XEN Hypervisor

The Xen approach

- Support for unmodified binaries (but not OS) essential
  - Important for app developers
  - Virtualized system exports has same Application Binary Interface (ABI)
- Modify guest OS to be aware of virtualization
  - Gets around problems of x86 architecture
  - Allows better performance to be achieved
- Expose some effects of virtualization
  - Translucent VM OS can be used to optimize for performance
- Keep hypervisor layer as small and simple as possible
  - Resource management, Device drivers run in privileged VMM
  - Enhances security, resource isolation

Paravirtualization

- Solution to issues with x86 instruction set
  - Don’t allow guest OS to issue sensitive instructions
  - Replace those sensitive instructions that don’t trap to ones that will trap
- Guest OS makes “hypercalls” (like system calls) to interact with system resources
  - Allows hypervisor to provide protection between VMs
- Exceptions handled by registering handler table with Xen
  - Fast handler for OS system calls invoked directly
  - Page fault handler modified to read address from replica location
- Guest OS changes largely confined to arch-specific code
  - Compile for ARCH=xen instead of ARCH=i686
  - Original port of Linux required only 1.36% of OS to be modified

XEN Paravirtualization

- Paravirtualization (PV) is an efficient and lightweight virtualization technique introduced by the Xen Project.
- PV does not require virtualization extensions from the host CPU and thus enables virtualization on hardware architectures that do not support Hardware-assisted virtualization.
- However, PV guests and control domains require kernel support and drivers that in the past required special kernel builds, but are now part of the Linux kernel as well as other operating systems.
- Paravirtualization implements the following functionality
  a. Disk and Network drivers
  b. Interrupts and timers
  c. Emulated Motherboard and Legacy Boot
  d. Privileged Instructions and Page Tables

Xen Basics

- Guest virtual machines running on a Xen Project Hypervisor are known as “domains”.
- A special domain known as domain0 (or dom0) is responsible for controlling the hypervisor and starting other guest operating systems.
- These other guest operating systems are called domUs. This is because these domains are “unprivileged” in the sense they cannot control the hypervisor or start/stop other domains.

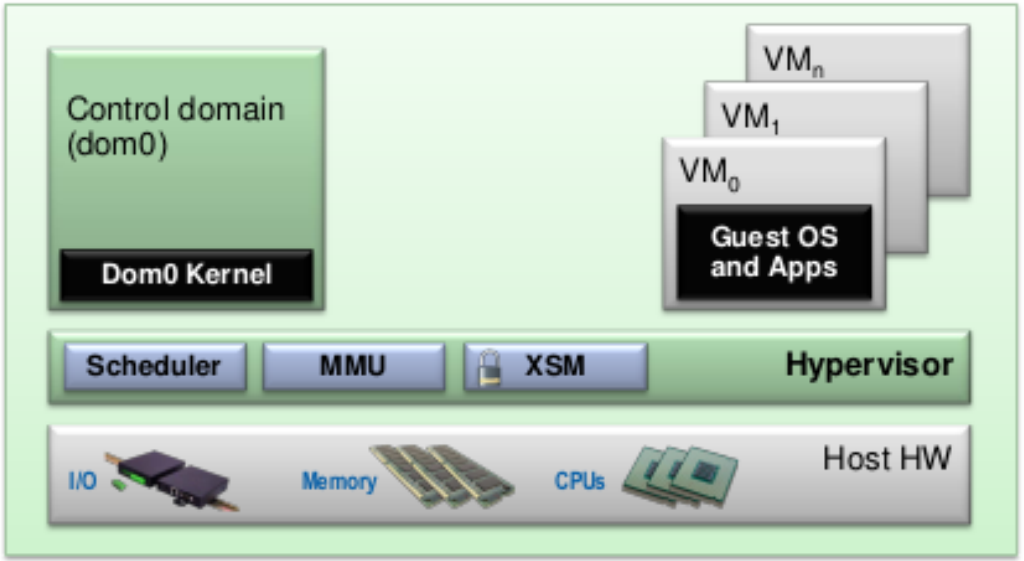
XEN Basics

- Xen hypervisor supports two primary types of virtualization:
  - paravirtualization (PV)
  - hardware virtualized machine (HVM) also known as “full virtualization”.
- Paravirtualization uses modified guest operating systems that we refer to as "enlightened" guests.
- These operating systems are aware that they are being virtualized and as such don’t require virtual hardware devices.
- Instead they make special calls to the hypervisor that allow them to access CPUs, storage and network resources.

Xen Basics

- In contrast, HVM guests need not be modified, as the hypervisor will create a fully virtual set of hardware devices for the machine resembling a physical x86 computer.
- This emulation requires more overhead than the paravirtualization approach but allows unmodified guest operating systems like Microsoft Windows to run on top of the hypervisor. HVMs are supported through virtualization extensions in the CPU.
- Several iterations of these extensions have been introduced in the last decade or so, collectively known as Intel VT and AMD-V and development continues.
- The technology is now prevalent; all recent servers, many desktops and some mobile systems should be equipped with at least some extensions.

Xen Project architecture



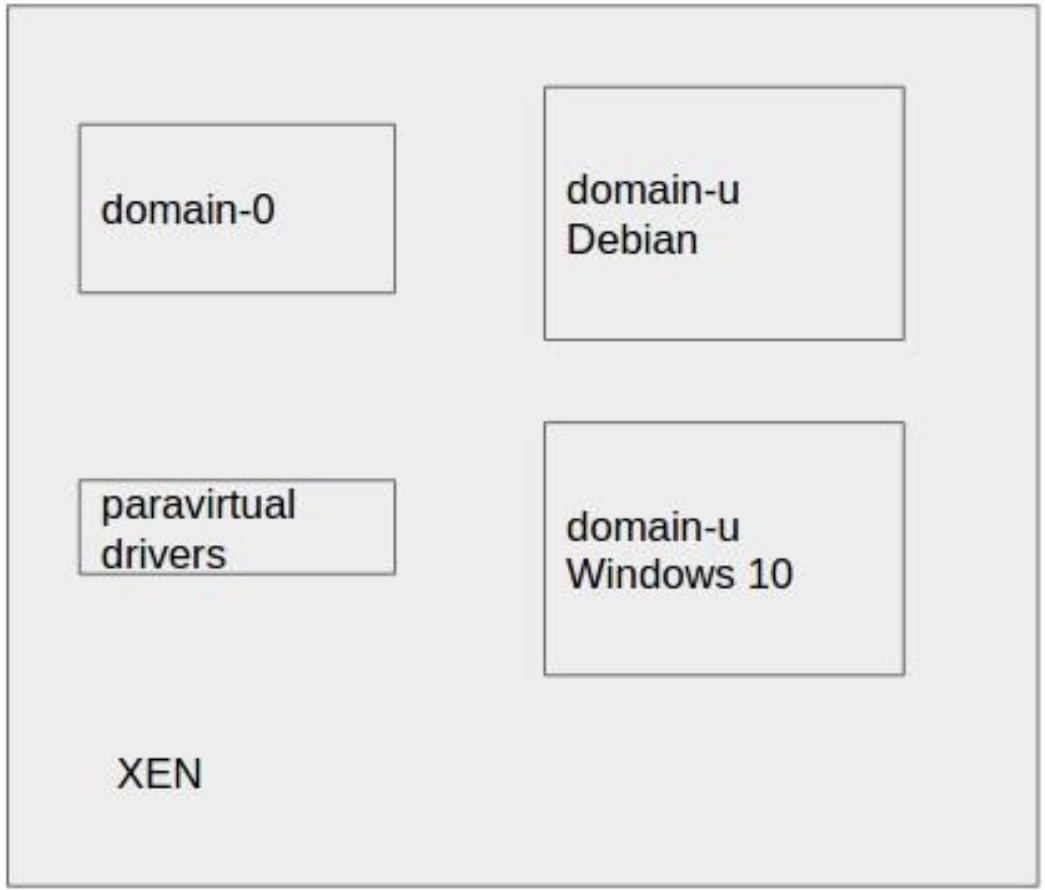
Control Domain aka Dom0

- Dom0 kernel with drivers

Guest Domains

- Your apps

Xen Arch


domain-0
domain-u
Debian
paravirtual
drivers
domain-u
Windows 10
XEN

Xen Arch

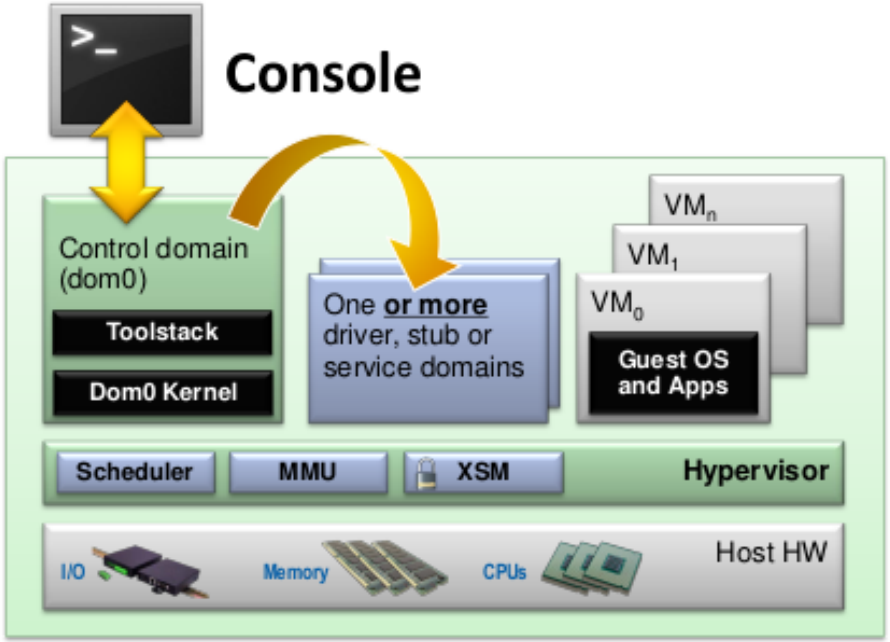
- **Xen** – this is the software that boots up when you boot the server.
- **domain-0** – this controls Xen and all the virtual machines added to the machine.
- **domain-u** – these are the virtual machines.
- **paravirtual drivers** – performance with Windows will be really slow unless you instill the open-source network, storage, bus, and other paravirtual drivers.

Xen Arch

This is the basic architecture of the Xen Project Hypervisor. We see that the hypervisor sits on the bare metal (the actual computer hardware). The guest VMs all sit on the hypervisor layer, as does dom0, the "Control Domain". The Control Domain is a VM like the guest VMs, except that it has two basic functional differences:

- The Control Domain has the ability to talk to the hypervisor to instruct it to start and stop guest VMs.
- The Control Domain by default contains the device drivers needed to address the hardware. Xen Project uses the drivers in the Control Domain's operating system to access many types of hardware.

Components



Console

- Interface to the outside world

Control Domain aka Dom0

- Dom0 kernel with drivers
- Xen Project Management Toolstack

Guest Domains

- Your apps

Driver/Stub/Service Domain(s)

- A "driver, device model or control service in a box"
- De-privileged and isolated
- Lifetime: start, stop, kill

Components

Dom0 forms the interface to the hypervisor. Through special instructions dom0 communicates to the Xen Project software and changes the configuration of the hypervisor. This includes instantiating new domains and related tasks.

- Another crucial part of dom0's role is as the primary interface to the hardware. The hypervisor doesn't contain device drivers. Instead the devices are attached to dom0 and use standard Linux drivers. Dom0 then shares these resources with guest operating systems.
- To implement paravirtualization, each paravirtualized datapath consists of two parts:
  - a "backend" that lives in dom0, which provides the virtual device and
  - a "frontend" driver within the guest domain, which allows the guest OS to access the virtual device. The backend and frontend use a high-speed software interface based on shared memory to transfer data between the guest and dom0.

Preparation

This guide requires a number of items, this checklist is what you will need:

- 64bit x86 computer with at least 1GB of RAM (this can be a server, desktop or laptop)
- Intel VT or AMD-V support (optional for PV, required for HVM and some PV optimizations)
- Sufficient storage space for your dom0 and whatever guests you want to install
- You will need the ISO image of the operating system you want to use.

Installing Xen

sda1 - /boot 200MB

sda2 - / 15GB

sda3 - swap

sda4 - reserved for LVM

Install Ubuntu on an empty machine and use the LVM (logical volume manager) option when you do that. Leave space for a logical volume for VMs when you are asked to create partitions. Or use another drive for that when you set up the VMs later.

Installing the Xen

1. Install XEN:

   sudo apt-get install xen-hypervisor-amd64

   sudo reboot

   GRUB will automatically choose to boot Xen first if Xen is installed.

1. Now you have a Xen hypervisor. Run this to confirm:

   sudo xl list

   You will see domain-0:

```
Name                                ID   Mem VCPUs      State   Time(s)
Domain-0                             0 10789     1     r--     2643.0
```

Network Configuration Using Linux Bridge

It's assumed that you are using a wired interface for your network configuration. WiFi networks are tricky to virtualize and many times not even possible.

For this example, we will use the most common Xen network setup: **bridged.**

- **Disable Network Manager :** If you are using Network Manager to control your internet connections, then you must first disable it as we will be manually configuring the connections.

  ```
  $ sudo stop network-manager
  ```

  ```
  $ echo "manual" | sudo tee /etc/init/network-manager.override
  ```

- Using bridge-utils

  ```
  $ sudo apt-get install bridge-utils
  ```

Network Configuration Using Linux Bridge

- In a bridged setup, it is required that we assign the IP address to the bridged interface. Configure network interfaces so that they persist after reboot:

```
$ sudo vi /etc/network/interfaces
auto lo eth0 xenbr0
iface lo inet loopback
iface xenbr0 inet dhcp
bridge_ports eth0
iface eth0 inet manual
```

- Restart networking to enable xenbr0 bridge:

```
$ sudo ifdown eth0 && sudo ifup xenbr0 && sudo ifup eth0
```

Network Configuration using Open vSwitch

- Install Open vSwitch

```
$ sudo apt-get install openvswitch-switch
```

- In a bridged setup, it is required that we assign the IP address to the bridged interface. Configure network interfaces so that they persist after reboot:

```
$ sudo vi /etc/network/interfaces
# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo eth0
iface lo inet loopback
iface eth0 inet manual
allow-hotplug ovsbr0
iface ovsbr0 inet dhcp
```

Network Configuration using Open vSwitch

- Set up Open vSwitch

```
$ sudo ovs-vsctl add-br ovsbr0
```

```
$ sudo ovs-vsctl set Bridge ovsbr0 stp_enable=false
other_config:stp-max-age=6 other_config:stp-forward-delay=4
```

```
$ sudo ovs-vsctl list Bridge
```

```
$ sudo ovs-vsctl add-port ovsbr0 eth0
```

- Now bring the interfaces up and acquire an IP address through DHCP. You should have your internet connection back at this point.

```
$ sudo ip link set eth0 up
```

```
$ sudo dhclient ovsbr0
```

Recommended Bridge Settings

For performance and security reasons, it's highly recommended that netfilter is disabled on all bridges.

```
$ sudo vi /etc/sysctl.conf
net.bridge.bridge-nf-call-ip6tables = 0
net.bridge.bridge-nf-call-iptables = 0
net.bridge.bridge-nf-call-arptables = 0
$ sudo sysctl -p /etc/sysctl.conf
```

```
# Note: These settings are created in /proc/sys/net. The bridge folder only appears to be created after first creating a bridge with the 'brctl' command.
```

Creating VMS

There are many options for installing guest images:

- **virt-builder:** A program for building VM disk images; part of the libguestfs set of tools
- **xen-tools:** A set of scripts for creating various PV guests
- **virt-manager:** A management system using libvirt
- Converting an existing installation
- Downloading pre-build guest images

Manually Create a PV Guest VM

Create a PV guest in LVM logical volume (LV) by doing a network installation of Ubuntu and other distributions.

List your existing volume groups (VG) and choose where you'd like to create the new logical volume.

```
$ sudo vgs
```

Create the logical volume (LV).

```
$ sudo lvcreate -L 10G -n lv_vm_ubuntu /dev/<VGNAME>
```

Confirm that the new LV was successfully created.

```
$ sudo lvs
```

Get Netboot Images

Choose an archive mirror:

```
$ sudo mkdir -p /var/lib/xen/images/ubuntu-netboot/trusty14LTS
$ cd /var/lib/xen/images/ubuntu-netboot/trusty14LTS
$ wget http://<mirror>/ubuntu/dists/trusty/main/installer-amd64/current/images/netboot/xen/vmlinuz
$ wget  http://<mirror>/ubuntu/dists/trusty/main/installer-amd64/current/images/netboot/xen/initrd.gz
```

Set Up Initial Guest Configuration

```
$ cd /etc/xen
$ cp xlexample.pvlinux ubud1.cfg
$ vi ubud1.cfg

name = "ubud1"

kernel = "/var/lib/xen/images/ubuntu-netboot/trusty14LTS/vmlinuz"
ramdisk = "/var/lib/xen/images/ubuntu-netboot/trusty14LTS/initrd.gz"
#bootloader = "/usr/lib/xen-4.4/bin/pygrub"

memory = 1024
vcpus = 1

# Custom option for Open vSwitch
vif = [ 'script=vif-openvswitch,bridge=ovsbr0' ]
disk = [ '/dev/<VGNAME>/lv_vm_ubuntu,raw,xvda,rw' ]

# You may also consider some other options
# [[http://xenbits.xen.org/docs/4.4-testing/man/xl.cfg.5.html]]
```

Start the VM

- Start the VM and connect to the console to perform the install.
- `$ sudo xl create -c /etc/xen/ubud1.cfg`

- Once installed and back to command line, modify guest configuration to use the pygrub bootloader. These lines will change

- `$ vi /etc/xen/ubud1.cfg`
- `#kernel = "/var/lib/xen/images/ubuntu-netboot/trusty14LTS/vmlinuz"`
- `#ramdisk = "/var/lib/xen/images/ubuntu-netboot/trusty14LTS/initrd.gz"`
- `bootloader = "/usr/lib/xen-4.4/bin/pygrub"`

Restart the VM

Now let's restart the VM with the new bootloader. (If the VM didn't shutdown after the install above, you may manually shut it down.)

```
$ sudo xl shutdown ubud1
```

```
$ sudo xl create -c /etc/xen/ubud1.cfg
```

Connect to the VM console : `$ sudo xl console ubud1`

Disconnect from the console : `Ctrl+]`

Manually installing an HVM Guest VM

Download Install ISO. (http://www.ubuntu.com/download/desktop)

```
sudo pvs
```

choose your VG

Create a LV (Logical Volume)

```
sudo lvcreate -L 4G -n ubuntu-hvm /dev/<VG>
```

Create Guest Config File

Create a guest config file /etc/xen/ubuntu-hvm.cfg

```
builder = "hvm"
name = "ubuntu-hvm"
memory = "512"
vcpus = 1
vif = ['']
disk = ['phy:/dev/<VG>/ubuntu-hvm,hda,w','file:/root/ubuntu-12.04-
       desktop-amd64.iso,hdc:cdrom,r']
vnc = 1
boot="dc"

xl create /etc/xen/ubuntu-hvm.cfg
vncviewer localhost:0
```

Finalizing the VM

After the install you can optionally remove the CDROM from the config and/or change the boot order.

For example /etc/xen/ubuntu-hvm.cfg:

```
builder = "hvm"
name = "ubuntu-hvm"
memory = "512"
vcpus = 1
vif = ['']
#disk = ['phy:/dev/<VG>/ubuntu-hvm,hda,w','file:/root/ubuntu-12.04-
        server-amd64.iso,hdc:cdrom,r']
disk = ['phy:/dev/<VG>/ubuntu-hvm,hda,w']
vnc = 1
boot="c"
#boot="dc"
```