**UNIT 2:**

**Cloud Enabling Technologies Service Oriented Architecture: REST and Systems of Systems – Web Services – Publish, Subscribe Model – Basics of Virtualization – Types of Virtualization –  
Implementation Levels of Virtualization – Virtualization Structures – Tools and Mechanisms –  
Virtualization of CPU – Memory – I/O Devices –Virtualization Support and Disaster Recovery.**

**What is service-oriented architecture?**

* Service-oriented architecture (SOA) is a method of software development that uses software components called services to create business applications. Each service provides a business capability, and services can also communicate with each other across platforms and languages. Developers use SOA to reuse services in different systems or combine several independent services to perform complex tasks.
* For example, multiple business processes in an organization require the user authentication functionality. Instead of rewriting the authentication code for all business processes, you can create a single authentication service and reuse it for all applications. Similarly, almost all systems across a healthcare organization, such as patient management systems and electronic health record (EHR) systems, need to register patients. These systems can call a single, common service to perform the patient registration task.

**Benefits of service-oriented architecture:**

Service-oriented architecture (SOA) has several benefits over the traditional monolithic architectures in which all processes run as a single unit. Some major benefits of SOA include the following:

**Faster time to market**

Developers reuse services across different business processes to save time and costs. They can assemble applications much faster with SOA than by writing code and performing integrations from scratch.

**Efficient maintenance**

It’s easier to create, update, and debug small services than large code blocks in monolithic applications. Modifying any service in SOA does not impact the overall functionality of the business process.

**Greater adaptability**

SOA is more adaptable to advances in technology. You can modernize your applications efficiently and cost effectively. For example, healthcare organizations can use the functionality of older electronic health record systems in newer cloud-based applications.

**How SOA Supports Cloud Computing**

When a business decides to migrate into the cloud, there are numerous decisions that must be made. However, if that business has already employed SOA, many of these important issues are automatically solved. A benefit of incorporating SOA is that you can have a display of horizontal services that run through your entire organization. It is this horizontal display that comes to mind when businesses contemplate virtualizing their IT environment.

More specifically, once SOA comes into the picture, services that were already built can be recycled, repurposed and redistributed, making them rapidly available throughout the company’s network. How does this apply to cloud computing? SOA’s ability to reuse services means that cloud costs can be kept down as well as made more agile. This, in turn, allows companies the ability to increase their rate of change (scalability).

Additionally, incorporating SOA allows companies the option of deploying a software-as-a-service in their cloud platform. Meanwhile, the cloud pumps out processing power as demanded. This relationship allows for greater flexibility and robustness, providing ample time to meet your operational needs.

SOA lays down a framework that simplifies the management of information technology (IT) systems. SOA employs mechanisms that allows IT systems to work together cohesively within one enterprise cloud platform.

For the strongest cloud computing platform, a business needs interfaces and architectures that are capable of reaching cloud resources. This cannot be achieved by slapping links between cloud resources and core enterprise information systems. That’s where SOA comes in. SOA provides the ability to look at business systems as an entire set of services. When you combine this with the cloud, you also have the ability to extend this to cloud resources as well.

The information stored within the cloud needs structure for documentation and organization within the architecture. In the past, this has critical step has often been overlooked in favor of focusing on “ad-hoc hype-drive” components. However, this aspect of the cloud must be readdressed and one solution is incorporating SOA.

**REST:** REST stands for REpresentational State Transfer and API stands for Application Program Interface. REST is a software architectural style that defines the set of rules to be used for creating web services. Web services which follow the REST architectural style are known as RESTful web services. It allows requesting systems to access and manipulate web resources by using a uniform and predefined set of rules. Interaction in REST based systems happen through Internet’s Hypertext Transfer Protocol (HTTP).

A Restful system consists of a:

* client who requests for the resources.
* server who has the resources.

It is important to create REST API according to industry standards which results in ease of development and increase client adoption.

**Architectural Constraints of RESTful API:** There are six architectural constraints which makes any web service are listed below:

* Uniform Interface
* Stateless
* Cacheable
* Client-Server
* Layered System
* Code on Demand

The only optional constraint of REST architecture is code on demand. If a service violates any other constraint, it cannot strictly be referred to as RESTful.

**Uniform Interface:** It is a key constraint that differentiate between a REST API and Non-REST API. It suggests that there should be an uniform way of interacting with a given server irrespective of device or type of application (website, mobile app). ‘

**Stateless:** It means that the necessary state to handle the request is contained within the request itself and server would not store anything related to the session. In REST, the client must include all information for the server to fulfill the request whether as a part of query params, headers or URI. Statelessness enables greater availability since the server does not have to maintain, update or communicate that session state.

**Cacheable:** Every response should include whether the response is cacheable or not and for how much duration responses can be cached at the client side.

**Client-Server:** REST application should have a client-server architecture. A Client is someone who is requesting resources and are not concerned with data storage, which remains internal to each server, and server is someone who holds the resources and are not concerned with the user interface or user state.

**Layered system:** An application architecture needs to be composed of multiple layers. Each layer doesn’t know any thing about any layer other than that of immediate layer and there can be lot of intermediate servers between client and the end server.

**Code on demand:** It is an optional feature. According to this, servers can also provide executable code to the client. The examples of code on demand may include the compiled components such as Java applets and client-side scripts such as JavaScript.

**Rules of REST API:** There are certain rules which should be kept in mind while creating REST API endpoints.

* REST is based on the resource or noun instead of action or verb based. It means that a URI of a REST API should always end with a noun. Example: /api/users is a good example, but /api?type=users is a bad example of creating a REST API.
* HTTP verbs are used to identify the action. Some of the HTTP verbs are – GET, PUT, POST, DELETE, GET, PATCH.
* A web application should be organized into resources like users and then uses HTTP verbs like – GET, PUT, POST, DELETE to modify those resources. And as a developer it should be clear that what needs to be done just by looking at the endpoint and HTTP method used.

| **URI** | **HTTP verb** | **Description** |
| --- | --- | --- |
| api/users | GET | Get all users |
| api/users/new | GET | Show form for adding new user |
| api/users | POST | Add a user |
| api/users/1 | PUT | Update a user with id = 1 |
| api/users/1/edit | GET | Show edit form for user with id = 1 |
| api/users/1 | DELETE | Delete a user with id = 1 |
| api/users/1 | GET | Get a user with id = 1 |

**Systems of Systems:**

* A system of systems (SoS) is the collection of multiple, independent systems in context as part of a larger, more complex system. A system is a group of interacting, interrelated and interdependent components that form a complex and unified whole.
* These independent and possibly distributed systems pool their resources together, creating a new and more complex system. Individual systems in an SoS work together to provide functionalities and performance that none of the independent systems, or constituent systems, could accomplish on their own.
* The goal of an SoS architecture is to get maximum value out of a large system by understanding how each of the smaller systems works, interfaces and is used.
* It is a type of architecture that uses a single interface to allow multiple systems to be used as one. A classic example is the Internet. Internet protocols such as HTTP allow the use of information and services across millions of physical machines using a single interface such as a web browser. Cloud computing is also a system of systems approach to computing that provides a single platform to access the computing power of many physical machines.

**Web Services:**

* The Internet is the worldwide connectivity of hundreds of thousands of computers belonging to many different networks.
* A web service is a standardized method for propagating messages between client and server applications on the World Wide Web. A web service is a software module that aims to accomplish a specific set of tasks. Web services can be found and implemented over a network in cloud computing.
* The web service would be able to provide the functionality to the client that invoked the web service.
* A web service is a set of open protocols and standards that allow data exchange between different applications or systems. Web services can be used by software programs written in different programming languages and on different platforms to exchange data through computer networks such as the Internet. In the same way, communication on a computer can be inter-processed.

**Web Service Components**

* XML and HTTP is the most fundamental web service platform. All typical web services use the following components:

**1. SOAP (Simple Object Access Protocol)**

* SOAP stands for "Simple Object Access Protocol". It is a transport-independent messaging protocol. SOAP is built on sending XML data in the form of SOAP messages. A document known as an XML document is attached to each message.
* Only the structure of an XML document, not the content, follows a pattern. The great thing about web services and SOAP is that everything is sent through HTTP, the standard web protocol.
* Every SOAP document requires a root element known as an element. In an XML document, the root element is the first element.
* The "envelope" is divided into two halves. The header comes first, followed by the body. Routing data, or information that directs the XML document to which client it should be sent, is contained in the header. The real message will be in the body.

### 2. UDDI (Universal Description, Search, and Integration)

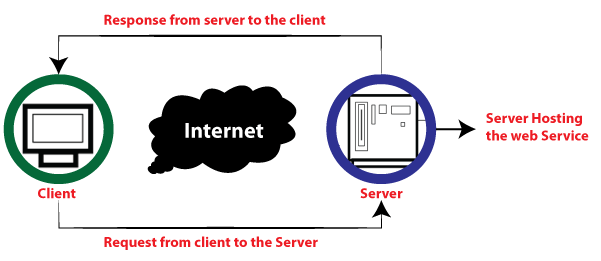
* UDDI is a standard for specifying, publishing and searching online service providers. It provides a specification that helps in hosting the data through web services.
* UDDI provides a repository where WSDL files can be hosted so that a client application can search the WSDL file to learn about the various actions provided by the web service. As a result, the client application will have full access to UDDI, which acts as the database for all WSDL files.
* The UDDI Registry will keep the information needed for online services, such as a telephone directory containing the name, address, and phone number of a certain person so that client applications can find where it is.

### 3. WSDL (Web Services Description Language)

* The client implementing the web service must be aware of the location of the web service. If a web service cannot be found, it cannot be used. Second, the client application must understand what the web service does to implement the correct web service.
* WSDL, or Web Service Description Language, is used to accomplish this. A WSDL file is another XML-based file that describes what a web service does with a client application. The client application will understand where the web service is located and how to access it using the WSDL document.

**How does web service work?**

The diagram shows a simplified version of how a web service would function. The client will use requests to send a sequence of web service calls to the server hosting the actual web service.



Remote procedure calls are used to perform these requests. The calls to the methods hosted by the respective web service are known as Remote Procedure Calls (RPC). Example: Flipkart provides a web service that displays the prices of items offered on Flipkart.com. The front end or presentation layer can be written in .NET or Java, but the web service can be communicated using a programming language.

The data exchanged between the client and the server, XML, is the most important part of web service design. XML (Extensible Markup Language) is a simple, intermediate language understood by various programming languages. It is the equivalent of HTML.

As a result, when programs communicate with each other, they use XML. It forms a common platform for applications written in different programming languages to communicate with each other.

Web services employ SOAP (Simple Object Access Protocol) to transmit XML data between applications. The data is sent using standard HTTP.

## Features of Web Service

**Web services have the following characteristics:**

**(a) XML-based:** A web service's information representation and record transport layers employ XML. There is no need for networking, operating system, or platform bindings when using XML.

**Loosely Coupled:** The subscriber of an Internet service provider may not necessarily be directly connected to that service provider. The user interface for a web service provider may change over time without affecting the user's ability to interact with the service provider. A strongly coupled system means that the decisions of the mentor and the server are inextricably linked, indicating that if one interface changes, the other must be updated.

A loosely connected architecture makes software systems more manageable and easier to integrate between different structures.

**(c) Ability to be synchronous or asynchronous:** Synchronicity refers to the client's connection to the execution of the function. Asynchronous operations allow the client to initiate a task and continue with other tasks.

**(e) Supports remote procedural calls:** Consumers can use XML-based protocols to call procedures, functions, and methods on remote objects that use web services. A web service must support the input and output framework of the remote system.

**(f) Supports document exchanges:** One of the most attractive features of XML for communicating with data and complex entities.

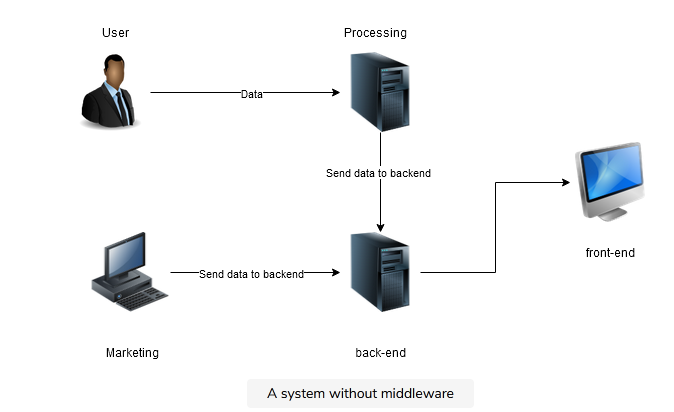
**Functions of Web services:**

* It’s possible to access it via the internet or intranet networks.
* XML messaging protocol that is standardized.
* Operating system or programming language independent.
* Using the XML standard, it is self-describing.
* A simple location approach can be used to locate it

**Publish, Subscribe Model:**

Consider a scenario where we might face failures due to tight dependency.

In the case of **tight dependency**, every component is dependent on some other element for its smooth functioning. The system relies on things to happen in a specific order. When some component fails to perform its function, the functioning of other components is also affected, and that may cause a temporary system failure.

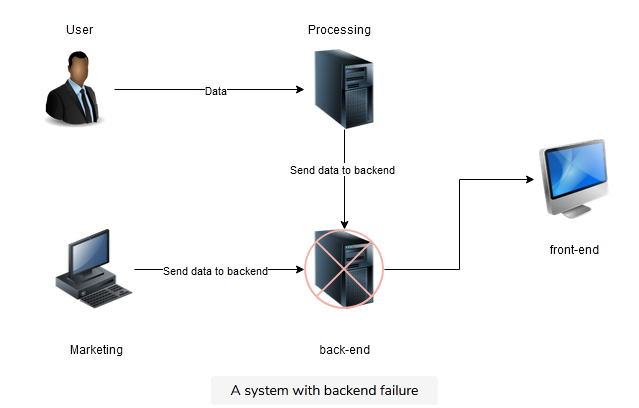


Here, the backend receives data from two sources:

1. The user (user data is processed and then sent to the backend)
2. Marketing

Once the data is received in the backend, it is moved to the frontend.

Let’s suppose that the backend fails at a certain point, maybe due to some hardware issues. In that case, all the messages from these sources are lost, and the frontend stops working because it doesn’t have any data to work with (backend).

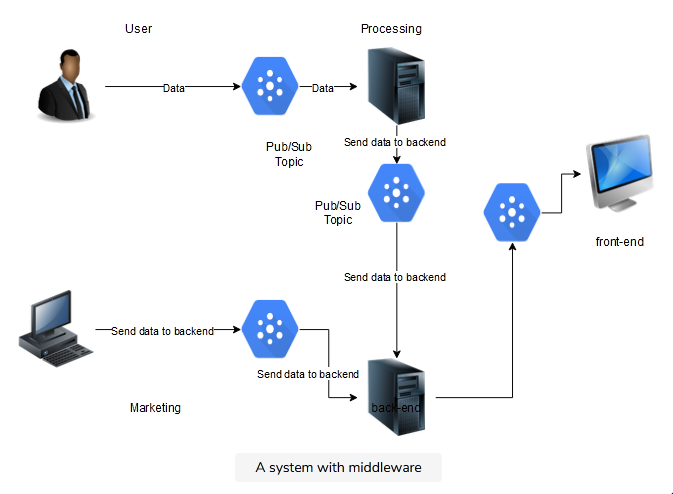


To avoid such failures, we use a **message bus**. There are different types of message buses, but we will only learn about a publish/subscribe model (Pub/Sub) in this shot.

### Using the Pub/Sub model

**Pub/Sub** is a layer between the components where we pass the message directly.

In this model, we can split our message bus into different groups of messages. In Pub/Sub, we call these **topics**. A message can either be sent to or received from a topic. Let’s look at the system using Pub/Sub:



By introducing Pub/Sub, we have loosely coupled our services and presented resilience in component failure. Pub/Sub in Google Cloud is a **fully managed service**, that is, messages are queued, and made ready to be consumed again when a component is restored. In this way, we can think of Pub/Sub as a shock absorber for the system.

Pub/Sub can also act on triggers or trigger some actions. Here are a few essential points to know about Pub/Sub:

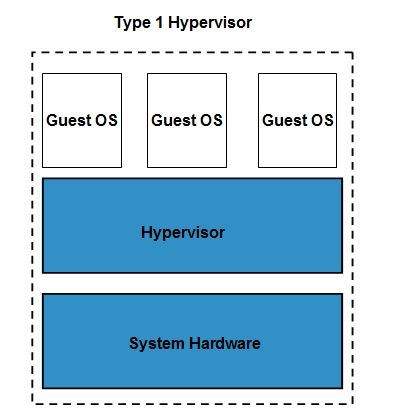
* It is Google’s global messaging and event ingestion service.
* It is a server-less and fully-managed service.
* We can create multiple publisher/subscriber patterns—for example, one to one, many to one, and so on.
* It guarantees at least one delivery of every message.
* We can process messages in real-time or in batches.

**Basics of Virtualization:**

* **Virtualization** is the "creation of a virtual (rather than actual) version of something, such as a server, a desktop, a storage device, an operating system or network resources".
* It is a technique, which allows to share a single physical instance of a resource or an application among multiple customers and organizations. It does by assigning a logical name to a physical storage and providing a pointer to that physical resource when demanded.
* Creation of a virtual machine over existing operating system and hardware is known as Hardware Virtualization. A Virtual machine provides 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.**

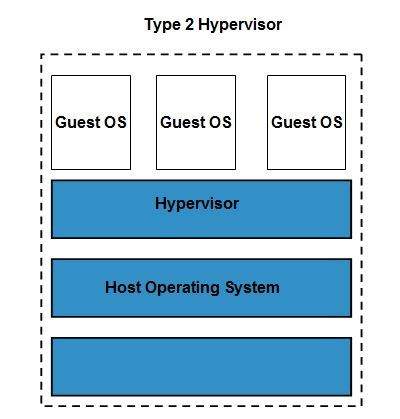
**Hypervisor**

* 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. LynxSecure, RTS Hypervisor, Oracle VM, Sun xVM Server, VirtualLogic VLX are examples of Type 1 hypervisor. The following diagram shows the Type 1 hypervisor.



The **type1 hypervisor** does not have any host operating system because they are installed on a bare system.

**Type 2 hypervisor** is a software interface that emulates the devices with which a system normally interacts. Containers, KVM, Microsoft Hyper V, VMWare Fusion, Virtual Server 2005 R2, Windows Virtual PC and **VMWare workstation 6.0** are examples of Type 2 hypervisor. The following diagram shows the Type 2 hypervisor.



**Types of Virtualization:**

1. Hardware Virtualization.
2. Operating system Virtualization.
3. Server Virtualization.
4. Storage Virtualization.

**1) Hardware Virtualization:**

* When the virtual machine software or virtual machine manager *(VMM) is directly installed on the hardware system* is known as hardware virtualization.
* The main job of hypervisor is to control and monitoring the processor, memory and other hardware resources.
* After virtualization of hardware system we can install different operating system on it and run different applications on those OS.

**Usage:**

Hardware virtualization is mainly done for the server platforms, because controlling virtual machines is much easier than controlling a physical server.

**2) Operating System Virtualization:**

When the virtual machine software or virtual machine manager *(VMM) is installed on the Host operating system* instead of directly on the hardware system is known as operating system virtualization.

**Usage:**

Operating System Virtualization is mainly used for testing the applications on different platforms of OS.

**3) Server Virtualization:**

When the virtual machine software or virtual machine manager *(VMM) is directly installed on the Server system* is known as server virtualization.

**Usage:**

Server virtualization is done because a single physical server can be divided into multiple servers on the demand basis and for balancing the load.

**4) Storage Virtualization:**

Storage virtualization is the *process of grouping the physical storage from multiple network storage devices so that it looks like a single storage device*.

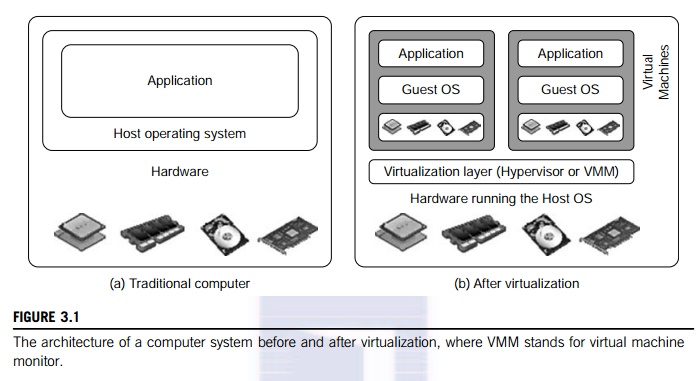
Storage virtualization is also implemented by using software applications.

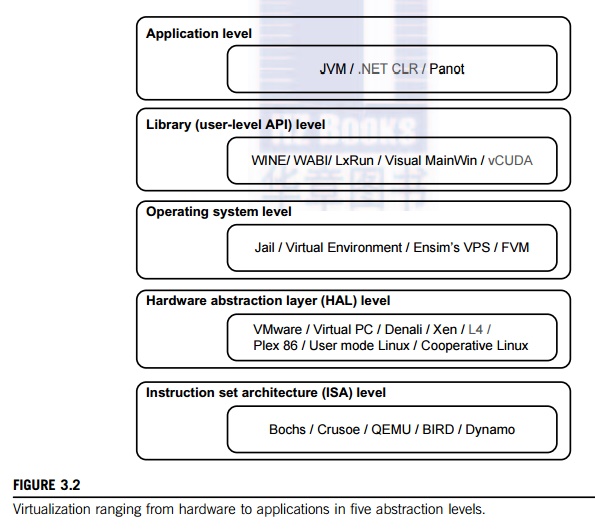
Usage:

Storage virtualization is mainly done for back-up and recovery purposes.

**IMPLEMENTATION LEVELS OF VIRTUALIZATION**

* It is not simple to set up virtualization. Your computer runs on an operating system that gets configured on some particular hardware. It is not feasible or easy to run a different operating system using the same hardware. To do this, you will need a hypervisor which is a bridge between the hardware and the virtual operating system, which allows smooth functioning.
* A traditional computer runs with a host operating system specially tailored for its hardware architecture, as shown in **Figure 3.1(a).** After virtualization, different user applications managed by their own operating systems (guest OS) can run on the same hardware, independent of the host OS. This is often done by adding additional software, called a virtualization layer as shown in Figure 3.1(b). This virtualization layer is known as hypervisor or virtual machine monitor (VMM) [54]. The VMs are shown in the upper boxes, where applications run with their own guest OS over the virtualized CPU, memory, and I/O resources.





**1) Instruction Set Architecture Level (ISA)**

ISA virtualization can work through ISA emulation. This is used to run many legacy codes written for a different hardware configuration. These codes run on any virtual machine using the ISA. With this, a binary code that originally needed some additional layers to run is now capable of running on the x86 machines. It can also be tweaked to run on the x64 machine. With ISA, it is possible to make the virtual machine hardware agnostic.

For the basic emulation, an interpreter is needed, which interprets the source code and then converts it into a hardware format that can be read. This then allows processing. This is one of the five implementation levels of virtualization in Cloud Computing..

**2) Hardware Abstraction Level (HAL)**

True to its name HAL lets the virtualization perform at the level of the hardware. This makes use of a hypervisor which is used for functioning. The virtual machine is formed at this level, which manages the hardware using the virtualization process. It allows the virtualization of each of the hardware components, which could be the input-output device, the memory, the processor, etc.

Multiple users will not be able to use the same hardware and also use multiple virtualization instances at the very same time. This is mostly used in the cloud-based infrastructure.

**3) Operating System Level**

At the level of the operating system, the virtualization model is capable of creating a layer that is abstract between the operating system and the application. This is an isolated container on the operating system and the physical server, which uses the software and hardware. Each of these then functions in the form of a server.

When there are several users and no one wants to share the hardware, then this is where the virtualization level is used. Every user will get his virtual environment using a dedicated virtual hardware resource. In this way, there is no question of any conflict.

**4) Library Level**

The operating system is cumbersome, and this is when the applications use the API from the libraries at a user level. These APIs are documented well, and this is why the library virtualization level is preferred in these scenarios. API hooks make it possible as it controls the link of communication from the application to the system.

**5) Application Level**

The application-level virtualization is used when there is a desire to virtualize only one application and is the last of the implementation levels of virtualization in Cloud Computing. One does not need to virtualize the entire environment of the platform.

This is generally used when you run virtual machines that use high-level languages. The application will sit above the virtualization layer, which in turn sits on the application program.

It lets the high-level language programs compiled to be used at the application level of the virtual machine run seamlessly.

**Virtualization Structures-Tools and Mechanisms:**

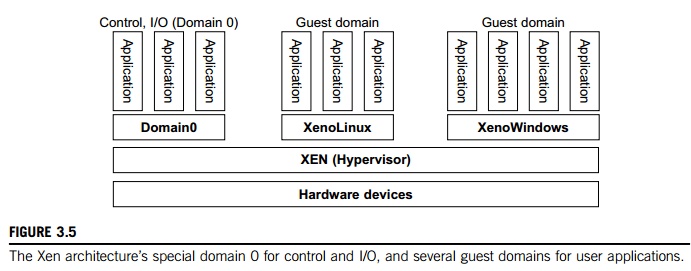
Depending on the position of the virtualization layer, there are several classes of VM architectures, namely the hypervisor architecture, para- virtualization, and host-based virtualization. The hypervisor is also known as the VMM (Virtual Machine Monitor). They both perform the same virtualization operations.

**1. Hypervisor and Xen Architecture**

* The hypervisor supports hardware-level virtualization (see Figure 3.1(b)) on bare metal devices like CPU, memory, disk and network interfaces. The hypervisor software sits directly between the physi-cal hardware and its OS. This virtualization layer is referred to as either the VMM or the hypervisor. The hypervisor provides hypercalls for the guest OSes and applications. Depending on the functional-ity, a hypervisor can assume a micro-kernel architecture like the Microsoft Hyper-V. Or it can assume a monolithic hypervisor architecture like the VMware ESX for server virtualization.
* A micro-kernel hypervisor includes only the basic and unchanging functions (such as physical memory management and processor scheduling). The device drivers and other changeable components are outside the hypervisor. A monolithic hypervisor implements all the aforementioned functions, including those of the device drivers. Therefore, the size of the hypervisor code of a micro-kernel hyper-visor is smaller than that of a monolithic hypervisor. Essentially, a hypervisor must be able to convert physical devices into virtual resources dedicated for the deployed VM to use.

1.1 The Xen Architecture

* 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 core components of a Xen system are the hypervisor, kernel, and applications. The organi-zation of the three components is important. Like other virtualization systems, many guest OSes can run on top of the hypervisor. However, not all guest OSes are created equal, and one in particular controls the others.



* The guest OS, which has control ability, is called Domain 0, and the others are called Domain U. Domain 0 is a privileged guest OS of Xen. It is first loaded when Xen boots without any file system drivers being available. Domain 0 is designed to access hardware directly and manage devices. Therefore, one of the responsibilities of Domain 0 is to allocate and map hardware resources for the guest domains (the Domain U domains).
* For example, Xen is based on Linux and its security level is C2. Its management VM is named Domain 0, which has the privilege to manage other VMs implemented on the same host. If Domain 0 is compromised, the hacker can control the entire system. So, in the VM system, security policies are needed to improve the security of Domain 0. Domain 0, behaving as a VMM, allows users to create, copy, save, read, modify, share, migrate, and roll back VMs as easily as manipulating a file, which flexibly provides tremendous benefits for users. Unfortunately, it also brings a series of security problems during the software life cycle and data lifetime.

**2. 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, nonvirtualizable 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.

2.1 Full Virtualization:

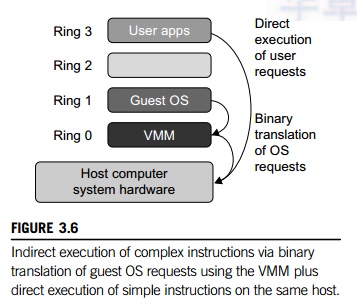
With full virtualization, noncritical instructions run on the hardware directly while critical instructions are discovered and replaced with traps into the VMM to be emulated by software. Both the hypervisor and VMM approaches are considered full virtualization. Why are only critical instructions trapped into the VMM? This is 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.

**2.2 Binary Translation of Guest OS Requests Using a VMM:**

This approach was implemented by VMware and many other software companies. As shown in Figure 3.6, 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.

2.3 Host-Based Virtualization

 An alternative VM architecture is to install a virtualization layer on top of the host OS. This host OS is still responsible for managing the hardware. The guest OSes are installed and run on top of the virtualization layer. Dedicated applications may run on the VMs. Certainly, some other applications.



can also run with the host OS directly. This host-based architecture has some distinct advantages, as enumerated next. First, the user can install this VM architecture without modifying the host OS.  The virtualizing software can rely on the host OS to provide device drivers and other low-level services. This will simplify the VM design and ease its deployment.

Second, the host-based approach appeals to many host machine configurations. Compared to the hypervisor/VMM architecture, the performance of the host-based architecture may also be low. When an application requests hardware access, it involves four layers of mapping which downgrades performance significantly. When the ISA of a guest OS is different from the ISA of the underlying hardware, binary translation must be adopted. Although the host-based architecture has flexibility, the performance is too low to be useful in practice.

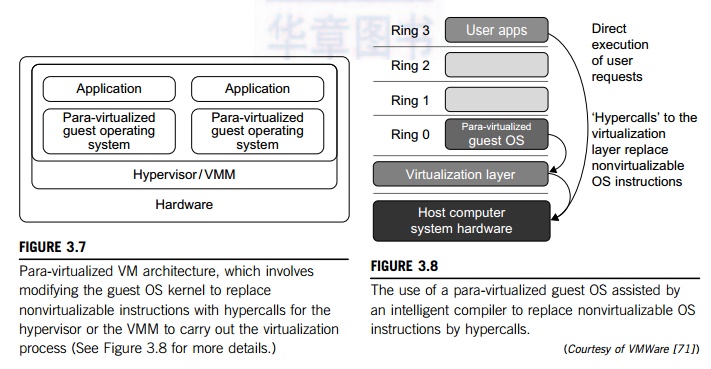
**3. Para-Virtualization with Compiler Support**

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 soft-ware 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 nonvirtualizable 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. The best example of para-virtualization is the KVM to be described below.

**3.1 Para-Virtualization Architecture:**

When the x86 processor is virtualized, a virtualization layer is inserted between the hardware and the OS. According to the x86 ring definition, the virtualization layer should also be installed at Ring 0. Different instructions at Ring 0 may cause some problems. In Figure 3.8, we show that para-virtualization replaces nonvirtualizable instructions with hypercalls that communicate directly with the hypervisor or VMM. However, when the guest OS kernel is modified for virtualization, it can no longer run on the hardware directly.



Although para-virtualization reduces the overhead, it has incurred other problems. First, its compatibility and portability may be in doubt, because it must support the unmodified OS as well. Second, the cost of maintaining para-virtualized OSes is high, because they may require deep OS kernel modifications. Finally, the performance advantage of para-virtualization varies greatly due to workload variations. Compared with full virtualization, para-virtualization is relatively easy and more practical. The main problem in full virtualization is its low performance in binary translation. To speed up binary translation is difficult. Therefore, many virtualization products employ the para-virtualization architecture. The popular Xen, KVM, and VMware ESX are good examples.

**3.2 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.

 3.3 Para-Virtualization with Compiler Support

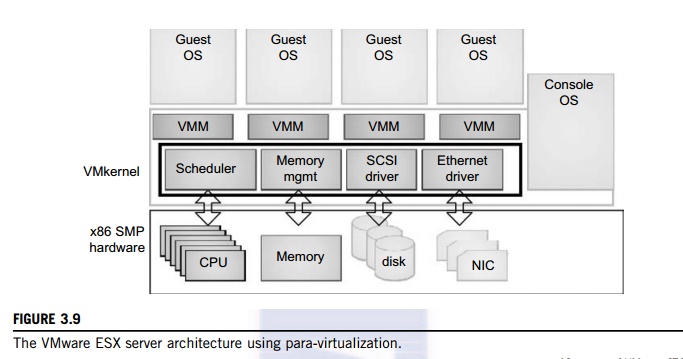
Unlike the full virtualization architecture which intercepts and emulates privileged and sensitive instructions at runtime, para-virtualization handles these instructions at compile time. The guest OS kernel is modified to replace the privileged and sensitive instructions with hypercalls to the hypervi-sor 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. On an UNIX system, a system call involves an interrupt or service routine. The hypercalls apply a dedicated service routine in Xen.

Example 3.3 VMware ESX Server for Para-Virtualization

VMware pioneered the software market for virtualization. The company has developed virtualization tools for desktop systems and servers as well as virtual infrastructure for large data centers. ESX is a VMM or a hypervisor for bare-metal x86 symmetric multiprocessing (SMP) servers. It accesses hardware resources such as I/O directly and has complete resource management control. An ESX-enabled server consists of four components: a virtualization layer, a resource manager, hardware interface components, and a service console, as shown in Figure 3.9. To improve performance, the ESX server employs a para-virtualization architecture in which the VM kernel interacts directly with the hardware without involving the host OS.

The VMM layer virtualizes the physical hardware resources such as CPU, memory, network and disk controllers, and human interface devices. Every VM has its own set of virtual hardware resources. The resource manager allocates CPU, memory disk, and network bandwidth and maps them to the virtual hardware resource set of each VM created. Hardware interface components are the device drivers and the VMware ESX Server File System. The service console is responsible for booting the system, initiating the execution of the VMM and resource manager, and relinquishing control to those layers. It also facilitates the process for system administrators.



**Various tools of virtualization:**

**Ganeti:**

* It is a virtual machine cluster management tool built on top of existing virtualisation technologies such as Xen or KVM and other open source software. Since 2007 Ganeti is developed and released as free and open-source software.
* Originally subject to the requirements of the GNU General Public License (GPL) version 2, the license was changed to the 2-clause BSD license in version 2.11.6, released September 2014.
* Ganeti acts as a convenient wrapper around existing hypervisors for system administrators to set up a cluster. It is used by Google for some of its computing infrastructure and also by the Linux Foundation (formerly Open Source Development Labs) for hosting open source projects.

**Kernel-based Virtual Machine (KVM):**

* It is an open source virtualization technology built into Linux. Specifically, KVM lets you turn Linux into a hypervisor that allows a host machine to run multiple, isolated virtual environments called guests or virtual machines (VMs).
* *KVM is part of Linux*. If you’ve got Linux 2.6.20 or newer, you’ve got KVM. KVM was first announced in 2006 and merged into the mainline Linux kernel version a year later. Because KVM is part of existing Linux code, it immediately benefits from every new Linux feature, fix, and advancement without additional engineering.

**OVirt:**

* It is an open source data center virtualization platform developed and promoted by Red Hat.
* OVirt offers large-scale, centralized management for server and desktop virtualization, was designed as an open source alternative to VMware vCenter.
* OVirt provides kernel-based virtual machine management for multi-node virtualization. Kernel-based Virtual Machines (KVMs) are part of a virtualization infrastructure that turns the Linux kernel into a hypervisor.

**Packer:** is an open source tool for creating identical Virtual Machine (VM) images for multiple platforms from a single source configuration.

**Xen:**

* The Xen Cloud Platform is an open source virtualization product that provides both virtualization and cloud computing capabilities.
* The Xen Cloud Platform includes VM lifecycle management, resource pools, event tracking, Open vSwitch support, real-time performance monitoring and Storage XenMotion.

**Vagrant:**

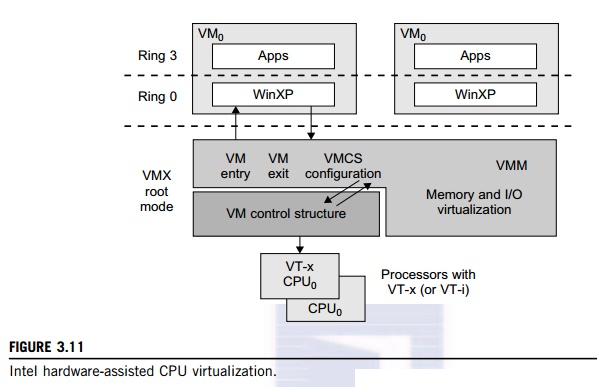
* It is an open-source software product for building and maintaining portable virtual software development environments; e.g., for VirtualBox, KVM, Hyper-V, Docker containers, VMware, and AWS.
* It tries to simplify the software configuration management of virtualization in order to increase development productivity.

**Virtualization of CPU:**

* 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.

**2.1 Hardware-Assisted CPU Virtualization**

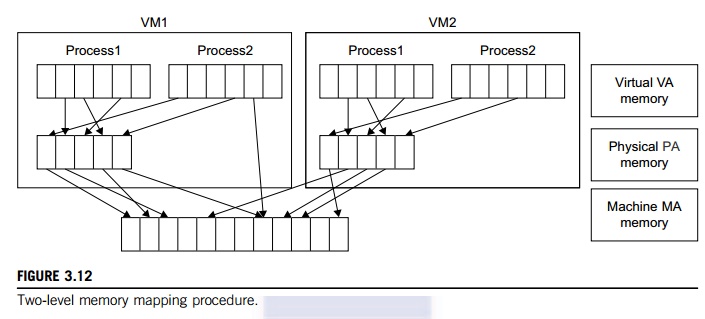
* This technique attempts to simplify virtualization because full or paravirtualization is complicated. Intel and AMD add an additional mode called privilege mode level (some people call it Ring-1) to x86 processors. Therefore, operating systems can still run at Ring 0 and the hypervisor can run at Ring -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.
* Example 3.5 Intel Hardware-Assisted CPU Virtualization:



Although x86 processors are not virtualizable primarily, great effort is taken to virtualize them. They are used widely in comparing RISC processors that the bulk of x86-based legacy systems cannot discard easily. Virtuali-zation of x86 processors is detailed in the following sections. Intel’s VT-x technology is an example of hardware-assisted virtualization, as shown in Figure 3.11. Intel calls the privilege level of x86 processors the VMX Root Mode. In order to control the start and stop of a VM and allocate a memory page to maintain the CPU state for VMs, a set of additional instructions is added. At the time of this writing, Xen, VMware, and the Microsoft Virtual PC all implement their hypervisors by using the VT-x technology.

**Memory Virtualization:**

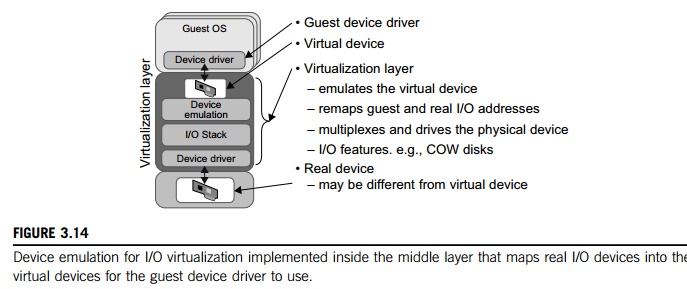
* Virtual memory virtualization is similar to the virtual memory support provided by modern operating systems. In a traditional execution environment, the operating system maintains mappings of virtual memory to machine memory using page tables, which is a one-stage mapping from virtual memory to machine 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.



* 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.

**I/O Virtualization**

* I/O virtualization involves managing the routing of I/O requests between virtual devices and the shared physical hardware. At the time of this writing, there are three ways to implement I/O virtualization: full device emulation, para-virtualization, and direct I/O. 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 full device emulation approach is shown in Figure 3.14.
* A single hardware device can be shared by multiple VMs that run concurrently. However, software emulation runs much slower than the hardware it emulates. 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.

**Virtualization Support:**

* Virtualization enables users to disjoint operating systems from the underlying hardware, i.e, users can run multiple operating systems such as Windows, Linux, on a single physical machine at the same time. Such operating systems are known as guest Oses (operating systems).
* Essentially, Virtualization **allows cloud providers to deliver users along with existing physical computer hardware**. As a simple process, it enables cloud users to purchase only necessary computing resources when they actually need it, and to sustain those resources cost-effectively when the workload expands.
* Virtualization **relies on software to simulate hardware functionality and create a virtual computer system**. This enables IT organizations to run more than one virtual system – and multiple operating systems and applications – on a single server.
* Security has been the advantageous concern for adopting virtualization. The security is served through firewalls that prevent from any unreliable access and preserve the data safe and confidential.
* With the deployment of virtualization, users can work efficiently as the working process is very streamlined and agile. Presently, the employed network switch is easy to use, flexible and saves time. Virtualization is also helpful in troubleshooting technical errors, occurring in any of the connected devices. It eradicates the issues of retaining or recovering lost data due to corrupted or crashed devices, and therefore promotes ROI and saves time.

**Virtualization Disaster Recovery:**

* Virtual disaster recovery is a type of DR that typically involves replication and allows a user to fail over to virtualized workloads.
* For the most efficient virtual disaster recovery, an organization should copy virtual machine (VM) workloads off-site on a regular basis. Replication can essentially make a real-time copy of VMs in a separate location, thus strengthening DR.
* Virtualization provides flexibility in disaster recovery. When servers are virtualized, they are containerized into VMs, independent from the underlying hardware. Therefore, an organization does not need the same physical servers at the primary site as at its secondary disaster recovery site.
* Other benefits of virtual disaster recovery include ease, efficiency and speed. Virtualized platforms typically provide high availability in the event of a failure. Virtualization helps meet recovery time objectives (RTOs) and recovery point objectives (RPOs), as replication is done as frequently as needed, especially for critical systems. DR planning and failover testing is also simpler with virtualized workloads than with a physical setup, making disaster recovery a more attainable process for organizations that may not have the funds or resources for physical DR.

### Virtual disaster recovery planning and testing:

Virtual infrastructures can be complex. In a recovery situation, that complexity can be an issue, so it's important to have a comprehensive DR plan.

A virtual disaster recovery plan has many similarities to a traditional DR plan. An organization should:

* Decide which systems and data are the most critical for recovery, and document them.
* Get management support for the DR plan
* Complete a risk assessment and business impact analysis to outline possible risks and their potential impacts.
* Document steps needed for recovery.
* Define RTOs and RPOs.
* Test the plan.

As with a traditional DR setup, you should clearly define who is involved in planning and testing, and the role of each staff member. That extends to an actual recovery event, as staff should be ready for their tasks during an unplanned incident.