Virtual Machines and Containers
Outline

• Fundamental idea
• History
• Benefits and features
• Building Blocks
• Types of virtual machines and their implementations
• Virtualization and operating system components
• Examples
Fundamental Idea

- Abstract hardware of a single computer into several different execution environments
  - Similar to layered approach
  - But layer creates virtual system (virtual machine, or VM) on which operation systems or applications can run

- Several components
  - Host – underlying hardware system
  - Virtual machine manager (VMM) or hypervisor – creates and runs virtual machines by providing interface that is identical to the host
    - Except in the case of paravirtualization
  - Guest – process provided with virtual copy of the host
    - Usually an operating system
  - Single physical machine can run multiple operating systems concurrently, each in its own virtual machine
System Models

Non-virtual machine

<table>
<thead>
<tr>
<th>hardware</th>
<th>kernel</th>
<th>processes</th>
</tr>
</thead>
</table>

Virtual machine

machine
Implementation of VMMs

Vary greatly, with options including:

- **Type 0 hypervisors** - Hardware-based solutions that provide support for virtual machine creation and management via firmware
  - IBM LPARs and Oracle LDOMs are examples

- **Type 1 hypervisors** - Operating-system-like software built to provide virtualization
  - including VMware ESX, Joyent SmartOS, and Citrix XenServer

- **Type 1 hypervisors** – Also includes general-purpose operating systems that provide standard functions as well as VMM functions
  - including Microsoft Windows Server with HyperV and RedHat Linux with KVM

- **Type 2 hypervisors** - Applications that run on standard operating systems but provide VMM features to guest operating systems
  - including VMware Workstation and Fusion, Parallels Desktop, and Oracle VirtualBox
Implementation of VMMs – Other Variations

• Other variations include:
  ° Paravirtualization - Technique in which the guest operating system is modified to work in cooperation with the VMM to optimize performance
  ° Programming-environment virtualization - VMMs do not virtualize real hardware but instead create an optimized virtual system
    • used by Oracle Java and Microsoft.Net
  ° Emulators – Allow applications written for one hardware environment to run on a very different hardware environment, such as a different type of CPU
  ° Application containment - Not virtualization at all but rather provides virtualization-like features by segregating applications from the operating system, making them more secure, manageable
    • including Oracle Solaris Zones, BSD Jails, and IBM AIX WPARs

• Much variation due to breadth, depth and importance of virtualization in modern computing
History

- First appeared in IBM mainframes in 1972
- Allowed multiple users to share a batch-oriented system
- Formal definition of virtualization helped move it beyond IBM
  - A VMM provides an environment for programs that is essentially identical to the original machine
  - Programs running within that environment show only minor performance decreases
  - The VMM is in complete control of system resources
- In late 1990s Intel CPUs fast enough for researchers to try virtualizing on general purpose PCs
  - Xen and VMware created technologies, still used today
  - Virtualization has expanded to many OSes, CPUs, VMMs
Benefits and Features (1)

- Host system protected from VMs
- VMs protected from each other
  - Sharing is provided via shared file system volume, network communication
- Freeze, suspend, running VM
  - Then can move or copy somewhere else and resume
  - Snapshot of a given state, able to restore back to that state
    - some VMMs allow multiple snapshots per VM
  - Clone by creating copy and running both original and copy
- Run multiple, different OSes on a single machine
  - Consolidation, app dev, …
Benefits and Features (2)

- Enables OS research
- Can improve system development efficiency
- *Templating* – create an OS + application VM, provide it to customers, use it to create multiple instances of that combination
- *Live migration* – move a running VM from one host to another!
  - No interruption of user access
- All those features taken together ➔ cloud computing
  - Using APIs, programs tell cloud infrastructure (servers, networking, storage) to create new guests, VMs, virtual desktops, …
Building Blocks

Generally difficult to provide an exact duplicate of underlying machine

- Especially if only dual-mode operation available on CPU
- But getting easier over time as CPU features and support for VMM improves
- Most VMMs implement virtual CPU (VCPU) to represent state of CPU per guest as guest believes it to be
  - when guest context switched onto CPU by VMM, information from VCPU loaded and stored
- Several techniques, as described in next slides
Trap and Emulate (1)

Dual mode CPU means guest executes in user mode

- Kernel runs in kernel mode
- Not safe to let guest kernel run in kernel mode too
- So VM needs two modes – virtual user mode and virtual kernel mode
  - both of which run in real user mode
- Actions in guest that usually cause switch to kernel mode must cause switch to virtual kernel mode
Trap and Emulate (2)

• How to switch from virtual user mode to virtual kernel mode?
  ◦ Attempting a privileged instruction in user mode causes an error -> trap
  ◦ VMM gains control, analyzes error, executes operation as attempted by guest
  ◦ Returns control to guest in user mode
  ◦ Known as trap-and-emulate
  ◦ Most virtualization products use this at least in part

• User mode code in guest runs at same speed as if not a guest

• But kernel mode privileged code runs slower due to trap-and-emulate
  ◦ Especially a problem when multiple guests running, each needing trap-and-emulate

• CPUs adding hardware support, more CPU modes to improve virtualization performance
Trap and Emulate Implementation

User Processes

Privileged Instruction

Operating System

Trap

Guest

VMM

Emulate Action

Update

VMM

VCPU

Return

User Mode

Kernel Mode

CIS 415, Spring 2017
Virtual Machines and Containers
Binary Translation (1)

Some CPUs don’t have clean separation between privileged and nonprivileged instructions

- Earlier Intel x86 CPUs are among them
  - earliest Intel CPU designed for a calculator

- Backward compatibility means difficult to improve

- Consider Intel x86 popf instruction
  - loads CPU flags register from contents of the stack
  - if CPU in privileged mode -> all flags replaced
  - if CPU in user mode -> some flags replaced
    - No trap is generated
Binary Translation (2)

- Other similar problem instructions we will call special instructions
  - Caused trap-and-emulate method considered impossible until 1998

- Binary translation solves the problem
  - Basics are simple, but implementation very complex
  - If guest VCPU is in user mode, guest can run instructions natively
  - If guest VCPU in kernel mode (guest believes it is in kernel mode)
    - VMM examines every instruction guest is about to execute by reading a few instructions ahead of program counter
    - non-special-instructions run natively
    - special instructions translated into new set of instructions that perform equivalent task (for example changing the flags in the VCPU)
Binary Translation (3)

- Implemented by translation of code within VMM
- Code reads native instructions dynamically from guest, on demand, generates native binary code that executes in place of original code
- Performance of this method would be poor without optimizations
  - Products like VMware use caching
    - translate once, and when guest executes code containing special instruction cached translation used instead of translating again
    - testing showed booting Windows XP as guest caused 950,000 translations, at 3 microseconds each, or 3 second (5%) slowdown over native
Binary Translation Implementation

User Processes

(VMM Reads Instructions)
Special Instruction
Operating System

Guest
VMM

Translate
Execute Translation

Update

VMM
User Mode
Kernel Mode

Return

VCPU
Nested Page Tables

- Memory management another general challenge to VMM implementations
- How can VMM keep page-table state for both guests believing they control the page tables and VMM that does control the tables?
- Common method (for trap-and-emulate and binary translation) is nested page tables (NPTs)
  - Each guest maintains page tables to translate virtual to physical addresses
  - VMM maintains per guest NPTs to represent guest’s page-table state
    - just as VCPU stores guest CPU state
  - When guest on CPU → VMM makes that guest’s NPTs the active system page tables
  - Guest tries to change page table → VMM makes equivalent change to NPTs and its own page tables
  - Can cause many more TLB misses → much slower performance
Nested Page Tables

```
Guest Virtual Address
          ↓
                     Kernel Paging Data Structures
                     ↓
          Guest Physical Address
```

```
PML4 Directory Ptr Directory Table Offset
```

```
VMM Nested Page Table Data Structure
```

```
PML4E PDPT PDE PTE Phy Addr
```

```
Host Physical Address
```

CIS 415, Spring 2017 Virtual Machines and Containers 19
Hardware Assistance

- All virtualization needs some HW support
- More support \(\Rightarrow\) more feature rich, stable, better performance of guests
- Intel added new VT-x instructions in 2005 and AMD the AMD-V instructions in 2006
  - CPUs with these instructions remove need for binary translation
  - Generally define more CPU modes – “guest” and “host”
  - VMM can enable host mode, define characteristics of each guest VM, switch to guest mode and guest(s) on CPU(s)
  - In guest mode, guest OS thinks it is running natively, sees devices (as defined by VMM for that guest)
    - access to virtualized device, priv instructions cause trap to VMM
    - CPU maintains VCPU, context switches it as needed
- HW support for NPTs, DMA, interrupts as well, over time
Linux Containers

Includes material extracted from two presentations available on the web:

- https://docs.google.com/presentation/d/1SEWJ1nuVFOXifcDhLOZXQ1qt-d-rYcsSddYlrDTfOV1s/edit#slide=id.p4
- https://docs.google.com/presentation/d/1WdByuxWgayPb-RstO-XaENSqVPGP7h6t3GS6W4jk4tk/edit#slide=id.p
Virtual Machine vs Linux Container

- A virtual machine (VM) is a software implementation of a physical machine (for example, a computer) that executes programs like a physical machine.
- A Linux Container (LXC) provides operating system-level virtualization through a virtual environment that has its own process and network space, instead of creating a full-fledged virtual machine.
  - LXC is often referred to as lightweight virtualization.
In other words …

The primary difference between VM virtualization and Linux Containers is that virtual machines require a separate kernel instance upon which to execute, while containers can be deployed from the host operating system.
VM & LXC Diagrams

Figure 1: Hypervisor Diagram
Figure 2: Container Diagram
### History of Containers

<table>
<thead>
<tr>
<th>Year</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>2006</td>
<td>Google begins developing Containers/cgroups</td>
</tr>
<tr>
<td>2008</td>
<td>Some of the cgroup technology used by Google makes its way into the Linux Kernel and the LXC (LinuX Containers) project is born</td>
</tr>
<tr>
<td>2010</td>
<td>Enterprises start using cloud technologies</td>
</tr>
<tr>
<td>2013</td>
<td>Release of Linux Kernel 3.8 in which all Linux container technologies (at least as far as the kernel goes) are unified</td>
</tr>
<tr>
<td>2013</td>
<td>Docker introduced – simplifies deployment</td>
</tr>
<tr>
<td>2014</td>
<td>(September) – Docker is backed by venture capital</td>
</tr>
</tbody>
</table>
Container characteristics

- Containers are lightweight virtual machines.
- A large number of containers can run simultaneously on a host machine
  - Theoretical maximum is 6,000 containers and 12,000 bind mounts of root file system directories
- Containers are faster to create (than VMs) and have low start-up times
- Average spin-up time of a hypervisor virtual machine is measured in tens of seconds
- Containers can boot 1000x faster than VMs; their disk and memory footprint are also much lower
Underlying kernel mechanisms

- cgroups — manager resources for groups of processes
- namespaces — per process resource isolation
- seccomp — limit available system calls
- capabilities — limit available privileges
- CRIU — checkpoint/restore (with kernel support)
Namespaces

- Namespaces limit the scope of kernel-side names and data structures at process granularity

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mnt</td>
<td>mount points, filesystems</td>
</tr>
<tr>
<td>pid</td>
<td>processes</td>
</tr>
<tr>
<td>net</td>
<td>network stack</td>
</tr>
<tr>
<td>ipc</td>
<td>system V IPC</td>
</tr>
<tr>
<td>uts</td>
<td>unix timesharing – domain name, etc</td>
</tr>
<tr>
<td>user</td>
<td>UIDs</td>
</tr>
</tbody>
</table>
Other considerations

• Everything at Google runs in a Container; Google starts $\sim 2 \times 10^9$ Containers/week

• Docker – a container solution; a Linux container engine with:
  ◦ multiple backend drivers
  ◦ it is application- rather than machine-centric
  ◦ it provides application build tools
  ◦ supports diff-based deployment of updates
  ◦ supports versioning (git-like) and reuse
  ◦ supports links (tunnels) between containers