EE 542, Lectures 6 and 7
Sept. 11 and 13, 2017

Lecture 5 : Virtual Machines nd Hypervisors
Lecture 6 : Docker Containers and Unikernel

Reading Assignments : All sections in Chapter 3 except Section 3.6

Kai Hwang, USC

Difference between Traditional Computer and Virtual machines

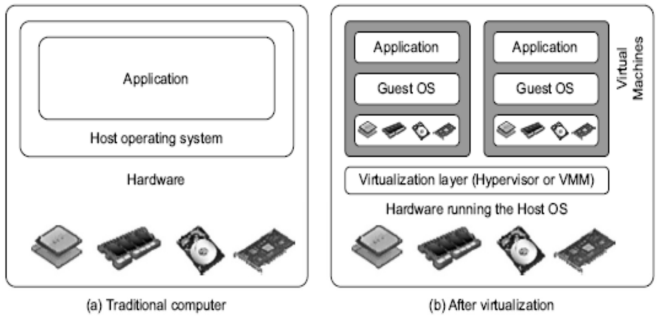
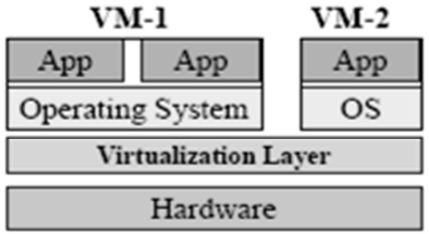


FIGURE 3.1
The architecture of a computer system before and after virtualization, where VMM stands for virtual machine monitor.

(Courtesy of VMWare, 2008)

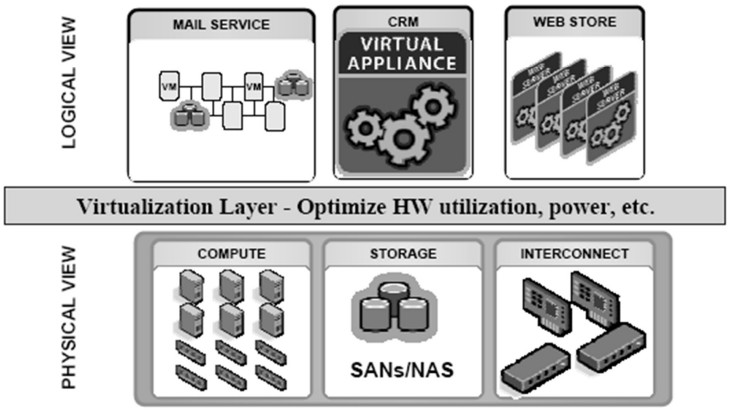
What is Virtualization ?

- A level of indirection between hardware and software.



- Virtual Machine abstraction
  - Run all software written for physical machine.

User's view of virtualization



(Courtesy of VMWare, 2008)

Various Interface Abstraction Levels in Modern Computer Architecture

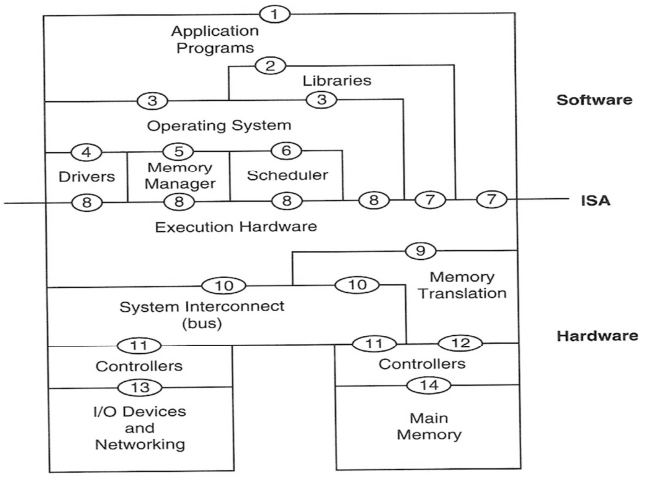
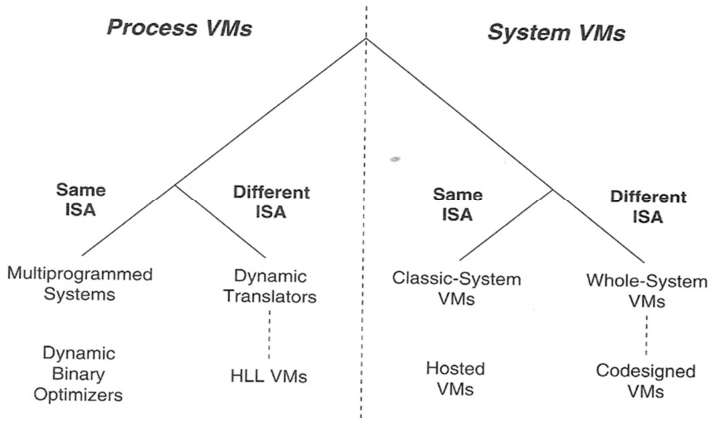


Figure 1.4 Computer System Architectures. *Implementation layers communicate vertically via the shown interfaces. This view of architecture is styled after one given by Glenford Myers (1982).*

(Courtesy of Smith and Nail, 2005)

A Taxonomy of Virtual Machines



(Courtesy of Smith and Nail, 2005)

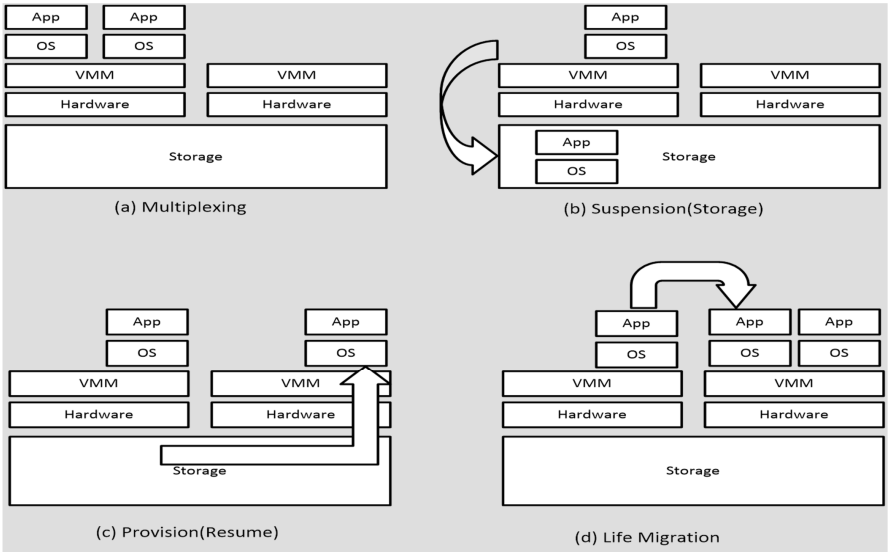


Figure 3.2 Virtual machine multiplexing, suspension, provision, and migration in a distributed computing environment

Virtualization at Various Abstraction Levels

Table 1.7 Relative Merits of Virtualization at Five Abstraction Levels

Level of Virtualization	Functional Description	Example Pakages	Relative Merits, App Flexibility/Isolation, and Implementation Complexity
Instruction Set Architecture	*Emulation of a guest ISA by the host ISA*	*Dynamo, Bird, Bochs, Crusoe*	*Very Low performance, high app flexibility, and median complexity and Isolation*
Hardware-level Virtualization	*Virtualization on top of bare metal hardware*	*XEN, VMWare, Virtual PC*	*High performance and complexity, median app flexibility, and good app isolation*
Operating System Level	*Isolated containers as OS instances*	*Docker Engine, Jail, FVM,*	*Highest performance, Low App flexibility and Isolation, and average complexity*
Run-Time Library Level	*Creating VM via run-time library through API hooks*	*Wine, cCUDA, WABI, LxRun*	*Average performance, low app flexibility and isolation, and low compexity*
User Application Level	*Deploy HLL VMs at user app level*	*JVM, .NET CLR, Panot*	*Low performance and app flexibility, very high complexity and app isolation*

Relative Performance of Virtualization at 5 Abstraction Levels

Table 3.1 Relative Merits of Virtualization at Various Levels

Level of Implementation	Higher Performance	Application Flexibility	Implementation Complexity	Application Isolation
ISA	X	XXXXX	XXX	XXX
Hardware-level virtualization	XXXXX	XXX	XXXXX	XXXX
OS-level virtualization	XXXXX	XX	XXX	XX
Runtime library support	XXX	XX	XX	XX
User application level	XX	XX	XXXXX	XXXXX

Virtualization at ISA level:

Emulating a given ISA by the ISA of the host machine. For example, MIPS binary code can run on an x-86-based host machine with the help of an ISA emulator. Example systems: Bochs, Crusoe, Quemu, BIRD, Dynamo

Advantage: It can run a large amount of legacy binary codes written for various processors on any given new hardware host machines; best application flexibility

Shortcoming & limitation: One source instruction may require tens or hundreds of native target instructions to emulate, which is rather slow. The V-ISA requires adding a processor-specific software translation layer to do the emulation aided by the complier.

Virtualization at Hardware Abstraction level:

This is known as hypervisor - Virtualization is performed right on top of the bare metal hardware. It generates virtual hardware environments for VMs.

Example systems: Xen, Hyper V, KVM, Virtual PC, Denali

Advantage: has higher performance and good application isolation

Shortcoming & limitation: very expensive to implement due to its complexity

Full Virtualization vs. Para-Virtualization

Full virtualization does not need to modify guest OS, and critical instructions are emulated by software through the use of binary translation. On the other hand, para virtualization needs to modify guest OS, and non-virtualizable instructions are replaced by hypercalls that communicate directly with the hypervisor or VMM. As of full virtualization, the advantage is no need to modify OS. However, this approach of binary translation slows down the performance a lot.

Para virtualization reduces the overhead, but the cost of maintaining paravirtualized OS is high. The improvement depends on the workload. VMware Workstation applies full virtualization, which uses binary translation to automatically modify x86 software on-the-fly to replace critical instructions. The para virtualization is supported by Xen, Denali and VMware ESX.

Full Virtualization

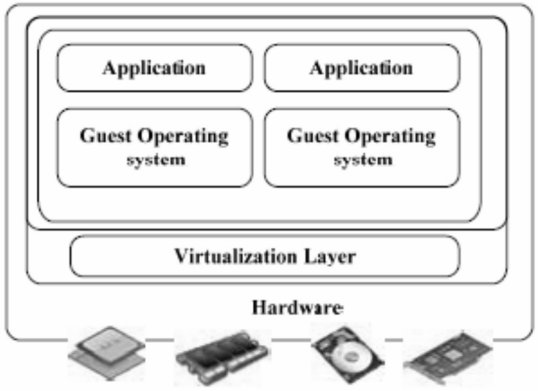


Figure 6.9 The concept of full virtualization using a hypervisor or a VMM directly sitting on top of the bare hardware devices. Note that no host OS is used here as in Figure 6.11.

Hypervisor

A hypervisor is a hardware virtualization technique allowing multiple operating systems, called guests to run on a host machine. This is also called the Virtual Machine Monitor (VMM).

Type 1 hypervisor or the bare metal hypervisor sits on the bare metal computer hardware like the CPU, memory, etc. All the guest operating systems are a layer above the hypervisor. So the hypervisor is the first layer over the hardware. The original CP/CMS hypervisor developed by IBM was of this kind. Examples are Microsoft Hyper-V.

Type 2 or the hosted hypervisor do not run over the bare metal hardware but they run over a host operating system. The hypervisor is the second layer over the hardware. The guest operating systems run a layer over the hypervisor and so they form the third layer. Examples are FreeBSD. The operating system is usually unaware of the virtualization

Virtual Machine Architectures

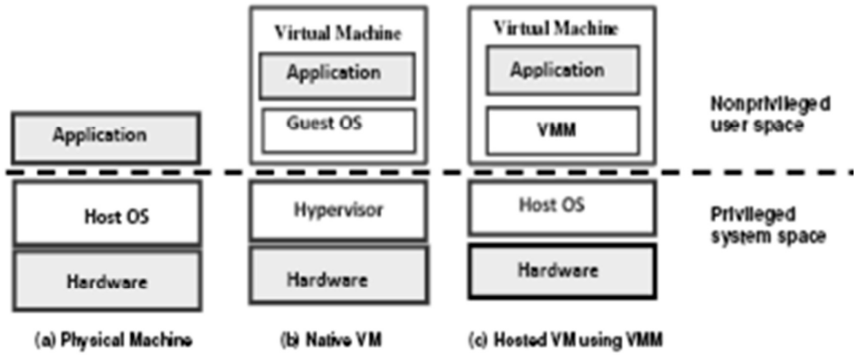


Figure 1.20 Three major components of a conventional physical machine is shown in Part (a). Native VM shown in Part (b) is created by a hypervisor sitting on top of bare metal hardware. A hosted VM shown in Part (c) is created by a VMM middleware in cooperation with the host OS.

Table 3.4
Hypervisors or VM monitors for generating VMs

Hypervisor	Host CPU	Host OS	Guest OS	Architecture, Applications, and User Community
XEN	x86, x86–64, IA-64	NetBSD, Linux	Linux, Windows, BSD, Solaris	Native hypervisor (Example 3.1) developed at Cambridge University
KVM	x86, x86–64, IA-64, S-390, PowerPC	Linux	Linux, Windows, FreeBSD, Solaris	Hosted hypervisor based on paravirtualization at the user space
Hyper V	x86 based	Server 2003	Windows servers	Windows-based native hypervisor, marketed by Microsoft
VMWare Workstation, VirtualBox	x86, x86–64	Any host OS	Windows, Linux, Darwin Solaris, OS/2, FreeBSD	Hosted hypervisor with a paravirtualization architecture

Hypervisor and the XEN Architecture

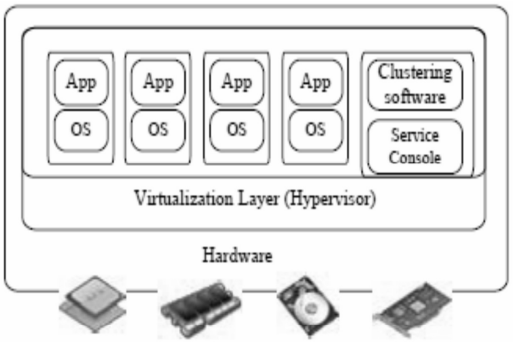


Figure 6.7 A hypervisor is the software layer for virtualization of the bare metal hardware. This layer can be implemented as a micro-kernel of the OS. It converts physical devices into virtual resources for user applications to run on the virtual machines deployed.

The XEN Architecture (1)

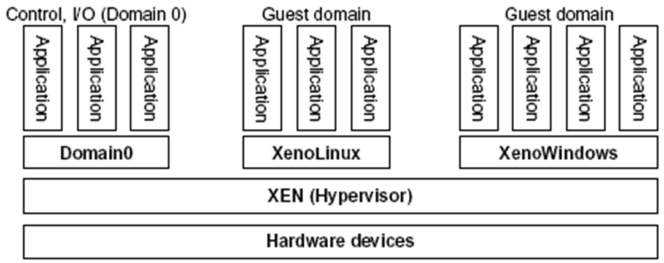


FIGURE 3.5

The Xen architecture's special domain 0 for control and I/O, and several guest domains for user applications.

The XEN Architecture (2)

Xen is an open source hypervisor program developed by Cambridge University. Xen is a micro-kernel hypervisor, which separates the policy from the mechanism. The Xen hypervisor implements all the mechanisms, leaving the policy to be handled by Domain 0, as shown in Figure 3.5. Xen does not include any device drivers natively [7]. It just provides a mechanism by which a guest OS can have direct access to the physical devices. As a result, the size of the Xen hypervisor is kept rather small. Xen provides a virtual environment located between the hardware and the OS. A number of vendors are in the process of developing commercial Xen hypervisors, among them are Citrix XenServer [62] and Oracle VM [42].

The XEN Architecture (2)

Xen is an open source hypervisor program developed by Cambridge University. Xen is a micro-kernel hypervisor, which separates the policy from the mechanism. The Xen hypervisor implements all the mechanisms, leaving the policy to be handled by Domain 0, as shown in Figure 3.5. Xen does not include any device drivers natively [7]. It just provides a mechanism by which a guest OS can have direct access to the physical devices. As a result, the size of the Xen hypervisor is kept rather small. Xen provides a virtual environment located between the hardware and the OS. A number of vendors are in the process of developing commercial Xen hypervisors, among them are Citrix XenServer [62] and Oracle VM [42].

Host-based Virtualization

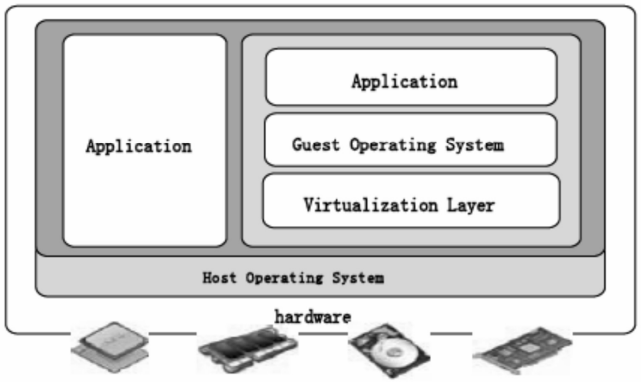


Figure 6.11 A hosted VM installs a guest OS on top of the host OS. This differs from the full virtualization architecture shown in Figure 6.9.

Para-Virtualization

In para-virtualization, the guest operating system has to be modified. Para-virtualization provides specially defined 'hooks' to do some tasks in the host and guest operating systems, which would otherwise have been done in a virtual environment, which is slower. The VMM in a para-virtualized platform is simpler because the critical tasks are now performed in the operating system rater than by the VMM. Since the virtualization overhead decreases the performance increases.

Some of the disadvantages are the compatibility and the portability is reduces because of the modified operating system. Also the cost of maintenance is high because of the deep OS modifications.
Some examples of Para-virtualization are KVM and the VMware ESX, vSphere, etc.

Table 3.5: VMware Hardware Virtualization (Hypervisors) and hosted Software for Virtualization (http://vmware.com/products/vsphere/, 2016):

Category	VMware Software Products or Third Party Software
Native (Hypervisor)	Adeos, CP/CMS, Hyper-V, KVM (Red Hat Enterprise Virtualization), LDoms / Oracle VM Server for SPARC, LynxSecure, SIMMON, VMware ESXi (VMware vSphere, vCloud), VMware Infrastructure, Xen XenClient), z/VM
Hosted (Specialized)	Basilisk II, bhyve, Bochs, Cooperative Linux, DOSBox, DOSEMU, LLinux, Mac-on-Linux, Mac-on-Mac, SheepShaver, SIMH, Windows on Windows, Virtual DOS machine, Win4Lin
Hosted (Independent)	Microsoft Virtual Server, Parallels Workstation, Parallels Desktop forMac, Parallels Server for Mac, PearPC, QEMU, VirtualBox, Virtual Iron, VMware Fusion, VMware Player, VMware Server VMware Workstation, Windows Virtual PC
OtherTools	Ganeti, oVirt, Virtual Machine Manager

VMWare ESX Server for Para-Virtualization

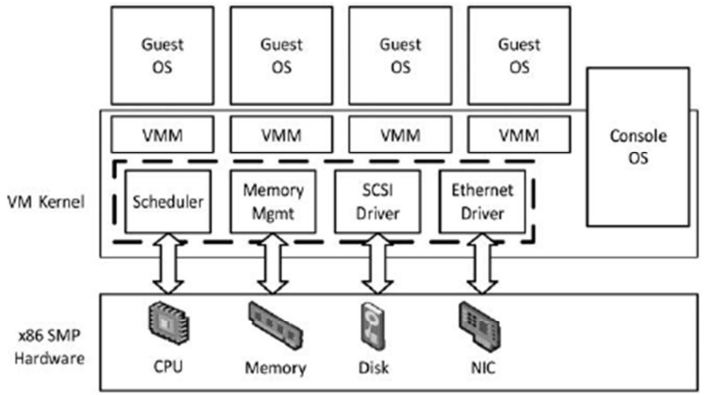


Figure 3.11
The VMware ESX server architecture using paravirtualization. Courtesy of VMware, 2011, http://www.vmware.com/products.

HW and SW Support for CPU, Memory and I/O Virtualization :

- CPU virtualization demands hardware-assisted traps of sensitive instructions by the VMM
- Memory virtualization demands special hardware support (shadow page tables by VMWare or extended page table by Intel) to help translate virtual address into physical address and machine memory in two stages.
- I/O virtualization is the most difficult one to realize due to the complexity if I/O service routines and the emulation needed between the guest OS and host OS.

Virtualization Support at Intel

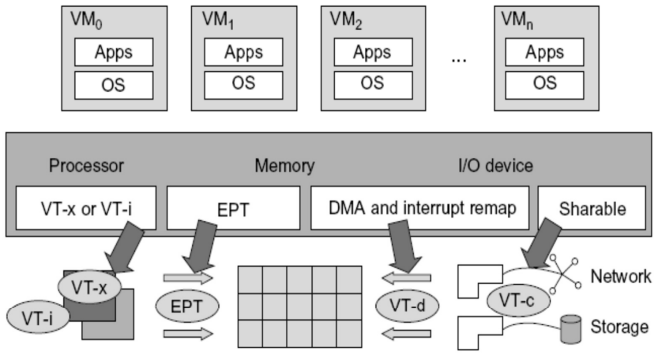


FIGURE 3.10

Intel hardware support for virtualization of processor, memory, and I/O devices.

(*Modified from [68], Courtesy of Lizhong Chen, USC*)

Lecture 6 : Docker Engine and Containers

for Virtualization at the Linux Kernel Level

- Docker engine work with host OS to produce application software containers
- The App software container does not use a guest OS, thus highly scalable
- Virtual clusters are studied with the use either VMs or containers

Virtualization at Operating System level:

It is an abstraction layer between traditional OS and user placations. This virtualization creates isolated containers on a single physical server and the OS-instance to utilize the hardware and software in datacenters. Typical systems: Jail / Virtual Environment / Ensim's VPS / FVM

Advantage: have minimal starup/shutdown cost, low resource requirement, and high scalability; synchronize VM and host state changes.

Shortcoming & limitation: all VMs at the operating system level must have the same kind of guest OS; poor application flexibility and isolation.

Virtualization at OS Level

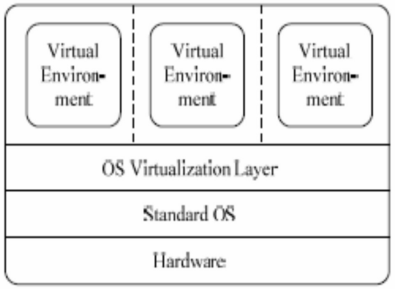


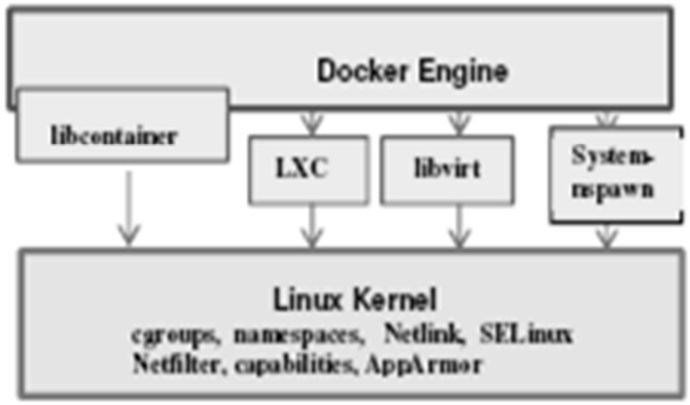
Figure 6.3 The virtualization layer is inserted inside an OS to partition the hardware resources for multiple VMs to run their applications in virtual environments

Advantages of OS Extension for Virtualization

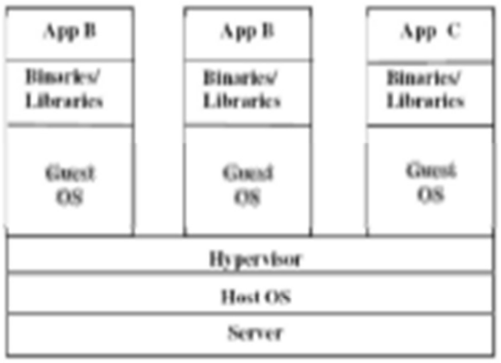
1. VMs at OS level has minimum startup/ shutdown costs
2. OS-level VM can easily synchronize with its environment

Disadvantage of OS Extension for Virtualization

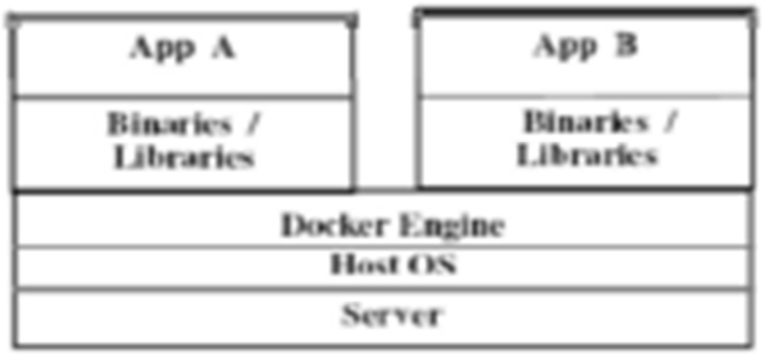
All VMs in the same OS container must have the same or similar guest OS, which restrict application flexibility of different VMs on the same physical machine.



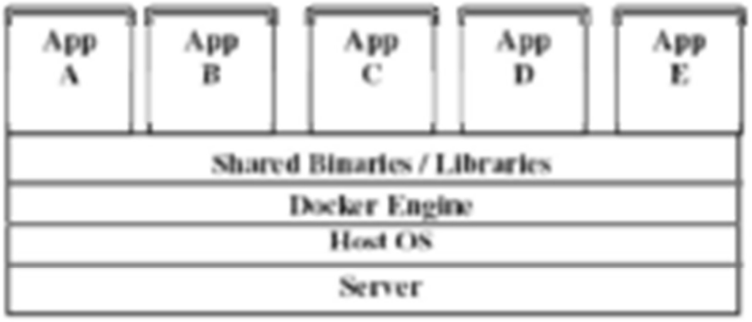
The Docker engine access the Linux kernel features for isolated virtualization of different application containers.



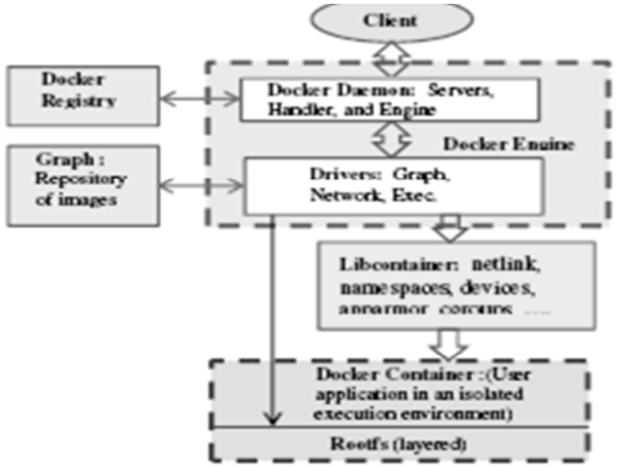
(a). Three hypervisor-created virtual machines (VMs) on the same hardware host, each VM is heavily loaded with its own guest OS and specific binaries and libraries.



(b). Each container is loaded with its own binaries/libraries which are not shared.



(c) The Docker engine creates many light-weight app containers, which are isolated , bur share OS and ,



Docker schematic block diagram illustrating the process of creating and management of Docker containers for isolated execution, separately.

Table 1.9: Comparison Between Hypervisor-created VMs and Docker Containers

VM Type	Strength and Weakness	Suitable Applications
Hypervisor-created Virtual Machines	Higher app flexibility in launching different OS apps, but demand more memory and overhead to create and launch the VMs	More suitable for use in multiple apps without orchestration. VMs appeal to run on wide variety of operating systems. .
Docker Containers	Light-weight application containers with low overhead to create and running with better protection under an isolated execution environment.	More suitable for scalable use of the same app in multiple copies under orchestration. Works better with a particular OS version. It may save the operation costs on clouds.

Assessing the Use of Virtual Machines and Docker Containers in Today's Clouds

- Both VMs and software containers will co-exist for some time in today clouds
- The VMs have high software portability on different types of hardware platforms
- VMs are heavily weighted with the use of heavy duty guest OS. This may weaken its acceptance in the future.
- The Docker containers are light-weight and more cost-effective to implement and to apply in scalable applications. Eventually, most clouds will use containers over Linux hosts.

Software Tools for Container Management

Table 3.11 Host Provisioning and Container Scheduling Tools

Tool Name	Brief Descrption of Tool Functionality
fleet	*Scheduling and cluster management component of CoreOS*
marathon	*Scheduling and service manage,ment component a Mesosphere installation*
Swarm	*Docker's robust scheduler to spin up containers on provisioned hosts*
mesos	*Apache mesos abstracts and manage the resources of all hosts in a cluster*
Kubernetes	*Google's scheduler over the containers running on your cloud infrastructure*
compose	*Docker's tool that allows group management of containers, declaratively*

Virtualization at Library Support level:

It creates execution environments for running alien programs on a platform rather than creating VM to run the entire operating system. It is done by API call interception and remapping. Typical systems: Wine, WAB, LxRun , VisualMainWin

Advantage: It has very low implementation effort

Shortcoming & limitation: poor application flexibility and isolation

Virtualization with Middleware/Library Support

Table 3.4 Middleware and Library Support for Virtualization

Middleware or Runtime Library and References or Web Link	Brief Introduction and Application Platforms
WABI (http://docs.sun.com/app/docs/doc/802-6306)	Middleware that converts Windows system calls running on x86 PCs to Solaris system calls running on SPARC workstations
Lxrun (Linux Run) (http://www.ugcs.caltech.edu/~steven/lxrun/)	A system call emulator that enables Linux applications written for x86 hosts to run on UNIX systems such as the SCO OpenServer
WINE (http://www.winehq.org/)	A library support system for virtualizing x86 processors to run Windows applications under Linux, FreeBSD, and Solaris
Visual MainWin (http://www.mainsoft.com/)	A compiler support system to develop Windows applications using Visual Studio to run on Solaris, Linux, and AIX hosts
vCUDA (Example 3.2) (IEEE *IPDPS* 2009 [57])	Virtualization support for using general-purpose GPUs to run data-intensive applications under a special guest OS

User-Application level:

It virtualizes an application as a virtual machine. This layer sits as an application program on top of an operating system and exports an abstraction of a VM that can run programs written and compiled to a particular abstract machine definition. Typical systems: JVM , NET CLI , Panot

Advantage: has the best application isolation

Shortcoming & limitation: low performance, low application flexibility and high implementation complexity.

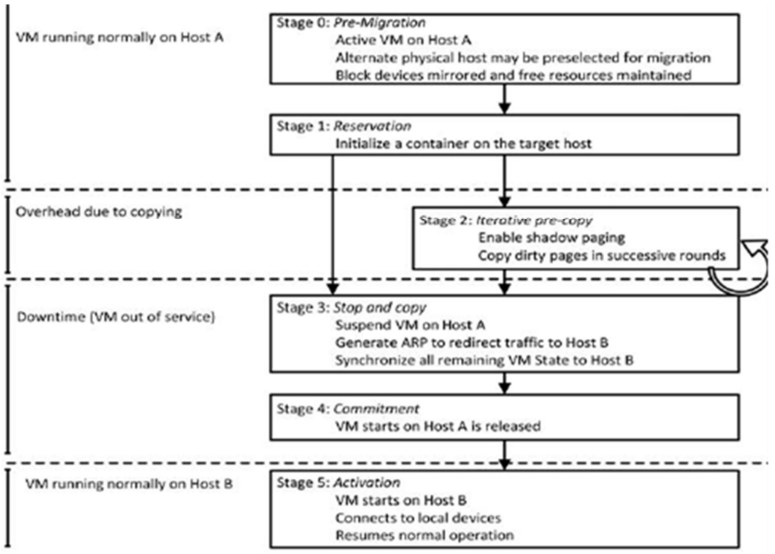


Figure 3.18
Steps for live migration of VMs from host computer A to another host B. Reprinted with permission fro C. Clark et al., "Live Migration of Virtual Machines," *Proc. of the Second Symposium on Networked Systen Design and Implementation* (NSDI'05), 2005.

Live Migration of Virtual Machines

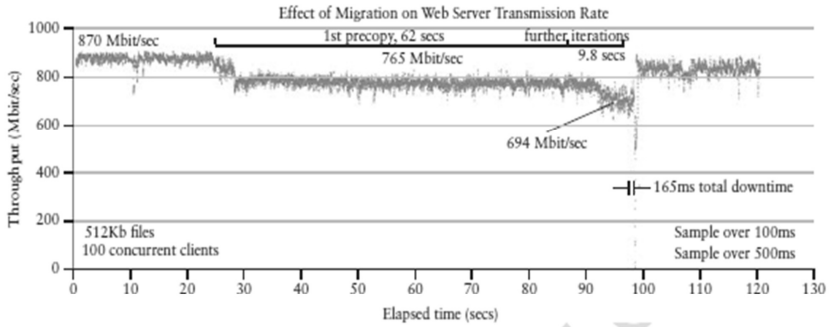


Figure 3.19
Effect on data transmission rate of a VM migrated from one failing web server to another. The downtime of 165 ms was observed. Reprinted with permission from C. Clark et al., "Live Migration of Virtual Machines," *Proc. of the Second Symposium on Networked Systems Design and Implementation* (NSDI'05), 2005.

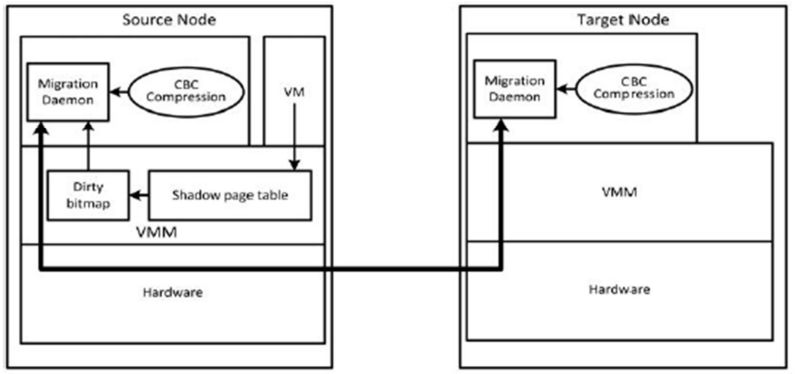


Figure 3.20
Live migration of VM from the Domain0 to that of an Xen-enabled target host.

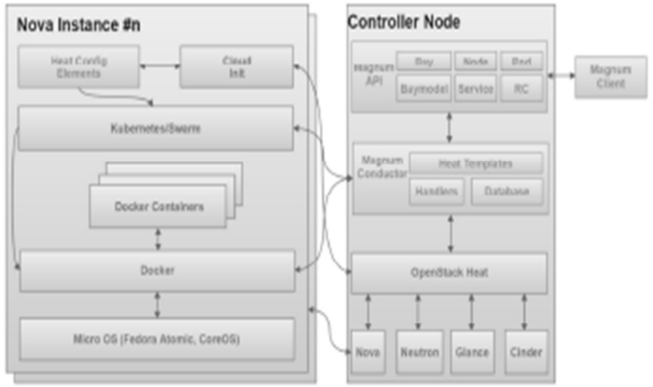
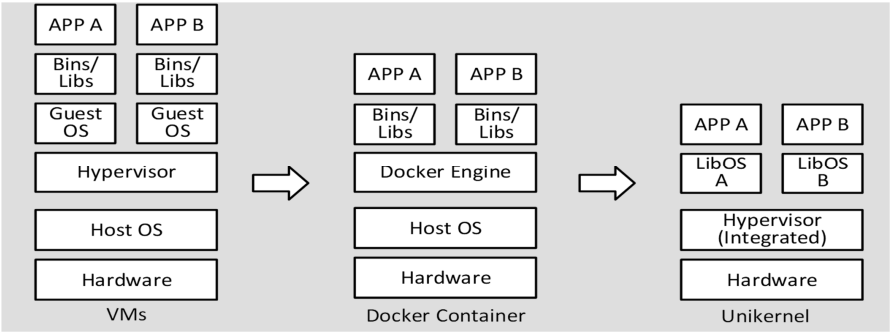


Figure 3.22 Docker container orchestration Illustrated by using OpenStack Magnum to deploy container clusters in multiple Nova instances *(Courtesy of https://wiki.openstack.org/wiki/Magnum, retrieved Aug. 2015).*

Figure 3.17: Architectural evolution from VMs to Docker containers and Unikerkels



Switching from x86 to ARM, Power, and Sparc as Building Nodes in Clouds

- The x86 processors are used in almost all server clusters deployed in cloud platforms. There is a rise of using ARM, Power and SPARC processors in virtualized cloud computing.
- In 2013, IBM announced one billion investment in upgrading its Power series for Linux applications. IBM has provided support plans for commercial clouds to switch to Linux-based Power hosts.
- In 2015, Oracle starts to shift towards cloud computing. ARM processors used in many smartphone have demonstrated their strength in low power consumption and low cost to deliver high performance.
- Dell, HP, Microsoft, and Amazon are all developing ARM servers cloud constructions. Future clouds may enter an era of keen competition among x86, ARM, Power and SPARC processors.

Reading Assignments :

1. K. Hwang, *Cloud Computing for Machine Learning and Cognitive Applicarios*, Chapter 3, MIT Press, 2017
2. M. Rosenblum and T. Garfinkel, "Virtual Machine Monitors: Current Technology and Future Trends", *IEEE Computer Magazine*, May 2005, pp.39-47.
3. VM Ware, Inc., "Virtualization Overview ", White paper, http://www.vmware.com , 2006.
4. *Virtual Machines* by James Smith and Ravi Nair, Morgan Kaufmann, an Elesevier imprint, 2005