Chapter 28 Virtualization Overview - Notes

28.2 Introduction

Virtualization entails creating (in software) an abstracted complete machine environment, under which end user software runs unaware of the physical details of the machine it is running on.

Quite a few approaches to accomplish this mission, but generally involve host operating system and filesystem which directly uses hardware. One or more guest systems run on top (or underneath, depending on how you look at it) of host system at lower level of privilege. Access to hardware such as network adapters usually done through some kind of virtual device, which may access physical device when required.

Many uses for virtual machine. Major ones: more efficient use of hardware, software isolation, security, stability, safe development options, ability to run multiple operating systems at same time without rebooting.

28.3 Learning Objectives:

- Understand the concepts behind **virtualization** and how it came to be widely used.
- Understand the rules of **hosts** and **guests**.
- Discuss the difference between **emulation** and **virtualization**.
- Distinguish between the different types of **hypervisors**.
- Be familiar with how Linux distributions use and depend on **libvirt**.
- Use the qemu hypervisor.
- Install, use, and manage KVM.

28.4 What Is Virtualization?

In this chapter, will be concerned with creation, deployment, and maintenance of Virtual Machines (VMs).

Virtual Machines: virtualized instance of an entire operating system, may fulfill the role of a **server** or a **desktop/workstation**.

Outside world sees the VM as if it were actual physical machine, present somewhere on network. Applications running in VM generally unaware of their non-physical environment.

Other kinds of virtualizations include:

- **Network**

  Details of actual physical entwork, such as type of hardware, routers, etc. abstracted and need not be known by software running on it and configuring it: Software Defined Networking or SDN.

- **Storage**

  Multiple network storage devices configured to look like one big storage unit, such as disk: Network Attached Storage or NAS.

- **Application**

  Application isolated into standalone format, such as **container**, which will be discussed later.

There are differences between physical and virtual machines that are not always easy to ignore. Eg. performance tuning at low level needs to be done (separately) on both VM and physical machine residing underneath it.

28.5 Virtualization History

Virtualization has long proud history. Implemented originally on mainframes decades ago:

- Enables better hardware utilization
- Operating systems often progress more quickly than hardware
- It is microcode driven
- Not particularly user-friendly

Later on, virtualization technology migrated to PCs and workstations:

- Initially, it was done using **Emulation**
- CPUs enhanced to support virtualization led to boost in performance, easier configuration, more flexibility in VM installation and migration

From early **mainframes** to **mini-computers**, virtualization has been used for expanding limit, debugging and administration enhancements.

Today, virtualization found everywhere in many forms. Each of the different forms and implementations that are available provide particular advantages.

28.6 Hosts and Guests

Host: underlying physical operating system managing one or more virtual machines.

Guest: the VM which is an instance of a complete operating system, running one or more applications. Sometimes, guest also called **client**. In most cases, guest should not care what host it is running on, can be **migrated** from one host to another, sometimes while actually running.

In fact, possible to convert physical machines to virtual ones, by copying the entire contents into a VM. Specialized software utilities can make this easier.

Low-level performance tuning on areas such as CPU utilization, networking throughput, memory utilization, often best done on host, as guest may have only simulated quantities not directly useful.

Application tuning done most on guest.

28.7 Emulators

First implementations of virtualization on PC architecture -> through the use of **emulators**. Application running on current operating system would appear to another OS as specific hardware environment. Emulators generally do not require special hardware to operate.

Qemu -> one such emulator.

virt-emu

Emulator

28.8 Emulation vs. Virtualization

An Emulator runs completely in software. Hardware constructs replaced by software. Useful for running virtual machines on

different architectures, eg. running pretend ARM guest machine on X86 host. Emulation often used for developing operating system for new CPU, even before hardware is available. Performance relatively slow.

28.9 Types of Virtualization Hypervisors

Host system (besides functioning normally with respect to software that it runs) also acts as hypervisor that initiates, terminates, manages guests. Also called **Virtual Machine Monitor** (**VMM**).

Two basic methods of virtualization:

- **Hardware virtualization** (also known as **Full Virtualization**)

  Guest system runs without being aware it is running as virtualized guest, does not require modifications to be run in this fashion.

- **Para-virtualization**

  Guest system is aware it is running in virtualized environment, has been modified specifically to work with it.

Recent CPUs from Intel and AMD incorporate virtualization extensions to the x86 architecture that allow the hypervisor to run fully virtualized (ie. unmodified) guest operating systems with only minor performance penalties.

The Intel extension (Intel Virtualization Technology), usually abbreviated as VT, IVT, VT-32, or VT-64, also known under the development code name of Vanderpool. Has been available since the spring of 2005.

The AMD extension usually called AMD-V, still sometimes referred to by the development code name of Pacifica.

For detailed explanation and comparison of how these two extensions work, see the Xen and the new processors article.

Can check directly if your CPU supports hardware virtualization extensions by looking at `/proc/cpuinfo`. If you have an IVT-capable chip, will see `vmx` in `flags` field. If you have an AMD-V capable chip, will see `svm` in same field. May also have to ensure virtualization capability is turned on in your CMOS.

While choice of operating systems tends to be more limited for para-virtualized guests, originally they tended to run more efficiently than fully virtualized guests. Recent advances in virtualization techniques have narrowed or eliminated such advantages, and the wider availability of the hardware support needed for full virtualization has made para-virtualization less advantageous and less popular.

Most modern hardware has hardware virtualization abilities (must be turned on in BIOS).

28.10 External and in-Kernel Hypervisors

Hypervisor can be:

- External to the host operating system kernel: VMware
- Internal to the host operating system kernel: KVM

In this course, will concentrate on KVM, as it is all open source, and requires no external third party hypervisor program.

28.11 Dedicated Hypervisor

Going past emulation, merging of hypervisor program into specially-designed lightweight kernel was next step in Virtualization deployment.

VMware ESX (and related friends): an example of hypervisor embedded into operating system.

virt-hyper

Dedicated Hypervisor

28.12 Hypervisor in the Kernel

The KVM project added hypervisor capabilities into the Linux kernel.

As discussed, specific CPU chip functions and facilities were required and deployed for this type of virtualization.

virt-kvm

Hypervisor in the Kernel

28.13 libvirt

The libvirt project: toolkit to interact with virtualization technologies. Provides management for virtual machines, virtual networks, and storage. Available on all enterprise Linux distributions.

Many application programs interface with libvirt. Some of the most common ones: **virt-manager**, **virt-viewer**, **virt-install**, **virsh**.

Complete list of currently supported hypervisors can be found on libvirt website:

- QEMU/KVM
- Xen
- Oracle VirtualBox
- VMware ESX
- VMware Workstation/Player
- Microsoft Hyper-V
- IBM PowerVM (phyp)
- OpenVZ
- UML (User Mode Linux)
- LXC (Linux Containers)
- Virtuozzo
- Bhyve (The BSD Hypervisor)
- Test (Used for testing)

28.14 Programs Using libvirt

Many utilities using **libvirt**. Exact list will depend on Linux distribution. Full list can be found on libvirt website.

In this course, will work through the use of robust GUI, **virt-manager**, rather than make much use of command line utilities, which lead to more flexibility, as well as use on non-graphical servers.

virtprogs

28.15 What is QEMU?

QEMU stands for **Q**uick **EMU**lator. Originally written by Fabrice Bellard in 2002. (Bellard is also known for feats such as holding, at one point, the world record for calculating **π**, reaching 2.7 trillion digits.)

QEMU: hypervisor that performs hardware **emulation**, or **virtualization**. Emulates CPUs by dynamically translating binary instructions between host architecture and emulated architecture.

Host and emulated architectures may be different, or the same. Numerous choices for both host and guest operating systems.

QEMU can also be used to emulate just particular applications, not entire operating systems.

By itself, QEMU much slower than host machine. But, can be used together with **KVM** (**K**ernel **V**irtual **M**achine) to reach speeds close to those of native host.

Guest operating systems do not require rewriting to run under QEMU. QEMU can save, pause, restore a virtual machine at any time. QEMU is a free software licensed under the GPL.

QEMU has the capability of supporting many architectures, including: IA-32 (i386), x86-64, MIPS, SPARC, ARM, SH4, PowerPC, CRIS, MicroBlaze, etc.

Cross-compilation abilities of QEMU make it extremely useful when doing development for embedded processors.

In fact, QEMU has often been used to develop processors which have either not yet been physically produced, or released to market.

28.16 Third Party Hypervisor Integration

Used by itself, QEMU relatively slow. However, can be integrated with third party hypervisors, and then reach near native speeds. Note: some of these systems are very close cousins of QEMU; others are more remote. A few main ones:

- KVM offers particularly tight integration with QEMU. WHen host and target are same architecture, full acceleration and high speed delivered. KVM native to **Linux**. Will discuss this is detail.
- Xen, also native to Linux, can run in hardware virtualization mode if architecture offers it, as does x86 and some ARM variants.
- Oracle Virtual Box can use qcow2 formatted images, and has very close relationship with QEMU.

In this course, recommend using (and will use in labs) **virt-manager** to configure and run virtual machines. Will also give instructions on how to run them using **qemu** command line utilities.

28.17 Image Formats

QEMU supports many formats for disk image files. However, only two used primarily, with the rest being available for historical reasons and for conversion utilities. These two main formats:

- **raw** (default)

  The simplest and easiest format to export to other non-QEMU emulators. Empty sectors do not take up space.

- **qcow2**

  COW stands for **C**opy **O**n **W**rite. Many options. See **man qemu-img** for more details.

To get list of supported formats:

```
c7:/tmp> qemu-img --help | grep formats:
Supported formats: vfat vpc vmdk vhdx vdi ssh sheepdog rbd raw host_cdrom host_floppy host_device file \
qed qcow2 qcow parallels nbd iscsi gluster dmg tftp ftps ftp https http cloop bochs blkverify blkdebug
```

Note in particular:

- **vdi**: Used by Oracle VirtualBox
- **vmdk**: Used by VMware

Note: `qemu-img` can be used to translate betw een formats, eg.:

```
$ qemu-img convert -O vmdk myvm.qcow2 myvm.vmdk
```

28.18 KVM and Linux

KVM uses the Linux kernel for computing resources including memory management, scheduling, synchronization, and more. When running virtual machine, KVM engages in co-processing relationship w ith Linux kernel.

In this format, KVM runs the virtual machine monitor w ithin one or more of the CPUs, using VMX or SVM instructions. At the same time, Linux kernel is executing on other CPUs.

The virtual machine monitor runs the guest, w hich is running at full hardw are speed, until it executes an instruction that causes the virtual machine monitor to take over.

At this point, the virtual machine monitor can use any Linux resource to emulate a guest instruction, restart the guest at the last instruction, or do something else.

By loading the KVM modules and starting a guest, turn Linux into hypervisor. The Linux personality is still there, but also have the hardw are virtual machine monitor. Can control the Virtual Machine using standard Linux resource and process control tools, such as **cgroups**, **nice**, **numactl**, etc.

KVM first appeared (pre-merge) as part of a Window s Virtual Desktop product. At the time of its upstream merge in 2007, KVM required recent x86-64 processors. On x86-64 platforms, KVM primarily (but not alw ays) a driver for the processor's virtualization subsystem.

KVM appeared as a trio of Linux kernel modules in 2007. When paired w ith modified version of QEMU (also provided at the same time), created a hypervisor that used the Linux kernel for most of its runtime services.

Shortly after Avi Kivity, the author of KVM, submitted source code to the Linux development community, Linus merged KVM into his Linux tree. This w as surprising to a lot of folks.

28.19 Managing KVM

Many low level commands for creating, converting, manipulating, deploying, maintaining virtual machine images.

Managing KVM can be done w ith both command line and graphical interfaces.

Command line tools include: **virt-*** and **qemu-***. Graphical interfaces include virt-manager, kimchi, OpenStack, oVirt, etc.

As you develop more expertise, w ill become practiced in using them. But, for all basic operations, **virt-manager** w ill suffice and that is w hat w e w ill use.

lsvirtqemu

##

Back to top