Cloud Computing Lecture Notes

Distributed Computing/Systems

Definition:

Distributed computing refers to a system where computing resources are distributed across multiple locations rather than being centralized in a single system. This enables task distribution and efficient resource utilization.

Why Use Distributed Systems?

- **Scalability Issues:** Traditional computing faces bottlenecks due to hardware limitations, whereas distributed systems allow for hardware scaling.
- **Connected Devices:** In a networked system, connected devices communicate, but this does not necessarily make them distributed.
- **IoT (Internet of Things):** IoT is one of the largest examples of distributed computing.
- **Multi-layered System Design:** Distributed computing enables systems to function in multiple layers, with each layer acting as a distributed entity.
- **User Perspective:** Although the system consists of multiple machines, distributed computing presents a unified system to users.

Parallel Computing

Parallel computing involves executing multiple processes simultaneously to enhance speed and efficiency.

Key Aspects of Parallel Computing:

- **Distribution & Speed:** Distributing tasks does not always guarantee faster execution; efficiency depends on the nature of the task and system design.
- **Core Objective:** The primary goal is to **speed up** processing by leveraging parallel programming and hardware optimizations.
- **Use Cases:** Parallel computing is widely used in:

    - Vector processing
    - Image processing
    - Matrix multiplication
- **Limitations:**
    - Not all applications can be parallelized.
    - Some components of code can be executed in parallel, while others may not.
    - Specific programming languages are required for parallel computing.
- **Computing Infrastructure:**
    - Dedicated machines are needed for parallel computing.
    - **Clusters:** A collection of similar types of systems working together.

Building Distributed Systems

Key Components:

1. **Shared Folders:** Simply sharing files does not make a system truly distributed. Proper access control and system design are needed.
2. **Middleware:** The software layer that facilitates communication between different system components.

Diagram in Notes:
A simple representation of middleware connecting multiple system components. The figure depicts three components (labeled 1, 2, 3) connected via middleware, highlighting its role in integrating distributed elements.

Role of Middleware:

A) Acts as a bridge between applications and networked devices.
B) Provides a unified interface for interacting with connected components.

Clusters vs. Grids

Clusters:

Clusters consist of multiple machines with similar hardware and operating systems, working together in a **symmetric** manner.

- **Key Features:**
  - Middleware abstracts hardware, OS, and software details from users.
  - All machines have the **same type of hardware and OS**.
  - Parallel computing is implemented efficiently in clusters.
  - Middleware in clusters has fewer complexities.
  - Clusters typically exist in **LAN (Local Area Networks)**, meaning all machines are geographically close and in the same environment.

Grids:

Grids, unlike clusters, consist of **heterogeneous** systems that may have different hardware, OS, and configurations. They are **geographically distributed** and operate in an **asymmetric** manner.

- **Key Features:**
  - Middleware in grids has to handle diverse configurations, leading to potential **performance degradation**.
  - Used for solving large-scale computational problems.
  - Systems join grids for incentives, such as **resource sharing** or **financial compensation**.
  - Users can sell extra computational resources within a grid.
  - Example: **Blockchain technology** operates in a grid-like manner.
  - **Drawbacks:** Security concerns due to distributed nature.

Comparison of Clusters & Grids:

- Organizations **do not** allow their internal systems to be part of a grid due to security concerns.
- Individual users, however, can participate in grid computing.
- **Example:** Torrents are a similar concept but not an exact representation of grids.
- **Clusters & Grids together form the foundation of cloud computing.**

- They also contribute to **High-Performance Computing (HPC).**

Cloud Computing

Definition:

Cloud computing is a form of **distributed computing** that provides on-demand computing resources over a network.

Characteristics of Cloud Computing:

- **No Operator Wait Time:** Users don't need to wait for an operator to grant access.
- **Ubiquitous Access:** Cloud services can be accessed from anywhere with an internet connection.
- **Utility Computing Model:** Users pay only for what they consume.
- **Existing Infrastructure:** Many foundational blocks of cloud computing are already present in **Data Centers (DCs)**.

Cloud Computing & HPC (High-Performance Computing):

- **HPC as a Cloud Service:** Cloud providers like **AWS** offer **HPC as a service**.
- **Cluster on Demand:** Users can request cluster-based resources directly from cloud providers.
- **Kubernetes Integration:** Cloud computing allows users to build their own **Kubernetes clusters** or use Kubernetes as a managed service.
- **Autoscaling:** Cloud computing dynamically adjusts resources based on demand.
- **Complex Workloads:** Supports operations like **Machine Learning (ML)** and **big data analytics**.

Supercomputing in the Cloud:

- **Top500.org:** Lists the world's top supercomputers.
- **Supercomputers:** Consist of **millions of cores**, forming clusters of multiple machines.
- **Difference from Cloud Computing:** Unlike the cloud, supercomputers grant access only to **specific users** requiring HPC.

Middleware in Cloud Computing:

- **Resource Management:** Middleware is responsible for managing resources in the cloud.
- **User Abstraction:** End users do not need to know where the cloud system's infrastructure is physically located.

Key Desired Attributes of Cloud Computing:

- **Backend Abstraction:** Cloud systems hide backend configurations and resource management from users.
- **Privacy & Compliance:** Some details, such as the **physical location of machines**, must be disclosed due to legal and compliance requirements.

Cloud Computing Overview

- If certain **essential characteristics** are present in distributed systems, they qualify as **cloud computing**.
- The **NIST (National Institute of Standards and Technology)** defines **cloud computing** as an extension of **distributed systems**.

Essential Characteristics of Cloud Computing

These characteristics distinguish **cloud computing** from traditional distributed systems.

1. On-Demand Self-Service

- Users can **configure and deploy services** independently, without requiring human intervention.
- No need to wait for an operator; services are **already deployed** and **provided via a web interface**.

2. Broad Network Access

- Cloud services establish a **fast** connection between servers and users.

- Since computing is performed **on the operator's side**, the results are delivered quickly to users.
- Users interact with the system via **APIs (Application Programming Interfaces)** or **web interfaces** without needing direct access to computing resources.

3. Resource Pooling

- The **resources of a cloud infrastructure** should be available to **all users**.
- Cloud providers combine **resources** into a shared pool.
- Users can utilize any resource as long as it is available, and once released, the resource **returns to the pool** for reallocation.

4. Rapid Elasticity

- Cloud computing **scales up or down** dynamically.
- Workloads can **expand or contract** based on demand.
- Scaling is **fast and automatic**, ensuring efficient resource utilization.
- Example: **AWS EC2 (Elastic Compute Cloud)** provides rapid elasticity.

5. Measured Service

- Cloud providers implement **pay-as-you-go** billing models.
- **Usage statistics** help with **future planning, security (e.g., anomaly detection), and billing transparency**.

Common Characteristics of Distributed & Cloud Systems

Cloud computing shares several **common characteristics** with general **distributed systems**.

1. Resource Pooling

- Present in **all distributed systems**, including **grid computing and clusters**.

2. Measured Services

- Found in **most distributed systems**.

- **Middleware** is responsible for **resource pooling**.
- Examples: **Hadoop clusters, Apache Spark**.

3. Essential vs. Common Characteristics

- The presence of common characteristics **alone does not** guarantee a system qualifies as **cloud computing**.
- Without these **common characteristics**, achieving the **essential characteristics** becomes extremely difficult.
- **Cloud construction without common characteristics is theoretically possible but practically impossible**.

4. Massive Scale

- A vast amount of **computing resources** is combined to form **large-scale cloud systems**.
- Some clouds operate on a **massive scale**, while others do not.
- **Private clouds for organizations** are **not massive**.
- **Public cloud market dominance: 95-96% of cloud infrastructure** belongs to a few major companies like:
    - **AWS (Amazon Web Services)**
    - **GCP (Google Cloud Platform)**
    - **Azure (Microsoft)**
    - **IBM Cloud**

5. Homogeneity

- Cloud systems prefer **symmetric resources** (same type of OS, servers, configurations, etc.).
- Although different resources can be used, **homogeneity** simplifies **management and performance**.
- **Asymmetric resources** lead to **performance degradation** due to compatibility challenges.

6. Virtualization

- Virtualization allows **creation of virtual objects** from physical resources, such as:

  - **Virtual Memory**
  - **Virtual Storage**
  - **Virtual Operating Systems**
- **Virtualization alone is not cloud computing**, but it is a key enabler.
- **Benefits of Virtualization:**
  - Supports **resource pooling**.
  - Provides **elasticity**.
  - **Eases workload migration** (moving virtual machines is easier than moving physical machines).
- **All cloud services are virtualized**.

7. Resilient Computing

- Cloud resilience ensures system availability despite **failures, redundancies, and faults**.
- Includes **backup, fault tolerance, early failure detection, and mitigation strategies**.
- **Resilience is relative:**
  - **Massive-scale clouds** offer **higher resilience**.
  - **Smaller/private clouds** may have **lower resilience**.

8. Service Orientation

- Cloud resources are provided as **web services**.
- Services include:
  - **Computing**
  - **Storage**
  - **Networking**
- **SOA (Service-Oriented Architecture)**: A design approach used for cloud services.

9. Advanced Security

- **Security breaches impact a large number of users** in cloud environments.
- Requires **significant investment** in **cloud security**.
- Security is **relatively easier** to maintain at a **large scale** than at a **small scale**.

10. Low-Cost Software

- Large-scale cloud usage **reduces software costs** due to **economies of scale**.
- **High production & demand** allow software costs to decrease **relatively**.

11. Geographical Distribution

- Cloud infrastructure is **distributed across multiple locations** to:
  - Improve **fault tolerance**.
  - Enhance **resilience, efficiency, and speed**.
  - Ensure **scalability and privacy compliance**.
- **CDN (Content Delivery Network):** A **distributed network of cloud instances** that **store and distribute content** to users efficiently.
- **Some clouds are not geographically distributed** but offer local compliance options.

Cloud Service Model

Cloud service models define:

1. **How users access services**
2. **How operators manage resources**
3. **Responsibility division between users and cloud service providers (CSPs)**

Responsibility Model

- **User vs. Cloud Service Provider (CSP)**
  - Defines **who manages what** in cloud-based systems.

(Diagram Present in Notes)

- A diagram represents a **responsibility model** where different layers of the cloud stack (Infrastructure, Platform, and Software) are managed either by the **user** or the **CSP**.

1. Infrastructure as a Service (IaaS)

- Example: **AWS EC2**
- Users manage:
  - **Applications**
  - **Operating Systems**
  - **Virtual Machines (VMs)**
  - **Servers**
  - **Networking**
- The **cloud provider** manages the underlying **hardware**.

(Diagram Present in Notes)

- A diagram illustrating **IaaS** shows how users handle **VMs, servers, and networking**, while the cloud provider manages the **infrastructure**.

2. Platform as a Service (PaaS)

- Users interact with **applications** but do not manage the **underlying platform**.
- Example: **Kubernetes**
- The **operator** manages the **platform**.

(Diagram Present in Notes)

- A visual representation of **PaaS** shows how users interact with applications while the cloud provider manages the platform layer.

3. Software as a Service (SaaS)

- Users access fully managed software applications.
- All layers (application, platform, and infrastructure) are **handled by the operator**.
- Users simply **consume the service** without managing any underlying components.

(Diagram Present in Notes)

- A **SaaS diagram** shows how all layers (app, platform, and infrastructure) are managed by the **CSP**.

4. Anything as a Service (XaaS)

- An **extended model** where **any IT service** is delivered as a cloud service.
- Underlying infrastructure **can be managed by either CSPs or third parties**.

Cloud Deployment Models

Cloud deployment models define how cloud infrastructure is set up, who can access it, and its intended use. The key deployment models are:

Model	Description
Public Cloud	Large-scale cloud infrastructure available to the general public.
Private Cloud	Cloud infrastructure dedicated to a single organization, offering better control and security.
Community Cloud	Shared cloud infrastructure among a specific group of organizations with common interests or policies.
Hybrid Cloud	A combination of private and public clouds for enhanced flexibility.
Multicloud	Utilization of multiple cloud services from different providers.
Federated Cloud	A collaboration of multiple cloud providers, maintaining interoperability.

Key Comparisons

- **Public Cloud**: Open to everyone, but security can be a concern.
- **Private Cloud**: More control and security but requires investment.
- **Community Cloud**: Restricted to a specific group (e.g., universities, research institutes).
- **Hybrid Cloud**: Best of both public and private clouds.
- **Multicloud**: Helps avoid vendor lock-in by using services from multiple providers.
- **Federated Cloud**: Enables seamless migration across different cloud providers.

Example Implementations:

- **Public Cloud Providers**: AWS, Google Cloud, Microsoft Azure.
- **Private Cloud Examples**: NED.

- **Federated Cloud Use Case**: Used when organizations need cross-cloud credentials and registrations.

NIST Cloud Reference Architecture

Key Components:

1. **Stakeholders in the Cloud Ecosystem:**
    - A. **Users**
    - B. **Cloud Service Providers (CSPs)**
    - C. **Internet Service Providers (ISPs)**
2. **Cloud Broker**
    - A. Acts as an **interface** offering cloud services on behalf of a cloud operator.
    - B. In the past, **cloud brokers were crucial**, but now they are **less common** due to improved **self-service capabilities**.
3. **Cloud Broker Roles:**
    - A. Aggregates services from multiple clouds.
    - B. Less relevant today since cloud providers now offer direct self-service.
    - C. **Example:** AWS provides its own cloud services without external brokers.
4. **Cloud Balancer**
    - A. Functions as a **service arbitrator**.
    - B. Manages communication and resource distribution between cloud providers and users.
5. **Cloud Auditor**
    - A. Conducts audits for **quality assurance, security compliance, and performance benchmarks**.
    - B. Auditors must be **independent** and not **internally controlled by the cloud provider** to ensure security and reliability.
6. **Cloud Carrier**
    - A. Acts as an **ISP**, delivering cloud services to end users.
7. **Cloud Provider**
    - A. Responsible for managing **hardware resources and service orchestration**.

Cloud Orchestration & Management

Cloud orchestration involves **automating the deployment, coordination, and management** of cloud resources.

Orchestration Levels:

1. **Cloud Service Provider (CSP) Level**
    A. Manages large-scale cloud infrastructure.
    B. Handles automation of cloud resources.
2. **User Level**
    A. Automates individual functions within the cloud system.

Security & Privacy

- Implemented at **all layers**:
    - **Hardware**
    - **Network**
    - **Software**
    - **User Interfaces**
- Ensures **secure data transmission and access control**.

Cloud Service Management Layers

1. **Business Support**: Provides services and **billing automation**.
2. **Provisioning & Configuration**: Helps in **cloud deployment and workload management**.
3. **Portability**: Enables **migration between cloud providers**.
4. **Interoperability**: Ensures compatibility between different **cloud platforms**.

Advantages & Disadvantages of Cloud Computing

Advantages

1. **Lower Infrastructure & Hardware Costs**
    A. Reduces the need for upfront capital investment.
    B. Organizations save money by using **pay-as-you-go** pricing models.

2. **Scalability**
    A. Cloud resources can expand or contract **on-demand**.
3. **Security & Compliance**
    A. **Sovereign clouds** offer high security and privacy compliance.
    B. Large-scale cloud operators invest in **robust security measures**.
4. **Efficient Resource Utilization**
    A. Large-scale cloud providers optimize **performance and storage**.

Disadvantages

1. **Handling Scale**
    a. Cloud adaptation is growing **exponentially**, leading to **scalability challenges**.
2. **Cost Management**
    a. Operational costs can **increase over time** if not managed properly.
3. **Privacy & Security Concerns**
    a. Despite major investments, cloud environments still face **security threats**.
4. **Distributed System Challenges**
    a. Network latency, system failures, and **data consistency issues** are common in distributed cloud architectures.

Virtualization

1. Virtualization Overview

- Virtualization can be implemented **natively** or **without** cloud computing.
- Most **cloud computing infrastructures rely on virtualization** since it is a **common characteristic** that helps achieve essential cloud features.
- **Purpose of Virtualization:**
    - Efficient utilization of key resources.
    - Provides **abstraction** by modeling hardware or software environments.
    - Enables the creation of **interfaces** that simulate physical components.
    - Utilized by **hypervisors** to create and manage virtual machines.

2. Benefits of Virtualization

- **Resource Management:** Optimizes computing power, storage, and network resources.
- **Ease of Migration:** Virtual machines (VMs) can be migrated easily compared to physical machines.
- **Flexibility:** Virtualization allows multiple applications to run on a single physical resource.
- **Cost Savings:** Reduces the need for physical hardware.

3. Levels of Virtualization

Virtualization can be implemented at different levels in cloud computing:

1. **Server/Machine Virtualization**
   A. Enables multiple VMs to run on a single **physical machine**.
   B. Abstracts hardware resources.
2. **Network Virtualization**
   A. Allows multiple **virtual networks** to operate on shared **physical infrastructure**.
   B. Example: **Software-defined networking (SDN)**.
3. **I/O Virtualization**
   A. Virtualizes input/output devices like **storage, network interfaces, and disk controllers**.

4. Virtualization Approaches

Virtualization involves creating **virtual objects** in different ways:

A. Multiplexing

- **Definition:** Creates multiple **virtual objects** from a single **physical resource**.
- **Example: Processor Virtualization** (One CPU appears as multiple virtual processors).

- **OS-Level Multiplexing:** The OS distributes processor time among multiple tasks, giving an illusion of parallel execution.

B. Aggregation

- **Definition:** Combines multiple **physical objects** to create a **single virtual object**.
- **Example: RAID (Redundant Array of Independent Disks)**
    - Combines multiple storage disks into one **logical storage unit**.
    - Provides redundancy for fault tolerance.
- **Diagram Present in Notes:**
    - A simple RAID structure showing multiple disks aggregated into a single logical storage entity.

C. Emulation

- **Definition:** Creates a **virtual object** from a **different type of physical object**.
- **Example: Virtual Memory**
    - The OS **uses disk storage as an extension of RAM**.
    - Allows systems to run applications that require more memory than physically available.

5. Types of Virtualization

A. Hardware-Level Virtualization

- **Large-scale virtualization** used in cloud environments.
- **Type 1 Hypervisor (Bare Metal)**
    - Runs **directly on hardware**.
    - Highly **scalable and efficient**.
    - Examples: **VMware ESXi, KVM, Xen**.
- **Type 2 Hypervisor (Hosted)**
    - Runs **on top of an existing OS**.
    - Less scalable but easier to deploy.
    - Examples: **VirtualBox, VMware Workstation**.
- **Diagram Present in Notes:**
    - Shows a **comparison** between Type 1 and Type 2 hypervisors.

B. System-Level Virtualization

- Allows multiple **operating systems** to run on a **single hardware system**.
- Example: **VirtualBox, VMware Workstation**.
- **Diagram Present in Notes:**
    - Illustrates the relationship between **Guest OS, Host OS, and Hardware** in system-level virtualization.

C. Application-Level Virtualization

- Virtualizes **software applications**, allowing them to run **independently** of the underlying OS.
- Example: **JVM (Java Virtual Machine)** runs Java applications across different platforms.

6. Nested & Memory Virtualization

- **Memory Virtualization** allows the OS to use part of a **physical disk** as virtual memory.
- **Nested Virtualization**:
    - Virtualization **within** virtualization (e.g., running a **Type 2 hypervisor inside a Type 1 hypervisor**).
    - Can degrade performance but is useful for **testing environments**.

7. Network Virtualization

- Inspired by traditional **hardware network functions**.
- Allows **networking components (switches, routers, firewalls, etc.)** to run as **virtual instances**.
- **Infrastructure as a Service (IaaS)** enables users to create virtual networks.
- **Examples:**
    - **VNF (Virtual Network Function)**
    - **NFV (Network Functions Virtualization)**
- **Diagram Present in Notes:**

    - Shows multiple **logical networks** running on shared **physical infrastructure**.

Hypervisors

1. Hypervisor Overview

A **hypervisor** is a software layer that allows multiple virtual machines (VMs) to run on a single physical machine by **abstracting hardware resources**.

Responsibilities of a Hypervisor

- **Process Isolation**: Ensures that VM processes do not interfere with each other's memory space (prevents buffer overflow).
- **Resource Management**: Allocates CPU, memory, and storage to VMs efficiently.
- **Virtual Machine Execution**: All VMs are created and managed on top of a hypervisor.
- **Hardware Access Control**: Ensures that instructions requiring hardware access execute under the hypervisor's control.
- **Operating System Modes**:
    - **Privileged Mode**: System-level access.
    - **Non-Privileged Mode**: Application-level access.

2. Key Features of Hypervisors

A. VM Process Isolation (Kernel-Level)

- Prevents VMs from **overlapping or interfering** with each other's memory or processing space.

B. Device Mediation & Access Control

- **Hypervisors emulate hardware devices**, allowing VMs to interact with peripherals.
- Example: When using **VirtualBox** or **VMware Workstation**, pressing a key inside the VM is handled by the hypervisor.

C. Direct Execution of Commands from VMs

- **Privileged Instructions** from guest VMs (e.g., hardware access) must go through the hypervisor.
- **Type 1 Hypervisors** execute commands directly.
- **Type 2 Hypervisors** depend on the base OS for execution.

D. VM Lifecycle Management

- Manages **creation, backup, migration, and deletion** of virtual machines.
- **Live Migration**: Moving a running VM from one server to another without downtime.
- **Snapshot & Backup**: VMs can take periodic snapshots to restore state in case of failures.
- **Data Retention Policies**: Every cloud provider defines policies for VM data storage and backup.

E. Hypervisor Platform Management

- **Updates & Version Control**: Hypervisors receive periodic updates for security and performance improvements.
- **Management Interfaces**:
  - **CLI (Command Line Interface)**: Preferred for large-scale deployments.
  - **Web Interfaces**: Available for ease of use.
- **Security Considerations**:
  - **Type 1 Hypervisors** are more secure since they run directly on hardware.
  - **Type 2 Hypervisors** introduce additional risk due to dependency on the host OS.

3. Types of Hypervisors

Hypervisors are classified based on how they access hardware:

A. Type 1 Hypervisor (Bare Metal)

- Runs **directly on the physical hardware** without an underlying OS.
- **More efficient, secure, and scalable**.
- Examples:
    - **VMware ESXi**
    - **KVM (Kernel-based Virtual Machine)**
    - **Xen**
- **Used in:** Large-scale cloud environments.

B. Type 2 Hypervisor (Hosted)

- Runs **on top of an existing OS**, which manages hardware access.
- **Less efficient and secure** due to OS dependency.
- Examples:
    - **VMware Workstation**
    - **VirtualBox**
- **Used in:** Testing and development environments.

Comparison Diagram in Notes:

- Illustrates **Type 1 Hypervisor running directly on hardware** vs. **Type 2 Hypervisor relying on an OS**.

4. Virtualization Models

There are two primary models for virtualization:

A. Full Virtualization

- **Hypervisor exposes the same hardware interface** to the virtual machine as it does in physical hardware.

- **OS inside VM does not need modification**.
- **Higher overhead** due to full emulation.

B. Para-Virtualization

- **Hypervisor provides a specialized interface** that does not exist in physical hardware.
- **Guest OS must be modified** to communicate with the hypervisor.
- **Performance optimization** due to lower overhead.

Comparison Table in Notes:

Feature	Full Virtualization	Para-Virtualization
Hardware Interface	Same as physical	Optimized for VM
OS Modification	Not required	Required
Performance	Lower due to emulation	Higher due to direct hypervisor interaction

5. Hypervisors Used in Cloud Computing

- **Xen Hypervisor**:
  - Used by **AWS** (with modifications).
  - **AWS Nitro Hypervisor** is based on Xen.
  - Supports both **para-virtualization and full virtualization**.
  - Used in **ARM processors** for mobile and embedded systems.
- **KVM (Kernel-based Virtual Machine)**
  - Open-source Linux hypervisor.
  - **Technically a Type 1 hypervisor** but runs alongside the Linux OS.
  - Allows direct access to hardware.
  - **Default hypervisor in Linux-based cloud environments**.
- **VMware ESXi**
  - A **commercial Type 1 hypervisor** developed by VMware.
  - **Bare-metal installation** for cloud computing.
  - Expensive licensing but widely used in enterprise environments.

Diagram Present in Notes:

- Shows **KVM architecture**, depicting **VMs running on a Linux kernel hypervisor**.

6. SWOT Analysis of Hypervisors

A SWOT (Strengths, Weaknesses, Opportunities, Threats) analysis was conducted to assess hypervisors' role in cloud computing.

Strengths

- **High scalability** with Type 1 hypervisors.
- **Efficient resource allocation** and isolation.
- **Support for live migration and snapshots**.

Weaknesses

- **Type 2 hypervisors depend on the host OS**, making them less efficient.
- **Security vulnerabilities in Type 2 hypervisors** due to OS dependency.

Opportunities

- **Increased adoption of open-source hypervisors** (e.g., KVM) in cloud computing.
- **Integration with AI and automation** for smarter VM management.

Threats

- **Cybersecurity risks** in cloud environments.
- **Vendor lock-in** when using proprietary hypervisors like VMware ESXi.

Containers & Kubernetes

1. Containers

Overview

- **Inspired by application-level virtualization**.
- **Difference from Virtual Machines (VMs):**
    - **VMs** contain a full-fledged OS and take up several GBs.
    - **Containers** are lightweight, providing an **isolated environment** to run specific applications.
- **Diagram in Notes:**
    - Shows a **comparison of containers and VMs**, with VMs running on a hypervisor and containers running on a container engine.

Advantages of Containers

- **Smaller Footprint**: Consumes fewer resources than traditional VMs.
- **Faster Execution**: Containers spawn quickly compared to VMs.
- **Efficiency**: Instead of running an entire OS, containers **share the host OS kernel**.

Containerization Evolution

- **Previously**: Workloads were executed using **Virtual Machines**.
- **Now**: Workloads are executed in **Containers**.
- **Popular Container Platforms**:
    - **Docker**: Leading containerization technology.
    - **Open Container Initiative (OCI)**: Standardization of container runtimes.

Container Optimization

- Optimization depends on the **container engine**, which impacts:
    - **Performance**
    - **Security**
    - **Scalability**
- **Orchestration Tools**:
    - **Kubernetes**
    - **Docker Swarm**
- **Elasticity**:

    - Containers are **more elastic** than virtual machines.
    - They can be created and destroyed rapidly.

Challenges with Containers

- **Security Issues**:
    - Containers have a **short lifespan**, making security auditing and investigations difficult.
- **Tracking Issues**:
    - Short-lived containers make **monitoring and logging** challenging.

2. Cloud-Native Applications (CNA)

Definition

- Applications designed **specifically for cloud environments**.
- Built using **containers and microservices**.

Cloud-Native Characteristics

- **Highly Scalable**
- **Flexible**
- **Fault-Tolerant**
- **Lightweight & Efficient**
- **Supports Continuous Deployment**

Microservices Architecture

- **Modular Approach**: Applications are split into **small, independent services**.
- **Cloud-Optimized**: Unlike monolithic applications, microservices leverage cloud elasticity.
- **Examples**:
    - **YouTube**
    - **Netflix**
    - **AWS Cloud Services**

Diagram in Notes:

- Illustrates **Cloud-Native Architecture**, showing the breakdown into **microservices, containers, and Kubernetes**.

3. Kubernetes

Overview

- Kubernetes is a **container orchestration platform** that **manages and scales containers** across multiple machines.
- **Inspired by Google's internal Borg system**.

Kubernetes Architecture

- **Control Plane**:
    - Manages and schedules workloads across worker nodes.
- **Worker Nodes (Computing Plane)**:
    - Run the actual containerized workloads.

Diagram in Notes:

- Shows a **Kubernetes cluster** with:
    - **Control Plane**
    - **Worker Nodes**
    - **Pods running containers**

Key Kubernetes Components

1. **Node**:
    A. A machine (physical or virtual) where workloads run.
2. **Pod**:
    A. Smallest deployable unit in Kubernetes.
    B. Can contain **one or more containers**.
3. **Scheduler**:
    A. Distributes workloads across nodes.
4. **etcd**:
    A. Persistent storage (key-value store for configuration data).
5. **API Server**:

    A. Manages API requests.
6. **Controller Manager**:
    A. Maintains the state of Kubernetes objects.
7. **Kube-proxy**:
    A. Manages network communication between pods.

Diagram in Notes:

- Depicts the **Kubernetes Control Plane and Components**.

4. Kubernetes & Docker

Docker vs. Kubernetes

- **Docker**:
    - Creates and runs containers.
- **Kubernetes**:
    - Manages, orchestrates, and scales containers across clusters.

Kubernetes Managed Services

- **Amazon Elastic Kubernetes Service (EKS)**
- **Azure Kubernetes Service (AKS)**
- **Google Kubernetes Engine (GKE)**

Kubernetes Deployment on Cloud

Two Approaches

1. **System Build (Manual Deployment)**
    A. Build nodes (master, slave/worker).
    B. Not auto-scalable.
2. **Kubernetes as a Service (Managed)**
    A. Fully managed service that provides:
        i. **Cluster provisioning**
        ii. **Elastic scaling**
        iii. **Auto-management**

Kubernetes Scalability

- Kubernetes **adds worker nodes dynamically** as needed.
- In research-based tasks, **Kubernetes as a Service may not be suitable** because it bypasses certain configurations.

Security & Logging Considerations

- Since containers **spawn and terminate quickly**, **logging is crucial** for tracking container events.
- **Red Hat Kubernetes** provides **enterprise-level** Kubernetes solutions.

AWS & Virtual Private Cloud (VPC)

1. Amazon Web Services (AWS)

Overview

- **AWS** is the **largest public cloud service provider**.
- Provides **Infrastructure as a Service (IaaS)**, allowing users to build **entire cloud-based infrastructure**.

Key Benefits of AWS

1. **Security** – Ensures data protection and access control.
2. **Availability** – Highly redundant infrastructure with global coverage.
3. **Performance** – Optimized cloud performance with scalability.
4. **Scalability** – Allows seamless growth and resource provisioning.
5. **Flexibility** – Multiple configurations for different workloads.
6. **Global Footprint** – Data centers distributed worldwide.

Security & Availability Considerations

- Security is the **top priority** since AWS is a **public cloud**.
- **Availability** is an attribute of security but is **listed separately** because:

    - The region where a service is available impacts **privacy and performance**.
- **Latency is a key benchmark** for IoT-based applications on AWS.

Edge & Fog Computing in AWS

- **Edge Computing** reduces latency by processing data closer to users.
- **Fog Computing** acts as a middle layer between cloud servers and edge devices, improving real-time processing.
- **Used for IoT applications** requiring real-time processing.

Scalability & Resource Management

- **Over-provisioning** ensures resources are always available when needed.
- Scalability **has limits**, making it a **classic challenge in cloud computing**.
- **Admission Control** enforces **resource limits per user**.
- **Flexibility** in AWS includes choosing **data center locations** and **storage options**.

Global Infrastructure of AWS

- AWS **data centers** are typically located near **sea ports** for high-speed **fiber-optic connectivity**.
- **Networking backbone** consists of **Wide Area Network (WAN) and Local Area Network (LAN)** for **efficient data transfer**.

2. Virtual Private Cloud (VPC)

What is a VPC?

- **VPC (Virtual Private Cloud)** is a **logically isolated section** within AWS, providing **dedicated cloud space for clients**.
- **Multi-Tenant Architecture**:
    - Multiple users can use AWS, each assigned **their own VPC**.
    - Users **can have multiple VPCs** based on **billing and workload requirements**.

Diagram Present in Notes:

- Shows a **VPC with a VM** inside it, demonstrating logical isolation.

VPC Hierarchy

1. **VPC → Virtual Machines (VMs) → Operating System (OS) → Platform → Applications**
2. **For IaaS:** The cloud provider **allocates a VPC** and users build their infrastructure.
3. **For PaaS/SaaS:** The stack is managed by the cloud provider.

Logical Isolation in AWS

- **Hypervisors manage VM isolation**.
- **Network Policies & Access Controls** ensure isolation within a VPC.
- **Logical isolation** is enforced, but **physical isolation is difficult**.
- **VPCs from different clients must be isolated**.
- **Same IP addresses can be used by different clients** since they are only valid within their **VPC scope**.

3. AWS Compute Services: Elastic Compute Cloud (EC2)

Overview

- **AWS EC2** provides virtual machines (VMs) in the cloud with **on-demand resource allocation**.
- **Features:**
  - **VM Creation & Management** – Automated provisioning.
  - **Compute Power (CPU, RAM, Storage)** – Configurable based on workload.
  - **Pre-Built OS Images** – OS is already installed and can be attached to VMs.
  - **Network-Based Storage** – Uses AWS **Storage Servers** for backups.

Diagram Present in Notes:

- Shows **two VMs with OS and storage connected to a network-based storage server**.

EC2 Variants

1. **Standard EC2** – General-purpose virtual machines.
2. **GPU-Enabled EC2** – Specialized VMs with **dedicated GPU resources**.
3. **Dedicated EC2** – Provides **physical isolation**, bypassing the hypervisor.
4. **HPC EC2** – High-performance computing for **massive parallel workloads**.

AWS EC2 Pricing Models

- **Auctioned EC2 Instances**:
  - Allows **bidding on spare EC2 capacity**.
  - **Cost-effective** for non-critical workloads.
  - **Highest bid wins** the instance.
- **Billing Variations**:
  - **Pricing varies by region** due to energy and infrastructure costs.
  - **Different services have different pricing** based on the region.

EC2 Deployment Considerations

- **Users specify where their EC2 machine is created** (region selection).
- **Operators decide which physical server** to assign within that region.
- **Schedulers manage** the resource allocation from a pool of servers.

AWS Infrastructure Services

When a machine is created, AWS handles:

1. **Storage Data Placement** – Decides the exact physical server for data.
2. **Compute Services (EC2)**
3. **Network Services** – Assigns **IP addresses**.
4. **Identity & Access Management (IAM)** – Manages user roles and authentication.
5. **Storage Services** – Provides OS images and backup options.

Diagram Present in Notes:

- Shows **AWS infrastructure flow**, linking **IAM, EC2, Network, and Storage Services**.

AWS Service Dependencies

- Some services are **critical** for machine deployment.
- **A machine cannot be created without**:
    - **OS (Storage Service)**
    - **Network Services (IP Address Assignment)**

1. Elastic IP Address

Overview

- **Elastic IP (EIP)** is a **static IP address** allocated from **AWS's pool of public IPs**.
- It is **not tied to a single EC2 instance** but can be reassigned.
- When a machine is created, it receives an **IP assigned via DHCP**, which **changes over time** unless an **Elastic IP** is assigned.

Elastic IP Functionality

- **Used when a fixed IP address is required**, such as for **web hosting or load balancing**.
- **Diagram Present in Notes**:
    - Shows **EC2 instances with dynamic IPs** connected to a **load balancer** with an **Elastic IP assigned to the account**.

Key Points

- Web servers (EC2) have **dynamically assigned IPs**.
- Load balancers distribute traffic and manage the flow between **users and EC2 instances**.
- Elastic IP is **static** and remains fixed even if an instance is restarted.
- **Auto-scaling ensures availability**, creating new instances as needed.

2. Database Services

Overview

- AWS provides **multiple database solutions**, including **managed and self-hosted options**.
- **Key Database Types**:
  - **Relational Databases (RDS)** – Structured storage with SQL support.
  - **Non-Relational Databases (NoSQL)** – Schema-less, scalable databases.
  - **Database Migration Services** – Used for migrating data between databases.

3. Storage Services

Types of File Storage in AWS

1. **Network-Based Storage**:
   A. Distributed File System (DFS) ensures scalability and redundancy.
   B. Multiple storage options exist in **AWS VPC**.
2. **Object Storage (S3)**:
   A. Stores data as objects rather than files.
   B. **OS cannot be installed on S3**, as it is used for **storing backups, media, and application data**.
   C. **Used as Platform as a Service (PaaS)**.
   D. Capacity, access speed, and **pricing models vary**.
3. **Block Storage (EBS)**:
   A. Used for persistent storage on **EC2 instances**.
   B. Supports **OS installation**.
   C. Processing to retrieve blocks can be **slower compared to direct disk access**.
4. **Archival Storage (Glacier)**:
   A. Designed for **long-term data retention**.
   B. **Very low cost**, but **high latency** for retrieval.

Diagram Present in Notes:

- Shows **different AWS storage types, including Object, Block, and Archival storage**.

4. AWS Networking

Overview

- AWS provides a **robust networking infrastructure** to connect and manage cloud resources.
- Networking in AWS consists of:
    - **Backbone Networks** – Data center interconnectivity using **fiber-optic infrastructure**.
    - **Virtual Networks** – User-defined networks within AWS.

Key Components

- **VPN (Virtual Private Network)**:
    - Extends **on-premises networks** to AWS securely.
    - Makes users part of a **private cloud** setup.
- **VPC (Virtual Private Cloud)**:
    - Provides **logical network isolation** for AWS users.
    - **Allows multiple VMs to run within a secured environment**.
- **Virtual Routers & Subnets**:
    - AWS **Virtual Router (V. Router)** handles internal network traffic.
    - **Subnetting** allows segmentation within a VPC.

Diagram Present in Notes:

- Shows **VPC with connected VMs, Virtual Routers, and external networking infrastructure**.

5. AWS Deployment & Administration

Automation in AWS

- Infrastructure **can be automated** for deployment.
- **Key AWS Automation Tools**:
    - **AWS CloudFormation** – Infrastructure as Code (IaC).

- **AWS OpsWorks (Chef-based automation)** – Configuration management.

Monitoring Services

- **AWS CloudTrail** – Tracks **API requests** and logs actions.
- **AWS CloudWatch** – **Real-time monitoring** of AWS resources.

Service Level Agreements (SLAs)

- Defines the **level of service** between **AWS and the user**.
- Responsibilities differ **based on service type (IaaS, PaaS, SaaS)**.

6. AWS Networking Firewalls

Overview

- AWS **blocks all external access** by default.
- **Firewalls are necessary** for EC2 instances and other AWS resources.

Key Components

- **Firewall Policies**:
    - **Users control firewall rules** via **AWS Security Groups** and **Network Access Control Lists (NACLs)**.
- **SSH Security**:
    - Transport Layer Security (TLS) ensures **secure access to AWS resources**.
    - **Users are responsible** for setting up firewall rules.

AWS Content Delivery Networks (CDN)

- AWS **CloudFront** speeds up **content delivery globally**.
- **CDN provides region-based content delivery** for performance and compliance.

7. Deployment Models in AWS

Key Points

- Deployment can be **automated or manual** (via scripts).
- Multiple service configurations exist:
  - **2-Tier & 3-Tier Architectures**.
  - **Customized service models** based on use case.

Performance & Reliability Considerations

- **System performance and reliability** are major concerns for cloud deployment.
- **High reliability can only be provided by large cloud service providers (CSPs)**.

OpenStack

OpenStack is an **ecosystem** that allows for the construction of cloud environments using open-source software.

Cloud Construction Types

Using OpenStack, multiple types of cloud environments can be created:

- **Public Cloud**
- **On-premises Cloud (Private Cloud)**
- **Edge Cloud** (Created at the edge for IoT, Telecom systems)

Key Features

- **Open-source and free**
- **Commercialized versions** also exist
- **Apache Cloud Stack** serves as an alternative to OpenStack, along with **Open Nebula**.
- **Base metal, virtual machines, and containers** can be handled via OpenStack.

OpenStack and AWS

- AWS provides **bare metal as EC2 instances** along with other services similar to OpenStack.
- **The world runs on OpenStack:**
    - **Case Studies**: Exploring organizations using OpenStack.
    - **OpenInfra**: A project of OpenStack.
    - **Contributors**: NASA, Rackspace Cloud.
    - **Use cases** of OpenStack across industries.

OpenStack Deployment

- A cloud can be built on OpenStack, and additional services and features can be integrated.
- Deployment can be **automated** as discussed in prior sections.
- **OpenStack is primarily an IaaS provider.**
- OpenStack can be:
    - Downloaded as **open-source software**.
    - Acquired as **vendor-specific OpenStack solutions** (e.g., **RedHat OpenStack, Ubuntu OpenStack**).
- **Installation Requirements:**
    - **Multiple Machines** or
    - **DevStack (Single Machine Setup)**: A script used to construct a cloud on a **laptop** (usually not feasible for production but useful for **testing purposes**).

OpenStack Architecture

1. Dashboard Service

- Provides a **web interface** for managing OpenStack.
- Apache-based, designed for **end-users and administrators** to access control.

2. Compute Service

- Manages **VM lifecycle**.
- **Hypervisor**: **KVM** (default, but AWS uses different hypervisors).

- **Responsibilities:**
    - Distributes **resources across machines**.
    - Manages **VM scheduling decisions**.
- OpenStack's **compute service can be modified**, but AWS compute services cannot.

3. Networking Service

- Manages networking functionalities:
    - Firewall rules
    - IP assignment
    - VPN routing
- Uses **Open vSwitch & SDN controllers** for **DHCP, routing, and network firewall management**.

4. Storage Service

- Manages **distributed file systems**.
- Can be used as:
    - **A network service**
    - **An object service**
- Used to **build storage-as-a-service** in the cloud.
- **Supports multiple types of storage:**
    - **Block storage**
    - **Object storage** (default offerings).

5. Identity Service

- Provides **authentication and authorization** for OpenStack services and users.
- **Ensures isolation** in IaaS cloud:
    - **Access Control**
    - **Compute Service Isolation (Hypervisor-based)**
    - **Network-Level Isolation**
- Identity services can be modified for security enhancements.

6. Image Service

- Handles **VM image creation and management**.

- Similar to **Amazon Machine Images (AMIs)**.

7. Orchestration Service

- Automates the **management and scaling** of multiple machines.

8. Monitoring/Telemetry Service

- Collects metrics and performance data for **monitoring OpenStack services**.

9. Database Services

- Supports:
  - **Relational Databases**
  - **Non-Relational Databases**
  - **Migration between databases**

Virtual Machine Lifecycle in OpenStack

- When creating a **VM**, several services are involved:
  - **Telemetry/Monitoring Dashboard**
  - **Identity Service**
  - **Compute Service**
  - **Image Service**
  - **Networking Service**
  - **Storage Orchestration**
- These services operate on a **distributed system** and must be coordinated properly.
- **Issues that can occur:**
  - **Network service failure** → Machine is created but has **no IP**.
  - **Orchestration & Dashboard are optional**, but identity and image services **are necessary**.
  - **Storage service failure** → Machine is created, but **no storage** is allocated.

Master-Slave Architecture in OpenStack

- **Cloud Controller** → **Master Node** (Component Management)
- **Compute Nodes, Network Nodes, Storage Nodes** function under master-slave architecture.

OpenStack Node Architectures

Three-Tier Architecture of OpenStack

1. **Controller Node**
    a. Manages overall OpenStack services.
    b. Handles authentication, APIs, and orchestration.
    c. Responsible for monitoring and scheduling.
2. **Compute Node**
    a. Runs virtual machines (VMs) using a hypervisor (e.g., KVM).
    b. Manages VM execution and resource allocation.
3. **Networking Node**
    a. Handles network connectivity for OpenStack.
    b. Manages virtual networking, routing, and IP assignments.

Note: A **minimum of three separate machines** is required for this architecture.

Two-Tier Architecture of OpenStack

1. **Controller Node**
    a. Combines API, identity services, orchestration, and monitoring.
2. **Compute + Networking Node**
    a. A single node manages both virtual machines and networking.

Example: If **3,000 machines** are used to construct a cloud, **90% of them** are utilized for compute nodes.

- **Storage as a Service Cloud:**
    - Most machines are allocated for storage.
- **DevStack:**
    - Consolidates all nodes into a **single physical machine**.
    - Used **only for testing** due to its **limited scalability**.

Conceptual vs. Physical Architecture

- **Conceptual/Logical Architecture**
    - Defines how a VM is launched.
    - Identifies services involved.
    - Focuses on distributed services.
- **Physical Architecture**
    - **Single Node:** Minimum of **1 Controller**.
    - **Two Nodes:** Minimum of **1 Controller + Compute/Networking**.
    - Defines roles of machines and how resources are allocated.

Components of OpenStack

1. **SQL Database Service**
    a. Stores metadata of the cloud (logs, services).
    b. **Not** a Database-as-a-Service (DBaaS).
2. **Message Queue Service**
    a. Ensures services **communicate via messaging**.
3. **Network Time Service**
    a. Synchronizes machine time.
    b. Uses **GMU** (Global Master Unit) to ensure all nodes share the same clock.
    c. The **Controller Node** acts as the **time source**.
4. **Identity Service**
    a. Provides **authentication** and **authorization**.
5. **Networking ML2 Plugin**
    a. Manages **Layer 2 networking** services.

Compute Nodes

- **KVM Hypervisor**
    - Default hypervisor used in OpenStack.
- **Open vSwitch**
    - Allows **unlimited VMs** to connect.
    - Software-based **network switch** used for networking and compute.

Network Nodes

- **L3 Agent**
    - Connects cloud machines to the external world.
    - Provides **routing**.
- **DHCP Agent**
    - Assigns **IP addresses** dynamically.
- **Open vSwitch**
    - Enables software-defined networking (SDN) in OpenStack.

Note: Many networking and compute components are merged in **small-scale architectures**.

Storage Service

- **Mandatory for OpenStack** (not optional).
- **Telemetry Agent**
    - Manages **billing, resource monitoring, and prediction**.
    - **Collects stats** from OpenStack nodes.

Privacy & Security in OpenStack

- **Red Interface (Network Management)**
    - Manages internal communication.
- **Yellow Interface (Compute Node Communication)**

- Used for **fast data transfers** when a new machine is created.
- **Separate Storage Network**
  - Enables **high-speed data transfer** between compute nodes and storage servers.

External Networking in OpenStack

- **External Network**
  - Provides routing for **connections to the external world**.
  - Worker and compute nodes are **not exposed** by default for security reasons.
- **Logical vs. Physical Network**
  - **Logical Network:** Manages all resources (SDN-based).
  - **Physical Network:** Hardware-level networking.

Diagrams Present in Notes:

- **Storage Network Diagram**
  - Shows interaction between **VMs, network storage, and compute nodes**.
- **External Network Diagram**
  - Explains how OpenStack nodes connect to **external services** securely.