

2nd ISSNSM's Tutorial on

Using Xen Virtualization in Research Projects

(Tutorial T3)

Speaker:

Kyrre Begnum

June 4, 2008

ISSNM program chaired by Burkhard Stiller, David Hausheer, University of Zürich

ISSNM laboratory organization chaired by Cristian Morariu, Peter Racz, University of Zürich

Virtualization with Xen

Kyrre Begnum, Oslo University College

kyrre.begnum@iu.hio.no

Using Xen Virtualization in Research Projects

ISSNSM 2008

It's in the news



kyrre.begnum@iu.hio.no

Using Xen Virtualization in Research Projects

ISSNSM 2008

Used in a wide range of areas

- Server consolidation
- Education and training
- High Performance Computing
- Personal convenience
- Legacy application/ environment support
- Predictable computing environment
- Research and Development
- Product demos
- Virtual hosting

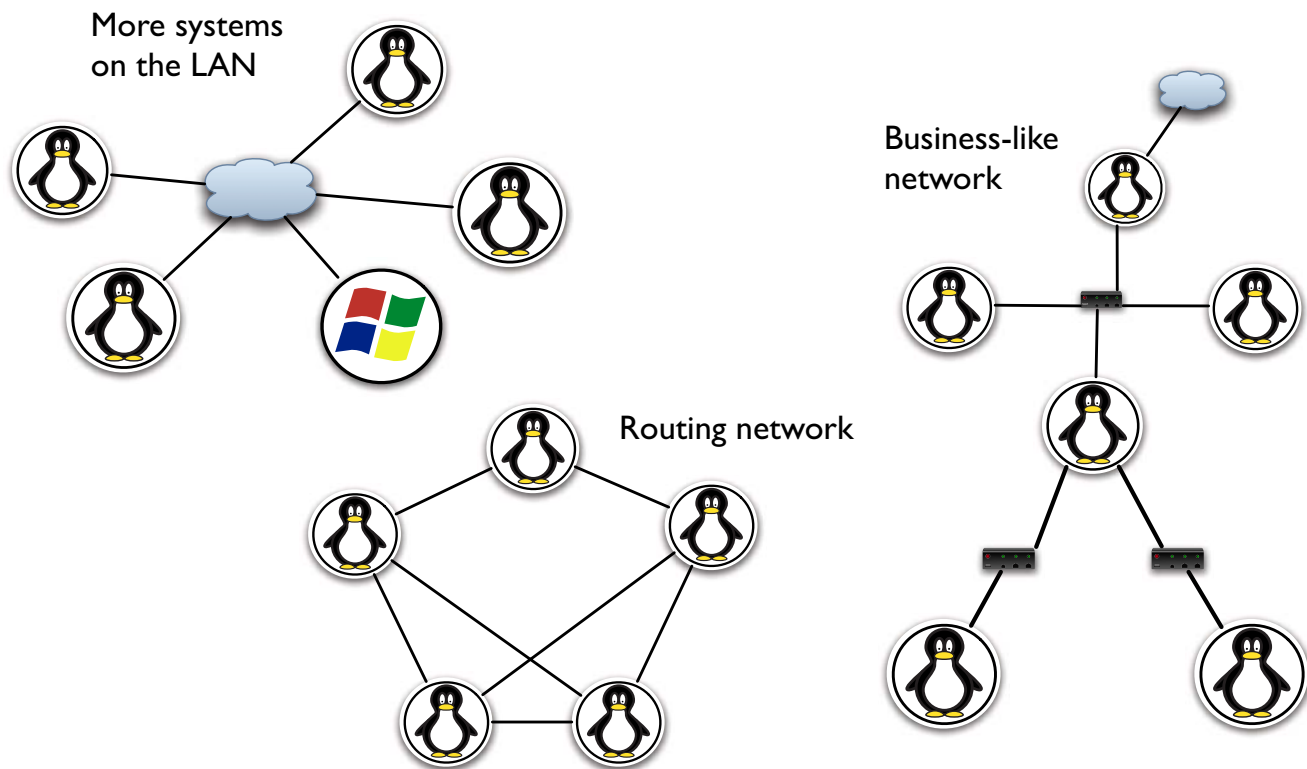
Used in a wide range of areas

- Server consolidation
- Education and training
- High Performance Computing
- Personal convenience
- Legacy application/ environment support
- Predictable computing environment
- Research and Development
- Product demos
- Virtual hosting

You are here!



Typical scenarios



What you should know after this course

- How to install Xen on a Debian-based system
- The fundamental components and under-the-hood functioning of Xen
- Building and running virtual machines
- Setting up networks and connecting virtual machines together
- Sufficient knowledge to be independent of tools and build your own solution

Basics

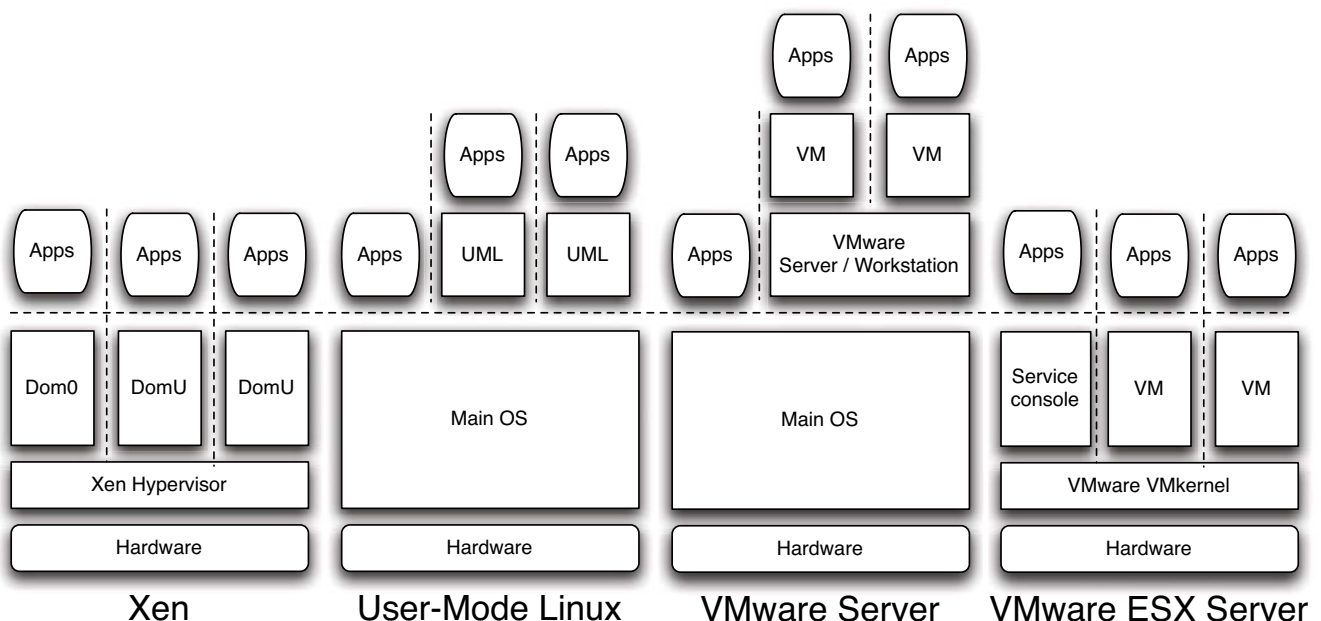
Xen

- OpenSource virtual machine monitor as part of a large architecture for dynamic service re-deployment
- Became famous with “*Xen and the art of virtualization*” in 2004
- Has become popular for data-centers where large numbers of virtual machines run on powerful servers
- Small entourage of commercial server-side management tools
- Bought by Citrix - lets keep our fingers crossed!

Xen terminology

- All virtual machines are called *domains*
- The host operating system is also a virtual machine called Domain0, or Dom0
- The other virtual machines are called DomU (User domains)
- The thin layer between the hardware and all the domains is called the *Hypervisor*

Xen architecture



The components of a virtual machine

1. **Filesystem** - This can be a filesystem image, or disk image. They are usually pre-built.
2. **Configuration file** - The name, hardware and network connections are defined here

The kernel is outside

- Paravirtualized guests have their kernel outside of the filesystem
- This goes for the initrd image as well
- Usually one uses the same kernel and initrd as dom0, but that can cause problems when using different distributions
- The kernel modules must be on the inside of the filesystem

Installing Xen

- One has three choices:
 - Install binary package from the distribution
 - Install binary package from Xen
 - Compile from source
- Each approach has benefits and drawbacks
- Xen is still under heavy development, and tends to change between releases

Xen 3.1 on Debian 4.0

Commands:

```
apt-get install bridge-utils libc6-xen
wget http://bits.xensource.com/oss-xen/release/3.1.0/bin.tgz/xen-3.1.0-install-x86\_32.tgz
tar xzf xen-3.1.0-install-x86_32.tgz
cd dist
./install.sh
depmod -a
mkinitramfs -o /boot/initrd.img-2.6.18-xen 2.6.18-xen
```

Add to /boot/grub/menu.lst

```
title Xen 3.1.0 / XenLinux 2.6
kernel /boot/xen.gz console=vga
module /boot/vmlinuz-2.6.18-xen root=/dev/hda1 ro console=tty0
module /boot/initrd.img-2.6.18-xen
```

Starting Xen

- Reboot (cross your fingers! Press your Thumbs! etc.)
- Choose the Xen hypervisor and kernel
- When booted up, run (as root):

```
/etc/init.d/xend start
```

Check status with:

```
xm list
```

Filesystems

- Paravirtualization relies on partitions with ready-made filesystems (cloning approach)
- Booting with a CD image and installing from it is not possible (in PV)
- It is the filesystem which decides the distribution of the VM
- Most tools for creating these filesystems are specific to classes of distributions

Creating Filesystems

On a Debian-based system using xen-tools

```
apt-get install xen-tools
```

```
xen-create-image --size 2Gb --hostname=first --dir . --dhcp \  
--kernel=/boot/vmlinuz-2.6.18-xen --initrd=/boot/initrd.img-2.6.18-xen \  
--debootstrap --dist=etch --mirror=http://ftp.no.debian.org/debian
```

Fetch the actual filesystem

```
cp domains/default/disk.img etch.ext3
```

In principle, you should be able to build filesystems from different distribution using xen-tools

Configuration Files

- The configuration file defines the “hardware” makeup of the virtual machine
- Here, you define memory, network cards, hard drives, number of CPUs and more
- In Xen, the configuration file is actually Python code, which is very picky on syntax!
- The benefit of using Python, is that one can “program” the configuration and make it more dynamic. (Just like PHP for web-pages)

Writing a configuration file

A simple VM:

```
# -*- mode: python; -*-
kernel = "/boot/vmlinuz-2.6.18-xen"
memory = 64
disk = [ 'file://root/etch.ext3,hda1,w' ]
root = '/dev/hda1'
extra = '2'
name = "etch"
vif = [ '' ]
```

- Note the declaration of the disk with the URI of the filesystem, device and permission
- Memory is specified in Megabytes without “M” or “MB”
- The kernel can point to any Xen-compatible kernel
- ‘vif’ is a list of network interfaces (only one is configured here)

Common commands

- Starting and stopping

```
xm create [-c] etch.cfg, xm shutdown etch
```

- Status

```
xm [--long] list, xentop
```

- Emergencies

```
xm destroy etch
```

- Console access

```
xm console etch
```

Notice, that you use the configuration file for starting, but use the VM name for the rest.



&



Useful GRUB options

```
title          Ubuntu, Xen 3.1, kernel 2.6.18
root           (hd0,0)
kernel         /boot/xen.gz noreboot console=vga dom0_mem=512M
module        /boot/vmlinuz-2.6.18-xen root=/dev/sda1 ro quiet splash
module        /boot/initrd.img-2.6.18-xen
savedefault
boot
```

- `noreboot` will stop the hypervisor from automatically rebooting in case of problems
- `dom0_mem=512M` will only assign 512MB of Ram for domain0, reserving the rest for virtual machines

Xend configuration file

- Located in `/etc/xen/xend-config.sxp`
- To adjust the networking, edit:
`(network-script 'network-bridge bridge=xenbr0')`
`(vif-script 'vif-bridge bridge=xenbr0')`
- Limit the minimum memory for dom0:
`(dom0-min-mem 196)`
- You need to restart the xend daemon after editing:
`/etc/init.d/xend restart`

SMP virtual machines

```
# -*- mode: python; -*-
kernel = "/boot/vmlinuz-2.6.18-xen"
memory = 64
disk = [ 'file://root/etch.ext3,hda1,w' ]
root = '/dev/hda1'
extra = '2'
name = "etch"
vif = [ '' ]
vcpu = 2
```

- Virtual machines get *virtual CPUs*
- The virtual CPUs are moved around between the real CPUs
- One can pin down a VCPU to a real CPU, but this might do more damage than good

Performance

- CPU usage scales about linear with the number of virtual machines
- Networking scales a bit worse, but not bad
- Disk scales worst, but it is hardest to predict
 - File images can perform very fast, if the VM is alone and dom0 has a lot of memory
 - Disk partitions (LVM) perform OK, but scale better than files

Scheduling of domains

- Xen offers basic credit scheduling for virtual machines as default
- More weight means more CPU-time
- A maximum CPU cap can be added as well
- Sufficient for most scenarios
- Use `xm sched-credit` to view credit policy
- Change the credit like this
`xm sched-credit -d domain [-w weight] [-c cpucap]`

Utilizing the configuration file

```
# -*- mode: python; -*-
name = 'myvm'
memory = 64

# Rest of code goes here
kernel = "/boot/vmlinuz-2.6-xen"
disk = [ 'file://root/' + name + '.img,hda1,w' ]
root = '/dev/hda1'
extra = '2 '
vif = [ 'bridge=xenbr0' ]
```

- Since the configuration file basically is Python code, we can utilize this to create dynamic configurations and limit errors
- The example above re-uses the name of the VM in order to point to the right filesystem

Working with different distributions

- There are few tools which can create filesystems belonging to other distributions
- Sites like <http://jailtime.org> offer downloadable filesystems for many free distributions (great for testing)
- The problem is usually the kernel and initrd (more specifically, the xen kernel modules)

Saving and restoring VMs

- Xen offers two methods for freezing a domain:
 - **Pause.** The VM exists in xm list and uses resources, but is not running:

```
xm pause <domid>
xm unpause <domid>
```
 - **Save.** The VMs memory is saved to a file and can later be restored.

```
xm save <domid> <state-file>
xm restore <state-file>
```

This is not entirely a snapshot, you need to make a copy of the disk, as well

HVM - on supported platforms

- HVM means we provide hardware emulation to the virtual machine
- This is useful for operating systems which do not support paravirtualization
- Recent CPUs and motherboards support HVM
 - Intel VT / AMD-V
 - You can test if you have HVM support:

```
xm dmesg | grep HVM
```

Installing a HVM domain

- HVM domains can be installed from a CD image, in which case they need an empty disk-file
- A HVM filesystem is a disk-image, not a filesystem image
- Xen HVM uses QEMU for device emulation, which performs slower than paravirtualization
- HVM domains can be accessed using SDL or VNC

Installing a HVM domain

```
# -*- mode: python; -*-
kernel = "/usr/lib/xen/boot/hvmloader"
builder='hvm'
usb=1
usbdevice='tablet'
boot = 'd'
vnc=1
vncviewer=0
vncunused=0
device_model = '/usr/lib/xen/bin/qemu-dm'
vncpasswd='akes9womb'
memory = 256
disk = [ 'phy:/dev/vg0/winXP,ioemu:hda,w', 'file://root/winXP.iso,hdd:cdrom,r' ]
root = '/dev/hda1'
extra = '2'
name = 'windows.os10'
vif = [ 'type=ioemu,bridge=lan.os10' ]
```

Installing a HVM domain

```
# -*- mode: python; -*-
kernel = "/usr/lib/xen/boot/hvmloader"
builder='hvm'
usb=1
usbdevice='tablet'
boot = 'd'
vnc=1
vncviewer=0
vncunused=0
device_model = '/usr/lib/xen/bin/qemu-dm'
vncpasswd='akes9womb'
memory = 256
disk = [ 'phy:/dev/vg0/winXP,ioemu:hda,w', 'file://root/winXP.iso,hdd:cdrom,r' ]
root = '/dev/hda1'
extra = '2'
name = 'winXP'
vif = [ 'type=ioemu,bridge=lan.os10' ]
```

Change to 'c'
after installation



Live migration

- Enables virtual machines to move between servers with no downtime
- Live migration is very fast, but depends on the amount of memory assigned to the VM
- There are two important dependencies:
 - The two servers have the same CPU architecture
 - Both servers have concurrent access to the filesystem

Enabling live migration

- Accept domains from other servers
In `/etc/xen/xend-config.sxp`
`(xend-address '')`
`(xend-relocation-address '')`
`(xend-relocation-hosts-allow '')`
- Restart the xend daemon
`/etc/init.d/xend restart`
- Live migration of a VM can be done like this
`xm migrate --live <domid> <new-server>`

Performance tips

- Create a swap file inside the VM filesystem
- Use separate disk for VM filesystems
- Make sure dom0 runs little / few services
- Disable screensavers!
- Ordinary performance tools will show “ALL OK” even if the dom0 is struggling, look for alternative measures

PCI-passthrough

- Xen supports PCI-passthrough, which will give a virtual machine direct, exclusive access to a hardware device on the PCI-bus.
- This is useful for hardware testing on multiple distributions
- Support is not superb at the moment, but should work with a little bit of mailing-list reading

Exercises

Basics

1. Install Xen
2. Create three virtual machines
3. Start all three virtual machines and connect to their console
4. Are the virtual machines online?
5. Look for more filesystems on <http://jailtime.org>
6. Start all three virtual machines and connect to their console



1. Install `hdparm` and run
`hdparm -Tt /dev/hda1`
What results do you get and what if you run it on two and three VMs at the same time?
2. For a quick CPU benchmark, run:
`time echo "1234567 ^ 123456" | bc`
Try on one VM, dom0 only and on several at the same time. Write down your results.
3. Use the credit scheduler to give one domain far less weight. Recreate the CPU test from above. Were you able to influence the end-result?
4. Edit `/boot/grub/menu.lst` and limit the memory of dom0
5. Create an SMP virtual machine and re-do some of the tests.
6. For a quick CPU benchmark, run:
`time echo "1234567 ^ 123456" | bc`
Try on one VM, dom0 only and on several at the same time. Write down your results.
7. Use the credit scheduler to give one domain a `cpucap`. Recreate the CPU test from above. Discuss the benefits / drawbacks of weight vs. `cpucap`.
8. Experiment with saving and restoring domains

Part II - Networking++

Networking

- All virtual machines are either connected to the LAN or to local bridge devices
- Bridge devices function like layer two switches
- One can create arbitrary many bridges and create complex network topologies
- Traffic shaping and QoS is less supported
- Ideal point to “tap” for network monitoring

Bridge devices

- Bridge devices can be created using the `brctl` command

```
brctl addbr myswitch
```

- Virtual machines are connected to the switches in the configuration file

```
vif = [ 'bridge=myswitch' ]
```

- The switch has to be created before the virtual machine is started
- Status can using `brctl show`
- Access to the LAN is through the bridge device `xenbr0` (renamed after Xen 3.2)

Several network cards

```
# -*- mode: python; -*-  
kernel = "/boot/vmlinuz-2.6.18-xen"  
memory = 64  
disk = [ 'file://root/etch.ext3,hda1,w' ]  
root = '/dev/hda1'  
extra = '2'  
name = "etch"  
vif = [ 'bridge=xenbr0', 'bridge=myswitch' ]
```

- Network cards are added to the `vif` array
- The order of the declaration mirrors the order of the devices (`eth0`, `eth1` etc.)

The network card seen from dom0

- Every NIC has a corresponding interface on dom0
- The syntax for the interface is
`vif[domid].[nic]`
- When collecting data from it using `ifconfig`, remember to switch RX and TX
- You can manually connect and disconnect these interfaces to other bridge-devices

Setting a MAC address

```
# -*- mode: python; -*-
kernel = "/boot/vmlinuz-2.6.18-xen"
memory = 64
disk = [ 'file://root/etch.ext3,hda1,w' ]
root = '/dev/hda1'
extra = '2'
name = "etch"
vif = [ 'bridge=xenbr0,mac=00:16:3E:51:89:6A' ]
```

- Some software licenses use the MAC address for authentication
- Xen can have arbitrary MAC addresses for its domains (unlike VMware)
- Changing the MAC address may confuse some distributions

Additional disks

```
# -*- mode: python; -*-
kernel = "/boot/vmlinuz-2.6.18-xen"
memory = 64
disk = [ 'file://root/etch.ext3,hda1,w', 'phy:/dev/vg0/vmdisk,hdb1,w' ]
root = '/dev/hda1'
extra = '2'
name = "etch"
vif = [ 'bridge=xenbr0' ]
```

- Disks are added to the disk array
- Notice the different URI of the block-device rather than the file
- Disks can only be shared if defined as read-only
- Note, that the device node needs to be present on the VM in order to use the disk

Creating empty filesystem files

- Disks can be created as simple files

```
dd if=/dev/zero of=/root/xendisk bs=1M count=1400
mkfs.ext3 /root/xendisk
# optional
mount -o loop /root/xendisk /mnt
```

- They can actually be added in run-time:

```
xm block-attach etch file://root/xendisk /dev/hdc ro
```

Next, inside the running VM:

```
mknod /dev/hdc b 22 0
mount /dev/hdc /mnt
```

Adding software to the VM while offline

- Mount the filesystem
mount -o loop etch.ext3 /mnt
- Chroot into the mounted filesystem
chroot /mnt bash
- Install packages and edit files
- Unmount the filesystem
umount /mnt

Automation

- In research projects, you need to re-run experiments a number of times
- Creation and re-creation of virtual machines may scale badly with the number of VMs and their network topology
- Tools like MLN <http://mln.sourceforge.net> can function as an abstraction layer above Xen
- Look for these kinds of solutions or write your own once you've decided to stick with Xen

Topics we left untouched

- What if you want to configure parameters of the filesystem itself (users, software packages, network, etc.)?
- Xen can change the number of CPUs and amount of memory **ONLINE!**
- VMs can also grow network cards (and simulate roaming)
- Monitoring
- Installing Xen from distribution packages

Exercises

Networking++

1. Create a star topology consisting of three virtual machines
2. Make one of the virtual machines become a gateway with two network cards
3. Create an empty filesystem and add it to a running VM
4. VM PING-PONG!
 1. Enable live migration (shown on one of the slides)
 2. Download ttylinux from:
http://legolas.iu.hio.no/ttylinux_xen.tar.gz
 3. Edit the name of the VM so that it has your name
 4. Send the VM to other machines when they are ready!

Thank you :)