CLOUD COMPUTING

U23CST71

Comprehensive Learning Material — All Five Units

University / College Level | L:3 T:0 P:0 C:3

Department of Computer Science & Technology

COURSE OBJECTIVES

1. Understand principles of cloud architecture, models and infrastructure.
2. Understand concepts of virtualization and virtual machines.
3. Gain knowledge about virtualization infrastructure.
4. Explore and experiment with various cloud deployment environments.
5. Learn about security issues in the cloud environment.

COURSE OUTCOMES (CO)

CO1: Understand the design challenges in the cloud.

CO2: Apply the concept of virtualization and its types.

CO3: Experiment with virtualization of hardware resources and Docker.

CO4: Deploy applications on cloud platforms like AWS, Azure, and OpenStack.

CO5: Analyze security threats and apply IAM policies in cloud environments.

TABLE OF CONTENTS

UNIT I

CLOUD ARCHITECTURE MODELS AND INFRASTRUCTURE

Cloud Architecture · System Models · Distributed Computing · NIST Definition · Reference Architecture · Deployment Models · Service Models · Infrastructure Design

1.1 Introduction to Cloud Computing

Cloud computing is one of the most transformative paradigms in modern information technology. It refers to the delivery of computing services — including servers, storage, databases, networking, software, analytics, and intelligence — over the Internet (the cloud) to offer faster innovation, flexible resources, and economies of scale.

In traditional IT, organizations purchased and maintained their own hardware and software. Cloud computing replaces this model with a pay-as-you-go approach, where resources are rented from a cloud provider. This eliminates the need for large upfront capital investments and allows organizations to scale capacity up or down based on actual needs.

Definition (NIST)

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. — NIST SP 800-145

1.1.1 Evolution of Cloud Computing

Era	Technology	Key Feature
1960s	Mainframe Time-Sharing	Multiple users share expensive mainframe
1980s	Client-Server Computing	Distributed processing, dedicated servers
1990s	Internet & Web	Browser-based access to remote services
2000s	Grid & Utility Computing	Resource pooling across organizations
2006+	Cloud Computing (AWS launch)	On-demand, pay-per-use, massive scale

2010s	Hybrid & Multi-Cloud	Combination of private and public clouds
2020s	Edge & Serverless	Ultra-low latency, event-driven computing

1.1.2 Key Advantages of Cloud Computing

• **Cost Efficiency:** Eliminates large upfront capital expenditure; shift to operational expenditure model.

• **Scalability:** Scale resources up or down on demand within minutes, globally.

• **Reliability:** Data backed up across multiple geographically distributed data centers.

• **Speed and Agility:** New IT resources provisioned in minutes rather than weeks.

• **Global Reach:** Deploy applications to data centers worldwide in a few clicks.

• **Focus on Core Business:** Eliminate time spent managing infrastructure.

• **Security:** Major cloud providers invest significantly in security, often exceeding on-premises capabilities.

1.2 System Models for Distributed Computing

Cloud computing builds on decades of distributed computing research. Understanding these foundational models is essential for appreciating how cloud architectures evolved and function today.

1.2.1 Client-Server Model

The client-server model is the simplest form of distributed computing. Clients are end-user devices that request services, while servers are powerful machines that respond to these requests. The Internet's HTTP/HTTPS architecture is fundamentally a client-server model.

• **Advantages:** Centralized management, easy updates, clear separation of roles.

• **Disadvantages:** Single point of failure at server, limited scalability, high server cost.

• **Examples:** Web browsers (client) and web servers (Apache, Nginx).

1.2.2 Peer-to-Peer (P2P) Model

In a P2P model, all participants (peers) have equal capability and can act as both clients and servers. Resources are shared directly between peers without centralized coordination.

• **Structured P2P:** Uses Distributed Hash Tables (DHT) for organized resource discovery (e.g., BitTorrent, Chord).

• **Unstructured P2P:** Random peer connections; flooding used for resource discovery (e.g., Gnutella).

• **Hybrid P2P:** Combines central server for coordination with P2P for data transfer.

1.2.3 Cluster Computing

A cluster is a group of tightly coupled computers connected via a high-speed LAN, working together as a single system. Clusters are commonly used for high-performance computing (HPC) and high-availability services.

Type	Description	Use Case

Compute Cluster	Parallel processing of computational tasks	Scientific simulations, rendering
Load-Balancing Cluster	Distributes requests across nodes for performance	Web server farms
High-Availability (HA) Cluster	Failover if one node fails — continuity guaranteed	Critical business systems
Storage Cluster	Shared distributed storage accessed by all nodes	Databases, file servers

1.2.4 Grid Computing vs Cloud Computing

Aspect	Grid Computing	Cloud Computing
Definition	Loosely coupled heterogeneous computers for large tasks	On-demand virtualized resource pool
Management	Decentralized, self-coordinating	Centralized, managed by provider
Resource Model	Dedicated allocation per job	Dynamic, shared, multi-tenant
Business Model	Research-oriented, free/donated	Commercial, pay-per-use
Standards	Open (OGSA, Globus Toolkit)	Mix of proprietary and open
Virtualization	Minimal	Core enabling technology
QoS	Best-effort, variable	SLA-guaranteed, consistent
Examples	SETI@Home, LHC Grid	AWS, Azure, Google Cloud

1.3 NIST Cloud Computing Reference Model

The National Institute of Standards and Technology (NIST) published Special Publication 800-145 in 2011, which remains the definitive reference for cloud computing definitions. The NIST model defines cloud computing through five essential characteristics, three service models, and four deployment models.

1.3.1 Five Essential Characteristics

Characteristic	Description	Technical Mechanism
On-Demand Self-Service	Provision resources without human interaction	APIs, web portals, CLI
Broad Network Access	Access over network using standard mechanisms	HTTP/S, TCP/IP, mobile clients

Resource Pooling	Shared resources dynamically assigned to consumers	Multi-tenancy, virtualization
Rapid Elasticity	Resources scale up/down quickly to meet demand	Auto-scaling, containers
Measured Service	Usage monitored, controlled, and reported	Metering, billing, dashboards

1.4 Cloud Reference Architecture — NIST CCRA

The NIST Cloud Computing Reference Architecture (CCRA) defines five major actors and their interactions. This framework provides a common language for understanding roles and responsibilities in complex cloud service chains.

1.4.1 Five Actors

Actor	Role	Example
Cloud Consumer	Uses services, pays provider	Enterprise using AWS EC2
Cloud Provider	Delivers and manages services	Amazon, Microsoft, Google
Cloud Auditor	Independent third-party assessor	ISO 27001 auditing body
Cloud Broker	Manages service use, negotiates between parties	Cloud management platforms
Cloud Carrier	Transport/connectivity between provider and consumer	ISP, CDN provider

1.4.2 Cloud Provider Activities

- **Service Deployment:** Setting up and configuring physical and virtual infrastructure.
- **Service Orchestration:** Managing resource abstraction layers (hardware, virtualization, core connectivity).
- **Cloud Service Management:** Business support, provisioning, portability, and interoperability.
- **Security:** Implementing security controls at all layers.
- **Privacy:** Ensuring proper handling of consumer data.

1.5 Cloud Deployment Models

- **Public Cloud:** Infrastructure owned and operated by a third-party provider, delivered over the Internet. Resources shared among multiple organizations (multi-tenancy). Cost-effective due to economies of scale. Examples: AWS, Microsoft Azure, Google Cloud Platform.
- **Private Cloud:** Dedicated infrastructure for a single organization. Can be hosted on-premises or by a third-party provider. Provides greater control, customization, and security. Suitable for regulated industries (banking, healthcare). Examples: VMware vSphere Private Cloud, OpenStack.

• **Community Cloud:** Shared by organizations with common concerns (mission, security requirements, policy, compliance). Costs and resources shared across the community. Example: Government cloud shared by multiple agencies.

• **Hybrid Cloud:** Integration of two or more distinct cloud types, enabling data and application portability. Enables 'cloud bursting' — run in private, burst to public during peak demand. Most common enterprise model. Examples: Azure Arc, AWS Outposts.

Hybrid Cloud Use Case: A bank stores sensitive customer data in a private cloud (regulatory compliance) but uses the public cloud for customer-facing web applications and data analytics workloads during peak periods.

1.6 Cloud Service Models

The cloud service model determines how much of the IT stack the provider manages versus the customer. Think of it as a spectrum from maximum control (IaaS) to maximum convenience (SaaS).

Layer	IaaS	PaaS	SaaS
Applications	Customer	Customer	Provider
Data	Customer	Customer	Provider
Runtime	Customer	Provider	Provider
Middleware	Customer	Provider	Provider
OS	Customer	Provider	Provider
Virtualization	Provider	Provider	Provider
Servers	Provider	Provider	Provider
Storage	Provider	Provider	Provider
Networking	Provider	Provider	Provider

Service Model	Target User	Examples	Billing
IaaS	System administrators, DevOps	AWS EC2, Azure VMs, GCP Compute	Per hour/second
PaaS	Developers	Google App Engine, Heroku, Azure App Service	Per app/resource
SaaS	End users	Gmail, Salesforce, Office 365, Zoom	Per user/month
FaaS (Serverless)	Developers	AWS Lambda, Azure Functions	Per invocation

DBaaS	Developers, DBAs	AWS RDS, MongoDB Atlas	Per query/storage

1.7 Cloud Infrastructure: Compute and Storage Design

1.7.1 Compute Cloud Architecture

• **Hypervisor Layer:** Software abstraction enabling multiple VMs on a physical host (KVM, Xen, VMware ESXi).

• **Instance Types:** General Purpose (balanced CPU/RAM), Compute Optimized (high CPU), Memory Optimized (large RAM), Storage Optimized (high I/O), GPU Instances (ML/rendering).

• **Auto-Scaling Groups:** Automatically add/remove instances based on CloudWatch metrics or custom triggers.

• **Load Balancer:** Distributes traffic across instances. Types: Application LB (Layer 7), Network LB (Layer 4), Classic LB. Enables high availability and fault tolerance.

• **Placement Groups:** Control physical placement of instances for low latency (cluster) or high availability (spread).

1.7.2 Storage Cloud Architecture

Storage Type	Characteristics	Access Method	Example
Object Storage	Flat namespace, metadata-rich, unlimited scale	REST API (S3 API)	AWS S3, Azure Blob
Block Storage	Low-latency, attached to single instance like a hard drive	iSCSI, Fibre Channel	AWS EBS, Azure Disk
File Storage	Shared filesystem, multiple instances mount simultaneously	NFS, SMB	AWS EFS, Azure Files
Archive Storage	Very low cost, high retrieval latency, long-term retention	Async retrieval	AWS Glacier, Azure Archive

1.7.3 Design Challenges in Cloud Architecture

Challenge	Description	Mitigation Strategy
Scalability	System must handle growing workload without redesign	Microservices, horizontal scaling, auto-scaling
Availability	Services must remain accessible despite failures	Multi-AZ, replication, failover automation

Performance	Minimize latency and maximize throughput	CDN, caching, edge computing, load balancing
Security	Protect data across shared multi-tenant infrastructure	Encryption, IAM, network segmentation, WAF
Cost Management	Cloud costs can spiral unexpectedly	Tagging, budgets, reserved instances, rightsizing
Vendor Lock-in	Dependency on proprietary services reduces portability	Open standards, multi-cloud, Terraform
Compliance	Regulatory requirements may restrict data location	Data residency controls, compliance certifications
Latency	Geographic distance introduces network delays	Edge locations, regional deployments

1.8 Case Study: Real-World Cloud Architecture

Netflix on AWS: Netflix serves 230+ million subscribers using AWS across 3 regions with 30+ availability zones. Their architecture uses microservices (700+), S3 for content storage, EC2 Auto Scaling for transcoding, CloudFront CDN for content delivery, DynamoDB for user session data, and Chaos Engineering (Chaos Monkey) to validate fault tolerance. This represents a real-world implementation of cloud architecture principles.

1.9 Unit Summary

Summary

Cloud computing evolved from mainframes through distributed computing to today's elastic, on-demand utility model. NIST defines it with 5 characteristics, 3 service models (IaaS/PaaS/SaaS), and 4 deployment models (Public/Private/Community/Hybrid). The reference architecture involves 5 actors. Infrastructure design addresses compute and storage with challenges in scalability, availability, cost, security, and compliance.

Review Questions

1. Define cloud computing according to NIST. Explain each of the five essential characteristics with examples.
2. Compare Grid Computing and Cloud Computing across six dimensions including architecture, business model, and QoS.
3. Explain the four cloud deployment models. Give real-world scenarios where each model is most appropriate.
4. Describe IaaS, PaaS, and SaaS with the shared responsibility breakdown for each layer.
5. What are the five actors in the NIST Cloud Reference Architecture? Explain the role of Cloud Broker with an example.

6. Discuss the key design challenges in building compute and storage clouds. How does auto-scaling address scalability?

7. Compare Object Storage, Block Storage, and File Storage. When would you use each in a cloud application?

8. Trace the evolution of cloud computing from mainframe time-sharing to edge computing.

UNIT II

VIRTUALIZATION BASICS

VM Basics · VM Taxonomy · Hypervisor Types · Key Concepts · Virtualization Structure · Implementation Levels · Full Virtualization · Paravirtualization · Hardware-Assisted · CPU/Memory/I/O

2.1 Introduction to Virtualization

Virtualization is the foundational technology that makes cloud computing possible. It enables the abstraction of physical hardware resources into multiple independent virtual instances, allowing greater utilization, flexibility, and isolation.

Before virtualization, servers ran a single operating system and typically operated at only 10–15% of their hardware capacity. Virtualization allows multiple workloads to share the same physical hardware, dramatically improving utilization rates to 70–80% or more.

Definition

Virtualization: A technique for hiding the physical characteristics of computing resources from the way in which other systems, applications, or end users interact with those resources. It creates a software-based abstraction layer between hardware and software.

2.1.1 Benefits of Virtualization

Benefit	Description	Business Impact
Server Consolidation	Multiple VMs on one physical server	Reduce hardware count by 10:1 or more
Hardware Isolation	Failures in one VM don't affect others	Improved stability and reliability
Portability	VMs can be moved between hosts	Flexible workload placement
Snapshot/Backup	VM state captured at any point	Easy backup, fast recovery
Test Environments	Isolated sandbox environments	Faster development cycles
Legacy Support	Run old OS versions alongside modern	Preserve legacy investments
Disaster Recovery	VM images replicated offsite	Faster RTO/RPO
Green IT	Fewer physical servers = less power	Reduced carbon footprint

2.2 Virtual Machine Basics

A Virtual Machine (VM) is a software emulation of a physical computer. It executes programs like a physical machine and includes a complete operating system with its own virtual hardware: CPU, memory, storage, and network interfaces.

2.2.1 Components of a Virtual Machine

- **Virtual CPU (vCPU):** Logical CPU allocated to the VM. Multiple vCPUs can be assigned; hypervisor maps these to physical CPU cores.
- **Virtual Memory (vRAM):** Portion of host physical memory allocated to the VM. Appears as full RAM to the guest OS.
- **Virtual Disk:** A file on the host filesystem (e.g., .vmdk, .vhd, .qcow2) appearing as a physical disk to the guest.
- **Virtual Network Interface Card (vNIC):** Software-defined NIC connected to a virtual switch for network connectivity.
- **Virtual BIOS/UEFI:** Firmware interface initialized when VM boots, configuring boot order and hardware.
- **Virtual Peripherals:** Emulated devices such as CD/DVD drives, USB controllers, and display adapters.

2.3 Taxonomy of Virtual Machines

VM Category	Description	Examples	Use Case
System VM (Full VM)	Provides complete platform for OS execution	VMware, VirtualBox, KVM	Server consolidation, cloud computing
Process VM	Executes a single process in platform-independent environment	JVM (Java), .NET CLR	Cross-platform app execution
Language VM	Interprets/executes high-level language code	CPython, Ruby MRI	Scripting language runtime
OS-Level VM (Container)	Multiple isolated user-space on single OS kernel	Docker, LXC, Podman	Microservices, DevOps
Application VM	Application isolated from underlying OS	App-V, ThinApp	Software distribution

2.4 Hypervisor (Virtual Machine Monitor — VMM)

The hypervisor is the core software layer enabling virtualization. It creates and manages virtual machines, controlling their access to physical hardware while maintaining isolation between them.

Without the hypervisor, virtual machines cannot exist.

2.4.1 Type 1 Hypervisor — Bare-Metal

Type 1 hypervisors run directly on the host's physical hardware without an underlying operating system. They are installed directly on the server hardware and are the first software layer above the hardware. This gives them direct control over hardware resources, resulting in excellent performance and strong security isolation.

Feature	Details
Architecture	Runs directly on hardware; no host OS dependency
Performance	Near-native; minimal overhead
Security	Strong VM isolation; smaller attack surface
Management	Remote management console (vSphere, Hyper-V Manager)
Use Case	Enterprise data centers, cloud provider infrastructure
Examples	VMware ESXi, Microsoft Hyper-V, Xen, KVM (Linux kernel module)

2.4.2 Type 2 Hypervisor — Hosted

Type 2 hypervisors run as an application on a conventional host operating system. The host OS handles hardware access, and the hypervisor relies on the host OS for resource management. This makes them easier to install and use but introduces additional overhead.

Feature	Details
Architecture	Application running on host OS (Windows, macOS, Linux)
Performance	Higher overhead due to host OS layer
Security	Depends on host OS security; larger attack surface
Management	Standard desktop application interface
Use Case	Developer workstations, testing, education
Examples	VMware Workstation, Oracle VirtualBox, Parallels Desktop, QEMU

Feature	Type 1 (Bare-Metal)	Type 2 (Hosted)
Runs On	Hardware directly	Host operating system

Performance	High (near-native)	Lower (host OS overhead)
Security Isolation	Excellent	Moderate
Installation	Complex, hardware-specific	Simple, like any app
Primary Use	Production, cloud	Development, testing
Examples	VMware ESXi, KVM, Xen	VirtualBox, VMware Workstation

2.5 Key Concepts in Virtualization

Term	Definition
Host Machine	Physical computer running the hypervisor software.
Guest Machine	Virtual machine running on top of the hypervisor.
vCPU	Virtual CPU — logical processing unit assigned to a guest VM.
vRAM	Virtual RAM — portion of host physical memory allocated to a guest.
Live Migration	Moving a running VM between physical hosts with zero or minimal downtime.
Cold Migration	Moving a powered-off VM between hosts — simpler but requires downtime.
Snapshot	Point-in-time capture of VM disk, memory, and settings for backup/rollback.
Template	Master VM image used to rapidly deploy identical VMs.
Balloon Driver	Guest OS driver that inflates/deflates to manage memory dynamically.
Thin Provisioning	Allocating virtual storage larger than physical, growing on demand.
Overcommitment	Allocating more virtual resources than physically available (CPU/RAM).
VM Density	Number of VMs running on a single physical host.

2.6 Virtualization Structure

The virtualization stack has distinct layers. Understanding this layered structure is critical for troubleshooting, performance tuning, and security analysis.

Layer	Components	Managed By
Application Layer	User applications (databases, web servers, etc.)	User/Developer
Guest OS Layer	Windows, Linux, etc. running inside VMs	VM Owner
Hypervisor/VMM Layer	Resource allocation, VM lifecycle, isolation enforcement	Cloud/IT Admin
Hardware Abstraction	Virtual devices (vNIC, vDisk, vCPU)	Hypervisor
Physical Hardware	Physical CPU, RAM, NIC, Storage	Data Center/Provider

2.7 Implementation Levels of Virtualization

Level	Where Applied	Technique	Example
Instruction Set Architecture (ISA)	CPU instruction set	Full emulation	QEMU emulating ARM on x86
Hardware Abstraction Layer (HAL)	Hardware interface level	Binary translation or HW assist	VMware ESXi, Xen
OS Level	Within a single OS kernel	Namespaces, cgroups	Docker, LXC containers
Library / API Level	System call / library API	API interception, wrapping	Wine (Windows API on Linux)
Application Level	Programming language runtime	Bytecode interpretation/JIT	JVM, .NET CLR

2.8 Types of Virtualization

2.8.1 Full Virtualization

In full virtualization, the VMM completely simulates the underlying hardware, enabling unmodified guest operating systems to run without any changes. The hypervisor intercepts all privileged instructions from the guest using binary translation.

Binary translation works by scanning guest code blocks and replacing sensitive instructions with safe equivalent code that achieves the same effect without actually executing privileged operations.

Translated blocks are cached in a translation cache for reuse.

Characteristic	Details
Guest OS Modification	Not required — runs unmodified
Performance	Slightly reduced due to binary translation overhead
Isolation	Complete — guest unaware of virtualization
Portability	High — any OS can be virtualized
Examples	VMware Workstation, early ESXi, VirtualBox

2.8.2 Paravirtualization

Paravirtualization modifies the guest OS to cooperate with the hypervisor, replacing privileged instructions with explicit hypercalls — direct calls to the VMM. This eliminates the overhead of binary translation at the cost of requiring OS modification.

- **Hypercalls:** Guest OS makes explicit calls to the hypervisor API instead of executing privileged hardware instructions.
- **Virtio:** A standard interface for paravirtualized I/O drivers (network, block) widely used in KVM.
- **Performance:** Closer to native performance than full virtualization for I/O-intensive workloads.
- **Limitation:** Cannot be used with unmodified proprietary OS (e.g., unmodified Windows).
- **Examples:** Xen (original paravirtualization), KVM with virtio drivers.

2.8.3 Hardware-Assisted Virtualization

Modern processors include hardware extensions specifically designed to support virtualization. These extensions introduce new CPU modes that allow the hypervisor to run guest OS in a special mode with direct hardware access for non-privileged instructions, while automatically trapping to the hypervisor for privileged operations.

Feature	Intel VT-x	AMD-V
Technology	Intel Virtualization Technology for IA-32/64	AMD Virtualization
CPU Modes	VMX root (hypervisor) / VMX non-root (guest)	Host mode / Guest mode
I/O Virtualization	Intel VT-d (IOMMU)	AMD-Vi (IOMMU)
Memory	Extended Page Tables (EPT)	Nested Page Tables (NPT)
Performance	Near-native for most workloads	Near-native for most workloads

2.9 CPU Virtualization

CPU virtualization is perhaps the most complex aspect of virtualization because it must handle the x86 architecture's privilege ring model while maintaining isolation and performance.

2.9.1 x86 Ring Architecture

Ring	Privilege Level	Normal Use	Virtualization Use
Ring -1	Highest (VMX root)	Not used traditionally	Hypervisor (with HW-assist)
Ring 0	Kernel mode	OS kernel	Guest OS (moved here with HW-assist, Ring 1 in software)
Ring 1	—	Device drivers (some OS)	Guest OS (full/paravirt without HW-assist)
Ring 3	User mode	Applications	Applications (unchanged)

2.9.2 vCPU Scheduling

- **Credit-based Scheduling (Xen):** Each vCPU gets a credit allocation; used credits consumed during execution; ensures fair sharing.
- **Proportional Share Scheduling:** vCPUs receive CPU time proportional to their allocated shares.
- **NUMA Awareness:** Scheduler places vCPUs on physical cores close to the memory they access (Non-Uniform Memory Access).
- **CPU Pinning:** Binding specific vCPUs to specific physical cores for predictable, low-latency performance.

2.10 Memory Virtualization

Technique	How It Works	Performance Impact
Shadow Page Tables	VMM maintains shadow tables mapping guest virtual to host physical addresses	High overhead; many TLB flushes
Extended Page Tables (EPT/NPT)	Hardware walks two-level page table (guest + host) automatically	Low overhead; near-native
Memory Ballooning	Guest balloon driver reclaims unused memory back to hypervisor	Minimal; requires cooperation
Transparent Page Sharing	Identical pages across VMs deduplicated into one shared copy (CoW)	Memory savings at slight CPU cost

Memory Swapping	VMM swaps out VM pages to disk when physical RAM exhausted	High overhead; avoid if possible
Large Pages (Huge Pages)	2MB pages instead of 4KB reduce TLB misses for large VMs	Significant performance boost

2.11 I/O Virtualization

Method	Description	Performance	Use Case
Full Emulation	Hypervisor emulates physical device in software	Low	Simple/legacy devices
Para-virtualized (virtio)	Guest uses special drivers to call hypervisor APIs directly	High	Network, storage in KVM/Xen
Device Passthrough	Physical device dedicated exclusively to one VM	Near-native	GPU compute, high-freq trading
SR-IOV	NIC presents multiple virtual functions (VFs) assignable to VMs	Near-native	High-performance networking
DPDK	Data Plane Development Kit bypasses kernel for network I/O	Extreme	Telecom, NFV workloads

2.12 Unit Summary

Summary

Virtualization abstracts physical hardware using a hypervisor (Type 1 bare-metal or Type 2 hosted). VMs have full virtual hardware (vCPU, vRAM, vDisk, vNIC). Virtualization types: Full (no OS modification, binary translation), Paravirtualization (modified OS, hypercalls), Hardware-Assisted (Intel VT-x/AMD-V). CPU virtualization handles ring architecture; memory uses EPT/shadow tables/ballooning; I/O uses emulation, virtio paravirtualization, or SR-IOV passthrough.

Review Questions

1. What is virtualization? Explain the benefits of virtualization for enterprise data centers.

2. Distinguish between Type 1 and Type 2 hypervisors with architecture diagrams described in words.

3. Describe the five levels of virtualization implementation with one example at each level.

4. Compare Full Virtualization, Paravirtualization, and Hardware-Assisted Virtualization across performance, OS modification, and examples.

5. Explain binary translation in full virtualization. Why was it needed and why is it less common now?

6. What is the x86 ring architecture? How does it create a challenge for CPU virtualization?

7. Describe four memory virtualization techniques. Which provides the best performance and why?

8. Compare full I/O emulation, virtio paravirtualization, and SR-IOV in terms of performance and use case.

UNIT III

VIRTUALIZATION INFRASTRUCTURE AND DOCKER

Desktop · Network · Storage Virtualization · OS-Level Virtualization · App Virtualization · Virtual Clusters · Containers vs VMs · Docker Architecture · Components · Images & Registries

3.1 Desktop Virtualization

Desktop virtualization separates the desktop environment and its applications from the physical client device. The desktop runs in a centralized data center or cloud, and users access it remotely. This approach centralizes management, improves security, and enables BYOD (Bring Your Own Device) policies.

Type	Description	Advantages	Limitations
VDI (Virtual Desktop Infrastructure)	Desktop VMs hosted on central servers, accessed by thin clients	Central management, strong security	Bandwidth dependent, latency sensitive
Remote Desktop Services (RDS)	Multiple users share one server OS — only screen/input transmitted	Cost-effective for many users	App conflicts, limited customization
Application Streaming	App delivered on-demand, runs locally	Offline capable, fast startup	Complex packaging
Client Hypervisor	Hypervisor on client device runs local VMs	Offline, secure isolation	Client hardware requirements
Desktop-as-a-Service (DaaS)	Cloud-hosted VDI managed by provider	No infrastructure to manage	Ongoing subscription cost

3.2 Network Virtualization

Network virtualization abstracts physical network infrastructure into multiple logical networks, each isolated from the others. It is essential for creating secure, flexible cloud network topologies without physical reconfiguration.

3.2.1 VLAN (Virtual Local Area Network)

VLANs logically segment a physical network switch into multiple isolated broadcast domains. Traffic between VLANs requires routing (Layer 3), providing security isolation. Defined by IEEE 802.1Q standard; uses VLAN ID tags in Ethernet frames.

3.2.2 Software Defined Networking (SDN)

SDN decouples the network control plane (deciding where traffic goes) from the data plane (actual packet forwarding). A centralized SDN Controller manages the entire network programmatically, enabling dynamic, automated network configuration.

SDN Layer	Function	Technology
Application Layer	Business applications, network policies	REST APIs, Intent-based networking
Control Layer	SDN Controller — centralized intelligence	OpenFlow, OpenDaylight, ONOS
Infrastructure Layer	Physical/virtual switches, routers — dumb forwarding devices	OpenFlow-enabled switches, OVS

3.2.3 Network Functions Virtualization (NFV)

NFV replaces dedicated network hardware appliances with software running on standard x86 servers. Traditional hardware firewalls, load balancers, and routers are replaced by Virtual Network Functions (VNFs).

- **Benefits:** Reduced hardware cost, faster deployment, easy scaling, vendor independence.
- **VNF Examples:** Virtual firewall, virtual router, virtual load balancer, virtual WAN optimizer.
- **NFV Infrastructure (NFVI):** Compute, storage, and networking hardware supporting VNFs.
- **Management (MANO):** NFV Orchestrator manages VNF lifecycle, resource allocation.

3.2.4 Virtual Switch (vSwitch)

A virtual switch operates within the hypervisor to forward traffic between VMs and between VMs and physical NICs. It supports VLANs, QoS, port mirroring, and flow control.

- **VMware vSphere Standard Switch (VSS):** Basic vSwitch per ESXi host; not centrally managed.
- **VMware Distributed Switch (VDS):** Centrally managed across multiple ESXi hosts via vCenter.
- **Open vSwitch (OVS):** Open-source production-quality virtual switch supporting OpenFlow; used in KVM, OpenStack.
- **Linux Bridge:** Simple kernel-level bridge for basic VM networking.

3.3 Storage Virtualization

Storage virtualization abstracts physical storage into a unified logical resource pool, hiding the complexity of underlying storage hardware from applications and administrators.

Concept	Description

SAN (Storage Area Network)	High-speed dedicated network (Fibre Channel or iSCSI) connecting servers to shared storage arrays. Block-level access.
NAS (Network Attached Storage)	File-level storage accessed over standard TCP/IP network (NFS, SMB/CIFS). Shared filesystem.
SDS (Software-Defined Storage)	Storage managed through software abstraction layer, hardware-independent. Examples: Ceph, VMware vSAN.
Thin Provisioning	Allocate large virtual storage volumes that consume only actual used physical space.
Storage Tiering	Automatically move data between fast (SSD) and slow (HDD) storage based on access frequency.
Data Deduplication	Eliminate redundant copies of data to reduce storage consumption.
Compression	Reduce data size before storing to save physical space.

3.4 OS-Level Virtualization

OS-level virtualization creates multiple isolated user-space environments (containers) on a single OS kernel. Unlike full VMs, containers share the host kernel, making them significantly more lightweight and faster to start.

3.4.1 Linux Kernel Features Enabling Containers

Feature	Purpose	Details
Namespaces	Process isolation	PID, Network, Mount, UTS (hostname), IPC, User namespaces
cgroups (Control Groups)	Resource limitation	Limit CPU, memory, I/O, network per process group
seccomp	System call filtering	Restrict which system calls a container can make
AppArmor / SELinux	Mandatory access control	Define allowed file and network access for processes
Capabilities	Fine-grained privileges	Break root into specific capabilities; grant only what's needed

OverlayFS	Layered filesystem	Union mount for container image layers

3.5 Application Virtualization

• **Concept:** Applications packaged with all dependencies into a self-contained virtual environment, eliminating conflicts with other applications or the OS.

• **VMware ThinApp:** Captures application and dependencies into a single portable executable (.exe) that runs without installation.

• **Microsoft App-V:** Streams applications on-demand from central server; runs in isolated virtual environment.

• **AppImage (Linux):** Self-contained executable bundle for Linux applications — no installation required.

• **Snap/Flatpak:** Modern Linux application sandboxing with dependency bundling and update management.

• **Benefits:** No DLL conflicts, easy deployment, instant rollback, side-by-side versioning.

3.6 Virtual Clusters and Resource Management

A virtual cluster pools VM resources across multiple physical hosts, providing the appearance of a unified cluster. Advanced resource management features ensure optimal performance and high availability.

Feature	Description	Benefit
Live Migration (vMotion)	Move running VMs between hosts without shutdown	Zero-downtime maintenance, load balancing
DRS (Distributed Resource Scheduler)	Automatically balance VM workloads across cluster hosts	Optimal resource utilization
High Availability (HA)	Automatically restart VMs on surviving hosts when a host fails	Fast recovery from hardware failure
Fault Tolerance (FT)	Shadow VM kept in sync with primary; zero downtime if primary fails	Zero RPO/RTO for critical VMs
Storage DRS	Balance VM disk I/O and capacity across datastores	Avoid storage hotspots
Resource Pools	Hierarchical grouping of VMs with guaranteed/reserved resources	Fair sharing and priority

3.7 Containers vs. Virtual Machines — Detailed Comparison

Aspect	Virtual Machines	Containers
Isolation Level	Hardware-level — separate OS kernel per VM	Process-level — shared OS kernel
Startup Time	1–5 minutes (OS boot)	Milliseconds to seconds
Image Size	Gigabytes (full OS + apps)	Megabytes (app + dependencies only)
Performance	Near-native (with hardware assist)	Near-native (no OS overhead)
Portability	Less portable (OS/driver dependencies)	Highly portable across any Linux host
Security Isolation	Strong (separate kernel)	Weaker (shared kernel; escape risks)
Resource Overhead	High (separate OS per VM)	Very low (shared kernel, no OS duplication)
Management Tools	vSphere, Hyper-V, KVM/libvirt	Docker, Kubernetes, Podman
Use Case	Full OS workloads, legacy apps, diverse OS	Microservices, DevOps, CI/CD pipelines
Persistence	Persistent by nature	Ephemeral by design; external volumes for persistence

Choosing Between VMs and Containers: Use VMs when you need strong isolation, multiple OS types, or legacy application support. Use containers for microservices, rapid deployment, CI/CD pipelines, and when startup time and density matter. Modern architectures often use both together — containers running inside VMs for security and flexibility.

3.8 Introduction to Docker

Docker is an open platform for developing, shipping, and running applications. Docker enables separating applications from infrastructure so that software can be delivered quickly. With Docker, you can manage infrastructure the same way you manage applications.

Docker was released in 2013 by dotCloud (later Docker, Inc.) and quickly became the de facto standard for containerization. It uses Linux kernel features (namespaces, cgroups) but packages them in an easy-to-use interface with a powerful ecosystem.

3.9 Docker Architecture

Docker uses a client-server architecture. The Docker client communicates with the Docker daemon using REST API, which builds, runs, and manages containers.

Component	Description	Communication
Docker Client (CLI)	User interface — runs docker commands	REST API over Unix socket or TCP
Docker Daemon (dockerd)	Background service managing all Docker objects	Listens on REST API; manages containers, images, networks, volumes
containerd	Container runtime handling container lifecycle	dockerd uses containerd internally
runc	Low-level container runner (OCI-compliant)	containerd delegates to runc for container creation
Docker Registry	Stores and distributes Docker images	HTTP push/pull (Docker Registry API v2)

3.10 Docker Components in Detail

3.10.1 Dockerfile

A Dockerfile is a text document containing instructions to build a Docker image. Each instruction creates a new read-only layer in the image filesystem.

Instruction	Purpose	Example
FROM	Specify base image	FROM ubuntu:22.04
RUN	Execute command during build	RUN apt-get install -y python3
COPY / ADD	Copy files into image	COPY app.py /app/
WORKDIR	Set working directory	WORKDIR /app
ENV	Set environment variable	ENV PORT=8080
EXPOSE	Document port the container listens on	EXPOSE 8080
CMD	Default command when container starts	CMD ["python3", "app.py"]
ENTRYPOINT	Fixed executable for the container	ENTRYPOINT ["nginx", "-g"]
VOLUME	Declare mount point for external volumes	VOLUME ["/data"]
USER	Switch to non-root user	USER appuser

Example Dockerfile for a Python Flask Application:

```
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
ENV FLASK_APP=app.py
EXPOSE 5000
USER nobody
CMD ["flask", "run", "--host=0.0.0.0"]
```

3.10.2 Docker Images

• **Layered Architecture:** Images consist of read-only layers. Each Dockerfile instruction adds a layer. Layers are cached — rebuilds only recreate changed layers and above.

• **Image ID:** Cryptographic SHA256 hash uniquely identifying each image.

• **Tags:** Human-readable labels (nginx:latest, nginx:1.25.3). 'latest' is default but not recommended for production.

• **Base Images:** Minimal starting points — scratch (empty), alpine (5MB), ubuntu (72MB), debian-slim.

• **Multi-stage Builds:** Use multiple FROM instructions; copy only artifacts from build stage to final image, dramatically reducing image size.

3.10.3 Docker Containers

Command	Purpose	Example
docker run	Create and start container	docker run -d -p 80:80 nginx
docker ps	List running containers	docker ps -a (all containers)
docker stop	Stop a running container	docker stop mycontainer
docker rm	Remove stopped container	docker rm mycontainer
docker exec	Run command in running container	docker exec -it mycontainer bash
docker logs	View container output logs	docker logs -f mycontainer
docker inspect	Show detailed container info	docker inspect mycontainer
docker stats	Real-time resource usage	docker stats mycontainer

3.10.4 Docker Volumes

Docker volumes provide persistent storage that outlives container restarts and removals. There are three types:

Volume Type	Description	Use Case
Named Volume	Docker-managed volume stored in /var/lib/docker/volumes	Databases, persistent app data

Bind Mount	Host directory/file mounted into container	Development (live code reload), config files
tmpfs Mount	Stored only in host memory — never persisted	Sensitive data, temporary scratch space

3.10.5 Docker Networks

Network Type	Description	Use Case
Bridge (default)	NAT-based isolation; containers communicate by name	Standalone apps, local development
Host	Container shares host network namespace — no isolation	Performance-critical apps
Overlay	Multi-host networking for Docker Swarm clusters	Distributed microservices
Macvlan	Container gets its own MAC address, appears as physical device	Legacy apps expecting physical network
None	No networking — completely isolated	Batch jobs, security-critical containers

3.11 Docker Images and Registries

• **Docker Hub:** Official public registry at hub.docker.com. 8+ million images. Official images curated by Docker and vendors.

• **Private Registries:** Self-hosted (Harbor, Docker Registry) or cloud-managed (AWS ECR, Azure ACR, Google GCR).

• **Image Naming:** [registry]/[namespace]/[repository]:[tag] — e.g., registry.example.com/myteam/myapp:v1.2.3

• **docker pull:** Download image from registry to local daemon.

• **docker push:** Upload local image to registry.

• **docker build:** Build image from Dockerfile.

• **Image Scanning:** Security scanning of images for vulnerabilities (Trivy, Snyk, Docker Scout).

• **Content Trust:** Sign images with Docker Content Trust (DCT) to verify authenticity.

3.12 Unit Summary

Summary

Virtualization infrastructure spans desktop (VDI, RDS), network (VLAN, SDN, NFV, vSwitch), and storage (SAN, NAS, SDS) domains. OS-level virtualization uses Linux namespaces and cgroups. Containers are lighter, faster, and more portable than VMs but offer weaker isolation. Docker provides a complete ecosystem: Dockerfile for image definition, layered images, named/bind/tmpfs volumes, bridge/overlay/host networks, and public/private registries.

Review Questions

1. What is VDI? Compare VDI with Remote Desktop Services (RDS) for 500 concurrent users.
2. Explain SDN architecture with its three layers. How does it differ from traditional networking?
3. Describe the Linux kernel features (namespaces, cgroups, seccomp) that enable containers.
4. Compare Containers and VMs across 8 dimensions. When would a production system use both together?
5. Explain Docker's layered image architecture. What is a multi-stage build and why is it beneficial?
6. Describe the five Docker network types with use cases for each.
7. Write a Dockerfile for a Node.js Express application. Explain each instruction.
8. What are Docker volumes? Compare named volumes, bind mounts, and tmpfs mounts.

UNIT IV

CLOUD DEPLOYMENT ENVIRONMENT

Google App Engine · Amazon AWS · Microsoft Azure · Cloud Software Environments · Eucalyptus · OpenStack · Multi-Cloud Strategy

4.1 Introduction to Cloud Platforms

Cloud deployment environments are the platforms, services, and software stacks through which cloud computing is realized. They range from public cloud giants offering hundreds of managed services to open-source platforms enabling private cloud deployments.

Platform	Type	Founded	Market Position
Amazon AWS	Public Cloud	2006	Market leader (~33% share)
Microsoft Azure	Public/Hybrid Cloud	2010	#2 (~22% share)
Google Cloud (GCP)	Public Cloud	2008	#3 (~11% share)
Google App Engine	PaaS (within GCP)	2008	Early PaaS pioneer
Eucalyptus	Open-Source Private Cloud	2008	AWS-compatible private cloud
OpenStack	Open-Source Cloud Platform	2010	Leading open-source private cloud

4.2 Google App Engine (GAE)

Google App Engine was one of the earliest PaaS offerings, launched in 2008. It allows developers to build and deploy web applications on Google's infrastructure without managing servers. The platform automatically handles scaling, load balancing, and infrastructure management.

4.2.1 GAE Environments

Feature	Standard Environment	Flexible Environment
Runtime	Sandboxed (Python, Java, Go, PHP, Node.js, Ruby)	Docker container (any language)
Scaling	Very fast (instances start in ms)	Slower (Docker startup)
Scaling to Zero	Yes — scale to 0 instances when idle	No — minimum 1 instance

Filesystem Access	Limited (read-only, no local disk)	Full (writable local disk)
Background Processes	No	Yes
Cost	More economical at low traffic	More predictable at sustained traffic
Use Case	Stateless web apps, APIs	Apps needing custom runtimes, background tasks

4.2.2 GAE Architecture

Component	Description
Application	Top-level container; associated with Google Cloud project; has global App Engine settings
Service	Independent component of app with own scaling config (previously called 'module')
Version	Code snapshot of a service; multiple versions can coexist; traffic splitting between versions
Instance	Running copy of a version; automatically created/deleted based on traffic
app.yaml	Configuration file defining runtime, scaling, environment variables, URL routing

4.2.3 GAE Built-in Services

- **Cloud Datastore/Firestore:** NoSQL document database natively integrated with GAE.
- **Memcache:** Distributed in-memory cache for session and computed data.
- **Task Queues:** Deferred execution of tasks outside request-response cycle.
- **BLOB Store:** Storage for large files (images, videos) separate from Datastore.
- **Search API:** Full-text search over structured data.
- **Mail API:** Send and receive email from applications.

4.3 Amazon Web Services (AWS)

Amazon Web Services (AWS) is the world's most comprehensive and widely adopted cloud platform. Launched in 2006 with SQS and S3, it pioneered commercial cloud computing and today offers over 200 fully featured services across compute, storage, database, networking, analytics, AI/ML, IoT, security, and more.

4.3.1 AWS Global Infrastructure

Infrastructure Component	Description	Count (approx.)

Regions	Geographically isolated locations; data sovereignty boundary	33 regions worldwide
Availability Zones (AZs)	Isolated data centers within a region; connected by low-latency links	105+ AZs
Edge Locations	Endpoints for CloudFront CDN and Route 53; content caching	400+ locations
Local Zones	AWS infrastructure in major metro areas for ultra-low latency	30+ locations
Wavelength Zones	AWS compute embedded at 5G network edge for mobile applications	Limited (carrier-specific)
Outposts	AWS hardware installed on-premises for hybrid scenarios	On-premises deployment

4.3.2 Core AWS Services

Category	Service	Key Feature
Compute	EC2 (Elastic Compute Cloud)	Virtual machines — 500+ instance types
Compute	Lambda	Serverless; run code without provisioning servers; pay per invocation
Compute	ECS / EKS	Managed Docker containers / Managed Kubernetes
Compute	Elastic Beanstalk	PaaS for web apps; auto-handles infrastructure
Storage	S3 (Simple Storage Service)	Object storage; 11 nines durability; unlimited scale
Storage	EBS (Elastic Block Store)	SSD/HDD block volumes for EC2
Storage	EFS (Elastic File System)	Managed NFS shared across multiple EC2 instances
Storage	Glacier / S3 Glacier	Ultra-low-cost archive storage

Database	RDS	Managed relational DB: MySQL, PostgreSQL, Oracle, SQL Server
Database	Aurora	AWS-designed MySQL/PostgreSQL-compatible; 5x MySQL performance
Database	DynamoDB	Serverless NoSQL; single-digit millisecond latency; auto-scaling
Networking	VPC (Virtual Private Cloud)	Isolated virtual network; subnets, route tables, security groups
Networking	CloudFront	Global CDN with edge caching
Networking	Route 53	DNS with health checking, traffic routing policies
Security	IAM	Identity and access management for all AWS services
Security	AWS WAF	Web Application Firewall against common exploits
AI/ML	SageMaker	End-to-end ML platform: build, train, deploy models

4.3.3 AWS Well-Architected Framework

AWS defines five pillars of a well-architected cloud system:

Pillar	Key Questions	Core Principles
Operational Excellence	Can you run and monitor systems to deliver business value?	IaC, frequent small changes, anticipate failure
Security	How do you protect data and systems?	Defense in depth, least privilege, encryption, audit
Reliability	How does system recover from failure?	Multi-AZ, auto-scaling, backup, chaos testing
Performance Efficiency	How do you use computing efficiently?	Right instance types, serverless, caching, global

Cost Optimization	How do you avoid unnecessary costs?	Right-sizing, reserved instances, spot, monitoring
Sustainability	How do you minimize environmental impact?	Efficient code, serverless, managed services

4.4 Microsoft Azure

Microsoft Azure was launched in 2010 and has grown into the second-largest cloud provider. Azure is particularly strong in hybrid cloud scenarios, enterprise Microsoft stack integration, and government/regulated industries. It spans 60+ regions globally.

4.4.1 Core Azure Services

Category	Azure Service	Description
Compute	Azure Virtual Machines	IaaS VMs; Windows and Linux; 700+ instance sizes
Compute	Azure App Service	PaaS for web apps; managed runtime and scaling
Compute	Azure Functions	Serverless; event-driven execution
Compute	AKS (Azure Kubernetes Service)	Managed Kubernetes cluster
Storage	Azure Blob Storage	Object storage for unstructured data
Storage	Azure Files	Managed file shares; SMB and NFS protocols
Storage	Azure Disk Storage	Managed block storage for Azure VMs
Database	Azure SQL Database	Managed SQL Server as a service
Database	Azure Cosmos DB	Globally distributed multi-model NoSQL
Networking	Azure Virtual Network (VNet)	Isolated private network in Azure
Networking	Azure CDN	Content Delivery Network
Identity	Azure Active Directory (AAD)	Cloud identity; SSO, MFA, conditional access

Hybrid	Azure Arc	Manage on-premises servers, Kubernetes, and SQL with Azure
Security	Microsoft Defender for Cloud	Cloud security posture management and threat protection

4.4.2 Azure Unique Strengths

- **Hybrid Cloud Leadership:** Azure Arc and Azure Stack extend Azure management to any infrastructure.
- **Microsoft 365 Integration:** Native integration with Office 365, Teams, Power Platform, and Dynamics 365.
- **Active Directory:** Best-in-class enterprise identity with Azure AD (now Microsoft Entra ID).
- **Compliance Coverage:** 90+ compliance certifications — largest in the industry.
- **Government Cloud:** Azure Government and specialized sovereign cloud offerings.
- **.NET Ecosystem:** First-class support for .NET applications and Visual Studio integration.

4.5 Eucalyptus

Eucalyptus (Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems) is an open-source software platform for implementing private and hybrid clouds. Its key differentiator is full API compatibility with Amazon AWS, allowing the same tools and workflows to work against both Eucalyptus and AWS.

4.5.1 Eucalyptus Architecture

Component	Location	Function
Cloud Controller (CLC)	Front-end	User-facing entry point; manages resources globally; hosts EC2 and IAM APIs
Walrus	Front-end	S3-compatible object storage service
Cluster Controller (CC)	Mid-tier (per cluster)	Manages VM instances within a cluster; networking; schedules to NCs
Storage Controller (SC)	Mid-tier (per cluster)	EBS-compatible block storage; manages volumes and snapshots
Node Controller (NC)	Back-end (per physical node)	Runs on each host; manages VM lifecycle via hypervisor (KVM/Xen)

VMware Broker (optional)	Mid-tier	Connects Eucalyptus to existing VMware infrastructure

AWS Service	Eucalyptus Equivalent
EC2 (Compute)	Cloud Controller + Node Controller
S3 (Object Storage)	Walrus
EBS (Block Storage)	Storage Controller
VPC (Networking)	Cluster Controller networking
IAM (Identity)	Cloud Controller IAM API
ELB (Load Balancer)	Elastic Load Balancing (ELB) service

4.6 OpenStack

OpenStack is the most widely deployed open-source cloud platform globally. It was co-founded by NASA and Rackspace in 2010 and is now governed by the OpenInfra Foundation. OpenStack provides a comprehensive suite of software tools for building and managing public and private clouds, with a modular, API-first architecture.

4.6.1 OpenStack Core Components

Component	Code Name	Function	AWS Equivalent
Compute	Nova	VM lifecycle management; schedules instances to compute nodes	EC2
Object Storage	Swift	Distributed object storage; highly durable, eventually consistent	S3
Block Storage	Cinder	Block storage volumes for Nova instances; snapshots	EBS
Networking	Neutron	Networking as a service; VLANs, routers, floating IPs, security groups	VPC

Image Service	Glance	Registry for VM images; upload, catalog, deliver to Nova	EC2 AMI service
Identity	Keystone	Authentication (tokens), authorization, service catalog	IAM + STS
Dashboard	Horizon	Web-based UI for all OpenStack services	AWS Console
Orchestration	Heat	Template-based infrastructure provisioning (YAML/JSON HOT templates)	CloudFormation
Telemetry	Ceilometer	Metering and monitoring; collects resource usage data	CloudWatch
DNS	Designate	DNS as a service; manage zones and records	Route 53
Shared File System	Manila	NFS/CIFS shared filesystem service	EFS
Bare Metal	Ironic	Provision physical servers alongside VMs	EC2 Bare Metal

4.6.2 OpenStack Request Flow Example (Launching a VM)

- 1. User authenticates with **Keystone** — receives a token.
- 2. User requests VM via **Horizon** dashboard or Nova CLI/API.
- 3. **Nova** scheduler selects the best compute node based on filters and weights.
- 4. Nova contacts **Glance** to retrieve the selected VM image.
- 5. Nova contacts **Cinder** to provision a boot volume (if block storage boot).
- 6. Nova contacts **Neutron** to create a virtual port and configure networking.
- 7. Nova's compute agent on the target node launches the VM via the hypervisor (KVM).
- 8. **Ceilometer** records metering events for billing and monitoring.

4.6.3 OpenStack vs AWS — Feature Comparison

Dimension	OpenStack	AWS
Ownership	Open-source; you own the infrastructure	Managed service; provider owns infrastructure

Control	Complete control over all layers	Control only within provider's abstractions
Cost Model	Infrastructure + staff cost; no per-use fee	Pay-per-use; no upfront infrastructure
Scalability	Limited by your hardware investment	Virtually unlimited; global infrastructure
Complexity	High; requires skilled cloud engineers	Low; managed services reduce operational burden
Vendor Lock-in	None — open standards throughout	Significant; proprietary services hard to migrate
Time to Deploy	Weeks/months for initial setup	Minutes; no setup required
Best For	Regulated industries, telcos, large enterprises with existing hardware	Startups, enterprises wanting speed and managed services

4.7 Multi-Cloud Strategy

Many organizations use services from multiple cloud providers simultaneously (multi-cloud) to avoid vendor lock-in, optimize costs, and leverage best-in-class services from each provider.

- **Benefits:** Avoid vendor lock-in, geographic flexibility, best-of-breed services, negotiating leverage.
- **Challenges:** Complexity, multiple skill sets required, cost visibility across providers, security consistency.
- **Tools:** Terraform (infrastructure as code), Kubernetes (portable container orchestration), Pulumi, Crossplane for managing multi-cloud deployments.
- **Common Pattern:** AWS for compute/storage, Azure for AD/identity, GCP for ML/BigQuery analytics.

4.8 Unit Summary

Summary

Cloud deployment environments span managed public clouds (AWS, Azure, GCP) and open-source private clouds (Eucalyptus, OpenStack). Google App Engine pioneered PaaS. AWS leads with 200+ services and the Well-Architected Framework. Azure excels in hybrid and enterprise scenarios. Eucalyptus provides AWS-compatible private cloud. OpenStack offers comprehensive open-source cloud infrastructure with modular components (Nova, Swift, Cinder, Neutron, Keystone, Glance, Heat). Multi-cloud strategies balance flexibility against complexity.

Review Questions

1. Compare GAE Standard and Flexible environments. When would you choose each for a Python web application?
2. Describe AWS global infrastructure: Regions, Availability Zones, and Edge Locations. Why does this design improve availability?
3. Explain the AWS Well-Architected Framework's six pillars with examples for each.
4. List and describe seven core AWS services across compute, storage, database, and networking categories.
5. Compare Microsoft Azure and AWS: architecture, services, strengths, and target markets.
6. Describe the five components of Eucalyptus. How does it achieve AWS API compatibility?
7. Explain the OpenStack VM launch request flow through all relevant components.
8. Compare OpenStack and AWS across ownership, control, cost, scalability, and vendor lock-in.

UNIT V

CLOUD SECURITY

Virtualization-Specific Attacks · Guest Hopping · VM Migration Attack · Hyper Jacking · Side-Channel Attacks · Data Security · IAM · IAM Challenges · IAM Architecture · Zero Trust

5.1 Introduction to Cloud Security

Cloud security encompasses the policies, controls, procedures, and technologies that protect cloud-based systems, data, and infrastructure. It is a shared responsibility: cloud providers secure the underlying infrastructure, while customers secure their applications, data, and access controls.

Security Domain	Provider Responsibility	Customer Responsibility
Physical Security	Data center, hardware, power, cooling	None
Network Infrastructure	Core networking, DDoS mitigation	VPC configuration, security groups, NACLs
Virtualization	Hypervisor security, VM isolation	VM configuration, OS patching
OS & Runtime	N/A for IaaS	OS patching, runtime updates, application hardening
Application	N/A	Secure coding, dependency scanning, OWASP controls
Data	Storage durability/availability	Encryption, classification, access control
Identity	IAM service availability	User management, MFA, least privilege

5.2 Virtualization-Specific Security Threats

The virtualization layer introduces unique attack surfaces not present in traditional computing. These threats target the hypervisor, VM isolation mechanisms, and the management infrastructure.

5.2.1 Guest Hopping (VM Escape)

VM escape is the most severe virtualization-specific attack. An attacker who has compromised a guest VM exploits vulnerabilities in the hypervisor or virtual device emulation to break out of the VM's isolated environment and gain access to the host or other VMs.

Aspect	Details

Attack Vector	Exploit bugs in virtual device emulation (NIC, sound card, floppy), shared memory, or hypervisor API
Impact	Complete compromise of hypervisor, host OS, and all co-located guest VMs
Notable CVE	CVE-2015-3456 (VENOM): QEMU virtual floppy controller buffer overflow — affected Xen, KVM, VirtualBox
Detection	Anomaly detection in hypervisor, memory analysis, IDS on host
Prevention	Patch hypervisors promptly, disable unnecessary virtual devices, use hardware-isolated VMs
Mitigation Depth	Minimal VM privilege, dedicated hardware per tenant (bare metal), micro-segmentation

5.2.2 VM Migration Attacks

Live VM migration transfers a running VM's memory, CPU state, and storage between physical hosts. If migration channels are not properly secured, they represent a significant attack surface.

Attack Type	Description	Mitigation
Man-in-the-Middle	Attacker on migration network intercepts unencrypted VM memory pages	Encrypt migration traffic with TLS 1.3 or IPSec
State Injection	Attacker modifies VM memory pages in transit to inject malicious code	Integrity protection (HMAC) on migration data
Rogue Destination	Attacker redirects migration to attacker-controlled host	Mutual authentication of migration endpoints
Timing Attack	Attacker gains information from timing of memory page transfers	Dedicated encrypted migration network; traffic shaping
Snapshot Theft	Stealing VM snapshot files to extract data at rest	Encrypt snapshots; restrict snapshot access with ACLs

5.2.3 Hyper Jacking

Hyper jacking is one of the most sophisticated and insidious attacks against virtualized systems. The attacker inserts a thin malicious hypervisor beneath the legitimate hypervisor, causing the original OS and hypervisor to run as guests on top of the attacker's hypervisor.

• **Attack Mechanism:** Exploit BIOS/UEFI firmware vulnerabilities or bootkit to install malicious hypervisor before OS boots.

• **Blue Pill Attack:** A famous conceptual hyper jacking attack using AMD SVM hardware to insert a transparent hypervisor.

• **Stealth:** Extremely difficult to detect — all security software runs above the malicious hypervisor and cannot see it.

• **Persistence:** Survives OS reinstalls as it operates at firmware/boot level.

• **Prevention:** Secure Boot (UEFI with TPM), firmware integrity verification, hardware attestation (TPM 2.0).

• **Trusted Execution Environment (TEE):** Intel TXT or AMD SEV allows remote attestation of hypervisor integrity.

5.2.4 Side-Channel Attacks

Side-channel attacks exploit physical characteristics of shared hardware to infer information about other VMs' processes without any software vulnerability.

Attack Type	Mechanism	Example
Cache Timing	Monitor LLC cache access patterns to infer data from co-tenant VM	Flush+Reload, Prime+Probe on AES keys
Spectre/Meltdown	CPU speculative execution allows cross-VM memory reads	CVE-2017-5753, CVE-2017-5754
Rowhammer	Flip bits in neighboring DRAM rows by rapidly reading adjacent rows	Escape from VMs on shared DRAM
Power Analysis	Measure power consumption to infer cryptographic operations	Requires physical access or smart meters
Network Timing	Infer VM co-location and traffic from network timing	Identify co-tenants on shared network

Attack	Description	Mitigation
VM DoS / Resource Exhaustion	One VM consumes all CPU/memory/I/O starving others	Strict resource quotas via cgroups; per-VM rate limiting
VM Sprawl	Uncontrolled proliferation of VMs creating unmanaged attack surface	VM lifecycle policy; automated decommission; auditing
Snapshot Attack	Access sensitive data via old VM snapshots	Encrypt all snapshots; strict access control; auto-expiry

API Exploitation	Attack hypervisor management API to gain control	Strong API authentication; rate limiting; API gateway WAF

5.3 Data Security in the Cloud

Data security in cloud environments must address data throughout its complete lifecycle: at rest (stored), in transit (moving), and in use (being processed).

5.3.1 Encryption at Rest

Method	Description	Example
Server-Side Encryption (SSE-S3)	Cloud provider manages keys; data encrypted before writing to disk	AWS S3 default encryption
SSE with Customer Key (SSE-C)	Customer provides encryption key per request; provider doesn't store key	S3 SSE-C
Client-Side Encryption	Customer encrypts data before sending to cloud	AWS Encryption SDK, client-side S3
Bring Your Own Key (BYOK)	Customer manages keys in cloud HSM; provider encrypts with customer key	AWS KMS CMK, Azure Key Vault
Hold Your Own Key (HYOK)	Key never leaves customer premises; data decrypted only in customer environment	Azure Information Protection HYOK

5.3.2 Encryption in Transit

• **TLS 1.3:** Current standard for data encryption in transit. Provides forward secrecy, faster handshake, no legacy algorithms.

• **Certificate Management:** Use managed certificate services (AWS ACM, Let's Encrypt) for automatic renewal.

• **mTLS (Mutual TLS):** Both client and server authenticate with certificates — required for service mesh (Istio, Linkerd).

• **VPN:** Site-to-site IPSec VPN for secure connectivity between on-premises and cloud (AWS Site-to-Site VPN).

• **Private Connectivity:** AWS Direct Connect / Azure ExpressRoute — dedicated private fiber, bypassing public Internet.

• **API Security:** HTTPS mandatory, OAuth 2.0/OpenID Connect for authorization, API keys with rate limiting.

5.3.3 Key Management

• **Hardware Security Modules (HSM):** Tamper-resistant physical devices for cryptographic key storage and operations.

- **Key Rotation:** Regularly replace encryption keys to limit exposure from compromised keys.
- **Key Hierarchy:** Master key encrypts data encryption keys (DEK); DEKs encrypt actual data.
- **AWS KMS:** Managed key management with CloudHSM integration, audit trails via CloudTrail.
- **Azure Key Vault:** Centralized secrets, keys, and certificates management with HSM backing.

5.4 Identity and Access Management (IAM)

IAM is the cornerstone of cloud security. It governs which principals (users, roles, services) can take which actions on which cloud resources under which conditions. Every cloud API call is authenticated and authorized through IAM.

5.4.1 Core IAM Concepts

Concept	Definition	Cloud Example
Identity	Unique representation of an entity (user, service, device)	AWS IAM User, Azure AD User
Authentication (AuthN)	Proving who you are	Password, MFA, certificate, biometric
Authorization (AuthZ)	Determining what you can do	IAM Policy, RBAC Role, ACL
Principal	Entity making a request	User, IAM Role, Service Account
Policy	Document defining permissions (allow/deny on resource/action)	AWS IAM Policy JSON, Azure Role Definition
Role	Collection of permissions assumable by principals	AWS IAM Role, Azure RBAC Role
Token	Temporary credential issued after authentication	JWT, SAML Assertion, AWS STS token
Scope	Boundary of token's validity (resource, time)	OAuth scope, Azure subscription scope

5.4.2 Authentication Methods

Method	Security Level	Use Case
Password	Low-Medium	Human user login (must combine with MFA)
MFA (TOTP/FIDO2)	High	All privileged access; recommended for all accounts
Client Certificate	High	Machine-to-machine authentication

API Keys	Medium	Programmatic access (rotate regularly)
IAM Roles with Instance Profiles	High	EC2/Lambda automatic short-lived credentials
SAML 2.0 / OIDC Federation	High	Enterprise SSO to cloud console
Biometric	High	Mobile/consumer apps

5.4.3 Access Control Models

Model	Full Name	How Access Determined	Strengths	Limitations
DAC	Discretionary Access Control	Owner sets permissions	Flexible, intuitive	Hard to audit at scale; owner errors
MAC	Mandatory Access Control	System enforces based on sensitivity labels	Strong enforcement	Complex, inflexible
RBAC	Role-Based Access Control	Permissions linked to role; role linked to user	Easy to manage, audit	Role explosion in complex orgs
ABAC	Attribute-Based Access Control	Policies based on user/resource/environment attributes	Fine-grained, dynamic	Complex policies, harder to debug
PBAC	Policy-Based Access Control	Explicit policy statements evaluated at runtime	Most flexible, auditable	Requires sophisticated policy engine

5.5 IAM Challenges in Cloud Environments

Challenge	Description	Solution
Identity Federation	Consistent identity across multiple cloud providers and on-prem	SAML 2.0, OIDC; central IdP (Okta, Azure AD)
Over-Privileged Accounts	Granting excessive permissions violating least privilege	Regular access reviews; permission boundaries; JIT access

Privilege Escalation	Attacker chains permissions to gain admin via misconfigured policies	Policy analysis tools (IAM Access Analyzer, Ermetic)
Shadow IT	Employees using unauthorized cloud services without IT oversight	CASB (Cloud Access Security Broker); enforced SSO
Service Account Abuse	Overpowered service accounts compromised by attackers	Workload Identity; minimize permissions; rotate keys
Multi-Cloud Identity	Different IAM models across AWS, Azure, GCP	Unified identity platform; consistent RBAC policies
Zombie Accounts	Inactive former employee accounts still active	Automated deprovisioning; regular access reviews
Token Theft	Stolen JWT/OAuth tokens grant unauthorized access	Short token lifetimes; token binding; anomaly detection
Credential Stuffing	Compromised credentials from breaches used on cloud	MFA everywhere; breach detection; password managers
Insider Threat	Malicious or negligent internal users	User behavior analytics (UEBA); least privilege; DLP

5.6 IAM Architecture

5.6.1 Core Architectural Components

Component	Function	Example
Identity Provider (IdP)	Central authority authenticating users; issues tokens	Azure AD, Okta, AWS IAM Identity Center
Service Provider (SP)	Application relying on IdP for authentication	AWS Console, Salesforce, custom apps
Directory Service	Stores user accounts, groups, attributes	Microsoft AD, LDAP, AWS Directory Service
Single Sign-On (SSO)	One authentication grants access to multiple services	Azure AD SSO, Okta SSO, AWS SSO
MFA Service	Enforces second factor authentication	TOTP apps, FIDO2 hardware keys, SMS (avoid)
PAM (Privileged Access Management)	Special controls for admin accounts	CyberArk, BeyondTrust, AWS SSM Session Manager

Token Service (STS)	Issues short-lived temporary credentials	AWS STS, Azure AD Token Endpoint
Policy Engine	Evaluates access requests against policy	AWS IAM, Azure RBAC, Open Policy Agent (OPA)

5.6.2 Zero Trust Architecture

Zero Trust is a modern security model coined by Forrester Research and popularized by Google's BeyondCorp implementation. It rejects the traditional 'trust but verify' perimeter model in favor of 'never trust, always verify' — treating every request as potentially hostile regardless of network origin.

Zero Trust Principle	Traditional Security	Zero Trust
Trust Model	Trust all inside the perimeter	Trust nothing and no one by default
Verification	Authenticate once at perimeter	Verify every request, every time
Access Scope	Network-level access (VPN)	Per-application, per-resource access
User Location	Internal = trusted	Location irrelevant — always verify
Device	Corporate device = trusted	Verify device health and compliance always
Lateral Movement	Easy once inside	Micro-segmentation prevents spread

Zero Trust Implementation Steps:

1. **Identify Protected Surfaces:** Map sensitive data, applications, assets (DAAS) to protect.
2. **Map Transaction Flows:** Understand how data flows to/from protected surfaces.
3. **Build Zero Trust Architecture:** Deploy micro-perimeters around each protected surface.
4. **Create Zero Trust Policy:** Who needs access? To what? When? From where? Via which device?
5. **Monitor and Maintain:** Continuous logging, analytics, and policy refinement.

5.7 IAM Best Practices

Practice	Description	Implementation
Least Privilege	Grant only minimum permissions needed for a task	IAM Access Analyzer; permission boundaries; condition keys
MFA Everywhere	Require MFA for all human accounts, especially privileged	TOTP (Google Authenticator), FIDO2 (YubiKey)

Rotate Credentials	Regularly rotate passwords, API keys, certificates	90-day rotation policy; automated via Secrets Manager
Prefer Roles over Keys	Use IAM roles with temporary credentials instead of long-term access keys	EC2 Instance Profiles; Lambda Execution Roles; EKS IRSA
Centralize Audit Logging	Log all AuthN/AuthZ events; immutable, centralized	AWS CloudTrail; Azure Activity Log; SIEM integration
Separate Environments	Different AWS accounts/Azure subscriptions per environment	AWS Organizations; Azure Management Groups
Break Glass Accounts	Emergency admin accounts with strong controls, rarely used	Sealed envelope procedures; alert on use
Just-in-Time (JIT) Access	Grant elevated access only when needed, automatically revoke	AWS SSM Session Manager; Azure PIM; CyberArk

5.8 Cloud Security Best Practices

Domain	Practice	Tools
Network Security	VPC with private subnets; security groups as stateful firewalls; NACLs; VPN/Direct Connect	AWS VPC, Azure VNet, NSGs
DDoS Protection	Multi-layer protection: volumetric at edge, protocol at LB, application at WAF	AWS Shield Advanced, Azure DDoS Protection
Vulnerability Management	Regular scanning; automated patching; CVE tracking	AWS Inspector, Azure Defender, Trivy
Secrets Management	Never hardcode secrets; use managed secrets stores	AWS Secrets Manager, Azure Key Vault, HashiCorp Vault
Logging & Monitoring	Comprehensive audit logs; real-time alerting; SIEM integration	AWS CloudTrail, CloudWatch, Azure Monitor, Splunk
Compliance Automation	Continuous compliance assessment; auto-remediation	AWS Config, Azure Policy, Cloud Custodian
CSPM	Identify and fix cloud misconfigurations continuously	Prisma Cloud, Wiz, Microsoft Defender for Cloud

Supply Chain Security	Scan IaC templates, container images, dependencies	Checkov, Snyk, AWS CodeGuru Security

5.9 Unit Summary

Summary

Cloud security addresses virtualization threats: VM escape (hypervisor exploitation), VM migration attacks (MITM on migration traffic), hyper jacking (rogue hypervisor insertion), and side-channel attacks (Spectre/Meltdown, cache timing). Data security uses encryption at rest (SSE, KMS, HSM) and in transit (TLS 1.3, mTLS). IAM governs identity, authentication (MFA, certificates), and authorization (RBAC, ABAC). IAM challenges include federation, over-privilege, shadow IT, and zombie accounts. Zero Trust architecture — 'never trust, always verify' — is the modern approach replacing perimeter-based security.

Review Questions

1. Explain the VM escape (guest hopping) attack. Describe the VENOM vulnerability and how similar attacks are mitigated.
2. What is hyper jacking? How does it exploit the virtualization stack? How does Secure Boot and TPM help prevent it?
3. Describe three types of VM migration attacks. What cryptographic controls prevent each?
4. Explain Spectre and Meltdown as side-channel attacks. Why are they especially dangerous in cloud multi-tenant environments?
5. Compare Server-Side Encryption (SSE-S3, SSE-C) and client-side encryption for cloud storage.
6. Define IAM. Explain the difference between authentication and authorization with cloud examples.
7. Compare RBAC and ABAC. Which is more suitable for fine-grained cloud access control?
8. List five IAM challenges in cloud environments. For each, propose a technical solution.
9. Explain Zero Trust Architecture. How does it differ from traditional perimeter security?
10. What is the shared responsibility model? Map responsibilities between provider and customer for IaaS.

EXAM PREPARATION GUIDE

Purpose

This section provides a structured exam preparation guide for U23CST71 Cloud Computing, including important 2-mark and 16-mark questions, model answers, and key formulas/diagrams.

Important 2-Mark Questions (Short Answer)

Unit I — Cloud Architecture

Question	Key Answer Points
Define cloud computing (NIST).	On-demand network access to shared configurable resources rapidly provisioned with minimal management effort.
List the five NIST characteristics.	On-demand self-service, Broad network access, Resource pooling, Rapid elasticity, Measured service.
What is IaaS? Give an example.	Infrastructure as a Service: virtualized compute/storage/network. Example: AWS EC2.
Differentiate public and private cloud.	Public: shared, provider-owned, Internet accessible. Private: dedicated, single org, on-premises or hosted.
What is cloud bursting?	Running workloads in private cloud, automatically scaling to public cloud during peak demand.
Define elasticity in cloud computing.	Ability to rapidly scale resources up or down in response to demand, appearing unlimited to user.
What is a Cloud Broker?	An entity that manages cloud service use, performance, and delivery; negotiates between providers and consumers.
Name four cloud deployment models.	Public, Private, Community, Hybrid.

Unit II — Virtualization Basics

Question	Key Answer Points
Define virtualization.	Creating a software-based abstraction of physical resources (CPU, memory, storage, network) into virtual equivalents.

What is a hypervisor? Give types.	Software managing VMs. Type 1: bare-metal (VMware ESXi). Type 2: hosted (VirtualBox).
Define full virtualization.	Hypervisor simulates complete hardware; guest OS runs unmodified; uses binary translation.
What is paravirtualization?	Guest OS is modified to cooperate with hypervisor via hypercalls; better performance than full virt.
What is hardware-assisted virtualization?	CPU hardware extensions (Intel VT-x, AMD-V) enabling efficient virtualization without binary translation.
What is a snapshot?	Point-in-time copy of VM state (disk, memory, settings) used for backup and rollback.
Define memory ballooning.	Guest OS balloon driver inflates to return unused pages to hypervisor, enabling memory over-commitment.
What is SR-IOV?	Single Root I/O Virtualization: NIC presents multiple virtual functions directly assignable to VMs.

Unit III — Virtualization Infrastructure & Docker

Question	Key Answer Points
What is VDI?	Virtual Desktop Infrastructure: desktop VMs hosted centrally, accessed by thin clients over network.
Define SDN.	Software-Defined Networking: separates control plane (where traffic goes) from data plane (packet forwarding).
What are Linux namespaces?	Kernel feature isolating process views of resources: PID, Network, Mount, UTS, IPC, User namespaces.
Differentiate container and VM.	Container: shares host kernel, lightweight, fast. VM: separate kernel, strong isolation, heavier.
What is a Dockerfile?	Text file with instructions (FROM, RUN, COPY, CMD) to build a Docker container image layer by layer.
What is Docker Hub?	Public container image registry with 8+ million images; default registry for docker pull/push.

What are Docker volumes?	Persistent storage for containers. Types: named volumes, bind mounts, tmpfs mounts.
Define SAN and NAS.	SAN: block-level storage over dedicated network (Fibre Channel). NAS: file-level storage over TCP/IP.

Unit IV — Cloud Deployment Environments

Question	Key Answer Points
What is Google App Engine?	PaaS platform on Google Cloud for hosting web apps without managing servers; auto-scaling.
Name five AWS compute services.	EC2, Lambda, ECS, EKS, Elastic Beanstalk.
What is an AWS Region?	Geographically isolated cloud location containing multiple Availability Zones.
Define Eucalyptus.	Open-source private cloud platform providing AWS-compatible APIs (EC2, S3, EBS, IAM).
What is Nova in OpenStack?	OpenStack Compute: manages VM lifecycle, schedules instances to compute nodes.
Name the OpenStack identity service.	Keystone — provides authentication (tokens) and service catalog.
What is Neutron in OpenStack?	OpenStack Networking service: provides virtual networks, routers, floating IPs, security groups.
What is Azure Arc?	Microsoft service extending Azure management to on-premises, multi-cloud, and edge infrastructure.

Unit V — Cloud Security

Question	Key Answer Points
What is VM escape (guest hopping)?	Attacker breaks out of VM isolation by exploiting hypervisor vulnerabilities to access host/other VMs.
Define hyper jacking.	Installing a malicious hypervisor beneath the legitimate one; legitimate OS becomes a guest VM.
What is a VM migration attack?	Exploiting unsecured live migration channels to intercept or modify VM memory/state during transfer.

Define IAM.	Identity and Access Management: framework controlling who can access which cloud resources.
What is RBAC?	Role-Based Access Control: permissions assigned to roles; users/services assigned to roles.
What is Zero Trust?	Security model: 'never trust, always verify' — authenticate and authorize every request regardless of source.
What is MFA?	Multi-Factor Authentication: requires 2+ verification factors (password + TOTP/biometric/hardware key).
Define Spectre and Meltdown.	CPU side-channel vulnerabilities exploiting speculative execution to read memory across privilege boundaries.

Important 16-Mark Questions (Long Answer)

Unit I

1. Explain the NIST cloud computing reference model with its five essential characteristics, three service models, and four deployment models. Draw the relationship diagram. (16 marks)
2. What is cloud architecture? Describe the design considerations for Compute Clouds and Storage Clouds. Explain any five design challenges with solutions. (16 marks)
3. Compare Grid Computing and Cloud Computing across eight dimensions. Trace the evolution of cloud computing from mainframe time-sharing to modern edge computing. (16 marks)

Unit II

1. Explain Full Virtualization, Paravirtualization, and Hardware-Assisted Virtualization in detail. Compare them across performance, OS modification requirement, and use cases. (16 marks)
2. Describe CPU virtualization, Memory virtualization, and I/O virtualization techniques. Explain binary translation, Extended Page Tables (EPT), and SR-IOV. (16 marks)
3. What is a Hypervisor? Compare Type 1 and Type 2 hypervisors. Explain the five levels of virtualization implementation with examples at each level. (16 marks)

Unit III

1. Compare Virtual Machines and Docker Containers across 10 dimensions. Explain the Docker architecture including client, daemon, containerd, runc, and registry. (16 marks)
2. Explain Docker components in detail: Dockerfile, images, containers, volumes, and networks. Write a Dockerfile for a Flask Python application with security best practices. (16 marks)
3. Describe Network Virtualization techniques: VLAN, SDN, NFV, and virtual switches. Explain the three-layer SDN architecture with OpenFlow protocol. (16 marks)

Unit IV

1. Describe the OpenStack cloud platform. Explain all 12 major components including Nova, Swift, Cinder, Neutron, Keystone, Glance, and Horizon with their functions and AWS equivalents. (16 marks)

2. Compare Amazon AWS and Microsoft Azure. Describe AWS global infrastructure, the Well-Architected Framework's six pillars, and five core AWS services in each category. (16 marks)

3. Explain Eucalyptus architecture with its five components. How does it achieve AWS API compatibility? Compare Eucalyptus and OpenStack for private cloud deployment. (16 marks)

Unit V

1. Explain the three major virtualization-specific attacks: VM escape, VM migration attack, and hyper jacking. Describe the attack mechanism, impact, and mitigation for each. (16 marks)

2. What is IAM? Explain IAM concepts (identity, authentication, authorization, principal, policy, role). Compare RBAC and ABAC. List 10 IAM challenges with solutions. (16 marks)

3. Describe the Zero Trust security model. How does it differ from traditional perimeter security? Explain data security measures for data at rest, in transit, and key management best practices. (16 marks)

Key Formulas and Metrics in Cloud Computing

Availability and SLA

Metric	Formula / Definition	Example
Availability	(Uptime / Total Time) × 100%	99.99% = 52 min downtime/year
MTBF (Mean Time Between Failures)	Total operational time / Number of failures	1000 hours MTBF
MTTR (Mean Time To Recover)	Total downtime / Number of failures	2 hours MTTR
RTO (Recovery Time Objective)	Maximum acceptable time to restore service after failure	RTO = 4 hours
RPO (Recovery Point Objective)	Maximum acceptable data loss (in time) after failure	RPO = 1 hour
SLA Nines	99% = 3.65d/yr, 99.9% = 8.76h/yr, 99.99% = 52.6min/yr, 99.999% = 5.26min/yr	AWS S3: 99.999999999% durability

Virtualization and Performance Metrics

Metric	Description	Typical Value

VM Density	Number of VMs per physical host	20–100 VMs per host
CPU Overcommit Ratio	vCPUs allocated / physical CPU cores	4:1 to 8:1 typical
Memory Overcommit	vRAM allocated / physical RAM	1.25:1 to 1.5:1 with ballooning
Storage IOPS	Input/Output Operations Per Second	SSD: 10K–1M IOPS; HDD: 100–200 IOPS
Container Startup Time	Time to start a container	50–500 milliseconds
VM Boot Time	Time to boot a virtual machine	30 seconds to 5 minutes
Live Migration Time	Time to complete vMotion/live migration	~1 second per GB of active memory

Cloud Cost Models

Pricing Model	Description	Best For
On-Demand	Pay per hour/second; no commitment	Unpredictable, short-term workloads
Reserved Instances	1 or 3-year commitment; 40–75% discount	Steady-state, predictable workloads
Spot Instances	Bid on unused capacity; 60–90% discount; can be interrupted	Batch jobs, fault-tolerant workloads
Savings Plans	Flexible commitment to spend; covers multiple services	Mixed workloads across services
Free Tier	Limited free usage for new accounts (12 months or always-free)	Learning and development
Egress Costs	Charges for data transferred OUT of cloud	Factor into architecture decisions

Architecture Patterns in Cloud Computing

Common Cloud Architecture Patterns

Pattern	Description	Use Case
Microservices	Application decomposed into small, independent services	Netflix, Uber, Amazon

Serverless	Event-driven functions; no server management	API backends, event processing, automation
Event-Driven	Components communicate via events/messages asynchronously	Order processing, real-time analytics
CQRS	Command Query Responsibility Segregation — separate read/write models	High-read applications, event sourcing
Circuit Breaker	Prevent cascading failures by stopping calls to failing services	Resilient microservices
Sidecar	Deploy helper containers alongside main container in same pod	Service mesh (Envoy), logging agents
Strangler Fig	Gradually replace legacy system by routing traffic to new services	Legacy modernization
Bulkhead	Isolate elements to prevent failures from spreading	Resource pool isolation
API Gateway	Single entry point routing requests to appropriate microservices	Frontend for microservices
Multi-Region Active-Active	Traffic served from multiple regions simultaneously; zero-downtime failover	Global applications

DevOps and Cloud Integration

Modern cloud deployments are tightly integrated with DevOps practices, enabling continuous delivery of software at cloud scale:

Practice	Description	Cloud Tool
Infrastructure as Code (IaC)	Define infrastructure in code; version control; reproducible	Terraform, AWS CloudFormation, Azure Bicep
CI/CD Pipeline	Automated build, test, and deploy on every code change	AWS CodePipeline, Azure DevOps, GitHub Actions
GitOps	Git as single source of truth; automated reconciliation	ArgoCD, Flux for Kubernetes
Immutable Infrastructure	Never update servers; replace with new image	AMI-based deployments; container images

Blue-Green Deployment	Two identical environments; switch traffic for zero-downtime	Route 53 weighted routing; ALB target groups
Canary Release	Gradually route traffic to new version; monitor; roll back if needed	CodeDeploy, Istio traffic management
Chaos Engineering	Intentionally inject failures to test resilience	AWS Fault Injection Simulator; Chaos Monkey
Observability	Logs, metrics, traces for understanding system behavior	CloudWatch, Datadog, Jaeger, ELK Stack

Cloud Compliance and Governance

Standard/Regulation	Applies To	Key Requirements
ISO/IEC 27001	Any organization	Information security management system (ISMS)
SOC 2 Type II	SaaS providers	Security, availability, confidentiality, integrity, privacy
GDPR	EU data subjects' data	Data protection, right to erasure, breach notification (72h)
HIPAA	US healthcare data	PHI protection, access controls, audit logs, encryption
PCI-DSS	Payment card data	Cardholder data protection, network segmentation, vulnerability scanning
FedRAMP	US federal agencies	Cloud services authorization for government use
CCPA	California residents	Data privacy rights, opt-out of data sale

Governance Best Practices

• **Cloud Center of Excellence (CCoE):** Dedicated team setting cloud standards, guardrails, and best practices for the organization.

• **Tagging Strategy:** Mandatory tags on all resources (environment, team, cost-center, project) for cost allocation and governance.

• **AWS Organizations / Azure Management Groups:** Hierarchical account/subscription structure with Service Control Policies.

• **Landing Zone:** Pre-configured, secure, multi-account cloud environment following best practices (AWS Control Tower, Azure Landing Zones).

• **FinOps:** Financial operations for cloud — collaboration between engineering, finance, and business to optimize cloud spend.

• **Cloud Asset Inventory:** Automated discovery and cataloging of all cloud resources for security and cost visibility.

Emerging Trends in Cloud Computing

Trend	Description	Relevance
Edge Computing	Processing data closer to source (IoT devices, factories) rather than central cloud	Reduces latency; enables real-time decisions
Serverless & FaaS	No server management; pay per invocation; infinite auto-scaling	Reduces operational overhead; cost-effective
Kubernetes & Container Orchestration	Managing thousands of containers across clusters	Industry standard for microservices at scale
AI/ML as a Service	Pre-built ML models via APIs; managed ML platforms	Democratizes AI; faster development
Quantum Computing Cloud	Quantum processors accessible via cloud APIs	Cryptography, optimization, drug discovery
Confidential Computing	Processing encrypted data using TEEs (Intel SGX, AMD SEV)	Protects data-in-use; regulatory compliance
FinOps	Engineering, finance, and business collaboration on cloud cost optimization	Cloud cost governance
Sustainable Cloud (Green IT)	Reducing carbon footprint of cloud infrastructure	Corporate sustainability goals
Multi-Cloud Management	Unified control plane across AWS, Azure, and GCP	Avoid vendor lock-in; best-of-breed
Cloud-Native Security (CNAPP)	Unified cloud security platform combining CSPM, CWPP, CIEM	Holistic cloud security posture

APPENDIX: COMPREHENSIVE GLOSSARY

ABAC: Attribute-Based Access Control — access decisions based on user, resource, and environment attributes.

AppArmor: Linux security module restricting program capabilities via per-program profiles.

Auto-Scaling: Automatic adjustment of compute capacity in response to demand changes.

Availability Zone (AZ): Isolated data center within a cloud region for fault tolerance.

Bare Metal: Physical server with no hypervisor — directly accessible hardware.

Binary Translation: Rewriting privileged guest instructions to safe equivalents in full virtualization.

BYOD: Bring Your Own Device — policy allowing employees to use personal devices for work.

cgroups: Linux Control Groups — limits and accounts for resource usage per process group.

CASB: Cloud Access Security Broker — enforces security policies for cloud service access.

CDN: Content Delivery Network — geographically distributed servers for fast content delivery.

Cloud Bursting: Running workloads in private cloud but 'bursting' to public cloud during peak demand.

Compliance: Meeting regulatory standards: GDPR, HIPAA, PCI-DSS, SOC 2, ISO 27001.

Container: Lightweight isolated environment sharing host OS kernel — lighter than a VM.

CSPM: Cloud Security Posture Management — continuously assess and remediate cloud misconfigurations.

DaaS: Desktop as a Service — cloud-hosted virtual desktops delivered over the network.

DPDK: Data Plane Development Kit — bypasses OS kernel for high-speed network packet processing.

Docker: Open-source platform for building, shipping, and running containers.

Dockerfile: Text file with instructions to build a Docker container image.

DRS: VMware Distributed Resource Scheduler — automatically balances VM load across cluster.

Elasticity: Ability to rapidly scale resources up or down based on demand.

EPT: Extended Page Tables — Intel hardware for memory virtualization (two-level page walk).

Eucalyptus: Open-source private cloud software compatible with AWS APIs.

FaaS: Function as a Service — serverless execution of individual functions (AWS Lambda).

Fault Tolerance: Zero-downtime failover using synchronized shadow VMs (VMware FT).

Firewall: Network security device/software filtering traffic based on rules.

Full Virtualization: Hypervisor simulates complete hardware; guest OS runs unmodified.

GAE: Google App Engine — PaaS platform for hosting web applications on Google infrastructure.

Guest OS: Operating system running inside a virtual machine.

HA: High Availability — automatic VM restart on surviving hosts when a host fails.

Host Machine: Physical computer running the hypervisor and guest VMs.

HSM: Hardware Security Module — tamper-resistant device for cryptographic key management.

Hypervisor: Software/firmware creating and managing VMs — Type 1 (bare-metal) or Type 2 (hosted).

Hyper Jacking: Installing a malicious hypervisor beneath the legitimate hypervisor.

IaaS: Infrastructure as a Service — virtualized compute, storage, and network as a service.

IAM: Identity and Access Management — control who can access which cloud resources.

IdP: Identity Provider — authenticates users and issues security tokens (SAML, OIDC).

IOMMU: Input-Output Memory Management Unit — enables secure device passthrough to VMs.

IPSec: Internet Protocol Security — suite for encrypting and authenticating IP packets.

JIT: Just-in-Time access — temporary privileged access granted only when needed.

Keystone: OpenStack Identity Service — authentication and service catalog.

KVM: Kernel-based Virtual Machine — Linux kernel module providing Type 1 hypervisor capability.

Live Migration: Moving a running VM between physical hosts with minimal or no downtime.

MAC: Mandatory Access Control — system enforces access based on security labels.

MFA: Multi-Factor Authentication — requires two or more verification factors for authentication.

Meltdown: CPU vulnerability allowing unprivileged access to kernel memory via speculative execution.

Micro-segmentation: Dividing network into small isolated zones each requiring separate authentication.

Multi-tenancy: Multiple customers sharing cloud infrastructure with logical isolation.

Namespace: Linux kernel isolation mechanism for processes, networks, filesystems, and users.

NFV: Network Functions Virtualization — software-based network services on standard hardware.

NIST: National Institute of Standards and Technology — defines cloud computing reference model.

Nova: OpenStack Compute — manages VM instances and schedules them to compute nodes.

NPT: Nested Page Tables — AMD hardware equivalent of Intel EPT for memory virtualization.

OpenStack: Comprehensive open-source cloud infrastructure platform (Nova, Neutron, Cinder, etc.).

OVS: Open vSwitch — open-source production virtual switch supporting OpenFlow protocol.

PaaS: Platform as a Service — managed runtime environment for application deployment.

PAM: Privileged Access Management — controls and monitors privileged account access.

Paravirtualization: Virtualization requiring guest OS modification to use hypercalls to the VMM.

RBAC: Role-Based Access Control — permissions assigned based on organizational roles.

SaaS: Software as a Service — complete software delivered over Internet, no installation.

SAN: Storage Area Network — dedicated high-speed network for block-level storage access.

SDN: Software-Defined Networking — separates network control plane from data forwarding plane.

seccomp: Linux secure computing mode — restricts system calls available to containers.

Snapshot: Point-in-time copy of VM disk, memory, and state for backup or rollback.

Spectre: CPU vulnerability exploiting speculative execution to read arbitrary memory across privilege boundaries.

SR-IOV: Single Root I/O Virtualization — NIC presents multiple virtual functions for direct VM assignment.

SSO: Single Sign-On — one authentication event grants access to multiple services.

STS: Security Token Service — issues temporary credentials with limited scope and duration.

TLS: Transport Layer Security — cryptographic protocol for encrypted data transmission.

TPM: Trusted Platform Module — hardware chip for secure key storage and system integrity measurement.

Thin Provisioning: Allocating virtual storage space greater than available physical space.

VDI: Virtual Desktop Infrastructure — centrally hosted desktop VMs delivered to thin clients.

VLAN: Virtual LAN — logical network segment isolated at Layer 2 using 802.1Q tags.

VM Escape: VM guest breaking out of isolation to access hypervisor or other VMs.

vMotion: VMware live migration technology for moving running VMs between hosts.

VNF: Virtual Network Function — software-based network appliance running on standard servers.

VPC: Virtual Private Cloud — isolated virtual network within a cloud provider.

VPN: Virtual Private Network — encrypted tunnel over public network for secure connectivity.

Xen: Open-source Type 1 hypervisor originally developed at Cambridge University.

Zero Trust: Security model: verify every request explicitly; never assume trust based on network location.

This material covers all five units of U23CST71 Cloud Computing. Recommended references: Buyya et al. 'Cloud Computing: Principles and Paradigms' (Wiley); Linthicum 'Cloud Computing and SOA Convergence' (Addison-Wesley); AWS, Azure, Google Cloud, Docker, and OpenStack official documentation.