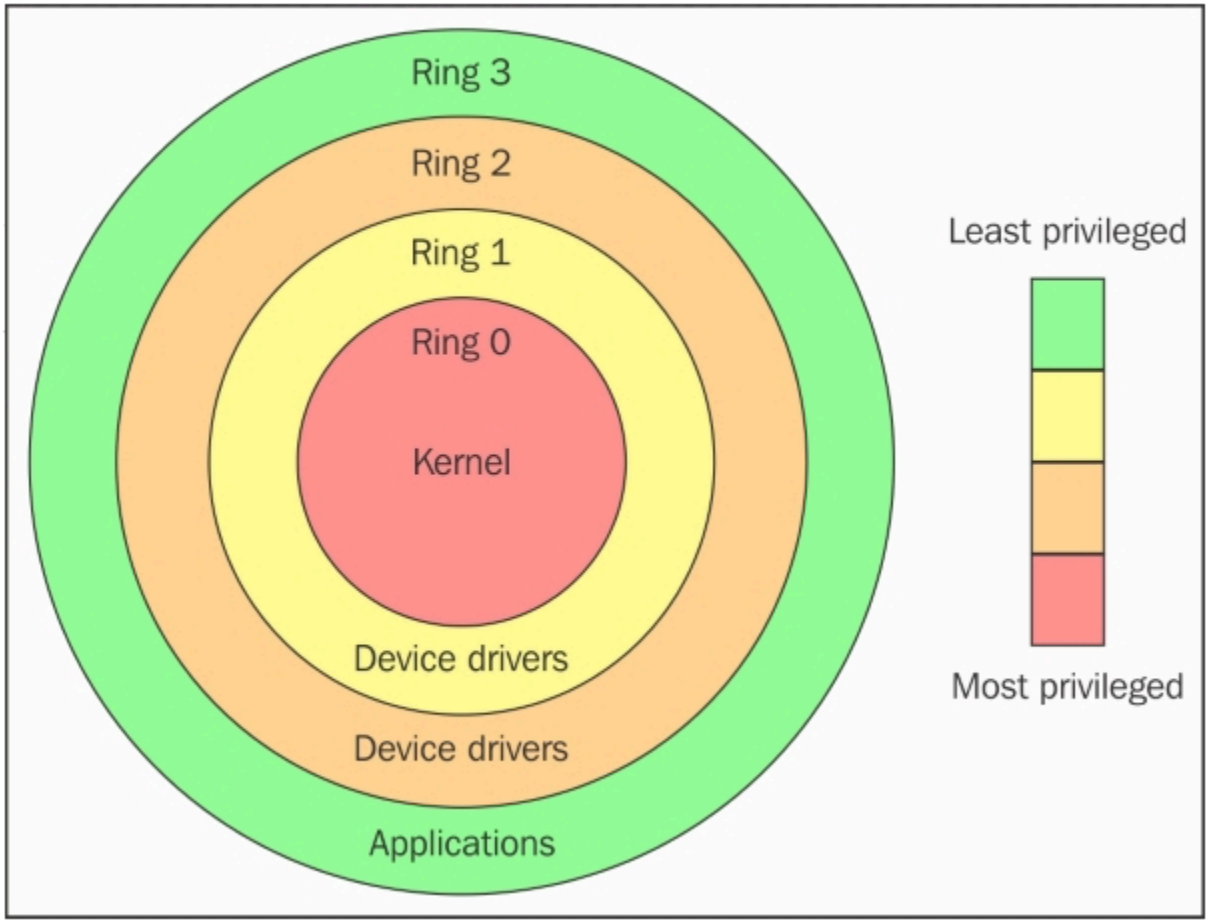
- OS Virtualization can be considered a type of container.
- In Unix, Linux the concept of "chroot", "chroot jail" has been used as a way to isolate a process and any processes spawned by a process from the rest of the system by changing the process's root directory.
- While chroot jails can be used to isolate a process from the rest of the system - they do not provide a secure platform in which you can run a suspect process , they are easy to break out of.

OS Virtualization

- Operating systems use different levels of "protection" to enforce access and security restraints on running software.

- The following diagram from reference 4 depicts these layers:

OS Virtualization



software running in ring 0 is the most privileged

software running in ring 3 is the least privileged

OS Virtualization

- Linux and MS-Windows generally only use two rings:
  - ring 0 is for **kernel mode** programs/drivers and core portions of the operating system

    Code running in ring zero, Kernel mode code is more privileged than code running in ring 3

  - ring 3 is for **user mode** programs

OS Virtualization

- User mode code (ring 3) must make calls into the Kernel, system calls, to perform I/O, access the network, etc.

- Calls from user mode into kernel mode are not regular function calls.

OS Virtualization

- When a user mode program makes a **system call** (for example to open a file). The compiler builds a stack frame or places value in CPU registers

- Next, a special type of instruction is called − a trap − that causes the CPU to switch to Kernel mode

OS Virtualization

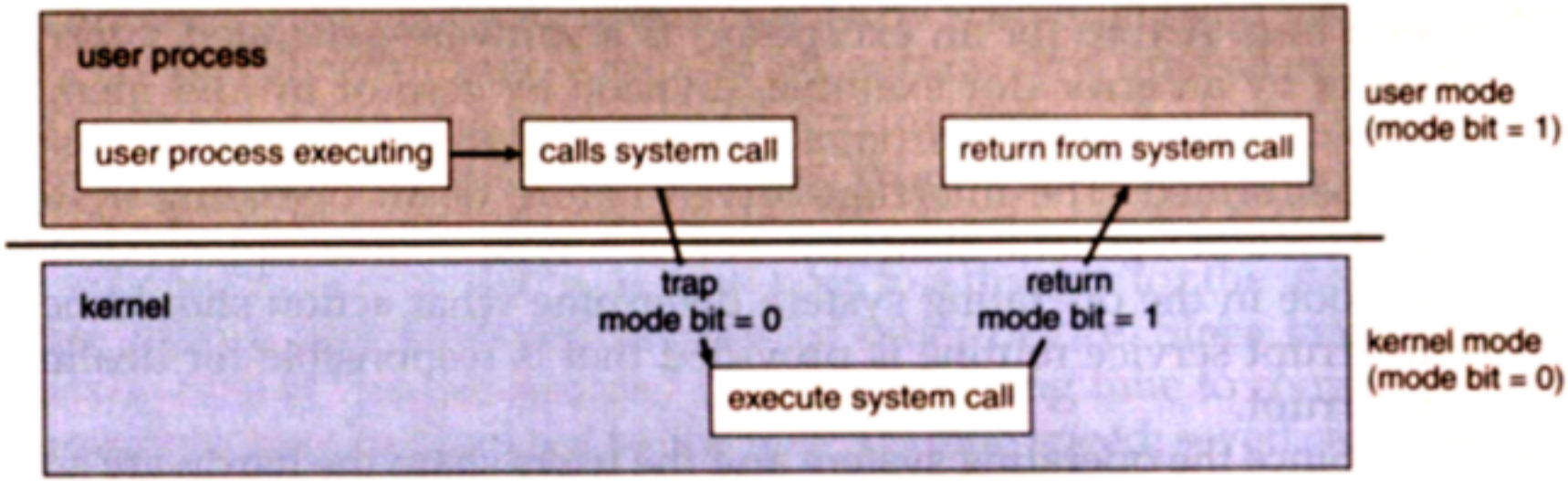
- The code that handles "the trap" is running in Kernel mode. That code looks at the system service requested − and invokes the code that handles the system code.
  - A different call stack is used in Kernel Mode
- After the system call completes, the Kernel code switches back into user mode

OS Virtualization

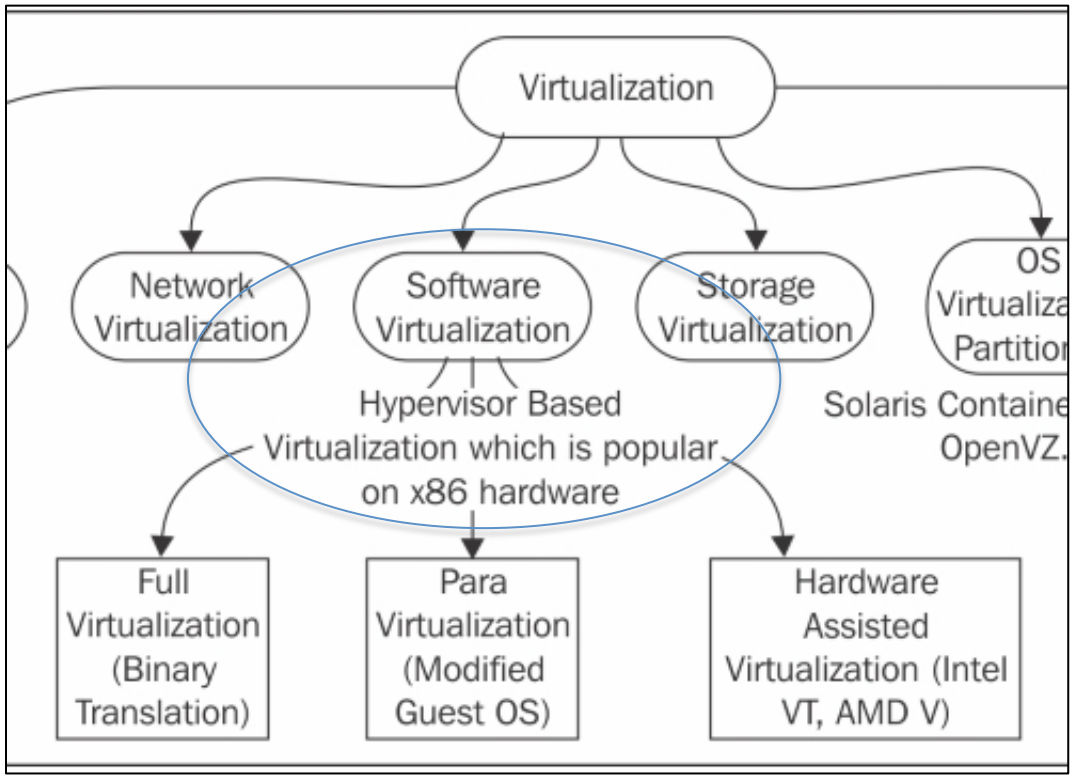
The following diagram from:

http://faculty.salina.k-state.edu/tim/ossg/Introduction/sys_calls.html

illustrates a system call:



Software Virtualization



Virtual Machine Monitor (VMM) – Hypervisor

- **A Virtual Machine Monitor** (VMM), often called a **hypervisor** is:
  - software, firmware, or hardware that creates and manage virtual machines

- Hypervisors (VMM's) can run directly on hardware or run within another operating system

Virtual Machine Monitor (VMM) – Hypervisor

- We will use terms hypervisor and VMM interchangeably

Virtual Machine Monitor (VMM) – Hypervisor

- The operating system that the hypervisor runs on is the **host** operating system

- The operating system run by the hypervisor is the **guest** operating system

Hypervisor

- Running one operating system − the guest − within another operating system − the host creates an interesting scenario

  The kernels of both the guest and host operating system expect to run in ring 0.

Hypervisor

- However, only one Kernel can run in ring 0!
- This had lead to different virtualization mechanisms:
  - full virtualization
  - paravirtualization

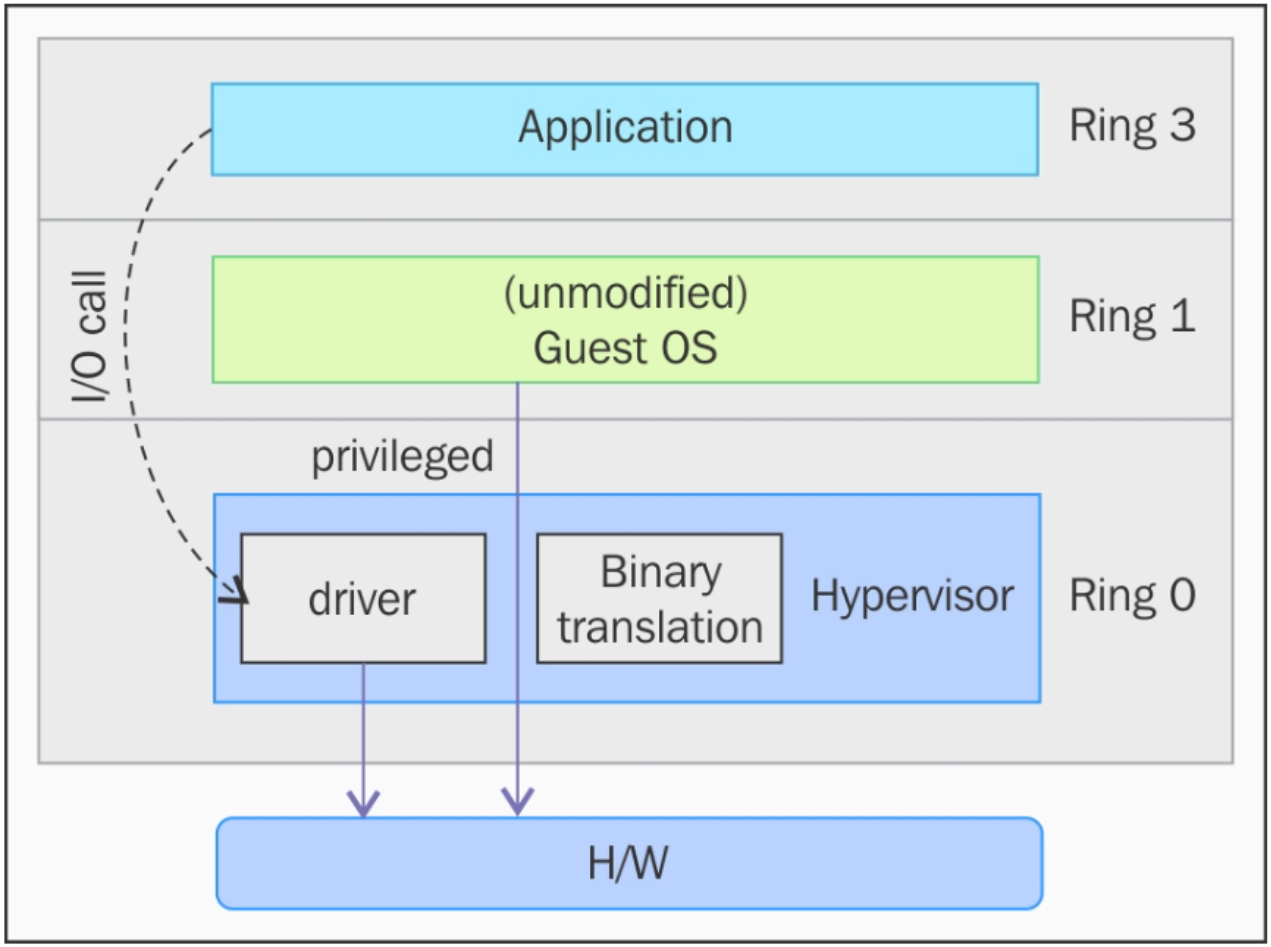
Full Virtualization

- Full virtualization
  - The guest OS runs in ring 1
  - The Hypervisor/VMM runs in ring 0
  - System calls from the guest OS are performed via:
    - binary (runtime) translation of system calls from the guest into the hypervisor (from ring 1 into ring 0)
    - emulation of the system call made by the guest OS in the hypervisor

Full Virtualization

- The following diagram from reference: Master KVM Virtualization depicts full virtualization

Full Virtualization



Full Virtualization

- The overhead of binary translation and emulation of guest OS functionality in the hypervisor can slow operations down

- However, the guest OS can run unmodified

Paravirtualization

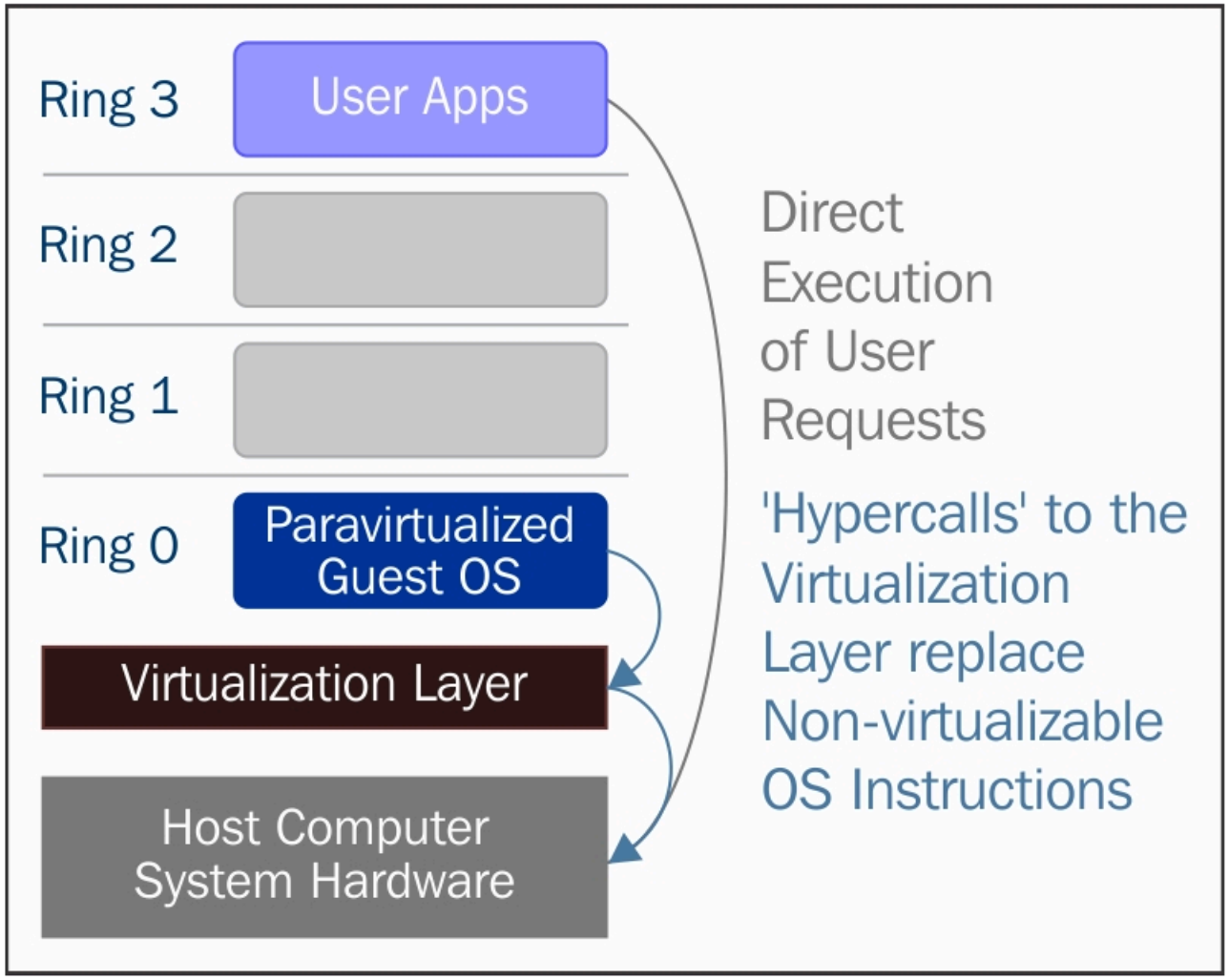
- Paravirtualization addresses the performance overhead of binary translation and emulation by using modified versions of guest operating systems that access ring 0.

- A hypervisor provide an API that the guest OS calls to execute in ring 0.

Paravirtualization

- The following diagram from reference: Master KVM Virtualization − depicts paravirtualization

Paravirtualization



Hardware Assisted Virtualization

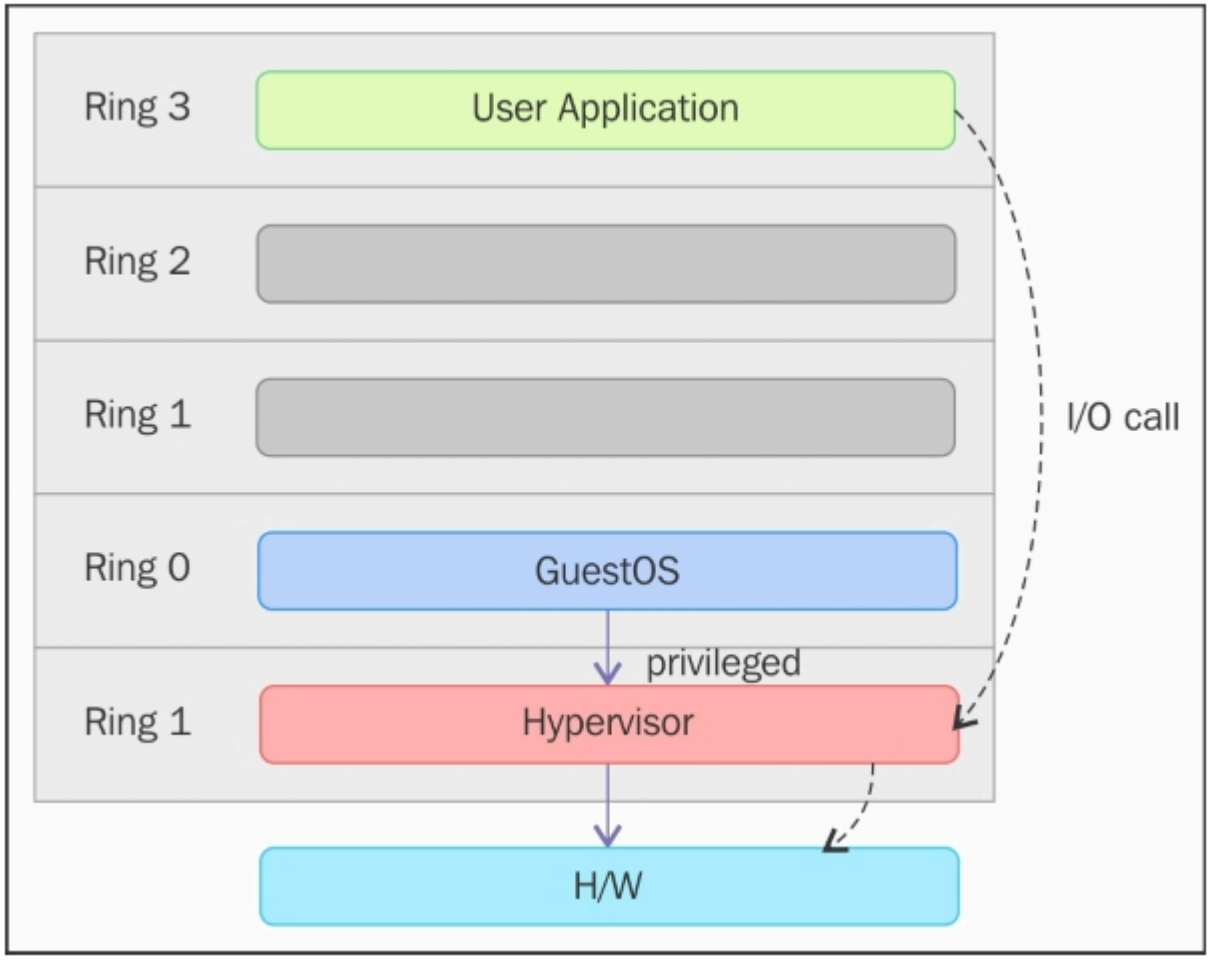
- Chip makers added support for faster virtualization via hardware assistance features in their chips.

- These extensions allow the hypervisor to run in ring 1, while the kernel of the guest OS runs in ring 0

Hardware Assisted Virtualization

- The following diagram from reference: Master KVM Virtualization − depicts a hardware assisted scenario

Hardware Assisted Virtualization



hardware assisted virtualization achieves better performances – the hypervisor has less work to does, it does not have to translate and emulate guest kernel mode functionality

Hypervisors

- Hypervisors (or Virtual Machine Monitors, VMM) control the operation of the guest operating system.

- Hypervisors provide "virtual hardware" that is seen by the guest OS as "real" hardware

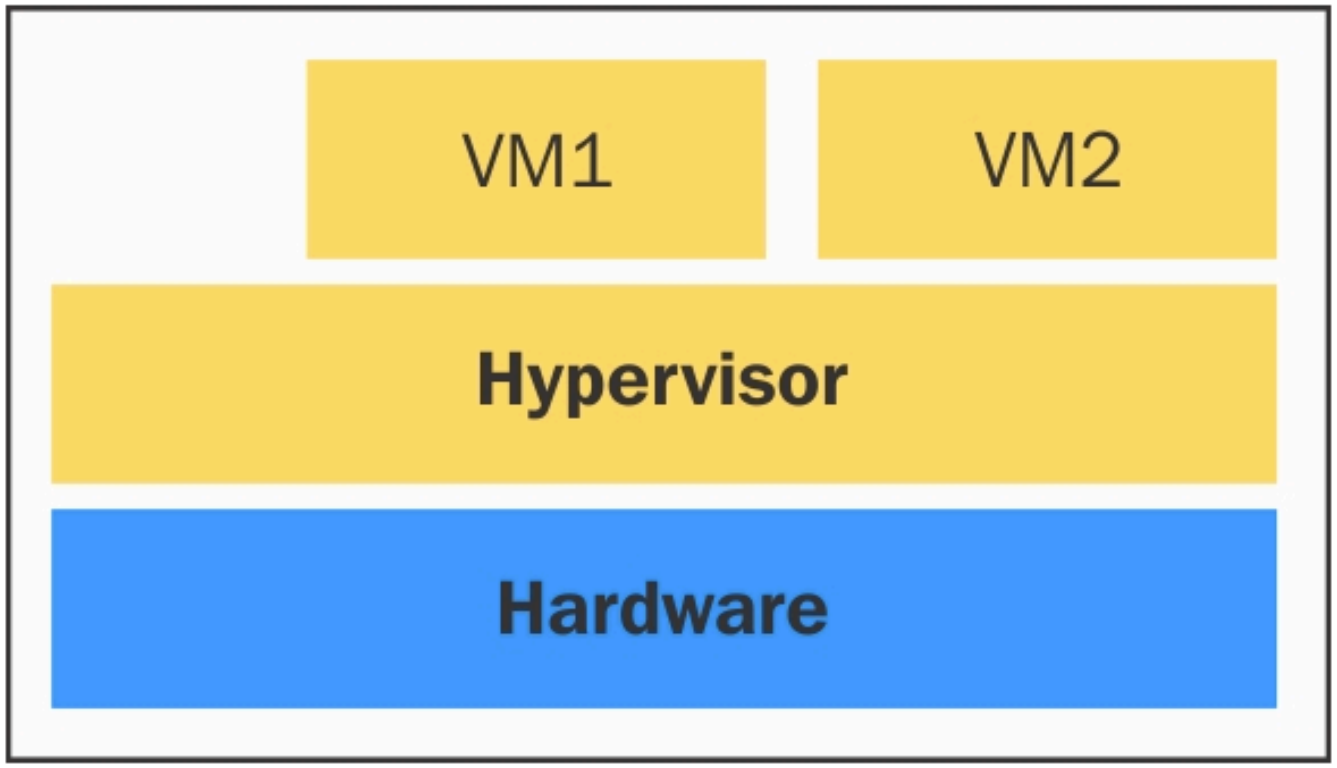
Hypervisors

- Hypervisors allocate resources to guest operating systems including:
  - processors, memory
- A hypervisor can support multiple, and different types of guest OS's

Hypervisors

- A general categorization of hypervisors is to classify them as "Type 1" and "Type 2"
- A Type 1 hypervisor runs directly on hardware

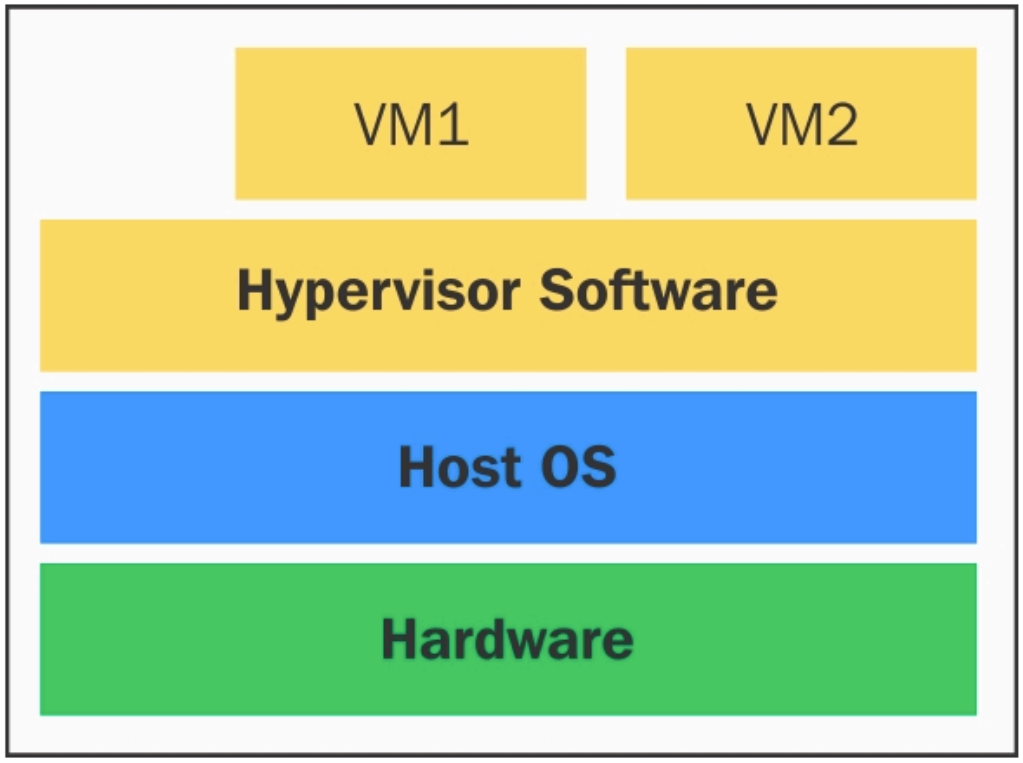
Type 1 Hypervisors



from reference 2

Type 2 Hypervisors

- A Type 2 hypervisor runs on top of a host OS



from reference 1

Type 1 vs. Type 2 Hypervisors

- Type 1:
  - easy to install, configure
  - small
  - less overhead − only has apps installs that it needs
- Type 2:
  - more flexible
  - wider range of hardware support

Open Source Virtualization

- Xen - http://www.xenproject.org/
  - supports full and paravirtualization
- KVM - http://www.linux-kvm.org/
  - supports full virtualization
- VirtualBox - http://www.linux-kvm.org/
  - supports full virtualization

Open Source Virtualization

- Xen - http://www.xenproject.org/
  - supports full and paravirtualization
- KVM - http://www.linux-kvm.org/
  - supports full virtualization
- VirtualBox - http://www.linux-kvm.org/
  - supports full virtualization

VirtualBox

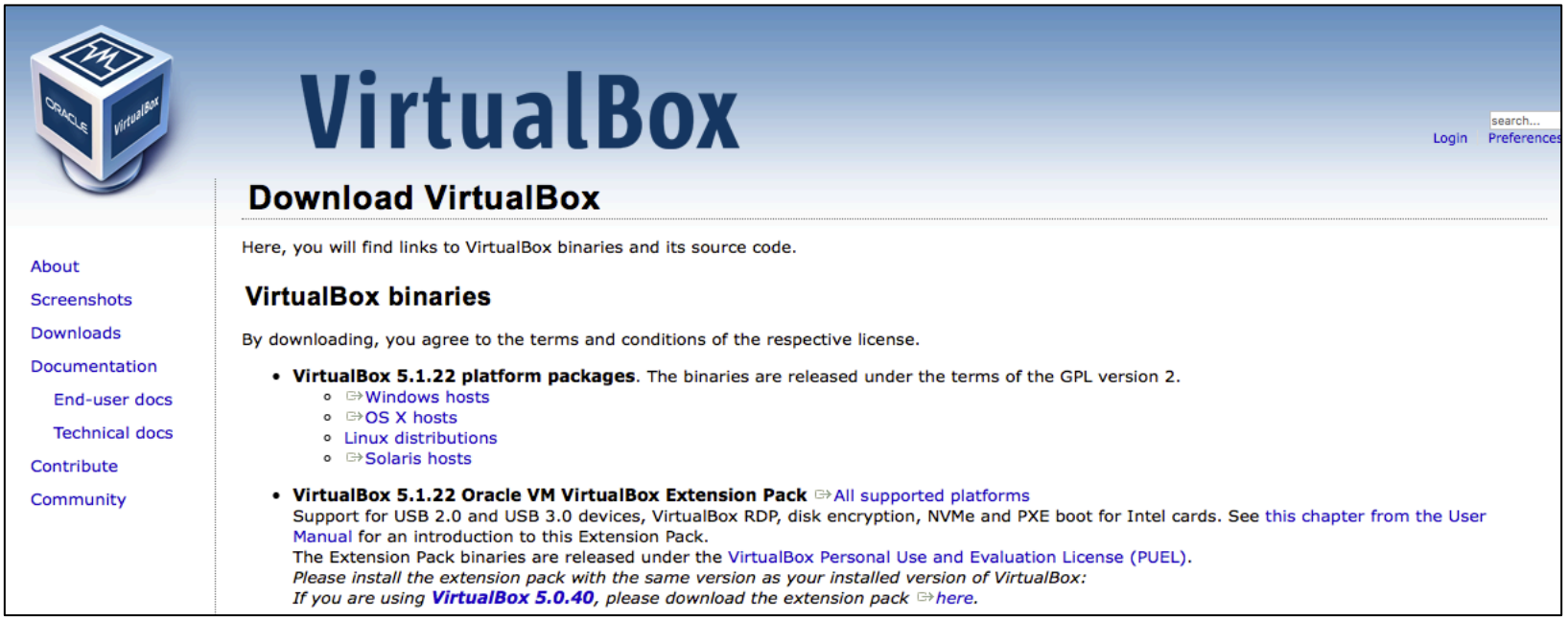
- Download VirtualBox from:

  https://www.virtualbox.org/wiki/Downloads

- Once downloaded , install it and make sure it starts up (you can close it once you confirm it runs).

- We will use VirtualBox later on in the section.

VirtualBox


VirtualBox
search...
Login
Preferences
Download VirtualBox
About
Screenshots
Downloads
Documentation
End-user docs
Technical docs
Contribute
Community
Here, you will find links to VirtualBox binaries and its source code.
VirtualBox binaries
By downloading, you agree to the terms and conditions of the respective license.
VirtualBox 5.1.22 platform packages. The binaries are released under the terms of the GPL version 2.
Windows hosts
OS X hosts
Linux distributions
Solaris hosts
VirtualBox 5.1.22 Oracle VM VirtualBox Extension Pack
All supported platforms
Support for USB 2.0 and USB 3.0 devices, VirtualBox RDP, disk encryption, NVMe and PXE boot for Intel cards. See this chapter from the User Manual for an introduction to this Extension Pack.
The Extension Pack binaries are released under the VirtualBox Personal Use and Evaluation License (PUEL).
Please install the extension pack with the same version as your installed version of VirtualBox:
If you are using VirtualBox 5.0.40, please download the extension pack here.

Vagrant

- https://en.wikipedia.org/wiki/Vagrant_(software)

  Vagrant is an open-source software product for building and maintaining portable virtual development environments

Vagrant

- Why use a tool like Vagrant?

  as deployment environments scale out, setup and management becomes more complex and more difficult to manage

- Vagrant can manage multiple platform configurations using different technology stacks

Vagrant

- Download Vagrant from:

  https://www.vagrantup.com/downloads.html

Vagrant

- After install Vagrant, verify it's installation as follows:
  - open a command prompt/terminal
  - type: vagrant

Vagrant

```
JeffsMacBookPro:~ jeffm$ vagrant
Usage: vagrant [options] <command> [<args>]

    -v, --version                    Print the version and exit.
    -h, --help                       Print this help.

Common commands:
     box             manages boxes: installation, removal, etc.
     connect         connect to a remotely shared Vagrant environment
     destroy         stops and deletes all traces of the vagrant machine
     global-status   outputs status Vagrant environments for this user
     halt            stops the vagrant machine
     help            shows the help for a subcommand
     init            initializes a new Vagrant environment by creating a Vagrantfile
     login           log in to HashiCorp's Atlas
     package         packages a running vagrant environment into a box
     plugin          manages plugins: install, uninstall, update, etc.
     port            displays information about guest port mappings
     powershell      connects to machine via powershell remoting
     provision       provisions the vagrant machine
     push            deploys code in this environment to a configured destination
     rdp             connects to machine via RDP
     reload          restarts vagrant machine, loads new Vagrantfile configuration
     resume          resume a suspended vagrant machine
     share           share your Vagrant environment with anyone in the world
     snapshot        manages snapshots: saving, restoring, etc.
     ssh             connects to machine via SSH
     ssh-config      outputs OpenSSH valid configuration to connect to the machine
     status          outputs status of the vagrant machine
     suspend         suspends the machine
     up              starts and provisions the vagrant environment
     validate        validates the Vagrantfile
     version         prints current and latest Vagrant version

For help on any individual command run `vagrant COMMAND -h`

Additional subcommands are available, but are either more advanced
or not commonly used. To see all subcommands, run the command
`vagrant list-commands`.
```

Vagrant

- Next, we will create a Vagrant project:
  - create a directory for you project.
  - On my machine I created:

    vagrant/vagrant_getting_started

    from my home directory. You can put this folder in a different location if you prefer.

    (https://www.vagrantup.com/intro/getting-started/project_setup.html)

Vagrant

- next, cd into vagrant_getting_started
- run command: vagrant init

```
JeffsMacBookPro:vagrant_getting_started jeffm$ vagrant init
A `Vagrantfile` has been placed in this directory. You are now
ready to `vagrant up` your first virtual environment! Please read
the comments in the Vagrantfile as well as documentation on
`vagrantup.com` for more information on using Vagrant.
```

Vagrant

- The init command created a new Vagrantfile in this directory (since one did not exist).

- Every Vagrant project uses a Vagrantfile to:
  - denote the root directory of the project
  - describe the virtual machine and other software to run in the vm

Vagrant Boxes

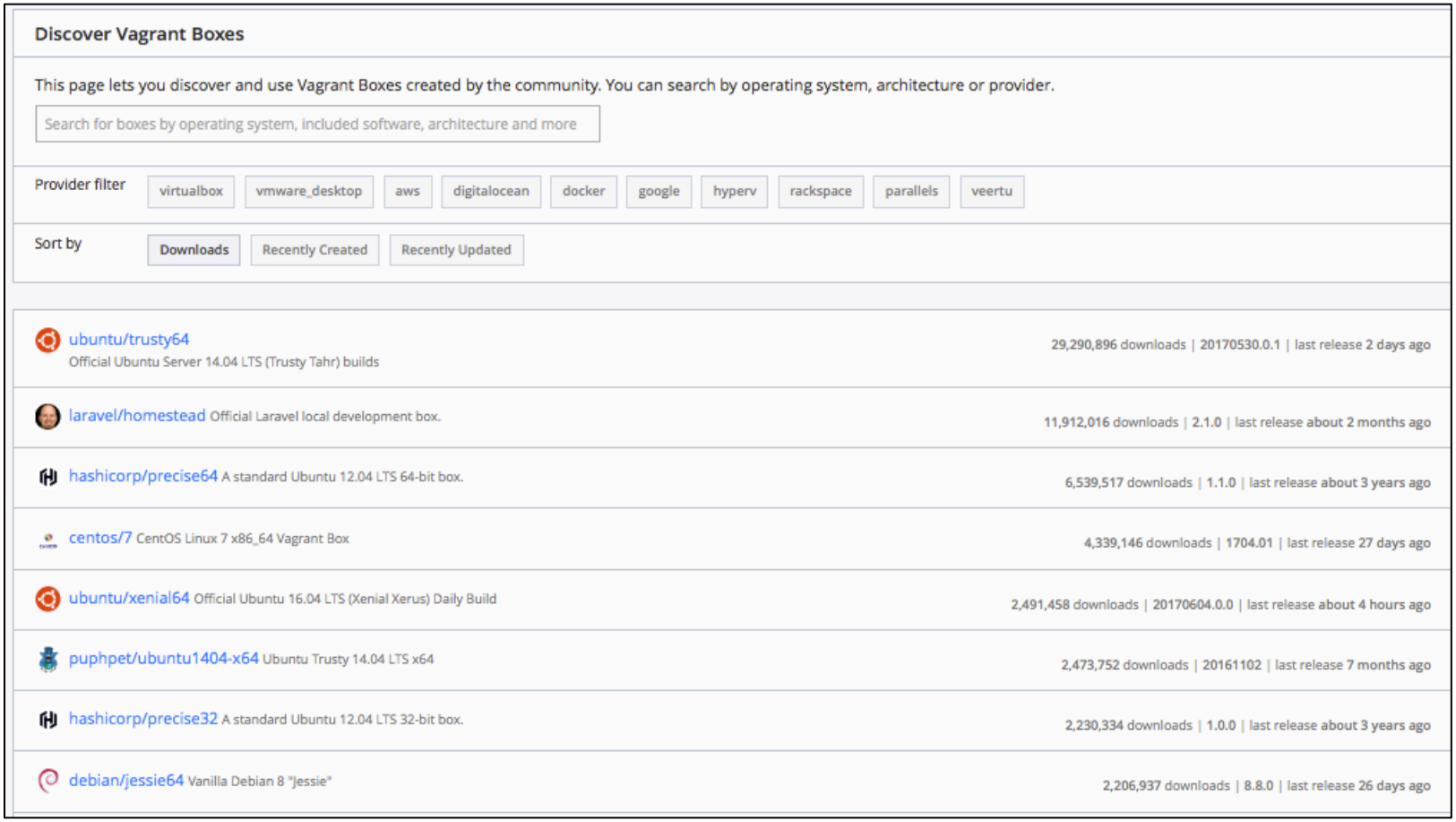
- Vagrant build VM's from images called "**boxes**"

- A number of predefined "boxes" are available at:

  https://vagrantcloud.com

  https://atlas.hashicorp.com/boxes/search
  (this url is deprecated but may still work)

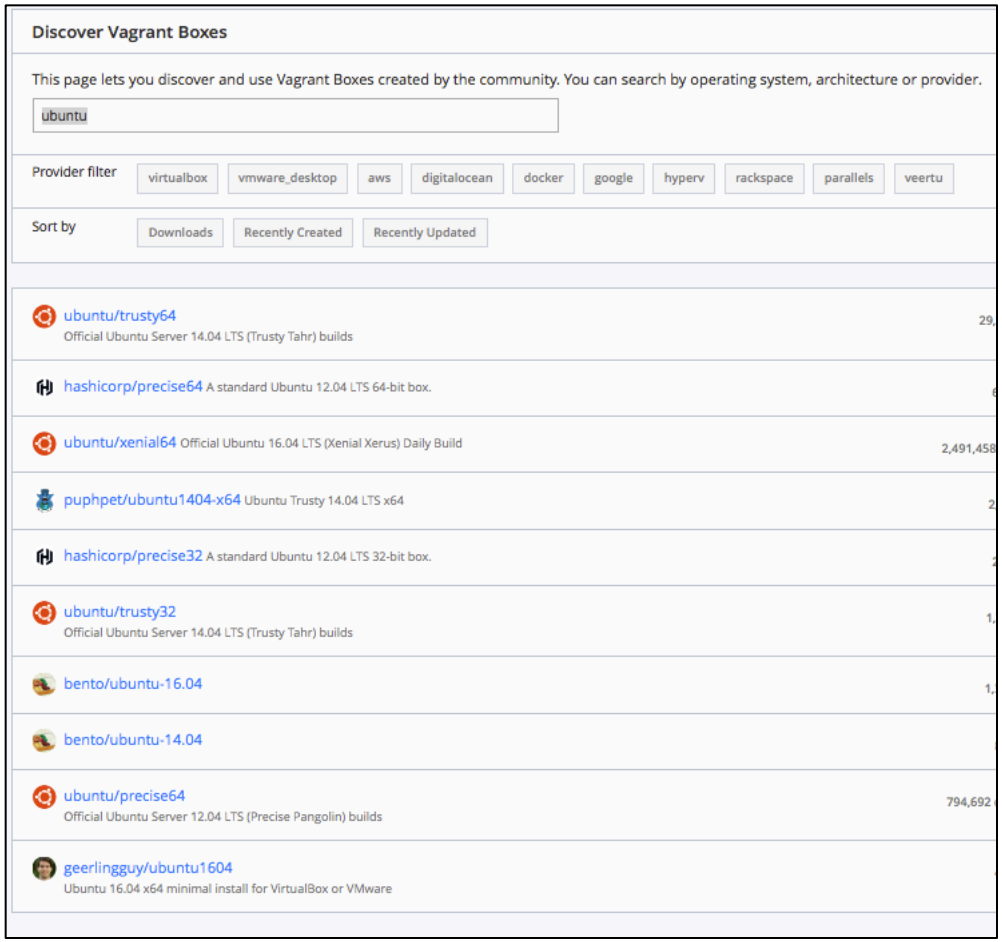
Vagrant Boxes



Vagrant Boxes

- You can search by "provider filter" (discussed later in this presentation)

Vagrant Boxes



Vagrant Boxes

- A Vagrant **Provider** is a service/APIs Vagrant uses to setup/create virtual environments.
- Vagrant contains built-in support for VirtualBox and Docker.
- Other Virtual Machine services are supported via plug-ins.

Vagrant Boxes

- Vagant **Provisioners** are services that allow you to revise the configuration of a box and install software in a box.

- Chef and Puppet are supported as Provisioners. We will look at Chef and Puppet later on in this course.

Vagrant Boxes

- Vagrant also support placing bash shell scripts into the guest OS within a box
- The guest OS can be reconfigured and software installed using Bash Shell scripts
- We will use Bash Shell Scripts in this section

Vagrant Boxes

```
JeffsMacBookPro:vagrant_getting_started jeffm$ vagrant box help
Usage: vagrant box <subcommand> [<args>]

Available subcommands:
     add
     list
     outdated
     prune
     remove
     repackage
     update

For help on any individual subcommand run `vagrant box <subcommand> -h`
```

The "box" command is used to install and manage boxes.

The image above shows command: **vagrant box help**

Vagrant Boxes

- Command: **vagrant box add** is used get and install a box:

Vagrant Boxes

```
JeffsMacBookPro:vagrant_getting_started jeffm$ vagrant box add -h
Usage: vagrant box add [options] <name, url, or path>

Options:

    -c, --clean                      Clean any temporary download files
    -f, --force                      Overwrite an existing box if it exists
        --insecure                   Do not validate SSL certificates
        --cacert FILE                CA certificate for SSL download
        --capath DIR                 CA certificate directory for SSL download
        --cert FILE                  A client SSL cert, if needed
        --location-trusted           Trust 'Location' header from HTTP redirects and use
uent urls as for the initial one
        --provider PROVIDER          Provider the box should satisfy
        --box-version VERSION        Constrain version of the added box

The box descriptor can be the name of a box on HashiCorp's Atlas,
or a URL, or a local .box file, or a local .json file containing
the catalog metadata.

The options below only apply if you're adding a box file directly,
and not using a Vagrant server or a box structured like 'user/box':

        --checksum CHECKSUM          Checksum for the box
        --checksum-type TYPE         Checksum type (md5, sha1, sha256)
        --name BOX                   Name of the box
    -h, --help                       Print this help
```

Vagrant Boxes

$ **vagrant box add -h**
Usage: vagrant box add [options] **<name, url, or path**>
…

The box descriptor can be the:

name of a box on HashiCorp's Atlas,

https://atlas.hashicorp.com/boxes/search

or a **URL**,

or a **local .box file**,

or **a local .json file** containing
the catalog metadata.

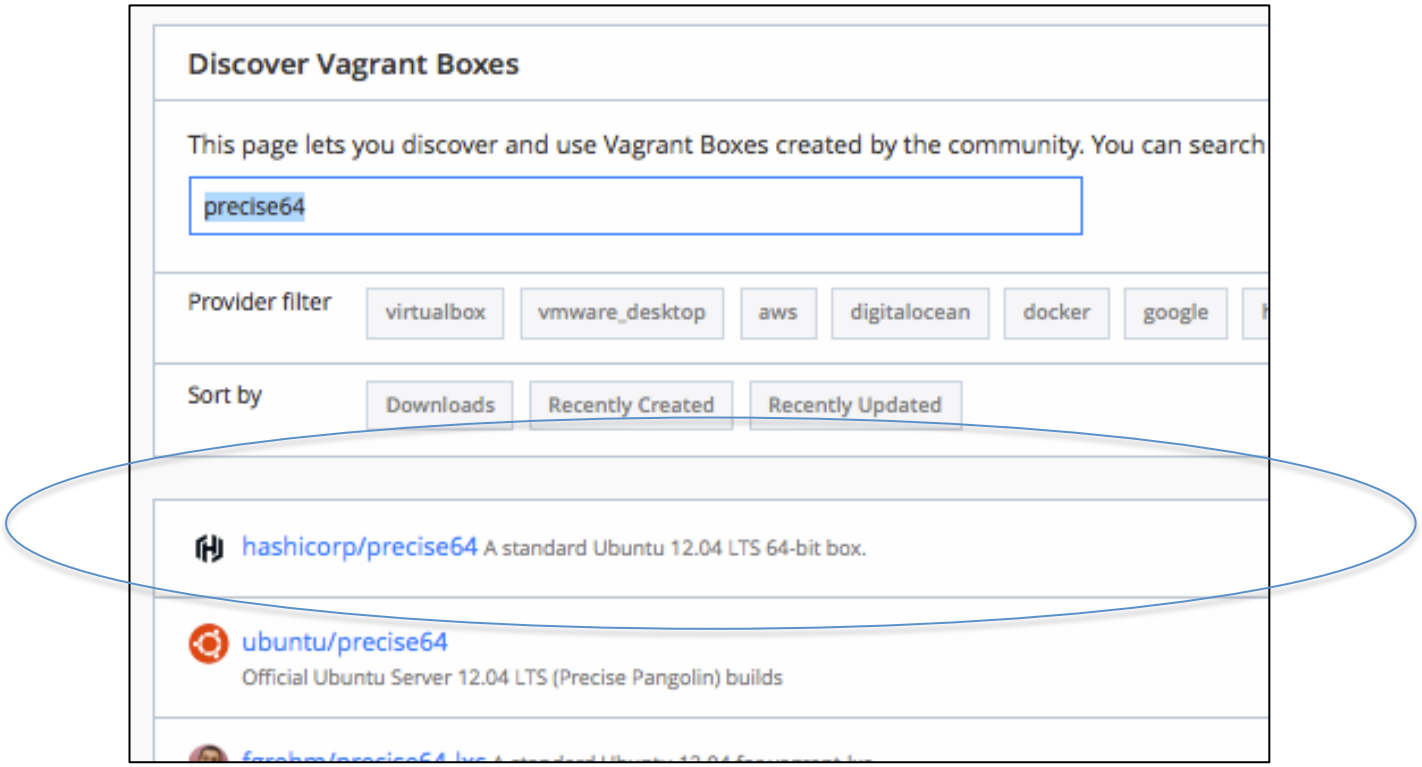
Vagrant Boxes

- Following the "Getting Started" guide (as of June 2017) from:

  https://www.vagrantup.com/intro/getting-started/boxes.html

- We will get "box" "hashicorp/precise64"

Vagrant Boxes



Vagrant Boxes

- To install hashicorp/precise64 run the following command in the home directory of our Vagrant project:

  vagrant box add hashicorp/precise64

  which is a Ubuntu Linux distro.

Vagrant Boxes

```
JeffsMacBookPro:vagrant_getting_started jeffm$ vagrant box add hashicorp/precise64
==> box: Loading metadata for box 'hashicorp/precise64'
    box: URL: https://atlas.hashicorp.com/hashicorp/precise64
This box can work with multiple providers! The providers that it
can work with are listed below. Please review the list and choose
the provider you will be working with.

1) hyperv
2) virtualbox
3) vmware_fusion

Enter your choice: 2
```

Select "2" for using the VirtualBox provider

Vagrant Boxes

```
JeffsMacBookPro:vagrant_getting_started jeffm$ vagrant box add hashicorp/precise64
==> box: Loading metadata for box 'hashicorp/precise64'
    box: URL: https://atlas.hashicorp.com/hashicorp/precise64
This box can work with multiple providers! The providers that it
can work with are listed below. Please review the list and choose
the provider you will be working with.

1) hyperv
2) virtualbox
3) vmware_fusion

Enter your choice: 2
==> box: Adding box 'hashicorp/precise64' (v1.1.0) for provider: virtualbox
    box: Downloading: https://atlas.hashicorp.com/hashicorp/boxes/precise64/versions/1.1.0/providers/virtualbox.box
    box: Progress: 2% (Rate: 278k/s, Estimated time remaining: 0:13:19)
```

The "box add" command will download the box image from the hashicorp repository.

NOTE: this command may take awhile to run − depending on the speed of your internet connection

Vagrant Boxes

```
JeffsMacBookPro:vagrant_getting_started jeffm$ vagrant box add hashicorp/precise64
==> box: Loading metadata for box 'hashicorp/precise64'
    box: URL: https://atlas.hashicorp.com/hashicorp/precise64
This box can work with multiple providers! The providers that it
can work with are listed below. Please review the list and choose
the provider you will be working with.

1) hyperv
2) virtualbox
3) vmware_fusion

Enter your choice: 2
==> box: Adding box 'hashicorp/precise64' (v1.1.0) for provider: virtualbox
    box: Downloading: https://atlas.hashicorp.com/hashicorp/boxes/precise64/versions/1.1.0/providers/virtualbox.box
==> box: Successfully added box 'hashicorp/precise64' (v1.1.0) for 'virtualbox'!
JeffsMacBookPro:vagrant_getting_started jeffm$
```

If the "box add" command ran successfully, you will see:

box: Successfully added box 'hashicorp/precise64' (v1.1.0) for 'virtualbox'!

Vagrant Boxes

- Once the box has been added into Vagrant's local list of boxes, we need to specify the box as the "base" for our Vagrant project

Vagrant Boxes

Open the Vagrantfile and change this line:

```
Vagrant.configure("2") do |config|
  #other comment here …
  config.vm.box = "base"
```

to

```
Vagrant.configure("2") do |config|
 #other comment lines here
 config.vm.box = "hashicorp/precise64
```

Starting Vagrant

- To start our Vagrant environment, run command **vagrant up**

  from the root directory of the Vagrant project

```
JeffsMacBookPro:vagrant_getting_started jeffm$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'hashicorp/precise64'...
==> default: Matching MAC address for NAT networking...
==> default: Checking if box 'hashicorp/precise64' is up to date...
==> default: Setting the name of the VM: vagrant_getting_started_default_1496636212419_73900
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
==> default: Forwarding ports...
    default: 22 (guest) => 2222 (host) (adapter 1)
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: vagrant
    default: SSH auth method: private key
    default:
    default: Vagrant insecure key detected. Vagrant will automatically replace
    default: this with a newly generated keypair for better security.
    default:
    default: Inserting generated public key within guest...
    default: Removing insecure key from the guest if it's present...
    default: Key inserted! Disconnecting and reconnecting using new SSH key...
==> default: Machine booted and ready!
==> default: Checking for guest additions in VM...
    default: The guest additions on this VM do not match the installed version of
    default: VirtualBox! In most cases this is fine, but in rare cases it can
    default: prevent things such as shared folders from working properly. If you see
    default: shared folder errors, please make sure the guest additions within the
    default: virtual machine match the version of VirtualBox you have installed on
    default: your host and reload your VM.
    default:
    default: Guest Additions Version: 4.2.0
    default: VirtualBox Version: 5.1
==> default: Mounting shared folders...
    default: /vagrant => /Users/jeffm/DevOps-Tech/vagrant/vagrant_getting_started
JeffsMacBookPro:vagrant_getting_started jeffm$
```

Accessing Vagrant VMs

- Vagrant started yp our virtual environment without a UI.

- To access our Vagrant VM, we need to use

  vagrant ssh

  from the root directory of our Vagrant project

Accessing Vagrant VMs

```
JeffsMacBookPro:vagrant_getting_started jeffm$ vagrant ssh
Welcome to Ubuntu 12.04 LTS (GNU/Linux 3.2.0-23-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
New release '14.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Welcome to your Vagrant-built virtual machine.
Last login: Fri Sep 14 06:23:18 2012 from 10.0.2.2
vagrant@precise64:~$
```

Accessing Vagrant VMs

- We did it! We setup Vagrant, added a box, ran the box, and accessed the box

- Now, logout of the ssh terminal accessing our vagrant box:

```
vagrant@precise64:~$ logout
Connection to 127.0.0.1 closed.
```

Accessing Vagrant VMs

- Now, let's tell Vagrant to stop running the VM that used our box as it image:

```
JeffsMacBookPro:vagrant_getting_started jeffm$ vagrant destroy
    default: Are you sure you want to destroy the 'default' VM? [y/N] y
==> default: Forcing shutdown of VM...
==> default: Destroying VM and associated drives...
```

The "vagrant destroy" command terminates the virtual machine running our box.

However − it does NOT remove the box. So we can start another VM using our box − as long as we do not remove the box (using the box remove command).