Unit 1: Introduction to Cloud Computing

Cloud Overview

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.
This cloud model is composed of five essential characteristics, three service models, and four deployment models. (According to NIST SP800-145)

Essential Characteristics:

- **On-demand self-service.** A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.
- **Broad network access**. Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).
- **Resource pooling.** The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, and network bandwidth.
- **Rapid elasticity.** Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.
- **Measured service.** Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.

Service Models:

- **Software as a Service (SaaS)**. The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure . The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user specific application configuration settings.
- **Platform as a Service (PaaS).** The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider.3 The consumer does not

manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.
- **Infrastructure as a Service (IaaS).** The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls).

Deployment Models:
- **Private cloud.** The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises.
- **Community cloud.** The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be owned, managed, and operated by one or more of the organizations in the community, a third party, or some combination of them, and it may exist on or off premises.
- **Public cloud.** The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider.
- **Hybrid cloud.** The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds).

Cloud vs Virtualization

Cloud	**Virtualization**
VM can be located on any host, depending on the availability of region	VM located on a specific host
Storage Persistence is provided through elastic block storage	Storage can be intrinsically persistent
Automated Management out of the box	Automated Management can be added with add-on
Distributed Control	Centralized Control
Suitable for end users	Suitable for IT Companies
On-Demand Computing	Manual Deployment and Retraction

What is Cloud API?

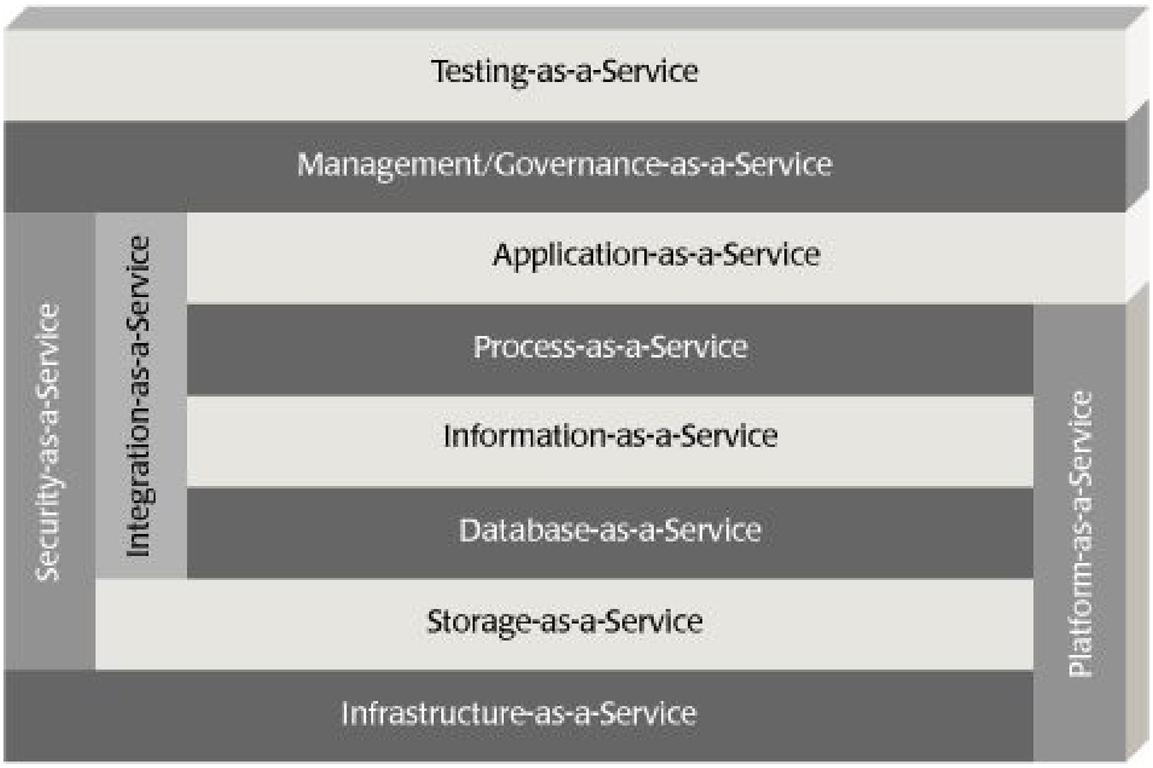
A Cloud Application Programming Interface (Cloud API) is a type of API that enables the development of applications and services used for the provisioning of cloud hardware, software, and platforms. A cloud API serves as a gateway or interface that provides direct and indirect cloud infrastructure and software services to users.

A cloud API is the core component behind any public cloud solution and is generally based primarily on the REST and SOAP frameworks, as well as cross-platform and vendor specific APIs.

Cloud APIs vary according to the provided service or solution, as follows:

- Infrastructure as a Service (IaaS): Infrastructure APIs provision raw computing and storage.
- Software as a Service (SaaS): Software or application APIs provision connectivity and interaction with a software suite.
- Platform as a Service (PaaS): Platform APIs provide back-end architecture for building intensive and feature rich applications.

Cloud Computing Components



Unit 2: Virtualization

Virtualization is a technique, which allows to share single physical instance of an application or resource among multiple organizations or tenants (customers). It does so by assigning a logical name to a physical resource and providing a pointer to that physical resource on demand.
Creating a virtual machine over existing operating system and hardware is referred as Hardware Virtualization. Virtual Machines provide an environment that is logically separated from the underlying hardware.
The machine on which the virtual machine is created is known as host machine and virtual machine is referred as a guest machine. This virtual machine is managed by a software or firmware, which is known as hypervisor.

Need of Virtualization

Virtualization essentially means to create multiple, logical instances of software or hardware on a single physical hardware resource. This technique simulates the available hardware and gives every application running on top of it, the feel that it is the unique holder of the resource. The details of the virtual, simulated environment are kept transparent from the application. Organizations may use this technique to actually do away with many of their physical servers and map their function onto one robust, evergreen physical server.
The advantage here is the reduced cost of maintenance and reduced energy wastage which is not very surprising. As you have fewer physical servers, you need only maintain them and therefore maintenance becomes much easier and cheaper too. As for energy conservation, it is fairly implicit. The amount of energy wasted is a function of the number of physical servers which is clearly much lower in a virtualized environment. Also as far as desktop virtualization is concerned, as the video points out, updates may now be made available much sooner as a single firmware update does not update one client machine, but several instances of the same.

Resource optimization

By virtualizing the hardware and allocating parts of it based on the real needs of users and applications, the available computing power, storage space and network bandwidth can be used much more effectively. Computers no longer need to be idle or performing below their capabilities because there are fewer connected users, or because the hosted application happens to be less demanding than the server can handle.

Virtual machines offer software developers isolated, constrained, test environments. Rather than purchasing dedicated physical hardware, virtual machines can be created on the existing hardware. Because each virtual machine is independent and isolated from all the other servers, programmers can run software without having to worry about affecting other applications, or external components affecting the execution of their code.

Consolidation

It is common practice to dedicate individual computers to a single application. If several applications only use a small amount of processing power, the administrator can consolidate several computers into one server running multiple virtual environments. For organizations that own hundreds or thousands of servers, consolidation can dramatically reduce the need for floor space, HVAC, A/C power, and co-location resources. This means the cost of ownership is reduced significantly, since less physical servers and floor and rack space are required, which in turn leads to less heat and power consumption, and ultimately a smaller carbon footprint.

Maximizing Uptime

Agility is all about being able to respond to changing requirements as quickly and flexibly as possible. Virtualization brings new opportunities to data center administration, allowing users to enjoy:

- Guaranteed uptime of servers and applications; speedy disaster recovery if large scale failures do occur.
- Instant deployment of new virtual machines or even aggregated pools of virtual machines via template images.
- Elasticity, that is, resource provisioning when and where required instead of keeping the entire data center in an *always-on* state.
- Reconfiguration of running computing environments without impacting the users.

Automatically Protect Applications from Server Failure

Server virtualization provides a way to implement redundancy without purchasing additional hardware. Redundancy, in the sense of running the same application on multiple servers, is a safety measure: if for any reason a server fails, another server running the same application takes over, thereby minimizing the interruption in service. This kind of redundancy works in two ways when applied to virtual machines:

- If one virtual system fails, another virtual system takes over.
- By running the redundant virtual machines on separate physical hardware you can also provide better protection against physical hardware failure.

Easily Migrate Workloads as Needs Change

Migration refers to moving a server environment from one place to another. With most virtualization solutions it is possible to move a virtual machine from one physical machine in the environment to another. With physical servers this was originally possible only if both physical machines ran on the same hardware, operating system and processor. In the virtual world, a server can be migrated between physical hosts with entirely different hardware configurations. Migration is typically used to improve reliability and availability: in case of hardware failure the guest system can be moved to a healthy server with limited downtime, if any. It is also useful if a virtual machine needs to scale beyond the physical capabilities of the current host and must be relocated to physical hardware with better performance.

Levels of Virtualization

Instruction Set Architecture Level
At the ISA level, virtualization is performed by emulating a given ISA by the ISA of the host machine. For example, MIPS binary code can run on an x86-based host machine with the help of ISA emulation. With this approach, it is possible to run a large amount of legacy binary code written for various processors on any given new hardware host machine. Instruction set emulation leads to virtual ISAs created on any hardware machine.
The basic emulation method is through code interpretation. An interpreter program interprets the source instructions to target instructions one by one. One source instruction may require tens or hundreds of native target instructions to perform its function. This process is relatively slow. For better performance, dynamic binary translation is desired. This approach translates basic blocks of dynamic source instructions to target instructions. The basic blocks can also be extended to program traces or super blocks to increase translation efficiency. Instruction set emulation requires binary translation and optimization. A virtual instruction set architecture (V-ISA) thus requires adding a processor-specific software translation layer to the compiler.

Hardware Abstraction Level
Hardware-level virtualization is performed right on top of the bare hardware. On the one hand, this approach generates a virtual hardware environment for a VM. On the other hand, the process manages the underlying hardware through virtualization. The idea is to virtualize a computer's resources, such as its processors, memory, and I/O devices. The intention is to upgrade the hardware utilization rate by multiple users concurrently. The idea was implemented in the IBM VM/370 in the 1960s. More recently, the Xen hypervisor has been applied to virtualize x86-based machines to run Linux or other guest OS applications.

Operating System Level
This refers to an abstraction layer between traditional OS and user applications. OS-level virtualization creates isolated containers on a single physical server and the OS instances to utilize the hardware and software in data centers. The containers behave like real servers. OS-level virtualization is commonly used in creating virtual hosting environments to allocate hardware resources among a large number of mutually distrusting users. It is also used, to a lesser extent, in consolidating server hardware by moving services on separate hosts into containers or VMs on one server.
Library Support Level
Most applications use APIs exported by user-level libraries rather than using lengthy system calls by the OS. Since most systems provide well-documented APIs, such an interface becomes another candidate for virtualization. Virtualization with library interfaces is possible by controlling the communication link between applications and the rest of a system through API hooks. The software tool WINE has implemented this approach to support Windows applications on top of UNIX hosts. Another example is the vCUDA which allows applications executing within VMs to leverage GPU hardware acceleration.

User-Application Level
Virtualization at the application level virtualizes an application as a VM. On a traditional OS, an application often runs as a process. Therefore, application-level virtualization is also known as process-level virtualization. The most popular approach is to deploy high level language (HLL) VMs.

In this scenario, the virtualization layer sits as an application program on top of the operating system, and the layer exports an abstraction of a VM that can run programs written and compiled to a particular abstract machine definition. Any program written in the HLL and compiled for this VM will be able to run on it. The Microsoft .NET CLR and Java Virtual Machine (JVM) are two good examples of this class of VM.

Server Virtualization

Server virtualization is the masking of server resources, including the number and identity of individual physical servers, processors, and operating systems, from server users. The server administrator uses a software application to divide one physical server into multiple isolated virtual environments. The virtual environments are sometimes called virtual private servers, but they are also known as guests, instances, containers or emulations.
There are three popular approaches to server virtualization: the virtual machine model, the paravirtual machine model, and virtualization at the operating system (OS) layer.

Application Virtualization

It refers to running an application on a thin client; a terminal or a network workstation with few resident programs and accessing most programs residing on a connected server. The thin client runs in an environment separate from, sometimes referred to as being encapsulated from, the operating system where the application is located.

Application virtualization fools the computer into working as if the application is running on the local machine, while in fact it is running on a virtual machine (such as a server) in another location, using its operating system (OS), and being accessed by the local machine. Incompatibility problems with the local machine’s OS, or even bugs or poor quality code in the application, may be overcome by running virtual applications.

Presentation Virtualization

Presentation Virtualization (PV) is an architecture whereby applications are deployed, managed, supported and executed on a server, not on a client device. Therefore, all that needs to transmit from the server to client is the screen information, and the keyboard and mouse information from the client to the server: it is also possible to connect client devices such as USB drives, local drives or printers and use them in the session as well.
In a PV environment, tasks of hardware and software upgrades, application deployment, technical support, and data storage and backup are simplified: applications reside on servers hosted in the datacentre rather than many of client devices deployed in numerous locations. Data is ideally stored on centralized file servers rather than on user PCs, or remote branch servers. This means end devices can be simpler, less expensive and most importantly easier to manage than traditional PC desktops.

Hypervisor and Xen Architecture

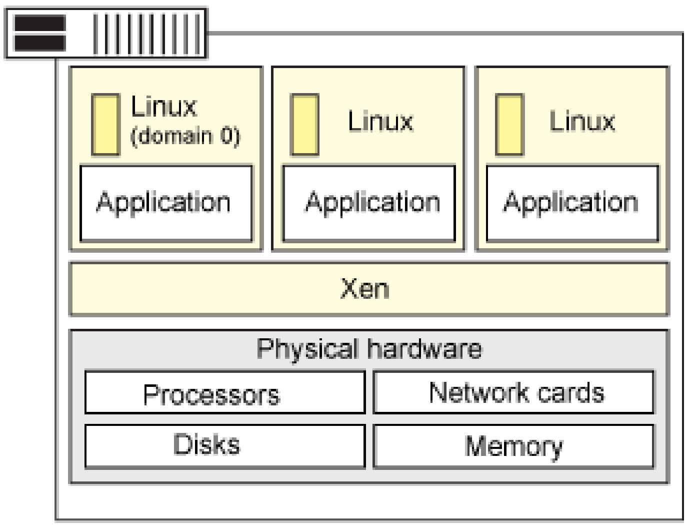
Hypervisor

The hypervisor is a firmware or low-level program that acts as a Virtual Machine Manager. There are two types of hypervisor:

- **Type 1 hypervisor** executes on bare system.
- **Type 2 hypervisor** is a software interface that emulates the devices with which a system normally interacts.

Xen

Xen is a type 1 hypervisor that creates logical pools of system resources so that many virtual machines can share the same physical resources.
Xen is a hypervisor that runs directly on the system hardware. Xen inserts a virtualization layer between the system hardware and the virtual machines, turning the system hardware into a pool of logical computing resources that Xen can dynamically allocate to any guest operating system. The operating systems running in virtual machines interact with the virtual resources as if they were physical resources.



The following figure shows a system with Xen running virtual machines.
Here Xen is running three virtual machines. Each virtual machine is running a guest operating system and applications independent of other virtual machines while sharing the same physical resources.

The following are key concepts of the Xen architecture:

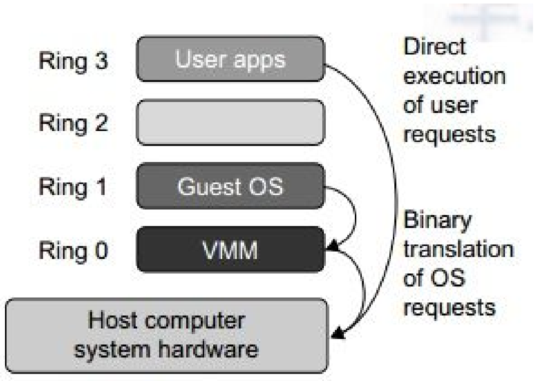
- Full virtualization.
- Xen can run multiple guest OS, each in its on VM.
- Instead of a driver, lots of great stuff happens in the Xen daemon, xend.

Xen can run several guest operating systems each running in its own virtual machine or domain. When Xen is first installed, it automatically creates the first domain, Domain 0 (or dom0).
Domain 0 is the management domain and is responsible for managing the system. It performs tasks like building additional domains (or virtual machines), managing the virtual devices for each virtual machine, suspending virtual machines, resuming virtual machines, and migrating virtual machines. Domain 0 runs a guest operating system and is responsible for the hardware devices.

The Xen daemon, xend, is a Python program that runs in dom0. It is the central point of control for managing virtual resources across all the virtual machines running on the Xen hypervisor. Most of the command parsing, validation, and sequencing happens in user space in xend and not in a driver.

Binary Translation with Full-Virtualization

Depending on implementation technologies, hardware virtualization can be classified into two categories: full virtualization and host-based virtualization. Full virtualization does not need to modify the host OS. It relies on binary translation to trap and to virtualize the execution of certain sensitive, non virtualizable instructions. The guest OSes and their applications consist of noncritical and critical instructions. In a host-based system, both a host OS and a guest OS are used. A virtualization software layer is built between the host OS and guest OS. These two classes of VM architecture are introduced next.
With full virtualization, noncritical instructions run on the hardware directly while critical instructions are discovered and replaced with traps into the VMM to be emulated by software. Both the hypervisor and VMM approaches are considered full virtualization. Only critical instructions trapped into the VMM, because binary translation can incur a large performance overhead. Noncritical instructions do not control hardware or threaten the security of the system, but critical instructions do. Therefore, running noncritical instructions on hardware not only can promote efficiency, but also can ensure system security.

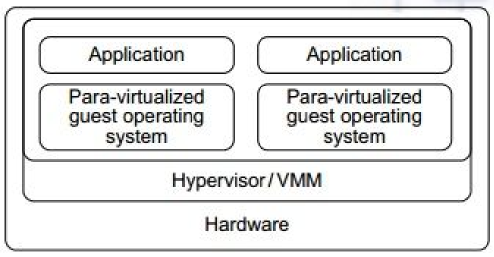

Ring 3
User apps
Ring 2
Ring 1
Guest OS
Ring 0
VMM
Host computer system hardware
Direct execution of user requests
Binary translation of OS requests

Binary Translation of Guest OS Requests Using a VMM
This approach was implemented by VMware and many other software companies. As shown in the figure, VMware puts the VMM at Ring 0 and the guest OS at Ring 1. The VMM scans the instruction stream and identifies the privileged, control- and behavior-sensitive instructions. When these instructions are identified, they are trapped into the VMM, which emulates the behavior of these instructions. The method used in this emulation is called binary translation. Therefore, full virtualization combines binary translation and direct execution. The guest OS is completely decoupled from the underlying hardware. Consequently, the guest OS is unaware that it is being virtualized.

Para-Virtualization with Compiler Support

Para-virtualization needs to modify the guest operating systems. A para-virtualized VM provides special APIs requiring substantial OS modifications in user applications. Performance degradation is a critical issue of a virtualized system. No one wants to use a VM if it is much slower than using a physical machine. The virtualization layer can be inserted at different positions in a machine software stack. However, para-virtualization attempts to reduce the virtualization overhead, and thus improve performance by modifying only the guest OS kernel.

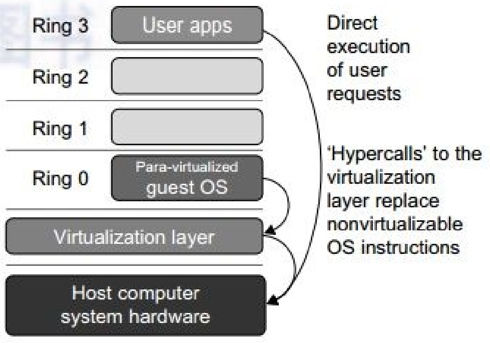
Figure 3.7 illustrates the concept of a paravirtualized VM architecture. The guest operating systems are para-virtualized. They are assisted by an intelligent compiler to replace the non virtualizable OS instructions by hypercalls as illustrated in Figure 3.8. The traditional x86 processor offers four instruction execution rings: Rings 0, 1, 2, and 3. The lower the ring number, the higher the privilege of instruction being executed. The OS is responsible for managing the hardware and the privileged instructions to execute at Ring 0, while user-level applications run at Ring 3.



Application
Application
Para-virtualized guest operating system
Para-virtualized guest operating system
Hypervisor/VMM
Hardware

FIGURE 3.7

Para-virtualized VM architecture, which involves modifying the guest OS kernel to replace nonvirtualizable instructions with hypercalls for the hypervisor or the VMM to carry out the virtualization process (See Figure 3.8 for more details.)



Ring 3
User apps
Ring 2
Ring 1
Ring 0
Para-virtualized guest OS
Virtualization layer
Host computer system hardware
Direct execution of user requests
'Hypercalls' to the virtualization layer replace nonvirtualizable OS instructions

FIGURE 3.8

The use of a para-virtualized guest OS assisted by an intelligent compiler to replace nonvirtualizable OS instructions by hypercalls.

(Courtesy of VMWare [71])

KVM (Kernel Based VM)

This is a Linux para-virtualization system—a part of the Linux version 2.6.20 kernel. Memory management and scheduling activities are carried out by the existing Linux kernel. The KVM does the rest, which makes it simpler than the hypervisor that controls the entire machine. KVM is a hardware-assisted para-virtualization tool, which improves performance and supports unmodified guest OSes such as Windows, Linux, Solaris, and other UNIX variants.

Para-Virtualization with Compiler Support

Unlike the full virtualization architecture which intercepts and emulates privileged and sensitive instructions at runtime, para-virtualization handles these instructions at compile time. The guest OS kernel is modified to replace the privileged and sensitive instructions with hypercalls to the hypervisor or VMM. Xen assumes such a para-virtualization architecture.

The guest OS running in a guest domain may run at Ring 1 instead of at Ring 0. This implies that the guest OS may not be able to execute some privileged and sensitive instructions. The privileged instructions are implemented by hypercalls to the hypervisor. After replacing the instructions with hypercalls, the modified guest OS emulates the behavior of the original guest OS.

CPU Virtualization

A VM is a duplicate of an existing computer system in which a majority of the VM instructions are executed on the host processor in native mode. Thus, unprivileged instructions of VMs run directly on the host machine for higher efficiency. Other critical instructions should be handled carefully for correctness and stability.
The critical instructions are divided into three categories:

- privileged instructions
- control-sensitive instructions, and
- behavior-sensitive instructions.

Privileged instructions execute in a privileged mode and will be trapped if executed outside this mode. Control-sensitive instructions attempt to change the configuration of resources used. Behavior-sensitive instructions have different behaviors depending on the configuration of resources, including the load and store operations over the virtual memory.

A CPU architecture is virtualizable if it supports the ability to run the VM's privileged and unprivileged instructions in the CPU's user mode while the VMM runs in supervisor mode. When the privileged instructions including control- and behavior-sensitive instructions of a VM are executed, they are trapped in the VMM. In this case, the VMM acts as a unified mediator for hardware access from different VMs to guarantee the correctness and stability of the whole system. However, not all CPU architectures are virtualizable. RISC CPU architectures can be naturally virtualized because all control- and behavior-sensitive instructions are privileged instructions. On the contrary, x86 CPU architectures are not primarily designed to support virtualization. This is because about 10 sensitive instructions, such as SGDT and SMSW, are not privileged instructions. When these instructions execute in virtualization, they cannot be trapped in the VMM.

Hardware-Assisted CPU Virtualization
This technique attempts to simplify virtualization because full or paravirtualization is complicated. Intel and AMD add an additional mode called privilege mode level (some people call it Ring-1) to x86 processors. Therefore, operating systems can still run at Ring 0 and the hypervisor can run at Ring -1. All the privileged and sensitive instructions are trapped in the hypervisor automatically. This technique removes the difficulty of implementing binary translation of full virtualization. It also lets the operating system run in VMs without modification.
Generally, hardware-assisted virtualization should have high efficiency. However, since the transition from the hypervisor to the guest OS incurs high overhead switches between processor modes, it sometimes cannot outperform binary translation. Hence, virtualization systems such as VMware now use a hybrid approach, in which a few tasks are offloaded to the hardware but the rest is still done in software. In addition, para-virtualization and hardware-assisted virtualization can be combined to improve the performance further.

Memory Virtualization

Virtual memory virtualization is similar to the virtual memory support provided by modern operating systems. In a traditional execution environment, the operating system maintains mappings of virtual memory to machine memory using page tables, which is a one-stage mapping from virtual memory to machine memory. All modern x86 CPUs include a memory management unit (MMU) and a translation lookaside buffer (TLB) to optimize virtual memory performance. However, in a virtual execution environment, virtual memory virtualization involves sharing the physical system memory in RAM and dynamically allocating it to the physical memory of the VMs.

That means a two-stage mapping process should be maintained by the guest OS and the VMM, respectively: virtual memory to physical memory, and physical memory to machine memory Furthermore, MMU virtualization should be supported, which is transparent to the guest OS. The guest OS continues to control the mapping of virtual addresses to the physical memory addresses of VMs. But the guest OS cannot directly access the actual machine memory. The VMM is responsible for mapping the guest physical memory to the actual machine memory. Figure 3.12 shows the two-level memory mapping procedure.

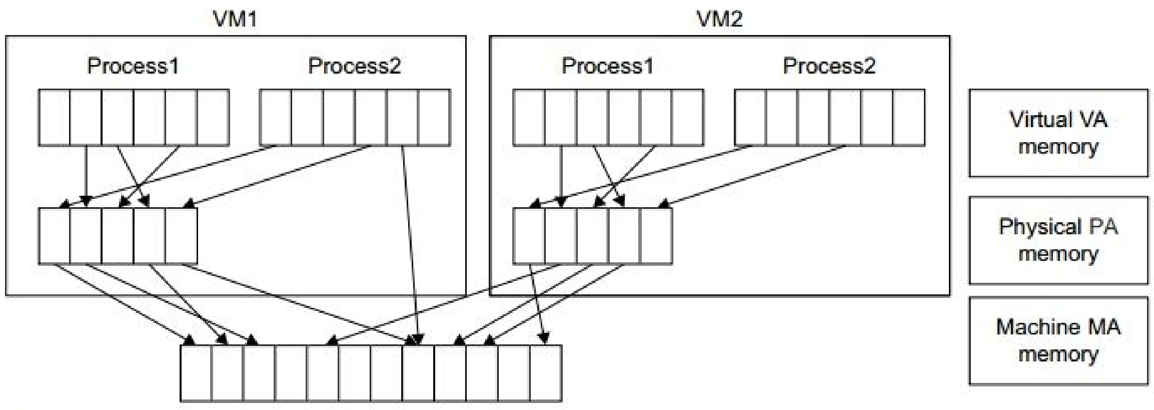


FIGURE 3.12

Two-level memory mapping procedure.

Since each page table of the guest OSes has a separate page table in the VMM corresponding to it, the VMM page table is called the shadow page table. Nested page tables add another layer of indirection to virtual memory. The MMU already handles virtual-to-physical translations as defined by the OS. Then the physical memory addresses are translated to machine addresses using another set of page tables defined by the hypervisor. Since modern operating systems maintain a set of page tables for every process, the shadow page tables will get flooded. Consequently, the performance overhead and cost of memory will be very high.

VMware uses shadow page tables to perform virtual-memory-to-machine-memory address translation. Processors use TLB hardware to map the virtual memory directly to the machine memory to avoid the two levels of translation on every access. When the guest OS changes the virtual memory to a physical memory mapping, the VMM updates the shadow page tables to enable a direct lookup. The AMD Barcelona processor has featured hardware-assisted memory virtualization since 2007. It provides hardware assistance to the two-stage address translation in a virtual execution environment by using a technology called nested paging.

IO Virtualization

I/O virtualization involves managing the routing of I/O requests between virtual devices and the shared physical hardware.There are three ways to implement I/O virtualization: full device emulation, para-virtualization, and direct I/O.

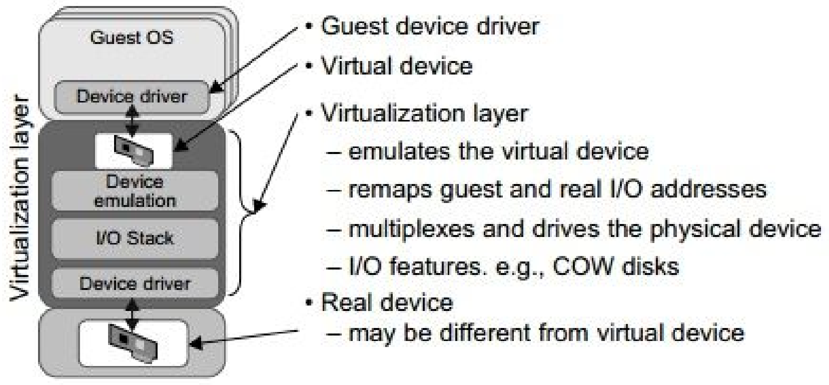

Guest OS
Device driver
Virtualization layer
Device emulation
I/O Stack
Device driver
• Guest device driver
• Virtual device
• Virtualization layer
– emulates the virtual device
– remaps guest and real I/O addresses
– multiplexes and drives the physical device
– I/O features. e.g., COW disks
• Real device
– may be different from virtual device

FIGURE 3.14

Device emulation for I/O virtualization implemented inside the middle layer that maps real I/O devices into the virtual devices for the guest device driver to use.

Full device emulation is the first approach for I/O virtualization. Generally, this approach emulates well-known, real-world devices. All the functions of a device or bus infrastructure, such as device enumeration, identification, interrupts, and DMA, are replicated in software. This software is located in the VMM and acts as a virtual device. The I/O access requests of the guest OS are trapped in the VMM which interacts with the I/O devices.

The para-virtualization method of I/O virtualization is typically used in Xen. It is also known as the split driver model consisting of a frontend driver and a backend driver. The frontend driver is running in Domain U and the backend driver is running in Domain 0. They interact with each other via a block of shared memory. The frontend driver manages the I/O requests of the guest OSes and the backend driver is responsible for managing the real I/O devices and multiplexing the I/O data of different VMs. Although para-I/O-virtualization achieves better device performance than full device emulation, it comes with a higher CPU overhead.

Direct I/O virtualization lets the VM access devices directly. It can achieve close-to-native performance without high CPU costs. However, current direct I/O virtualization implementations focus on networking for mainframes. There are a lot of challenges for commodity hardware devices. Since software-based I/O virtualization requires a very high overhead of device emulation, hardware-assisted I/O virtualization is critical. Intel VT-d supports the remapping of I/O DMA transfers and device-generated interrupts. The architecture of VT-d provides the flexibility to support multiple usage models that may run unmodified, special-purpose, or “virtualization-aware” guest OSes.

Virtualization in Multi-Core Processors

Virtualizing a multi-core processor is relatively more complicated than virtualizing a uni-core processor. Though multicore processors are claimed to have higher performance by integrating multiple processor cores in a single chip, multi-core virtualization has raised some new challenges to computer architect, compiler constructors, system designers, and application programmers. There are mainly two difficulties: Application programs must be parallelized to use all cores fully, and software must explicitly assign tasks to the cores, which is a very complex problem.

Concerning the first challenge, new programming models, languages, and libraries are needed to make parallel programming easier. The second challenge has spawned research involving scheduling algorithms and resource management policies. Yet these efforts cannot balance well among performance, complexity, and other issues. What is worse, as technology scales, a new challenge called dynamic heterogeneity is emerging to mix the fat CPU core and thin GPU cores on the same chip, which further complicates the multi-core or many-core resource management. The dynamic heterogeneity of hardware infrastructure mainly comes from less reliable transistors and increased complexity in using the transistors

Using Virtual Processor Cores

Wells proposed a multi-core virtualization method to allow hardware designers to get an abstraction of the low-level details of the processor cores. This technique alleviates the burden and inefficiency of managing hardware resources by software. It is located under the ISA and remains unmodified by the operating system or VMM (hypervisor). Figure 3.16 illustrates the technique of a software-visible VCPU moving from one core to another and temporarily suspending execution of a VCPU when there are no appropriate cores on which it can run.

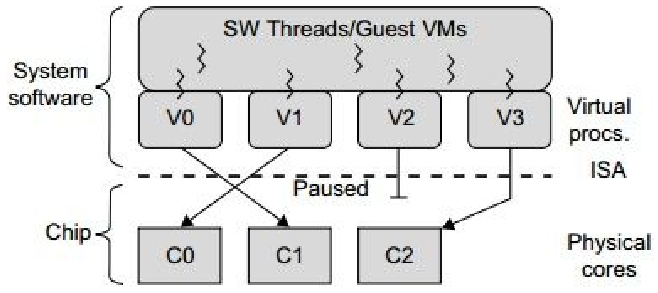


FIGURE 3.16

Multicore virtualization method that exposes four VCPUs to the software, when only three cores are actually present.

Virtual Hierarchy

Instead of supporting time-sharing jobs on one or a few cores, we can use the abundant cores in a space-sharing, where single-threaded or multithreaded jobs are simultaneously assigned to separate groups of cores for long time intervals. This idea was originally suggested by Marty and Hill. To optimize for space-shared workloads, they propose using virtual hierarchies to overlay a coherence and caching hierarchy onto a physical processor. Unlike a fixed physical hierarchy, a virtual hierarchy can adapt to fit how the work is space shared for improved performance and performance isolation.

Today's many-core CMPs use a physical hierarchy of two or more cache levels that statically determine the cache allocation and mapping. A virtual hierarchy is a cache hierarchy that can adapt to fit the workload or mix of workloads. The hierarchy's first level locates data blocks close to the cores needing them for faster access, establishes a shared-cache domain, and establishes a point of coherence for faster communication. When a miss leaves a tile, it first attempts to locate the block (or sharers) within the first level. The first level can also provide isolation between independent workloads. A miss at the L1 cache can invoke the L2 access.

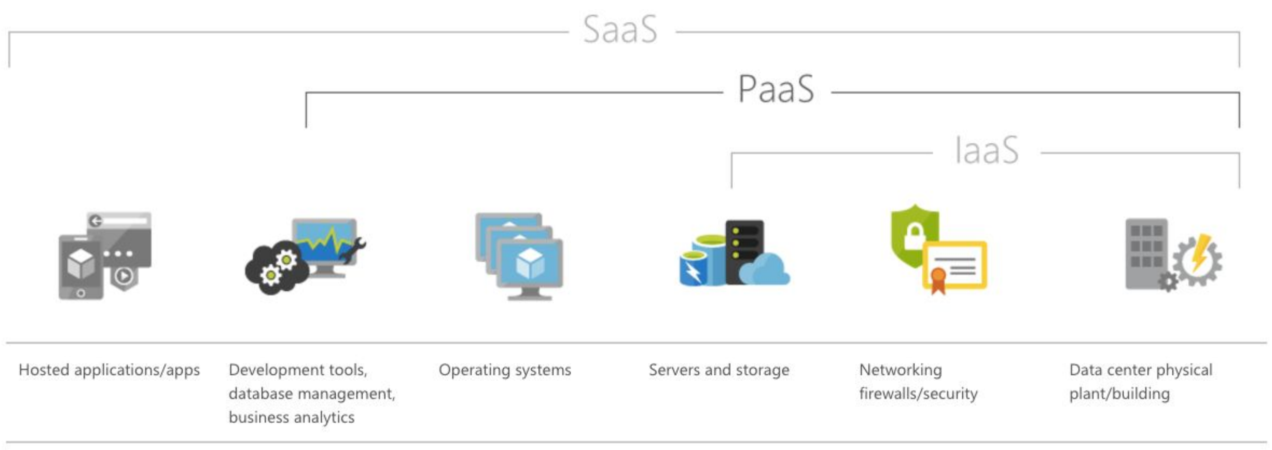
Unit 3: Cloud Computing Services

XaaS

XaaS is a collective term said to stand for a number of things including "X as a service," "anything as a service" or "everything as a service." The acronym refers to an increasing number of services that are delivered over the Internet rather than provided locally or on-site. XaaS is the essence of cloud computing.
The most common examples of XaaS are Software as a Service (SaaS), Infrastructure as a Service (IaaS) and Platform as a Service (PaaS). The combined use of these three is sometimes referred to as the SPI model (SaaS, PaaS, IaaS). Other examples of XaaS include storage as a service (SaaS), communications as a service (CaaS), network as a service (NaaS) and monitoring as a service (MaaS).

Offerings tagged with the '*as a service'* suffix have a number of common attributes, including:

- Low barriers to entry is a common method of offerings, with services typically being available to or targeting consumers and small businesses.
- Little or no capital expenditure as infrastructure is owned by the provider.
- Massive scalability is also common, though this is not an absolute requirement and many of the offerings have yet to achieve large scale.
- Multitenancy enables resources (and costs) to be shared among many users.
- Device independence[2] enables users to access systems regardless of what device they are using (e.g. PC, mobile,...etc.).
- Location independence[2] allows users remote access to systems.


SaaS
PaaS
IaaS
Hosted applications/apps
Development tools, database management, business analytics
Operating systems
Servers and storage
Networking firewalls/security
Data center physical plant/building

Infrastructure as a Service (IaaS)

NIST Definition of IaaS
The capability provided to the consumer is to provision processing, storage, networks and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage and deployed applications; and possibly limited control of select networking components (e.g., host firewalls).

In an IaaS model, a third-party provider hosts hardware, software, servers, storage and other infrastructure components on behalf of its users. IaaS providers also host users' applications and handle tasks including system maintenance, backup and resiliency planning.
IaaS platforms offer highly scalable resources that can be adjusted on-demand. This makes IaaS well-suited for workloads that are temporary, experimental or change unexpectedly. Other characteristics of IaaS environments include the automation of administrative tasks, dynamic scaling, desktop virtualization and policy-based services.
IaaS customers pay on a per-use basis, typically by the hour, week or month. Some providers also charge customers based on the amount of virtual machine space they use. This pay-as-you-go model eliminates the capital expense of deploying in-house hardware and software. However, users should monitor their IaaS environments closely to avoid being charged for unauthorized services.

Examples of IaaS

- Internal Business Networks
  Utilising pooled server and networking resources in which a business can store data and run applications. Expanding businesses can scale their infrastructure in accordance with growth
- Cloud Hosting
  Hosting of websites on virtual servers which are founded upon pooled resources from underlying physical servers
- Virtual Data Centre (Vdc)
  A virtualized network of interconnected virtual servers which can be used to offer enhanced cloud hosting capabilities, enterprise IT infrastructure or to integrate operations

Leading IaaS providers include Amazon Web Services (AWS), Windows Azure, Google Compute Engine, Rackspace Open Cloud, and IBM SmartCloud Enterprise.

Platform as a Service (IaaS)

NIST Definition of PaaS
The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.

In a PaaS model, a cloud provider delivers hardware and software tools -- usually those needed for application development -- to its users as a service. A PaaS provider hosts the hardware and

software on its own infrastructure. As a result, PaaS frees users from having to install in-house hardware and software to develop or run a new application.
In a PaaS environment, the service provider not only is responsible for provisioning and managing the lower level infrastructure resources, but also for providing a fully managed application development and deployment platform

Software developers, web developers and businesses can all benefit from PaaS:

- Software developers can take advantage of a PaaS solution to build an application which they are planning to offer over the internet or software to be sold out of the box
- Web developers can use individual PaaS environments at every stage of the process to develop, test and host their websites
- Businesses can develop their own internal software, particularly to create distinct ring-fenced development and testing environments

Common PaaS vendors include Salesforce.com's Force.com, which provides an enterprise customer relationship management (CRM) platform. PaaS platforms for software development and management include Appear IQ, Mendix, Amazon Web Services (AWS) Elastic Beanstalk, Google App Engine and Heroku.

Software as a Service (SaaS)

NIST Definition of SaaS
The capability provided to the consumer is to use the provider’s applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user specific application configuration settings.

Software as a service (SaaS) is a software distribution model in which a third-party provider hosts applications and makes them available to customers over the Internet.

SaaS provides a complete software solution which you purchase on a pay-as-you-go basis from a cloud service provider. You rent the use of an app for your organisation and your users connect to it over the Internet, usually with a web browser. All of the underlying infrastructure, middleware, app software and app data are located in the service provider’s data center. The service provider manages the hardware and software and with the appropriate service agreement, will ensure the availability and the security of the app and your data as well. SaaS allows your organisation to get quickly up and running with an app at minimal upfront cost.

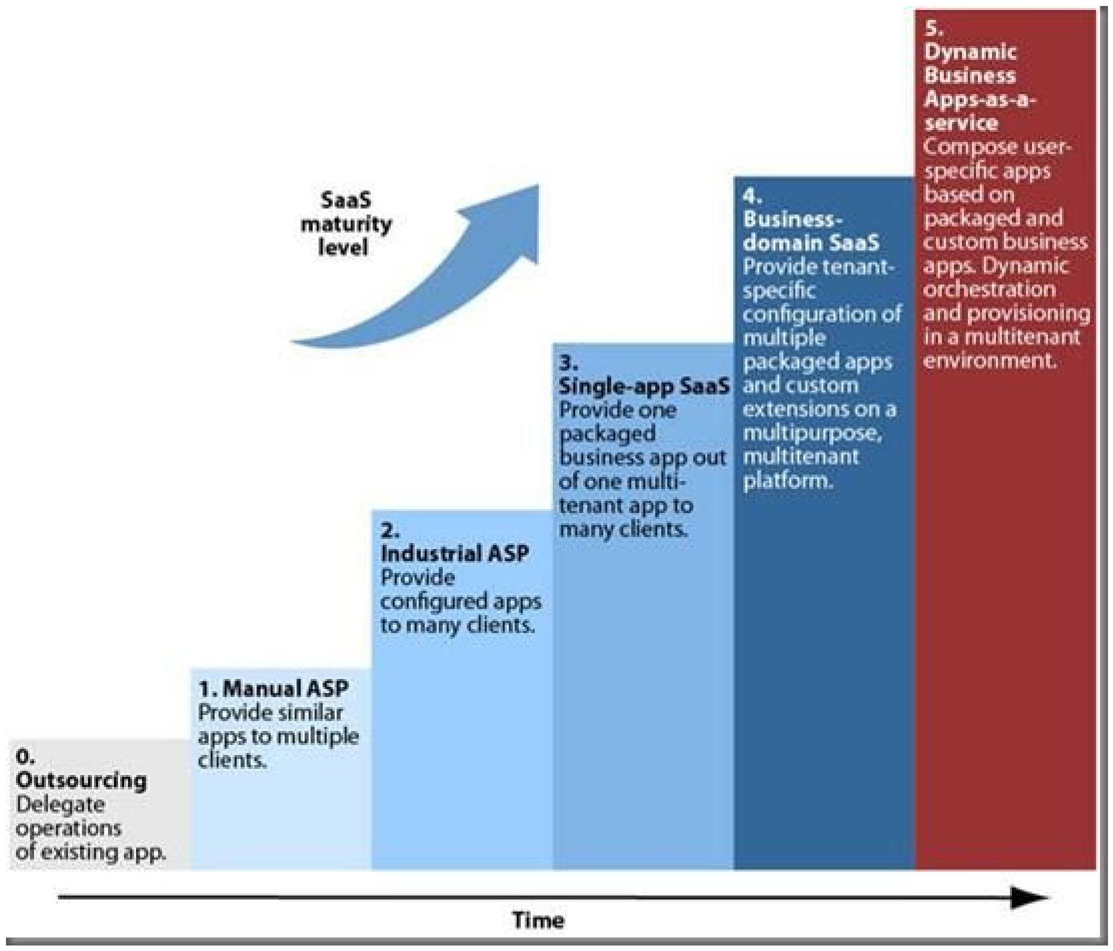
Advantages

- Gain access to sophisticated applications.
- Pay only for what you use.
- Use free client software. (For eg. Browser)
- Mobilise your workforce easily.
- Access app data from anywhere.

DataBase as a Service (DBaaS)

Database-as-a-Service (DBaaS) is a service that is managed by a cloud operator (public or private) that supports applications, without the application team assuming responsibility for traditional database administration functions. With a DBaaS, the application developers should not need to be database experts, nor should they have to hire a database administrator (DBA) to maintain the database. Application developers can simply call a database service and it works without even having to consider the database. Database can seamlessly scale and it would be maintained, upgraded, backed-up and handle server failure, all without impacting the developer in any way. DBaaS consists of a database manager component, which controls all underlying database instances via an API. This API is accessible to the user via a management console, usually a web application, which the user may use to manage and configure the database and even provision or deprovision database instances.
This is a more structured approach compared to storage as a service, and at its core it is really a software offering. In this model, payment may be charged according to the capacity used as well as the features and use of the database administration tools.

SaaS Maturity Model


SaaS maturity level
0. Outsourcing Delegate operations of existing app.
1. Manual ASP Provide similar apps to multiple clients.
2. Industrial ASP Provide configured apps to many clients.
3. Single-app SaaS Provide one packaged business app out of one multi-tenant app to many clients.
4. Business-domain SaaS Provide tenant-specific configuration of multiple packaged apps and custom extensions on a multipurpose, multitenant platform.
5. Dynamic Business Apps-as-a-service Compose user-specific apps based on packaged and custom business apps. Dynamic orchestration and provisioning in a multitenant environment.
Time

Normal Web Hosting vs PaaS hosting

NORMAL WEB HOSTING: Website hosted on single server

1. In traditional web hosting in addition to writing actual code developer has following responsibilities:
    a. Setting up the application server (e.g. Apache).
    b. Set up MySQL database.
    c. Setting up runtime platform like PHP, Ruby, etc.
    d. Set up FTP to deploy your IDE.
    e. Set up security and firewall.
2. Plus, if you want your application to be reliable, scalable and resilient against various failures, you'll have to deal with additional issues like setting up, monitoring, alerting load balancers, replication of application across multiple servers, deployment scripts, etc.
3. Thus, developers waste their time in setting up the production environment and managing servers.
4. E.g. DreamHost, Host Gator, GoDaddy.

PAAS HOSTING: website hosted on multiple servers at same time.

1. Here core competency of developer is to write the code and not to manage the servers.
2. It provides following benefits:
    a. No setup or configuration of servers.
    b. Easily deploy your application.
    c. Easily scale up or scale down your application.
    d. Redundancy, failover, backups are built into the system end to end. DNS, load balancers, caching server, application server and the database all run on multiple servers in different availability zones such that if failures occur there will be no or little downtime.
3. E.g. PHP Fog, AppFog.

Unit 5: Open Source Cloud Implementations

Eucalyptus

Eucalyptus is open source software for building AWS-compatible private and hybrid clouds. As an Infrastructure as a Service product, Eucalyptus allows you to flexibly provision your own collections of resources (both compute and storage), on an as-needed basis.

Whether you're looking for an on-premise complement to your public cloud setup, or already have a virtual data center and now want to move into a private or hybrid cloud setup, Eucalyptus can help you get started today. Download Faststart and have your cloud up and running in half an hour, or take advantage of Eucalyptus' seamless interoperation with Amazon Web Services' EC2 and S3 public cloud services, for an enterprise-grade hybrid cloud platform out of the box.

Features

Eucalyptus offers ways to implement, manage, and maintain your own collection of virtual resources (machines, network, and storage). The following is an overview of these features.

- **AWS API compatibility:** Eucalyptus provides API compatibility with Amazon Web Services, to allow you to use familiar tools and commands to provision your cloud.
- **Block- and bucket-based storage abstractions:** Eucalyptus provides storage options compatible with Amazon's EBS (block-based) and S3 (bucket-based) storage products.
- **Self-service capabilities:** Eucalyptus offers a Management Console, allowing your users to request the resources they need, and automatically provisioning those resources where available.
- **Web-based Interface:** The Eucalyptus Management Console is accessible from any device via a browser. The Console initial page provides a Dashboard view of components available to you to manage, configure, provision, and generate various reports.

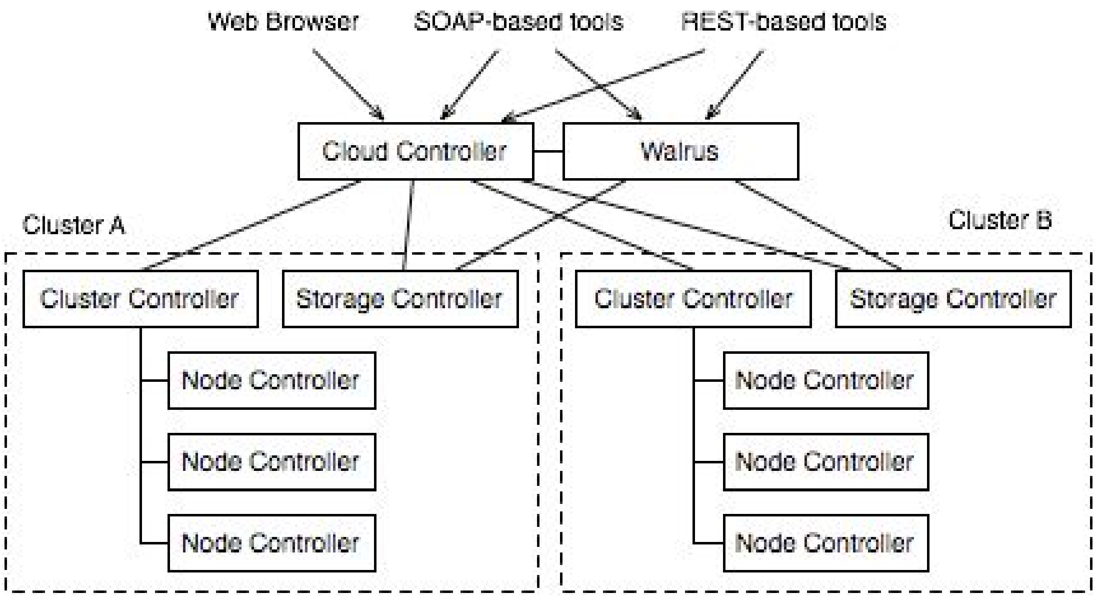
Other features include
- **Auto Scaling:** Allows application developers to scale Eucalyptus resources up or down based on policies defined using Amazon EC2-compatible APIs and tools
- **Elastic Load Balancing:** AWS-compatible service that provides greater fault tolerance for applications
- **CloudWatch:** An AWS-compatible service that allows users to collect metrics, set alarms, identify trends, and take action to ensure applications run smoothly
- **Resource Tagging:** Fine-grained reporting for showback and chargeback scenarios; allows IT/ DevOps to build reports that show cloud utilization by application, department or user
- **Expanded Instance Types:** Expanded set of instance types to more closely align to those available in Amazon EC2. Was 5 before, now up to 15 instance types.
- **Maintenance Mode:** Allows for replication of a virtual machine's hard drive, evacuation of the server node and provides a maintenance window.

- **SSH Key Management:** Eucalyptus employs public and private keypairs to validate your identity when you log into VMs using SSH. You can add, describe, and delete keypairs.
- **Image Management:** Before running instances, someone must prepare VM images for use in the cloud. This can be an administrator or a user.
- **Linux Guest OS Support:** Eucalyptus lets you run your own VMs in the cloud. You can run, describe, terminate, and reboot a wide variety of Linux-based VMs that were prepared using image management commands.
- **Windows Guest OS Support:** Eucalyptus allows you to run, describe, terminate, reboot, and bundle instances of Windows VMs.

Components of Eucalyptus

Eucalyptus is comprised of six components: Cloud Controller (CLC), Walrus, Cluster Controller (CC), Storage Controller (SC), Node Controller (NC) and an optional VMware Broker (Broker or VB).

Other than the VMware Broker, each component is a stand-alone web service. This architecture allows Eucalyptus both to expose each web service as a well-defined, language-agnostic API, and to support existing web service standards for secure communication between its components.



Cloud Controller

- The Cloud Controller (CLC) is the entry-point into the cloud for administrators, developers, project managers, and end-users.
- The CLC queries other components for information about resources, makes high-level scheduling decisions, and makes requests to the Cluster Controllers (CCs).
- As the interface to the management platform, the CLC is responsible for exposing and managing the underlying virtualized resources (servers, network, and storage).
- You can access the CLC through command line tools that are compatible with Amazon’s Elastic Compute Cloud (EC2) and through a web-based Eucalyptus Administrator Console.

Walrus

- Walrus allows users to store persistent data, organized as buckets and objects.

- You can use Walrus to create, delete, and list buckets, or to put, get, and delete objects, or to set access control policies.
- Walrus is interface compatible with Amazon's Simple Storage Service (S3), providing a mechanism for storing and accessing virtual machine images and user data.
- Walrus can be accessed by end-users, whether the user is running a client from outside the cloud or from a virtual machine instance running inside the cloud.

Cluster Controller

- The Cluster Controller (CC) generally executes on a machine that has network connectivity to both the machines running the Node Controllers (NCs) and to the machine running the CLC.
- CCs gather information about a set of NCs and schedules virtual machine (VM) execution on specific NCs.
- The CC also manages the virtual machine networks. All NCs associated with a single CC must be in the same subnet.

Storage Controller

- The Storage Controller (SC) provides functionality similar to the Amazon Elastic Block Store (Amazon EBS).
- The SC is capable of interfacing with various storage systems. Elastic block storage exports storage volumes that can be attached by a VM and mounted or accessed as a raw block device.
- EBS volumes persist past VM termination and are commonly used to store persistent data.
- An EBS volume cannot be shared between VMs and can only be accessed within the same availability zone in which the VM is running.
- Users can create snapshots from EBS volumes. Snapshots are stored in Walrus and made available across availability zones.
- Eucalyptus with SAN support lets you use your enterprise-grade SAN devices to host EBS storage within a Eucalyptus cloud.

Node Controller

- The Node Controller (NC) executes on any machine that hosts VM instances.
- The NC controls VM activities, including the execution, inspection, and termination of VM instances.
- It also fetches and maintains a local cache of instance images, and it queries and controls the system software (host OS and the hypervisor) in response to queries and control requests from the CC.
- The NC is also responsible for the management of the virtual network endpoint.

VMware Broker

- VMware Broker enables Eucalyptus to deploy virtual machines (VMs) on VMware infrastructure elements.
- VMware Broker mediates all interactions between the CC and VMware hypervisors (ESX/ESXi) either directly or through VMware vCenter.

OpenStack

OpenStack is a collection of open source technology projects. It provides an operating platform for orchestrating clouds on a massive scale. Its technology is hypervisor independent and includes software to provision virtual machines (VMs) on standard hardware. In addition, it offers a distributed object store and a wide range of optional functionality, including a network controller, authentication manager, management dashboard, and block storage

Features

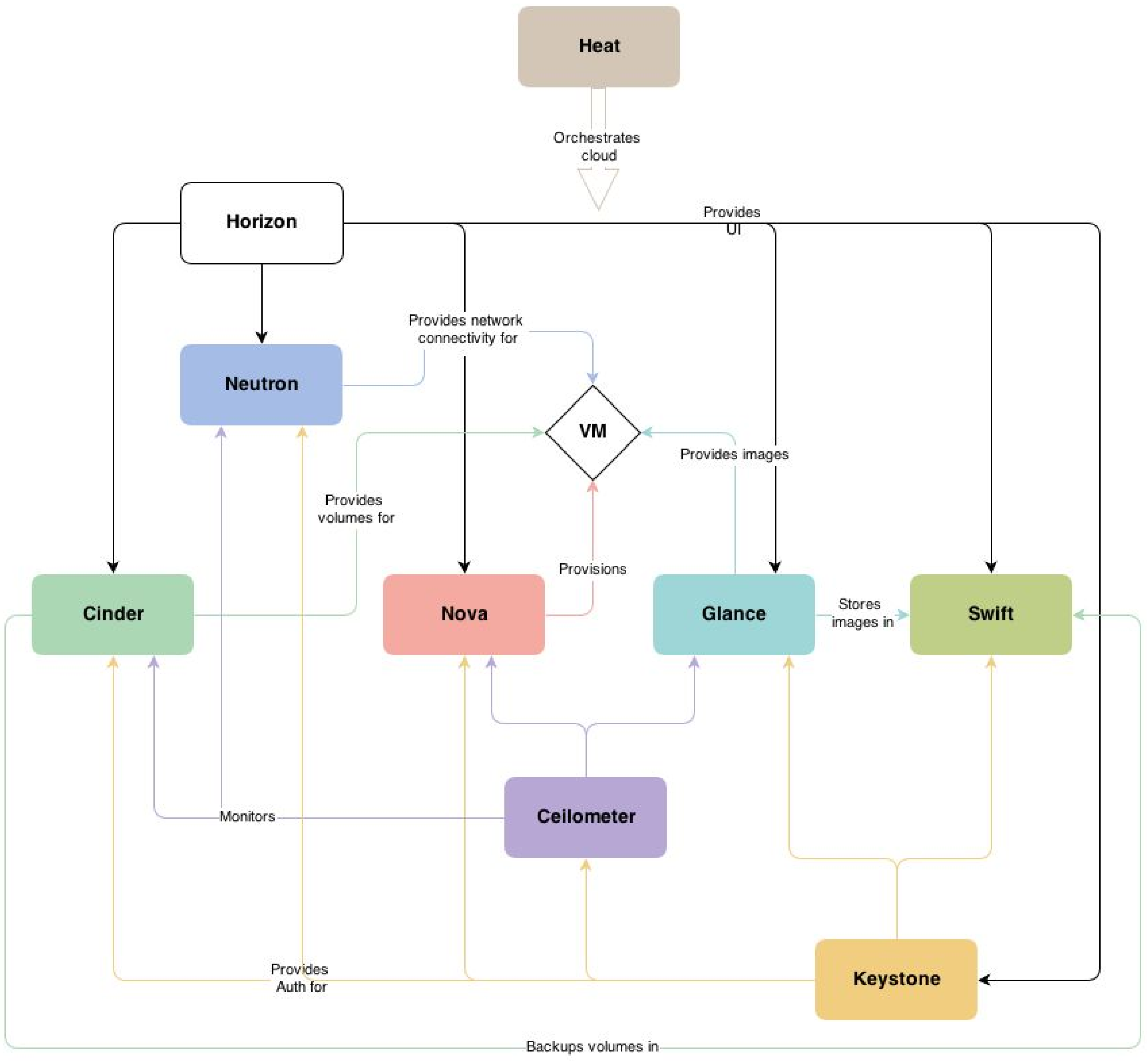
Features	Benefits
Leverages commodity hardware	No lock-in, lower price/GB.
HDD/node failure agnostic	Self-healing, reliable, data redundancy protects from failures.
Unlimited storage	Large and flat namespace, highly scalable read/write access, able to serve content directly from storage system.
Multi-dimensional scalability	Scale-out architecture: Scale vertically and horizontally-distributed storage. Backs up and archives large amounts of data with linear performance.
Account/container/object structure	No nesting, not a traditional file system: Optimized for scale, it scales to multiple petabytes and billions of objects.
Built-in replication 3✕ + data redundancy (compared with 2✕ on RAID)	A configurable number of accounts, containers and object copies for high availability.
Easily add capacity (unlike RAID resize)	Elastic data scaling with ease.
No central database	Higher performance, no bottlenecks.
RAID not required	Handle many small, random reads and writes efficiently.
Built-in management utilities	Account management: Create, add, verify, and delete users; Container management: Upload, download, and verify; Monitoring: Capacity, host, network, log trawling, and cluster health.
Drive auditing	Detect drive failures preempting data corruption.
Expiring objects	Users can set an expiration time or a TTL on an object to control access.
Direct object access	Enable direct browser access to content, such as for a control panel.
Realtime visibility into client requests	Know what users are requesting.
Supports S3 API	Utilize tools that were designed for the popular S3 API.

Restrict containers per account	Limit access to control usage by user.

Components

OpenStack consists of seven core projects:

1. Compute (Nova)
2. Networking (Neutron/Quantum)
3. Identity Management (Keystone)
4. Object Storage (Swift)
5. Block Storage (Cinder)
6. Image Service (Glance)
7. User Interface Dashboard (Horizon)



OpenStack Compute (Nova)

- OpenStack Compute (Nova) controls the cloud computing fabric (the core component of an infrastructure service).
- Written in Python, it creates an abstraction layer for virtualizing commodity server resources such as CPU, RAM, network adapters, and hard drives, with functions to improve utilization and automation.
- Its live VM management has functions to launch, resize, suspend, stop, and reboot through integration with a set of supported hypervisors.
- There is also a mechanism to cache VM images on compute nodes for faster provisioning.
- When the images are running, it is possible to store and manage files programmatically through an application programming interface (API).

OpenStack Networking (Neutron/Quantum)

- Networking (Neutron), formerly called Quantum, includes the capability to manage LANs with capabilities for virtual LAN (VLAN), Dynamic Host Configuration Protocol, and Internet Protocol version 6.
- Users can define networks, subnets, and routers to configure their internal topology, and then allocate IP addresses and VLANs to these networks.
- Floating IP addresses allow users to assign (and reassign) fixed external IP addresses to the VMs.

OpenStack Identity Management (Keystone)

- OpenStack Identity Management (Keystone) manages a directory of users as well as a catalog of OpenStack services they can access.
- Its purpose is to expose a central authentication mechanism across all OpenStack components.
- Rather than providing the authentication itself, Keystone can integrate with a variety of other directory services, such as Pluggable Authentication Module, Lightweight Directory Access Protocol (LDAP), or OAuth.
- Through these plug-ins, it's able to facilitate multiple forms of authentication ranging from simple user name-password credentials to sophisticated multifactor systems.
- OpenStack Identity makes it possible for administrators to configure centralized policies that apply across users and systems.
- They can create projects and users, assign them to administrative domains, define role-based resource permissions, and integrate with other directories like LDAP. A catalog contains a list of all of the deployed services in a single registry.
- Users and tools can retrieve a list of the services they can access either through programmatic requests or by logging in to the dashboard which they can also use to create resources and assign them to their account.

OpenStack Object Storage (Swift)

- OpenStack Object Storage (Swift) is based on the Rackspace Cloud Files product and is a redundant storage system ideal for scale-out storage.
- OpenStack ensures data replication and distribution across the devices in its pool, so users can employ commodity hard disks and servers rather than more expensive equipment.

- In the event of a component failure, OpenStack is able to replenish the content from other active systems to new cluster members.
- The architecture also enables horizontal scalability, because it's easy to extend storage clusters with additional servers, as required.
- Swift is a distributed storage system primarily for static data, such as VM images, backups, and archives.
- The software writes files and other objects to a set of disk drives that can be distributed on multiple servers around one or more data centers, ensuring data replication and integrity across the cluster.

OpenStack Block Storage (Cinder)

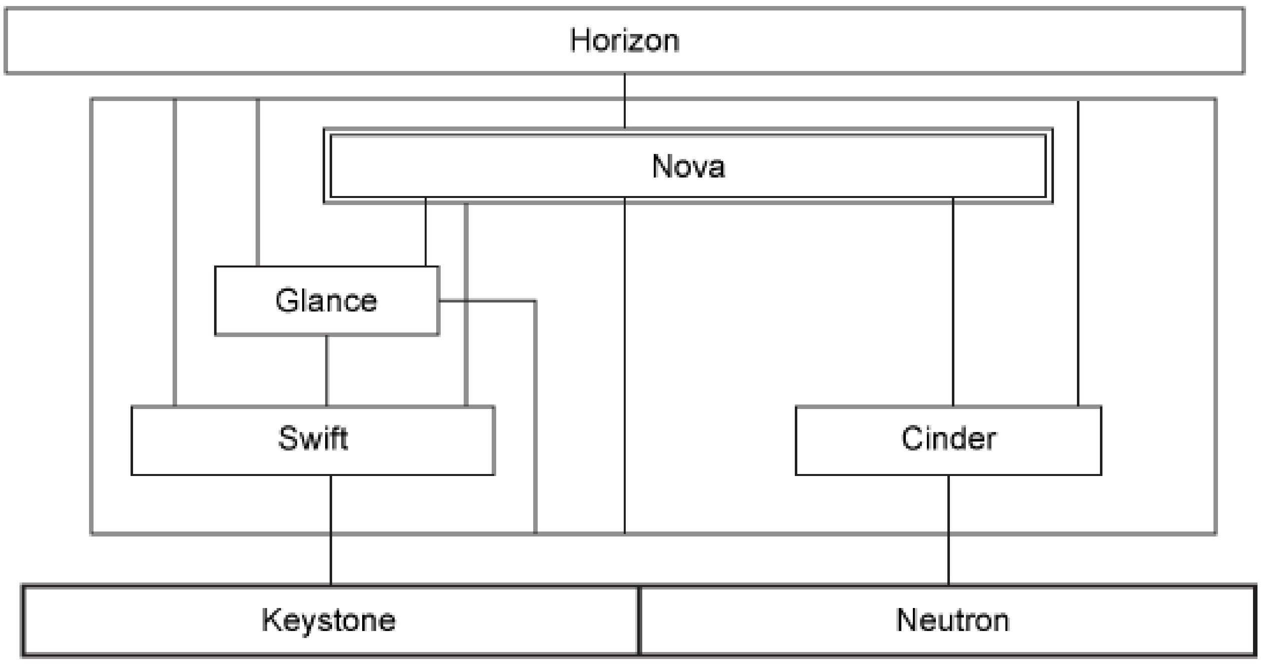
- OpenStack Block Storage (Cinder) manages block-level storage that compute instances use. Block storage lends itself well to scenarios with strict performance constraints, such as databases and file systems.
- The most common storage to use with Cinder is Linux server storage, but plug-ins exist for other platforms, as well, including Ceph, NetApp, Nexenta, and SolidFire.
- Cloud users can manage their storage requirements through the dashboard. The system provides interfaces to create, attach, and detach block devices from/to servers.
- It is also possible to back up Cinder volumes by using the snapshot capability.

OpenStack Image Service (Glance)

- OpenStack Image Service (Glance) provides support for VM images, specifically the system disks to be used in launching VM instances.
- In addition to discovery, registration, and activation services, it has capabilities for snapshots and backups.
- Glance images can function as templates to roll out new servers quickly and consistently.
- The API server exposes a Representational State Transfer (REST)-ful interface with which users can list and fetch virtual disk images that are assigned to an extensible set of back-end stores, including OpenStack Object Storage.
- Users can provide both private and public images to the service in a variety of formats, including VHD (Microsoft(® Hyper-V®), VDI (VirtualBox), VMDK (VMware), qcow2 (Qemu/Kernel-based Virtual Machine), and Open Virtualization Format.
- Functions exist to register new virtual disk images, query for information on publicly available disk images, and stream virtual disk images.

OpenStack Interdependencies

A typical OpenStack implementation will integrate most if not all of projects.



- Three elements interact with all the components in the system. Horizon is the graphical UI that administrators can most easily use to manage all the projects.
- Keystone handles the management of authorized users, and Neutron defines the networks that provide connectivity between the components.
- Nova can arguably be considered the core OpenStack. It handles the orchestration of workloads. Its compute instances usually require some form of persistent storage which can be either block-based (Cinder) or object-based (Swift).
- Nova also requires an image to launch an instance. Glance handles this request, whereby it can optionally use Swift as its storage back end.
- The OpenStack architecture had endeavored to make each project as independent as possible which gives users the option to deploy only a subset of the functionality and integrate it with other systems and technologies that offer similar or complementary functions.
- Nonetheless, this independence shouldn't mask the fact that a fully functional private cloud is likely to require virtually all the functionality to operate smoothly, and the elements will need to be tightly integrated.