CLOUD COMPUTING

Unit II — Virtualization in Cloud Computing

Topics Covered in this Unit

Virtualization Fundamentals: Concept, Motivation, and Core Architectural Benefits
Understanding Hypervisors: Type-1 (Bare-Metal) vs Type-2 (Hosted) Hypervisor Architectures
Types of Virtualization: Full, Para, Hardware-Assisted, OS-Level, and Device Virtualization
Processor (CPU) & Memory Virtualization: Popek-Goldberg Requirements, SPT, EPT & Ballooning
Virtual Clustering & Virtual Infrastructure: vDC, Dynamic Resource Scheduling & Live VM Migration
Network Virtualization (SDN/VXLAN/NFV) & Storage Virtualization (SAN/NAS/SDS)
Containerization vs Virtualization & Container Orchestration (Kubernetes Architecture)
Virtualization Applications, Pitfalls & Key Issues (VM Sprawl, Security & Noisy Neighbors)
High-Yield University Exam Question Bank with Detailed Model Answers & Unit Summary

Compiled in a structured, easy-to-understand, textbook style format
Specially curated for B.Tech / B.E. / BCA / MCA / B.Sc Computer Science & IT Students

COURSE SYLLABUS & MAPPING

[UNIT II SYLLABUS — PRESCRIBED UNIVERSITY CURRICULUM]

This section contains the official syllabus for Unit II. Verify with your university curriculum mapping.

Unit II Syllabus Breakdown:

Unit II Virtualization in Cloud Computing: Virtualization: What's virtualization, Benefits of Virtualization, Types of Virtualization: Processor virtualization, Memory virtualization, Full virtualization, Para virtualization, and Device virtualization, Virtual Clustering, Virtualization Architecture, Containerization and orchestration, Understanding importance of Hypervisors, Virtualization Applications, Issues with Virtualization, Virtualization and Cloud Computing: Virtualizations in Cloud, Virtual Infrastructure, CPU Virtualization, Network and Storage Virtualization

• Course Code: _____ • Semester / Year: _____

TABLE OF CONTENTS

2.1 Fundamentals & Concepts of Virtualization

Definition: Virtualization

Virtualization is the foundational technology that abstracts physical hardware resources (CPUs, memory, network interfaces, and storage drives) to create multiple simulated, isolated execution environments known as Virtual Machines (VMs) or containers on a single physical host machine.

Motivation Behind Virtualization

Before virtualization, the traditional 'one server, one application' enterprise deployment model resulted in severe operational and financial inefficiencies:

- **1. Underutilized Server Capacity:** Physical x86 servers operated at a meager 10%–15% average CPU utilization, leaving 85%+ of computing capacity wasted while consuming full electric power and cooling.
- **2. Datacenter Sprawl & High Real-Estate Costs:** Adding new applications required purchasing new physical rack-mounted server blades, leading to exponential datacenter footprint expansion and power supply saturation.
- **3. Hardware Dependency & Vendor Lock-in:** Operating systems and software were tied directly to specific physical motherboard chipsets and disk controllers, making hardware upgrades cumbersome.
- **4. Slow Provisioning Cycles:** Procuring, cabling, racking, and installing an operating system on a new physical server required weeks or months of manual engineering effort.

Core Benefits of Virtualization in Cloud Computing

- • **Server Consolidation:** Multiple virtual machines running heterogeneous operating systems (Linux, Windows) can run concurrently on a single high-capacity physical server blade, increasing overall hardware utilization to 75%–90%.
- • **Hardware Independence & Portability:** A virtual machine is encapsulated entirely as software files (e.g., .vmdk, .qcow2). It can be paused, copied, snapshotted, and moved seamlessly across different physical servers without compatibility errors.
- • **Rapid Provisioning & Elasticity:** New VM instances can be cloned from golden master images and booted in under 60 seconds through automated API commands.
- • **Strong Fault & Security Isolation:** A software crash, kernel panic, or malicious security intrusion inside one guest VM does not affect the host OS or neighboring tenant VMs on the same hardware.

- • **Live VM Migration & High Availability:** Active VMs can be migrated from one physical host to another across the network with zero perceptible application downtime (e.g., VMware vMotion, KVM Live Migration) for maintenance.

2.2 Hypervisors: Role, Importance & Classification

Definition: Hypervisor (Virtual Machine Monitor - VMM)

A Hypervisor or Virtual Machine Monitor (VMM) is a specialized low-level software, firmware, or hardware virtualization management layer that sits between the physical hardware and guest operating systems to allocate, isolate, schedule, and mediate access to CPU, memory, storage, and I/O devices.

Type-1 (Bare-Metal / Native) Hypervisors

Type-1 hypervisors run directly on the bare-metal physical host hardware without any underlying general-purpose host operating system. The hypervisor has direct ring-0 access to physical hardware registers, interrupts, and memory controllers:

- • **Architectural Characteristics:** Extremely lightweight, minimal overhead (1%–2%), high performance, microsecond I/O response, and hardened enterprise security perimeter.
- • **Target Deployment:** Hyperscale enterprise datacenters, public cloud infrastructure, and mission-critical server clusters.
- • **Prominent Examples:** VMware ESXi, Microsoft Hyper-V, Kernel-based Virtual Machine (KVM - Linux kernel module), Citrix XenServer.

Type-2 (Hosted) Hypervisors

Type-2 hypervisors run as an application layer on top of an existing host operating system (e.g., Windows 11, macOS, Ubuntu Linux). Hardware access from guest VMs must pass through the hypervisor and the host OS kernel:

- • **Architectural Characteristics:** Easier installation, extensive hardware device compatibility via host drivers, but suffers from higher performance overhead and latency due to double scheduling.
- • **Target Deployment:** Software development, QA testing, legacy software execution, and desktop virtualization.
- • **Prominent Examples:** Oracle VirtualBox, VMware Workstation, VMware Fusion, Parallels Desktop.

ARCHITECTURAL COMPARISON: TYPE-1 VS TYPE-2 HYPERVISORS

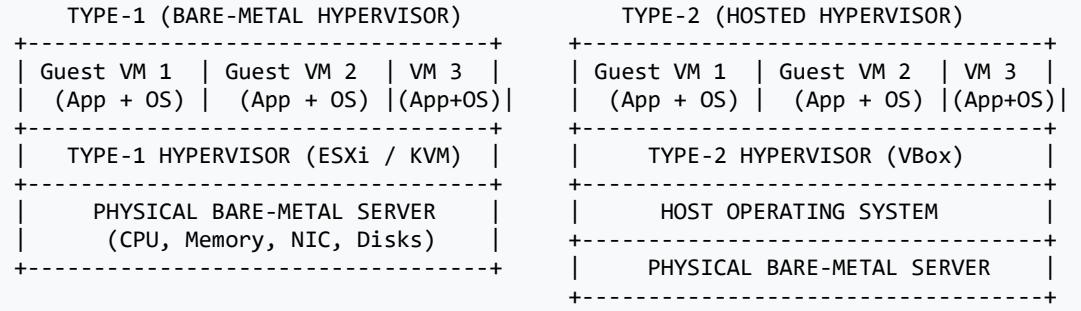

Figure: Structural architectural difference between Bare-Metal and Hosted Hypervisors

Comparison Parameter	Type-1 Hypervisor (Bare-Metal)	Type-2 Hypervisor (Hosted)
Installation Level	Directly on bare-metal physical server hardware.	As an application software on top of a Host OS.
Performance & Overhead	Near-native bare-metal speed; negligible overhead (~1-2%).	Moderate performance loss; higher latency due to Host OS context switches.
Hardware Control	Direct control over CPU registers, MMU, and PCI busses.	Indirect control mediated through Host OS device drivers.
Security Vulnerability	Extremely small attack surface; hardened microkernel.	Inherits all security vulnerabilities of the underlying Host OS.
Primary Use Case	Production cloud datacenters (AWS, Azure, private clouds).	Local software development, testing, student labs, desktop virtualization.
Industry Examples	VMware ESXi, KVM, Microsoft Hyper-V, Xen.	Oracle VirtualBox, VMware Workstation, Parallels Desktop.

2.3 Types and Techniques of Virtualization

Virtualization can be implemented across multiple architectural layers of the computing stack:

1. Full Virtualization (Binary Translation & Direct Execution)

In Full Virtualization, the hypervisor presents an exact, complete virtual simulation of the underlying hardware to each guest VM. The guest operating system is completely unaware that it is running in a virtualized environment and requires absolutely no source code modifications:

- • **Non-Privileged Instructions:** Executed directly on the physical CPU at native speed (Direct Execution).

- • **Privileged / Sensitive Instructions:** The hypervisor intercepts privileged CPU instructions issued by the guest kernel using dynamic binary translation (BT) and executes safe equivalent instructions in software.
- • **Advantage & Drawback:** Unmodified guest OS support (can run Windows, Linux, legacy OS), but binary translation incurs noticeable CPU overhead.

2. Para-Virtualization (Hypercalls)

In Para-Virtualization, the guest operating system is explicitly modified to be aware that it is executing in a virtualized environment:

- • **Hypercalls Mechanism:** Instead of issuing non-virtualizable privileged hardware instructions that must be trapped and emulated, the modified guest kernel communicates directly with the hypervisor using explicit software hooks called 'hypercalls' (similar to system calls in OS).
- • **Advantage & Drawback:** Drastically reduced CPU virtualization overhead and near-native I/O throughput. However, proprietary closed-source operating systems (e.g., Microsoft Windows) cannot be easily modified without vendor collaboration.
- • **Prominent Example:** Xen Hypervisor (dom0 / domU architecture).

3. Hardware-Assisted Virtualization

Modern x86 processors incorporate dedicated hardware virtualization instruction set extensions (Intel VT-x and AMD-V) that eliminate the need for software binary translation:

- • **Root vs Non-Root Execution Modes:** The CPU introduces two operational modes: VMX Root Mode (where the hypervisor executes with full hardware control) and VMX Non-Root Mode (where guest operating systems execute in Ring 0 without crashing the host).
- • **VMCS (Virtual Machine Control Structure):** A dedicated hardware data structure that saves and restores CPU register states automatically during VM-Entry and VM-Exit transitions.

4. Processor (CPU) and Memory Virtualization

- • **Popek-Goldberg Virtualization Theorem:** Formulates that an architecture is fully virtualizable if and only if all sensitive instructions (instructions that expose or modify hardware control registers) are a strict subset of privileged instructions (instructions that trap when executed in user mode). Classical x86 had 17 sensitive unprivileged instructions (the 'x86 virtualization hole'), which necessitated Binary Translation or Intel VT-x.
- • **Memory Virtualization (Three-Tier Memory Mapping):** Virtualization introduces three distinct layers of memory addresses:
  - **1. Guest Virtual Address (GVA):** Memory address used by processes inside the guest VM.

  - **2. Guest Physical Address (GPA):** Contiguous physical memory illusion presented to the guest OS.
  - **3. Host Physical Address (HPA):** Actual physical RAM addresses on the motherboard memory sticks.

- • **Shadow Page Tables (SPT) vs Extended Page Tables (EPT):** SPT emulates GVA-to-HPA mapping via software traps, causing high overhead. Hardware-assisted Nested Page Tables (Intel EPT / AMD NPT) perform two-dimensional page walks directly in hardware MMU.
- • **Memory Ballooning:** A dynamic memory management driver inside guest VMs that inflates to claim unused memory and returns it to the host hypervisor during host RAM shortages.

5. Device and I/O Virtualization

- • **Emulated Devices:** Software emulation of standard legacy hardware (e.g., Intel e1000 NIC, IDE disk). High CPU overhead due to multiple register traps.
- • **Para-virtualized Drivers (virtio):** High-performance shared-memory ring buffers between guest and hypervisor for disk and network I/O.
- • **Direct Assignment & SR-IOV (Single Root I/O Virtualization):** Bypasses hypervisor completely by allowing a physical PCIe device (e.g., 100GbE NIC, Nvidia GPU) to present itself as multiple separate Virtual Functions (VFs) directly mapped into guest VM memory.

2.4 Virtual Clustering & Virtual Infrastructure

A Virtual Infrastructure aggregates physical pools of compute, memory, storage, and networking across hundreds of physical servers into dynamic logical resource pools:

Virtual Clustering

- • **Definition:** A virtual cluster is an aggregation of multiple interconnected physical host servers managed by a centralized virtualization controller (e.g., VMware vCenter, OpenStack Nova) to form a unified, resilient pool of compute capacity.
- • **Distributed Resource Scheduling (DRS):** Continuously monitors CPU and RAM utilization across cluster hosts. If a host becomes overloaded, DRS automatically calculates load distribution and live-migrates VMs to underutilized nodes without service disruption.
- • **High Availability (HA) Clustering:** Heartbeat signals monitor physical host health. If a physical host experiences a hardware failure (power outage, motherboard crash), the cluster controller automatically restarts affected VMs on healthy nodes within seconds.

Live VM Migration Workflow

PRE-COPY LIVE VM MIGRATION STAGES

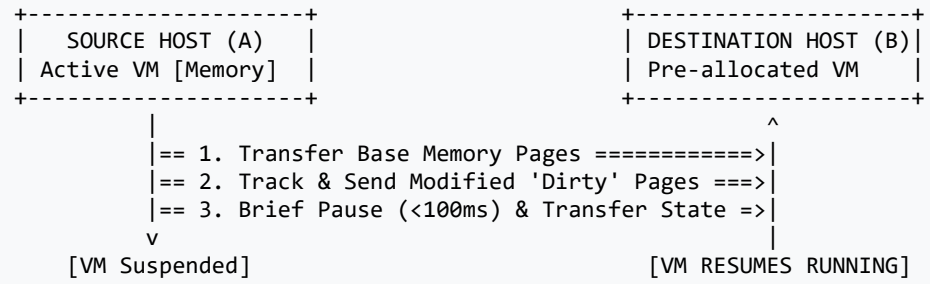

Figure: Pre-copy iterative memory migration guaranteeing sub-second downtime

2.5 Network and Storage Virtualization in Cloud

1. Network Virtualization

- • **Virtual Switches (vSwitch):** Software-based layer-2 switches operating inside the hypervisor kernel to route packets between co-located VMs and physical network cards.
- • **Overlay Networks (VXLAN / NVGRE):** Encapsulates Layer-2 Ethernet frames inside Layer-3 UDP packets, overcoming the 4096 VLAN limitation to support up to 16 million isolated virtual networks across cloud datacenters.
- • **Software-Defined Networking (SDN):** Decouples the Network Control Plane (decision-making software, OpenFlow controllers) from the Data Plane (packet forwarding hardware switches), enabling programmable multi-tenant routing and firewall rules.
- • **Network Function Virtualization (NFV):** Replaces proprietary hardware appliances (firewalls, routers, load balancers) with software running inside standard VMs.

2. Storage Virtualization

- • **Definition:** The pooling of physical storage from multiple network storage devices into what looks like a single storage device managed from a central console.
- • **Storage Area Networks (SAN) & LUNs:** High-speed Fibre Channel or iSCSI block storage aggregated into Logical Unit Numbers (LUNs) formatted with cluster file systems (e.g., VMFS).
- • **Software-Defined Storage (SDS):** Decouples storage software from underlying hardware (e.g., Ceph, VMware vSAN), providing automated policy-based tiering, thin provisioning (allocating disk blocks only upon data write), and deduplication.

2.6 Containerization and Orchestration

Definition: Containerization (OS-Level Virtualization)

Containerization is an operating-system-level virtualization method for deploying and running distributed applications without launching an entire guest OS for each app. Multiple isolated user-space instances (containers) share the single underlying Host OS kernel.

Core Linux Kernel Mechanisms Powering Containers

- **1. Linux Namespaces (Isolation):** Provides isolated views of system resources for each container: PID (process IDs), NET (network interfaces & routing tables), MNT (filesystem mount points), IPC (inter-process communication), and UTS (hostnames).
- **2. Control Groups (cgroups - Resource Governance):** Enforces strict limits, prioritization, and accounting for physical CPU cycles, memory usage, disk I/O, and network bandwidth per container.
- **3. Union File Systems (UnionFS / Overlay2):** Layered, copy-on-write (CoW) filesystems allowing lightweight, reusable container base images.

Comparison Dimension	Traditional Virtual Machines (VMs)	Application Containers (Docker)
Virtualization Layer	Hardware-level abstraction (Hypervisor emulates virtual CPU, RAM, disks).	Operating System-level abstraction (shares the Host OS kernel).
Guest Operating System	Requires a complete, independent Guest OS for every single VM.	No Guest OS needed; shares Host OS kernel; uses minimal rootfs.
Startup Boot Time	Minutes (full OS initialization, BIOS, systemd boot).	Milliseconds to seconds (starts as a standard isolated OS process).
Resource Footprint	Heavy (several gigabytes of RAM and disk per VM).	Extremely lightweight (megabytes of RAM and disk).
Isolation & Security	Strong hardware-level hypervisor boundary isolation.	Process-level isolation; shared kernel represents a broader attack surface.
Density per Host	Dozens of VMs per physical server blade.	Hundreds to thousands of containers per physical server blade.

Container Orchestration with Kubernetes

At enterprise cloud scale, managing thousands of ephemeral containers manually is impossible. Container Orchestration engines (Kubernetes / K8s) automate container scheduling, service discovery, load balancing, rolling updates, health monitoring, and self-healing:

- • **Pod:** The smallest deployable unit in Kubernetes, wrapping one or more tightly coupled containers sharing the same network IP and storage volumes.

- • **Control Plane:** Consists of kube-apiserver (REST API gateway), etcd (distributed key-value store), kube-scheduler (assigns pods to nodes), and kube-controller-manager (enforces desired state).
- • **Worker Nodes:** Executes pods via kubelet (node agent), kube-proxy (network packet forwarding), and container runtime (containerd).

2.7 Virtualization Applications, Issues & Pitfalls

Virtualization Applications

- • **Production Server Consolidation:** Merging hundreds of underutilized physical servers into a handful of dense multi-core blade chassis.
- • **Dynamic DevOps CI/CD Pipelines:** Spawning clean, isolated build and testing environments on-demand and tearing them down post-test.
- • **Disaster Recovery & High Availability:** Replicating VM snapshot images to off-site cloud datacenters for instantaneous recovery during natural disasters.
- • **Legacy OS Encapsulation:** Running obsolete operating systems (Windows Server 2003, Linux 2.4) securely on modern x86 server hardware.

Issues and Pitfalls with Virtualization

- **1. VM Sprawl:** Because creating virtual machines is effortless, administrators spawn hundreds of ad-hoc VMs and forget to decommission them, resulting in untracked resource consumption and unpatched security vulnerabilities.
- **2. Hypervisor Escape & Single Point of Failure:** A catastrophic security flaw in the hypervisor (e.g., Venom vulnerability) can allow an attacker inside a guest VM to break out into hypervisor memory and hijack all neighboring co-located tenant VMs.
- **3. Noisy Neighbor Resource Contention:** A CPU- or I/O-intensive workload on one VM can exhaust shared PCIe bus bandwidth or L3 CPU cache, degrading the latency of adjacent VMs.
- **4. Complex Licensing & Troubleshooting:** Software vendors often charge complex multi-core virtual licensing models. Diagnosing inter-VM network latency across virtual switches requires specialized monitoring tools.

2.8 High-Yield University Exam Question Bank & Model Answers

Part A: Short Answer Questions (2–3 Marks)

- **Q1. What is a Hypervisor? Differentiate Type-1 and Type-2 hypervisors.**

Answer: A Hypervisor (VMM) is a software/firmware layer that virtualizes hardware to run multiple guest operating systems. Type-1 (Bare-Metal, e.g., ESXi, KVM) runs directly on physical server hardware with maximum performance and minimal overhead. Type-2 (Hosted, e.g., VirtualBox) runs on top of an existing Host OS and is primarily used for desktop development.

- **Q2. Define Para-Virtualization and explain Hypercalls.**

Answer: Para-virtualization is a technique where the guest operating system kernel is modified to be aware of virtualization. Instead of executing non-virtualizable privileged CPU instructions, the guest issues explicit software requests called 'hypercalls' directly to the hypervisor, eliminating binary translation overhead.

- **Q3. What is VM Sprawl and how can it be mitigated?**

Answer: VM sprawl is the uncontrolled proliferation of unmanaged, idle, or forgotten virtual machines across a datacenter, leading to resource exhaustion and security vulnerabilities. It is mitigated using automated lifecycle policies, VM tagging, mandatory expiration leases, and centralized governance tools.

- **Q4. Explain the concept of Memory Ballooning.**

Answer: Memory ballooning is a hypervisor memory reclamation technique where a pseudo-device driver installed inside the guest OS 'inflates' by allocating unused guest memory to itself and releasing those underlying physical RAM pages back to the host hypervisor during host memory shortages.

Part B: Long Answer Questions (8–10 Marks)

- **Q5. Compare Virtual Machines and Containers in detail. Explain the role of Linux Namespaces and cgroups in containerization.**

Model Answer Outline:

- 1. Architectural definitions: Hardware-level abstraction (VMs + Hypervisor) vs OS-level abstraction (Containers + Host Kernel).
- 2. Linux Namespaces deep-dive: PID (Process isolation), NET (Network interfaces), MNT (Filesystem mounts), IPC, and UTS.
- 3. Control Groups (cgroups): Resource budgeting, rate-limiting, and memory/CPU quotas.
- 4. Comprehensive Comparison Table: Boot time, disk footprint, OS dependency, density, security boundary, and migration ease.
- 5. Architectural Diagrams illustrating VM stack (App -> Guest OS -> Hypervisor -> Hardware) vs Container stack (App -> Docker -> Host OS -> Hardware).

- **Q6. Describe the architecture and step-by-step process of Live VM Migration. Why is it vital for Cloud High Availability?**

Model Answer Outline:

- 1. Definition and significance of Live VM Migration (Zero-downtime workload migration across physical hosts).
- 2. Prerequisites: Shared network storage (SAN/NAS), identical CPU instruction family compatibility, gigabit interconnection.
- 3. Step-by-step Pre-Copy Migration Algorithm: Step 1: Pre-allocation on target; Step 2: Iterative page copying; Step 3: Dirty page tracking; Step 4: Final microsecond pause & register transfer; Step 5: ARP broadcast and resumption.
- 4. Applications: Dynamic cluster load rebalancing (DRS), proactive hardware maintenance, energy-efficient host shutdown during low traffic.

2.9 Unit II Summary & Quick Revision Sheet

Virtualization Concept	Key Architectural Takeaways & Exam Points
Hypervisors	Type-1 (Bare-Metal: ESXi, KVM, Hyper-V) vs Type-2 (Hosted: VirtualBox, Workstation).
Virtualization Types	Full (Binary translation), Para (Modified guest + Hypercalls), Hardware-assisted (Intel VT-x / AMD-V with Root/Non-Root modes).
Memory Virtualization	Three-tier mapping: GVA -> GPA -> HPA. Hardware Nested Page Tables (EPT) and Memory Ballooning.
Virtual Infrastructure	Virtual clusters, vSwitch, VXLAN overlay networks, Storage Area Networks (SAN), DRS, HA clustering.
Containers vs VMs	Containers share Host OS kernel via Namespaces & cgroups; VMs virtualize entire hardware stack with separate Guest OSs.