EE 660: Computer Architecture
Cloud Architecture: Virtualization

Yao Zheng
Department of Electrical Engineering
University of Hawai‘i at Mānoa

Based on the slides of Prof. Roy Campbell & Prof Reza Farivar
Agenda

- Introduction to Virtualization
- Implementations of Virtualization
- OS based Virtualization: Xen
- Containers: Dockers
Sharing Resources

• Economics of Clouds requires sharing resources

• How do we share a physical computer among multiple users?

• Answer: Abstraction
  • Introduce an abstract model of what a generic computing resource should look like
  • The physical computer resource then provides this abstract model to many users
Layers of Abstraction

- Introduce an abstract model of what a generic computing resource should look like
- The physical computer resource then provides this abstract model to many users
- Virtualization avoids creating dependencies on physical resources
Layers of Abstraction

- Introduce an abstract model of what a generic computing resource should look like
- The physical computer resource then provides this abstract model to many users
- Virtualization avoids creating dependencies on physical resources
Virtualization: Foundation of Cloud Computing

• Virtualization allows distributed computing models without creating dependencies on physical resources

• Clouds are based on Virtualization
  • offer services based mainly on virtual machines, remote procedure calls, and client/servers
  • provide lots of servers to lots of clients (e.g. phones)

• Simplicity of use and ease of programming requires allowing client server paradigms to be used to construct services from lots of resources
Types of virtualization

- Native, full
- Hardware assisted
- Para-virtualization
- OS level
  - Containers
  - Jails
  - Chroot
  - Zones
  - Open-VZ → Virtuozzo
Native and Full Virtualization

• the virtual machine simulates enough hardware to allow an unmodified "guest" OS (one designed for the same CPU) to be run in isolation.

• Examples:
  • VirtualBox
  • Virtual PC
  • Vmware
  • QEMU
Hardware enabled virtualization

• the virtual machine has its own hardware and allows a guest OS to be run in isolation.
• Intel VT (IVT)
• AMD virtualization (AMD-V)
• Examples:
  • VMware Fusion
  • Parallels Desktop for Mac
  • Parallels Workstation
Paravirtualization

• the virtual machine does not necessarily simulate hardware, but instead (or in addition) offers a special API that can only be used by modifying the "guest" OS.

• Examples:
  • XEN
Operating system-level virtualization

• virtualizing a physical server at the operating system level, enabling multiple isolated and secure virtualized servers to run on a single physical server.

• Examples:
  • Linux-Vserver
  • Solaris Containers
  • FreeBSD Jails
  • Chroot
  • CGroups
The “Software-defined Network”

1. Open interface to hardware

2. At least one good operating system that is extensible and possibly open source

3. Well-defined open API

- Network Operating System
- Simple Packet Forwarding Hardware
- Simple Packet Forwarding Hardware
Trend Virtualized OS + Virtualized Network

Simple common stable hardware substrate below + programmability + strong isolation model + competition above = faster innovation
Trend — Then Add Virtualized Storage

Virtualization Layer
- x86 (Computer)

Computer Industry

“Slicing”
- NOX (Network OS)
- OpenFlow
- Controller

Network Industry

Virtualization
- HDFS
- Controller

Storage Industry
- Distributed FS
- Storage Area Network
Agenda

• Introduction to Virtualization
• Implementations of Virtualization
• OS based Virtualization: Xen
• Containers: Dockers
Types of Virtualization

- Native, full
- Hardware assisted
- Para-virtualization
- OS level
  - Containers
  - Jails
  - Chroot
  - Zones
  - Open-VZ → Virtuozzo
The virtual machine simulates enough hardware to allow an unmodified "guest" OS (one designed for the same CPU) to be run in isolation.

Examples:
- VirtualBox
- Virtual PC
- Vmware
- QEMU
- Win4Lin
- XEN/Virtual Iron
Hardware Enabled Virtualization

- The virtual machine has its own hardware and allows a guest OS to be run in isolation.
- Intel VT (IVT)
- AMD virtualization (AMD-V)
- Examples:
  - VMware Fusion
  - Parallels Desktop for Mac
  - Parallels Workstation
Partial Virtualization

• The virtual machine simulates multiple instances of much (but not all) of an underlying hardware environment, particularly address spaces.
Paravirtualization

• The virtual machine does not necessarily simulate hardware, but instead (or in addition) offers a special API that can only be used by modifying the "guest" OS.

• Terminologies
  • Hypervisor, hypercall
  • Enomalism

• Examples:
  • XEN, KVM, Win4Lin 9x
Operating System-Level Virtualization

• Virtualizing a physical server at the operating system level, enabling multiple isolated and secure virtualized servers to run on a single physical server.

• Examples:
  • Parallels Workstation
  • Linux-VServer, Virtuozzo
  • OpenVZ, Solaris Containers
  • FreeBSD Jails
  • Chroot?
Operating System-Level Virtualization

<table>
<thead>
<tr>
<th>Hypervisor (VM)</th>
<th>Containers (CT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>One real HW, many virtual HWs, many Oss</td>
<td>One real HW (no virtual HW), one kernel, many userspace instances</td>
</tr>
<tr>
<td>High versatility – can run different Oss</td>
<td>Higher density, natural page sharing</td>
</tr>
<tr>
<td>Lower density, performance, scalability</td>
<td>Dynamic resource allocation</td>
</tr>
<tr>
<td>&lt;&lt;Lowers&gt;&gt; are mitigated by new hardware features (such as VT-D)</td>
<td>Native performance: [almost] no overhead</td>
</tr>
</tbody>
</table>
Thinner Containers, Better Performance

- Containers
  - Share host OS and drivers
  - Have small virtualization layer
  - Naturally share pages
- Hypervisors
  - Have separate OS plus virtual hardware
  - Hardware emulation requires VMM state
  - Have trouble sharing guest OS pages
- Containers are more elastic than hypervisors
- Container slicing of the OS is ideally suited to cloud slicing families on one server
- Hypervisors’ only advantage in IaaS is support for different OS
What Is the Trade-Off?
## Feature Comparison

<table>
<thead>
<tr>
<th>Hypervisor</th>
<th>Container</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple Kernels</td>
<td>❌</td>
</tr>
<tr>
<td>Load Arbitrary Modules</td>
<td>❌</td>
</tr>
<tr>
<td>Local Administration</td>
<td>❌</td>
</tr>
<tr>
<td>Live Migration</td>
<td>❌</td>
</tr>
<tr>
<td>Live System Update</td>
<td>❌</td>
</tr>
</tbody>
</table>

Hypervisor: Privileged Host VM | OS & Drivers | Hypervisor | Physical Hardware

Container: Privileged Root Context | VM1 | VM2 | VMn |

OS & Drivers | Physical Hardware

- **OpenVZ**
- **Zap**
Agenda

• Introduction to Virtualization
• Implementations of Virtualization
• OS based Virtualization: Xen
• Containers: Dockers
Xen 3.0 Guest VM

Processes
- init
- /bin/bash
- httpd ...

Filesystem
- /usr
- /proc
- /home

Guest VM Environment

Ports
- :80
- :3125
- :6881

Guest OS

Driver Domain

Hypervisor

Physical Hardware
Xen 3.0 Guest VM

- Driver Domain
  - Map Virtual Devices
  - CFQ for disk
  - HTB for network

Processes
- init
- /bin/bash
- httpd ...

Filesystem
- /usr
- /proc
- /home

Guest VM Environment

Guest OS

Driver Domain

Hypervisor

Physical Hardware

Ports
- :80
- :3125
- :6881
Xen 3.0 Guest VM

I/O Path
• Process to Guest OS
• Guest OS to IDD

Processes
init
/bin/bash
httpd ...

Filesystem

Ports
:80
:3125
:6881

Guest VM Environment

Guest OS
Xen 3.0 Guest VM

Security Isolation
- Hypervisor
- Access Physical Level
  - PCI Address
  - Virtual Memory
Xen 3.0 Guest VM

Security Isolation
- Hypervisor
- Access Physical Level
  - PCI Address
  - Virtual Memory

Resource Control
- Hypervisor
  - Allocate Resources
  - Schedule VMs
Agenda

• Introduction to Virtualization
• Implementations of Virtualization
• OS based Virtualization: Xen
• Containers: Dockers
Overview

• “Docker containers wrap up a piece of software in a complete filesystem that contains everything needed to run: code, runtime, system tools, system libraries – anything you can install on a server. This guarantees that the software will always run the same, regardless of its environment it is running in.” (Docker Inc., n.d.)
Overview

• Docker automates the deployment of applications inside software containers
• Additional layer of abstraction and automation of operating system–level virtualization on Linux.
History of Docker

• Solomon Hykes started Docker in France as an internal project within dotCloud, a platform-as-a-service company
• Docker was released as open source in 13 March 2013
• libcontainer library written in the Go programming language.
• Latest version 1.11.2 (31 May 2016)
• Docker team, Red Hat, IBM, Google, Cisco Systems and Amadeus IT Group.
Docker Structure
Containers built with Docker
Basics of Docker

Component A’s Docker File
Source Code Repository
Build

Docker Engine

Push

Docker Container
Image Registry

Docker Container A

Build

Docker Container B

Push

Docker Container C

Pull
Run

Developer Linux Host

Destination Linux Host

Source Code Repository

Build

Docker Engine

Push

Docker Container A

Pull
Run
Changes and Updates

Original Container

Docker Engine

App A

Bins + Libs

Update copies of container

Diffs packaged in Update Container

Docker Container Image Registry

Get Update

App A’’

Bins + Libs

Docker Engine