Independent Skill Development Mission

Independent Skill Development Mission NXT
EMPOWERING FUTURES, SHAPING TOMORROW

ISDM (INDEPENDENT SKILL DEVELOPMENT MISSION

CLOUD COMPUTING SERVICES & DEPLOYMENT (WEEKS 10-12)

AWS EC2, AZURE VIRTUAL MACHINES, GOOGLE COMPUTE ENGINE

CHAPTER 1: INTRODUCTION TO CLOUD-BASED VIRTUAL MACHINES

1.1 What Are Cloud Virtual Machines?

Cloud Virtual Machines (VMs) are **computing instances hosted on cloud platforms**, allowing users to run applications, store data, and manage workloads **without physical hardware**. They provide **on-demand computing resources** with **scalability, flexibility, and cost-efficiency**.

- **Why Use Cloud-Based VMs?**

✅ **Eliminates upfront hardware costs** – No need to purchase or maintain physical servers.

✅ **Scalable** – Increase or decrease computing power as needed.

✅ **High Availability & Reliability** – Cloud providers ensure uptime and disaster recovery.

✅ **Pay-as-You-Go Model** – Users only pay for the resources they consume.

- **Major Cloud VM Providers:**

✓ **AWS EC2** – Amazon's Elastic Compute Cloud.

✓ **Azure Virtual Machines** – Microsoft's cloud-based VM service.

✓ **Google Compute Engine (GCE)** – Google Cloud's VM platform.

📌 **Example:**

- A **startup can launch an AWS EC2 instance** to host a website instead of buying expensive servers.

CHAPTER 2: AWS ELASTIC COMPUTE CLOUD (EC2)

2.1 What is AWS EC2?

- **AWS EC2 (Elastic Compute Cloud)** is **Amazon's cloud-based VM service** that provides **resizable computing capacity** in the cloud.
- Users can **launch, configure, and manage virtual machines (EC2 instances)** as per their needs.
- EC2 supports **Windows, Linux, and macOS operating systems**.

2.2 Key Features of AWS EC2

Feature	Description
Instance Types	General-Purpose, Compute-Optimized, Memory-Optimized, Storage-Optimized.
Elasticity	Automatically scales up/down based on demand.
Security	Uses AWS Identity & Access Management (IAM), Security Groups, and Firewalls.
Storage Options	Supports Amazon Elastic Block Store (EBS) and Amazon S3.

Networking	Uses Virtual Private Cloud (VPC) for secure networking.

📌 **Example:**

- **Netflix runs its streaming services on AWS EC2,** dynamically scaling instances based on traffic spikes.

2.3 AWS EC2 Pricing Models

Pricing Model	Description	Best Use Case
On-Demand	Pay for compute capacity by the hour/second.	Short-term, unpredictable workloads.
Reserved Instances	Commit to a 1- or 3-year term for lower pricing.	Long-term workloads with predictable usage.
Spot Instances	Bid for unused EC2 capacity at discounted rates.	Cost-sensitive applications like batch processing.

📌 **Example:**

- **A research company uses AWS Spot Instances** to run large-scale machine learning models at low cost.

CHAPTER 3: MICROSOFT AZURE VIRTUAL MACHINES (VMS)

3.1 What is Azure Virtual Machines?

🔹 **Azure Virtual Machines (VMs)** provide **on-demand computing power** in Microsoft's cloud.

🔹 Allows users to **deploy Windows and Linux virtual machines**

with **full administrative control**.

- Integrates with **Microsoft's ecosystem (Office 365, Active Directory, SQL Server)**.

3.2 Key Features of Azure Virtual Machines

Feature	Description
VM Series	General-Purpose, Compute-Optimized, GPU-Optimized, and Confidential VMs.
Scalability	Uses Azure Auto-Scaling and Azure Load Balancer.
Security & Compliance	Supports Azure Security Center and Active Directory integration.
Networking	Uses Virtual Networks (VNet) for private and public access.
Hybrid Cloud	Supports on-premise and cloud hybrid environments.

📌 **Example:**

- **Starbucks uses Azure VMs** for real-time analytics and personalized customer experiences.

3.3 Azure Virtual Machines Pricing Models

Pricing Model	Description	Best Use Case
Pay-As-You-Go	Pay per hour or second for flexible usage.	Startups and small businesses.
Reserved VM Instances	Lower pricing for long-term commitments.	Large enterprises with predictable workloads.

Spot VMs	Unused capacity offered at a discount.	Non-critical and fault-tolerant applications.

📌 **Example:**

- **A financial company uses Azure Reserved VMs** for cost-effective database management.

CHAPTER 4: GOOGLE COMPUTE ENGINE (GCE)

4.1 What is Google Compute Engine?

🔹 **Google Compute Engine (GCE)** is **Google Cloud's Infrastructure-as-a-Service (IaaS) offering**, providing **high-performance VMs with automatic scaling**.

🔹 Integrates with **Google AI, Kubernetes, and BigQuery**.

🔹 Uses **Preemptible VMs for cost savings**.

4.2 Key Features of Google Compute Engine

Feature	Description
Machine Types	General-Purpose, Compute-Optimized, Memory-Optimized.
Auto-Scaling	Adjusts VM capacity dynamically.
Networking	Uses Google Cloud VPC, Load Balancing, and Firewalls.
Security	Uses IAM roles, encryption, and Shielded VMs.
Custom Machine Types	Users can define custom CPU/RAM configurations.

📌 **Example:**

- **PayPal runs its fraud detection models** on GCE for high-speed analytics.

4.3 Google Compute Engine Pricing Models

Pricing Model	Description	Best Use Case
Sustained Use Discounts	Automatic discounts for long-running workloads.	AI/ML workloads, Data analytics.
Preemptible VMs	Short-lived, low-cost instances for batch jobs.	Batch processing, video rendering.
Committed Use Discounts	Discounts for committing to 1 or 3 years.	Enterprise applications with long-term use.

📌 **Example:**

- **A biotech company uses Preemptible VMs** to run DNA sequencing simulations at reduced costs.

CHAPTER 5: AWS EC2 VS. AZURE VMS VS. GOOGLE COMPUTE ENGINE

Feature	AWS EC2	Azure Virtual Machines	Google Compute Engine
Best For	Enterprises, e-commerce, and startups	Windows-based	AI, big data, and

		workloads, hybrid cloud	Kubernetes workloads
Instance Customization	Wide range of instance types	Strong Microsoft integration	Custom machine types
Pricing	On-demand, Reserved, Spot Instances	Pay-as-you-go, Reserved, Spot VMs	Sustained discounts, Preemptible VMs
Security	IAM, Security Groups, VPC	Active Directory, Security Center	Shielded VMs, IAM
Scalability	Auto-Scaling, Load Balancer	Azure Auto-Scaling, Load Balancer	Auto-Scaling, Kubernetes Integration

📌 **Example:**

- **An AI startup using Google Cloud AI services would choose GCE**, while **a Microsoft-based enterprise might prefer Azure VMs**.

Exercise: Test Your Understanding

🔹 **What are the key differences between AWS EC2, Azure VMs, and Google Compute Engine?**

🔹 **How does pricing differ across cloud VM providers?**

🔹 **What are some common use cases for Preemptible VMs?**

🔹 **Why might a business choose Azure Virtual Machines over**

AWS EC2?

- **How do cloud-based VMs enhance scalability and security?**

Conclusion

✅ **AWS EC2, Azure Virtual Machines, and Google Compute Engine provide flexible, scalable, and cost-efficient cloud-based VMs.**

✅ **Choosing the right provider depends on business needs, pricing, security, and integration with existing systems.**

✅ **Cloud VMs power businesses worldwide, supporting AI, analytics, web hosting, and enterprise applications.**

SERVERLESS COMPUTING (AWS LAMBDA, AZURE FUNCTIONS)

CHAPTER 1: INTRODUCTION TO SERVERLESS COMPUTING

1.1 What is Serverless Computing?

Serverless computing is a cloud computing execution model where **cloud providers manage the infrastructure,** and developers focus only on writing code. Unlike traditional server-based computing, serverless platforms automatically **scale, deploy, and manage the execution of applications** without requiring developers to handle server maintenance.

- **Key Characteristics of Serverless Computing:**

✅ **No Server Management** – The cloud provider handles infrastructure maintenance.
✅ **Event-Driven Execution** – Code runs **only when triggered** (e.g., HTTP requests, file uploads).
✅ **Auto-Scaling** – Scales up or down automatically based on demand.
✅ **Pay-Per-Execution Pricing** – Users pay only for the computing time consumed.

- **Example:**

  - A company uses **AWS Lambda** to **automatically resize images** uploaded to an S3 bucket without managing a dedicated server.

CHAPTER 2: HOW SERVERLESS COMPUTING WORKS

2.1 Traditional vs. Serverless Architecture

Feature	Traditional Computing	Serverless Computing
Infrastructure Management	Developers manage servers	Cloud provider manages infrastructure
Scalability	Requires manual scaling	Auto-scales based on demand
Cost Model	Pay for provisioned resources	Pay per execution
Execution	Always running	Runs only when triggered

📌 **Example:**

- A company running a **web application** on traditional servers **pays for idle resources**, while a **serverless app incurs costs only when used**.

2.2 How Serverless Computing Works?

1️⃣ **Trigger Event:** A request is made (e.g., API call, database change, file upload).

2️⃣ **Function Execution:** The cloud provider **provisions resources and runs the function**.

3️⃣ **Response & Auto-Scaling:** The function returns a response and **scales automatically** if needed.

4️⃣ **Shutdown & Cost Savings:** After execution, resources **are freed, reducing costs**.

📌 **Example:**

- **A chatbot function** on Azure Functions is triggered **only when a user interacts with it,** saving costs when inactive.

CHAPTER 3: KEY SERVERLESS COMPUTING SERVICES

3.1 AWS Lambda

🔹 **What is AWS Lambda?**

✅ A serverless computing service from **Amazon Web Services (AWS)** that runs code in response to events.

✅ Supports multiple languages (**Python, Node.js, Java, C#**).

✅ Integrated with **AWS S3, DynamoDB, API Gateway, CloudWatch**.

🔹 **How AWS Lambda Works:**

1. A **user uploads an image** to **Amazon S3**.
2. The **S3 event triggers an AWS Lambda function**.
3. AWS Lambda **resizes the image** and stores it back in S3.

📌 **Example:**

- A **news website** uses AWS Lambda to **convert articles into audio** when a user requests it.

3.2 Microsoft Azure Functions

🔹 **What is Azure Functions?**

✅ Microsoft's **serverless compute** service that executes **code on demand**.

✅ Supports event-driven triggers (**HTTP requests, file uploads, message queues**).

✅ Integrates with **Azure Blob Storage, Cosmos DB, Event Grid, Power BI**.

🔹 **How Azure Functions Work:**

1. A **new record is added to an Azure database**.
2. An **Azure Function is triggered**, sending a **notification email**.

📌 **Example:**

- An **e-commerce website uses Azure Functions** to **process orders in real-time** when a customer makes a purchase.

CHAPTER 4: BENEFITS & CHALLENGES OF SERVERLESS COMPUTING

4.1 Benefits of Serverless Computing

✅ **No Infrastructure Management** – Developers focus on code, not servers.
✅ **Automatic Scaling** – Handles traffic spikes without manual scaling.
✅ **Lower Costs** – Pay only for **actual function execution time**.
✅ **Faster Time-to-Market** – Rapid development and deployment.

📌 **Example:**

- **Spotify uses AWS Lambda** to process **millions of music streaming requests** efficiently.

4.2 Challenges of Serverless Computing

❌ **Cold Start Latency** – Functions take time to **start if inactive** for a while.
❌ **Vendor Lock-In** – Applications may **depend heavily on a cloud**

provider's ecosystem.

❌ **Debugging & Monitoring Complexity** – Harder to troubleshoot compared to traditional servers.

❌ **Limited Execution Time** – Functions **must complete execution within time limits** (e.g., AWS Lambda: 15 min).

📌 **Example:**

- A banking app **switched from AWS Lambda to Kubernetes** due to high execution time limits.

Chapter 5: Serverless Use Cases Across Industries

5.1 Industries Benefiting from Serverless Computing

Industry	Use Case
E-commerce	Auto-processing orders using AWS Lambda
Media & Streaming	Video encoding and content personalization
IoT & Smart Devices	Event-driven automation for IoT devices
Banking & Finance	Fraud detection using real-time triggers
Healthcare	Secure processing of medical data in cloud functions

📌 **Example:**

- **Netflix uses AWS Lambda** to generate **personalized recommendations** based on user activity.

CHAPTER 6: SERVERLESS COMPUTING VS. CONTAINERIZATION

Feature	Serverless Computing	Containers (Docker, Kubernetes)
Deployment Model	Function-based	Containerized applications
Scaling	Auto-scales per request	Scales in groups (pods)
Execution Time	Short-lived	Persistent & long-running
Use Case	Event-driven applications	Microservices & large applications

📌 **Example:**

- **A chatbot** uses serverless, while a **full-fledged enterprise app** runs on Kubernetes containers.

Exercise: Test Your Understanding

- **What is the difference between AWS Lambda and Azure Functions?**
- **List three benefits of serverless computing.**
- **How does auto-scaling work in serverless computing?**
- **When should you use serverless vs. containerization?**
- **What are the major security concerns in serverless computing?**

Conclusion

Serverless computing **eliminates infrastructure management,** providing a **scalable and cost-effective solution** for cloud applications.

✅ **AWS Lambda & Azure Functions enable developers to build event-driven applications.**

✅ **Serverless is best for real-time processing, automation, and lightweight applications.**

✅ **Despite challenges, serverless computing is revolutionizing cloud-based software development.**

AWS S3, Azure Blob Storage, Google Cloud Storage

Chapter 1: Introduction to Cloud Storage

1.1 What is Cloud Storage?

Cloud storage is a **service that allows users to store, manage, and access data remotely via the internet** instead of relying on physical storage devices. Leading cloud providers, such as **AWS, Microsoft Azure, and Google Cloud**, offer scalable and highly available storage solutions.

🔹 **Key Features of Cloud Storage:**

✅ **Scalability** – Expands storage capacity on demand.

✅ **Durability** – Provides redundancy to prevent data loss.

✅ **Security** – Offers encryption, IAM policies, and access control.

✅ **Cost-Effectiveness** – Pay only for the storage you use.

📌 **Example:**

- A **media company** uses AWS S3 to store **high-resolution video content**, ensuring global access with minimal latency.

Chapter 2: Amazon Web Services (AWS) S3

2.1 What is AWS S3?

Amazon Simple Storage Service (S3) is an **object storage service** that provides **scalability, security, and high availability** for storing any type of data. It is designed to store **large volumes of unstructured data** (documents, images, videos, backups, etc.).

🔹 **Key Features of AWS S3:**

✅ **Object Storage Model** – Stores data as objects in buckets.

✅ **High Durability** – 99.999999999% (11 nines) of data durability.

✅ **Lifecycle Management** – Automates data retention and deletion.

✅ **IAM Policies & Access Control** – Provides fine-grained permissions.

✅ **Data Encryption** – Supports server-side and client-side encryption.

2.2 AWS S3 Storage Classes

AWS S3 offers multiple **storage classes** optimized for cost and performance.

Storage Class	Use Case	Pricing
S3 Standard	Frequently accessed data	Higher cost, low latency
S3 Intelligent-Tiering	Data with unknown access frequency	Automatic tiering
S3 Standard-IA (Infrequent Access)	Data accessed less often	Lower cost, retrieval fee
S3 Glacier	Archival storage	Low cost, high retrieval time
S3 Glacier Deep Archive	Long-term archival (10+ years)	Cheapest, slow retrieval

📌 **Example:**

- **A company uses S3 Standard for real-time applications** and **S3 Glacier for long-term data retention**.

2.3 AWS S3 Security & Access Management

✅ **IAM Policies & Bucket Policies** – Restrict who can access S3 buckets.
✅ **Server-Side Encryption (SSE)** – Encrypts data at rest.
✅ **AWS CloudTrail** – Logs S3 access for security monitoring.
✅ **VPC Endpoints** – Securely connect S3 to a Virtual Private Cloud (VPC).

📌 **Example:**

- **A healthcare provider encrypts medical records in S3** to comply with **HIPAA security standards**.

CHAPTER 3: MICROSOFT AZURE BLOB STORAGE

3.1 What is Azure Blob Storage?

Azure Blob Storage is **Microsoft Azure's object storage service**, designed for unstructured data like text, images, videos, and backups. It enables **scalable, cost-effective** cloud storage with global accessibility.

🔹 **Key Features of Azure Blob Storage:**
✅ **Supports Unstructured Data** – Ideal for media, logs, and backups.
✅ **Tiers for Cost Optimization** – Offers hot, cool, and archive storage.
✅ **Integration with Azure Services** – Works with **AI, analytics, and machine learning** tools.
✅ **Geo-Redundant Storage (GRS)** – Ensures data availability even if a region fails.

3.2 Azure Blob Storage Tiers

Azure Blob Storage provides three main **tiers** based on access frequency.

Storage Tier	Use Case	Pricing
Hot Tier	Frequently accessed data	Highest cost, lowest latency
Cool Tier	Infrequently accessed data	Lower cost, retrieval fee applies
Archive Tier	Long-term archival	Cheapest, highest retrieval time

📌 **Example:**

- **An enterprise stores recent logs in the Hot Tier**, while **older logs move to the Archive Tier to reduce costs**.

3.3 Azure Blob Storage Security & Access Management

✅ **Role-Based Access Control (RBAC)** – Assigns permissions using **Azure Active Directory (Azure AD)**.
✅ **Encryption at Rest & in Transit** – Uses **AES-256 encryption** for data security.
✅ **Private Endpoints** – Restrict public access using **Azure Virtual Network**.
✅ **Immutable Storage** – Protects data from modifications (useful for regulatory compliance).

📌 **Example:**

- **A financial company uses immutable storage in Azure Blob Storage** to comply with **SEC financial regulations**.

CHAPTER 4: GOOGLE CLOUD STORAGE (GCS)

4.1 What is Google Cloud Storage?

Google Cloud Storage (GCS) is a **fully managed, scalable object storage service** for businesses of all sizes. It integrates with **Google's AI, big data, and machine learning tools** for enhanced analytics.

🔹 **Key Features of Google Cloud Storage:**

✅ **Multi-Regional Availability** – Stores data across multiple locations for redundancy.

✅ **Automatic Lifecycle Management** – Moves data to cost-effective storage classes.

✅ **Strong Consistency** – Ensures immediate visibility of newly written data.

✅ **Fine-Grained Access Controls** – Uses IAM roles and Object ACLs (Access Control Lists).

4.2 Google Cloud Storage Classes

GCS offers **four storage classes** with automatic tiering for cost optimization.

Storage Class	Use Case	Pricing
Standard	High-performance, frequently accessed data	Higher cost, low latency

Nearline	Data accessed <1/month	Lower cost, retrieval fee applies
Coldline	Data accessed <1/year	Very low cost, long retrieval time
Archive	Long-term data retention	Cheapest, slow retrieval

📌 **Example:**

- **A research institution stores daily backups in Standard storage** but **archives historical research in Coldline storage**.

4.3 Google Cloud Storage Security & Access Control

✅ **IAM Policies & Object ACLs** – Controls user access to objects.
✅ **Bucket Lock** – Prevents modification of critical data.
✅ **Cloud Audit Logging** – Monitors and logs all access events.
✅ **VPC Service Controls** – Protects data by defining secure perimeters.

📌 **Example:**

- **A media company uses IAM roles in GCS** to allow **editors to upload content** while restricting access for viewers.

CHAPTER 5: COMPARISON OF AWS S3, AZURE BLOB STORAGE, AND GOOGLE CLOUD STORAGE

Feature	AWS S3	Azure Blob Storage	Google Cloud Storage

Storage Model	Object Storage	Object Storage	Object Storage
Access Control	IAM Policies, Bucket Policies	RBAC, Azure AD	IAM, Object ACLs
Data Encryption	Server-side (SSE), Client-side	AES-256 Encryption	AES-256 Encryption
Lifecycle Management	Yes	Yes	Yes
Pricing Model	Pay-as-you-go	Pay-as-you-go	Pay-as-you-go
Best For	Scalable web hosting, backups	Enterprise cloud solutions, analytics	AI, ML, and Big Data processing

📌 **Example:**

- **A startup needing high-speed AI model training** chooses **Google Cloud Storage** for **low-latency access**.

Exercise: Test Your Understanding

- **What are the primary benefits of cloud storage?**
- **How does AWS S3 differ from Google Cloud Storage?**
- **What is the advantage of Azure Blob Storage's Cool Tier?**
- **List three security features provided by cloud storage services.**
- **Which storage class would you choose for long-term backups? Why?**

Conclusion

✅ **AWS S3, Azure Blob Storage, and Google Cloud Storage** provide **secure, scalable, and cost-effective cloud storage solutions**.

✅ **Different storage classes** help optimize **costs and access speeds**.

✅ **Security features** like **encryption, IAM policies, and network isolation** ensure **data protection**.

CLOUD DATABASE SERVICES (AWS RDS, GOOGLE FIRESTORE, AZURE COSMOS DB)

CHAPTER 1: INTRODUCTION TO CLOUD DATABASE SERVICES

1.1 What is a Cloud Database?

A **cloud database** is a database that runs on **cloud infrastructure**, offering **scalability, high availability, automated maintenance, and security**. Unlike traditional on-premise databases, cloud databases are **fully managed**, reducing the need for database administration.

🔹 **Key Features of Cloud Databases:**

✅ **Automated Backups & Recovery** – Ensures data safety.

✅ **Scalability** – Scale resources up or down based on demand.

✅ **Global Accessibility** – Accessible from anywhere via the internet.

✅ **Security & Compliance** – Encryption, identity management, and compliance with regulations.

🔹 **Common Types of Cloud Databases:**

- **Relational Databases (SQL-based)** – AWS RDS, Azure SQL Database.
- **NoSQL Databases (Document-based, Key-Value, Columnar)** – Google Firestore, Azure Cosmos DB.

📌 **Example:**

- A **retail e-commerce website** uses **AWS RDS for structured sales data** and **Google Firestore for customer recommendations**.

CHAPTER 2: AMAZON RDS (RELATIONAL DATABASE SERVICE)

2.1 What is AWS RDS?

🔹 **Amazon Relational Database Service (RDS)** is a **fully managed relational database** service that supports multiple database engines such as:

✅ **MySQL** – Open-source, widely used.
✅ **PostgreSQL** – Advanced SQL database with NoSQL capabilities.
✅ **MariaDB** – Fork of MySQL, optimized for speed.
✅ **Oracle** – Enterprise-grade database.
✅ **Microsoft SQL Server** – Used in corporate environments.
✅ **Amazon Aurora** – AWS-optimized relational database, faster than MySQL.

🔹 **Key Features of AWS RDS:**

✅ **Automated Backups** – Daily backups & snapshots.
✅ **Replication & Multi-AZ Deployment** – Ensures high availability.
✅ **Performance Monitoring** – Uses **AWS CloudWatch** for real-time insights.
✅ **Security** – Supports **VPC, IAM, and encryption**.

2.2 Use Cases of AWS RDS

Industry	Use Case Example
E-commerce	Storing customer orders & payment records.
Banking & Finance	Securely managing financial transactions.
Healthcare	Patient data management in a structured SQL database.

📌 **Example:**

- **Netflix** uses **AWS RDS for storing subscription data** and transaction records.

2.3 Pros & Cons of AWS RDS

✅ **Pros:**

✓ Supports multiple database engines.

✓ Fully managed – No manual administration needed.

✓ High security & compliance.

❌ **Cons:**

✕ Expensive for large-scale applications.

✕ Limited customization compared to self-managed databases.

📌 **Hands-on Lab:**

- Deploy an **AWS RDS MySQL instance** and connect it to a web application.

CHAPTER 3: GOOGLE FIRESTORE (NOSQL DOCUMENT DATABASE)

3.1 What is Google Firestore?

🔹 **Google Firestore** is a **serverless NoSQL document database** that provides real-time synchronization, scalability, and ease of use for modern applications.

🔹 **Key Features of Firestore:**

✅ **Document-Based Storage** – Uses **collections and documents** instead of tables.

✅ **Real-Time Data Sync** – Ideal for **live chat, social media apps**.

✅ **Serverless & Auto-Scaling** – No need for infrastructure management.

✅ **Offline Support** – Works in mobile apps even without an internet connection.

3.2 Use Cases of Google Firestore

Industry	Use Case Example
Mobile Apps	Real-time messaging & notifications.
Gaming	Storing player profiles and game states.
E-commerce	Customer reviews, product catalog storage.

📌 **Example:**

- **Spotify uses Firestore** to manage real-time user playlists and social features.

3.3 Pros & Cons of Google Firestore

✅ **Pros:**

✓ Real-time sync for dynamic applications.

✓ Easy to use and scale.

✓ No infrastructure management required.

❌ **Cons:**

✕ Not optimized for **complex SQL queries**.

✕ Expensive for **high read/write workloads**.

📌 **Hands-on Lab:**

- Create a **Firestore database** and store user profiles for a chat application.

CHAPTER 4: AZURE COSMOS DB (GLOBALLY DISTRIBUTED NOSQL DATABASE)

4.1 What is Azure Cosmos DB?

🔹 **Azure Cosmos DB** is a **fully managed, globally distributed NoSQL database** that offers multi-model capabilities with high availability.

🔹 **Key Features of Cosmos DB:**

✅ **Multi-Model Support** – Works as a **document, key-value, columnar, or graph database**.

✅ **Global Distribution** – Data replicates across multiple regions automatically.

✅ **Guaranteed Low Latency** – **<10ms response time** for read/write operations.

✅ **Strong Consistency Models** – Ensures high data reliability.

4.2 Use Cases of Azure Cosmos DB

Industry	Use Case Example
Retail & E-commerce	Product catalog & inventory management.
IoT & Big Data	Storing sensor data from IoT devices.
Finance & Banking	Fraud detection & financial transactions.

📌 **Example:**

- **Coca-Cola uses Cosmos DB** for **global customer insights** and analytics.

4.3 Pros & Cons of Azure Cosmos DB

✅ **Pros:**

✓ Multi-model database (Key-Value, Document, Columnar).

✓ High availability & low-latency performance.

✓ Multi-region replication.

❌ **Cons:**

❌ More expensive than traditional NoSQL databases.

❌ Requires tuning for optimal performance.

📌 **Hands-on Lab:**

- Deploy an **Azure Cosmos DB instance** and create a globally distributed NoSQL database.

Chapter 5: Comparing AWS RDS, Google Firestore, and Azure Cosmos DB

Feature	AWS RDS	Google Firestore	Azure Cosmos DB
Type	Relational (SQL)	NoSQL (Document-based)	NoSQL (Multi-model)
Use Case	Structured business data	Real-time apps, chat, analytics	Global distribution, IoT, AI apps
Scalability	Vertical scaling	Auto-scaling	Horizontal scaling
Best For	E-commerce, Banking	Mobile apps, Gaming	Global businesses, IoT

📌 **Example:**

- A **banking application** uses **AWS RDS for transactions**, while a **social media app** uses **Firestore for real-time posts and likes**.

Exercise: Test Your Understanding

- **What are the key differences between relational and NoSQL databases?**
- **Which cloud database is best suited for real-time applications and why?**
- **How does Cosmos DB achieve global distribution?**
- **When should a company use AWS RDS instead of Google Firestore?**
- **What are the advantages of using serverless databases like Firestore?**

Conclusion

Cloud database services **revolutionized data storage, management, and scalability** for businesses worldwide.

✅ **AWS RDS** is best for **structured, relational data** like financial transactions.
✅ **Google Firestore** is ideal for **real-time NoSQL applications** like messaging apps.
✅ **Azure Cosmos DB** offers **global scalability** for IoT, AI, and distributed systems.

INFRASTRUCTURE AS CODE (TERRAFORM, AWS CLOUDFORMATION)

CHAPTER 1: INTRODUCTION TO INFRASTRUCTURE AS CODE (IAC)

1.1 What is Infrastructure as Code (IaC)?

Infrastructure as Code (IaC) is the practice of **managing and provisioning cloud infrastructure using machine-readable configuration files** instead of manual processes. IaC enables **automation, consistency, and scalability** in deploying cloud environments.

🔹 **Why is IaC Important?**

✅ **Automation & Efficiency** – Deploys infrastructure quickly and reduces human errors.

✅ **Consistency & Repeatability** – Ensures identical configurations across multiple environments.

✅ **Version Control** – Allows tracking and rollback of changes using Git.

✅ **Scalability** – Easily scales infrastructure up or down as per demand.

🔹 **Popular IaC Tools:**

✓ **Terraform** – Open-source multi-cloud infrastructure management.

✓ **AWS CloudFormation** – AWS-native infrastructure provisioning tool.

📌 **Example:**

- A **DevOps engineer uses Terraform to automate the deployment of AWS EC2 instances** instead of manually setting them up.

CHAPTER 2: TERRAFORM – MULTI-CLOUD INFRASTRUCTURE MANAGEMENT

2.1 What is Terraform?

🔹 **Terraform** is an **open-source Infrastructure as Code tool** developed by **HashiCorp** that enables users to define cloud resources in a **declarative configuration language**.

🔹 Supports **multiple cloud providers** such as **AWS, Azure, Google Cloud, and Kubernetes**.

2.2 Key Features of Terraform

Feature	Description
Multi-Cloud Support	Works across AWS, Azure, GCP, and more.
Declarative Configuration	Users define the desired state, and Terraform ensures deployment.
State Management	Keeps track of infrastructure changes via Terraform State.
Resource Dependency Management	Automatically manages dependencies between cloud resources.
Modularization	Allows reusable infrastructure components using Terraform Modules.

📌 **Example:**

- **A startup uses Terraform to deploy an AWS VPC, EC2 instances, and S3 storage buckets in a single command.**

2.3 How Terraform Works?

1. **Write Infrastructure Code** – Define resources in a .tf file (Terraform configuration file).
2. **Initialize Terraform** – Run terraform init to set up the Terraform environment.
3. **Plan Changes** – Use terraform plan to preview resource changes.
4. **Apply Configuration** – Execute terraform apply to deploy infrastructure.
5. **Destroy Resources** – Use terraform destroy to remove infrastructure.

📌 Example Terraform Code to Create an AWS EC2 Instance:

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_instance" "web_server" {
  ami           = "ami-12345678"
  instance_type = "t2.micro"
}
```

📌 Example:

- **An e-commerce company uses Terraform to automatically deploy EC2 instances whenever traffic spikes occur.**

CHAPTER 3: AWS CLOUDFORMATION – AWS-NATIVE IAC TOOL

3.1 What is AWS CloudFormation?

- **AWS CloudFormation** is an **AWS-native Infrastructure as Code (IaC) service** that enables users to define and deploy **AWS resources** using JSON or YAML templates.
- **Automates infrastructure provisioning** within AWS while ensuring consistency.

3.2 Key Features of AWS CloudFormation

Feature	Description
AWS Integration	Works natively with all AWS services.
Infrastructure as Code	Uses YAML/JSON templates to define cloud resources.
Stack Management	Deploys, updates, and deletes resources as a unit.
Rollback Support	Reverts changes if an update fails.
Drift Detection	Identifies configuration changes that deviate from the CloudFormation template.

📌 **Example:**

- **A financial company uses CloudFormation to deploy AWS Lambda functions, API Gateway, and DynamoDB in one automated process.**

3.3 How AWS CloudFormation Works?

1️⃣ **Write a CloudFormation Template** – Define infrastructure in YAML or JSON format.

2️⃣ **Create a Stack** – Upload the template to AWS CloudFormation.

3️⃣ **Deploy Resources** – AWS provisions the resources based on the template.

4️⃣ **Update Stack** – Modify infrastructure by updating the template.

5️⃣ **Delete Stack** – Remove resources when they are no longer needed.

📌 **Example CloudFormation YAML Template to Create an EC2 Instance:**

```
Resources:
  MyEC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: "ami-12345678"
      InstanceType: "t2.micro"
```

📌 **Example:**

- **A software company uses CloudFormation to deploy identical AWS environments for development, testing, and production.**

CHAPTER 4: TERRAFORM VS. AWS CLOUDFORMATION – KEY DIFFERENCES

Feature	Terraform	AWS CloudFormation

Cloud Support	Multi-cloud (AWS, Azure, GCP, Kubernetes)	AWS only
Configuration Language	HashiCorp Configuration Language (HCL)	YAML/JSON
Resource State Management	Uses Terraform State	AWS manages resources internally
Modularity & Reusability	Highly modular with Terraform Modules	Limited reusability with CloudFormation Stacks
Rollback Support	Manually handled	Automatic rollback on failure
Pricing	Free (Open Source)	Free, but AWS charges for resources

📌 **Example:**

- **A company using both AWS and Azure would prefer Terraform, while an AWS-only business may choose CloudFormation.**

Chapter 5: Best Practices for Using IaC (Terraform & CloudFormation)

✅ **Use Version Control (Git)** – Track changes and collaborate on infrastructure code.

✅ **Modularize Infrastructure** – Use reusable modules to simplify deployment.

✅ **Implement Security Best Practices** – Encrypt sensitive data, limit IAM permissions.

✅ **Enable Logging & Monitoring** – Use AWS CloudTrail, Terraform State, and audit logs.

✅ **Test Before Deployment** – Run terraform plan or CloudFormation Change Sets.

📌 **Example:**

- **A DevOps team uses GitHub to manage Terraform infrastructure code and runs automated tests before applying changes.**

Exercise: Test Your Understanding

- **What is the purpose of Infrastructure as Code (IaC)?**
- **How does Terraform differ from AWS CloudFormation?**
- **Why is Terraform preferred for multi-cloud environments?**
- **What are CloudFormation stacks, and how do they simplify deployment?**
- **List three best practices for implementing Infrastructure as Code.**

Conclusion

✅ **Terraform and AWS CloudFormation are essential Infrastructure as Code (IaC) tools** for **automating cloud infrastructure deployment and management**.

✅ **Terraform supports multi-cloud environments**, making it flexible for AWS, Azure, and Google Cloud.

✅ **AWS CloudFormation is best for AWS users**, offering deep integration and automatic rollback features.

✅ **Using IaC enhances scalability, consistency, and security in cloud computing.**

CI/CD IN CLOUD (JENKINS, GITHUB ACTIONS)

CHAPTER 1: INTRODUCTION TO CI/CD IN CLOUD

1.1 What is CI/CD?

Continuous Integration (CI) and Continuous Deployment (CD) are modern software development practices that **automate the process of building, testing, and deploying applications** in cloud environments.

🔹 **CI (Continuous Integration):** Developers **frequently merge code** into a shared repository, triggering **automated tests** to ensure software quality.

🔹 **CD (Continuous Deployment/Delivery):** After successful testing, code is **automatically deployed** to production or staging environments **without manual intervention**.

🔹 **Key Benefits of CI/CD:**

✅ **Faster Software Releases** – Reduces time between code commits and deployment.

✅ **Automated Testing** – Catches errors early, improving software quality.

✅ **Scalability** – Supports cloud-based, microservices-driven applications.

✅ **Consistency** – Ensures stable and repeatable deployments.

📌 **Example:**

- A **fintech startup uses GitHub Actions** to automate testing and deployment for its web application, reducing release times from **days to hours**.

CHAPTER 2: CI/CD PIPELINE IN CLOUD COMPUTING

2.1 How CI/CD Pipelines Work?

A **CI/CD pipeline** automates the **steps of integrating, testing, and deploying software** using cloud-based tools.

🔹 **CI/CD Pipeline Stages:**

1️⃣ **Code Commit:** Developers push new code to a shared repository (e.g., GitHub, GitLab).

2️⃣ **Build:** The system compiles the code and checks for syntax errors.

3️⃣ **Automated Testing:** Runs unit tests, integration tests, and security scans.

4️⃣ **Deployment:** Deploys the application to a **staging or production environment**.

5️⃣ **Monitoring & Feedback:** Monitors performance and logs errors.

📌 **Example:**

- **Netflix uses CI/CD pipelines** to **update its streaming application** multiple times a day without service interruptions.

2.2 Traditional vs. Cloud-Based CI/CD

Feature	Traditional CI/CD	Cloud-Based CI/CD
Infrastructure	Requires on-premises servers	Fully managed in the cloud
Scalability	Limited by local hardware	Scales dynamically
Cost	High setup and maintenance costs	Pay-as-you-go pricing

Speed	Slower, due to manual scaling	Faster, due to cloud automation

📌 **Example:**

- **GitHub Actions integrates natively with GitHub repositories,** eliminating the need for external servers.

CHAPTER 3: JENKINS FOR CI/CD IN CLOUD

3.1 What is Jenkins?

🔹 **Jenkins is an open-source CI/CD tool** that automates software development workflows. It supports **plugins for cloud-based deployments** in AWS, Azure, and Google Cloud.

🔹 **Key Features of Jenkins:**

✅ **Automates builds, testing, and deployment**.

✅ **Extensive Plugin Support** – Works with Docker, Kubernetes, and Terraform.

✅ **Supports Multi-Cloud Deployment** – Integrates with AWS CodeDeploy, Azure DevOps, and GCP Cloud Build.

📌 **Example:**

- **Facebook uses Jenkins to run thousands of automated tests** daily before code is deployed to production.

3.2 How Jenkins Works in Cloud CI/CD?

1️⃣ **Developer pushes code to a Git repository (GitHub, GitLab, Bitbucket).**

2️⃣ **Jenkins detects the change** and triggers a build process.

3️⃣ **Automated tests run** (unit tests, integration tests).

4️⃣ **Successful builds are deployed to cloud environments** (AWS, Azure, GCP).
5️⃣ **Feedback is provided** via logs and monitoring tools.

📌 **Example:**

- A software company **uses Jenkins on AWS EC2 instances** to automate testing for microservices.

3.3 Benefits & Challenges of Jenkins

✅ **Benefits of Jenkins**

✓ **Highly Customizable** – Extensive plugin ecosystem.
✓ **Self-Hosted or Cloud-Hosted** – Can run on AWS, Azure, or on-premises.
✓ **Integration with DevOps Tools** – Supports Docker, Kubernetes, and Terraform.

❌ **Challenges of Jenkins**

❌ **Complex Configuration** – Requires setup and maintenance.
❌ **High Resource Usage** – Consumes CPU and memory when running many jobs.
❌ **Security Risks** – Requires proper IAM and access controls.

📌 **Case Study:**

- **Amazon Prime Video uses Jenkins to automate deployments,** reducing software release cycles.

CHAPTER 4: GITHUB ACTIONS FOR CI/CD IN CLOUD

4.1 What is GitHub Actions?

🔹 **GitHub Actions is a cloud-native CI/CD tool** that allows developers to automate **build, test, and deployment workflows** directly within **GitHub repositories**.

🔹 **Key Features of GitHub Actions:**

✅ **Built-in CI/CD for GitHub projects**.

✅ **Event-driven workflow execution** (e.g., code push, pull requests).

✅ **Supports cloud deployments to AWS, Azure, and Google Cloud**.

📌 **Example:**

- **A fintech company uses GitHub Actions to deploy a serverless API on AWS Lambda.**

4.2 How GitHub Actions Works in Cloud CI/CD?

1️⃣ **A developer commits new code to a GitHub repository.**

2️⃣ **GitHub Actions triggers a CI/CD workflow.**

3️⃣ **Automated tests run to check for bugs.**

4️⃣ **Code is deployed to cloud platforms (AWS, Azure, GCP).**

5️⃣ **Logs & notifications** provide feedback to developers.

📌 **Example:**

- **A mobile app development team uses GitHub Actions to automatically deploy updates to Firebase Cloud.**

4.3 Benefits & Challenges of GitHub Actions

✅ **Benefits of GitHub Actions**

✓ **Easy Integration with GitHub** – No external setup needed.
✓ **Serverless Execution** – Runs workflows in cloud environments.
✓ **Built-in Secrets Management** – Securely stores API keys and credentials.

❌ **Challenges of GitHub Actions**

❌ **Limited Free Tier** – Paid plans for high workloads.
❌ **Vendor Lock-In** – Works best within GitHub ecosystem.
❌ **Slower Build Times for Large Projects** – May require self-hosted runners.

📌 **Case Study:**

- **A gaming startup uses GitHub Actions to deploy updates to AWS Lambda, reducing deployment time from hours to minutes.**

CHAPTER 5: CI/CD USE CASES IN CLOUD COMPUTING

5.1 Industries Using CI/CD in Cloud

Industry	Use Case
E-commerce	Automating deployment of new product features.
Banking & Finance	Ensuring secure, bug-free updates in financial applications.
Healthcare	Deploying telemedicine apps with HIPAA-compliant CI/CD pipelines.
Media & Streaming	Netflix, YouTube use CI/CD for real-time feature updates.

IoT & Edge Computing	Automating updates for smart devices and edge applications.

📌 **Example:**

- **Spotify deploys updates seamlessly** to its mobile and web apps **using cloud-based CI/CD pipelines**.

CHAPTER 6: JENKINS VS. GITHUB ACTIONS: WHICH TO CHOOSE?

Feature	Jenkins	GitHub Actions
Hosting Model	Self-hosted or cloud	Cloud-based (GitHub)
Ease of Use	Requires setup & maintenance	Fully integrated into GitHub
Flexibility	Highly customizable	Simplified workflow management
Best For	Large-scale, complex pipelines	Fast deployment for GitHub projects

📌 **Example:**

- A **large enterprise may prefer Jenkins for its flexibility**, while **a startup may choose GitHub Actions for its simplicity**.

Exercise: Test Your Understanding

- **What is the main difference between CI and CD?**
- **How does Jenkins integrate with cloud platforms?**
- **What are the benefits of using GitHub Actions for CI/CD?**

- **Name three industries that benefit from cloud CI/CD.**
- **Which CI/CD tool is best suited for startups and why?**

Conclusion

CI/CD in the cloud **enhances software development efficiency** by automating testing and deployment.

✅ **Jenkins is powerful but requires setup,** making it ideal for large enterprises.
✅ **GitHub Actions offers seamless GitHub integration,** making it perfect for small teams.
✅ **Cloud CI/CD helps businesses deploy applications faster and more securely.**

ASSIGNMENT:

DEPLOY A WEB APPLICATION USING A CLOUD PROVIDER.

📄 Assignment Solution: Deploy a Web Application Using a Cloud Provider

---

📌 Step 1: Choose a Cloud Provider & Deployment Method

1.1 Selecting a Cloud Provider

Choose one of the following cloud providers for deployment:

✅ **AWS (Amazon Web Services) – Amazon EC2, AWS Elastic Beanstalk**

✅ **Azure (Microsoft) – Azure App Service, Virtual Machines**

✅ **Google Cloud Platform (GCP) – Compute Engine, App Engine**

📌 **Example Choice:** Deploying a web application on **AWS EC2**.

📌 Step 2: Setting Up a Virtual Server (EC2 in AWS)

2.1 Launch an EC2 Instance

✅ **Step 1:** Log in to **AWS Console** → Go to **EC2 Dashboard**.

✅ **Step 2:** Click on **Launch Instance**.

✅ **Step 3:** Choose an **Amazon Machine Image (AMI)** – Select **Ubuntu 22.04 LTS**.

✅ **Step 4:** Choose an **Instance Type** – Select **t2.micro (Free Tier eligible)**.

✅ **Step 5:** Configure Security Group – Allow **HTTP (Port 80) and SSH (Port 22)**.

✅ **Step 6:** Assign a key pair for SSH access and click **Launch**.

🔹 **Why EC2?**

✓ Provides full control over the web application stack.

✓ Scales based on application needs.

📌 Step 3: Install Web Server & Configure Application

3.1 Connect to EC2 Instance via SSH

✅ **Step 1:** Open a terminal and connect using the key pair:

```
ssh -i "your-key.pem" ubuntu@your-ec2-public-ip
```

✅ **Step 2:** Update package lists and install dependencies:

```
sudo apt update && sudo apt upgrade -y
sudo apt install apache2 -y  # Install Apache web server
```

✅ **Step 3:** Start and enable Apache:

```
sudo systemctl start apache2
sudo systemctl enable apache2
```

📌 **Verify by visiting:**

- Open a browser and enter http://your-ec2-public-ip
- You should see the **Apache default page**.

📌 Step 4: Deploy the Web Application

4.1 Upload Web Application Files

✅ **Step 1:** Install Git and Clone the Repository:

```
sudo apt install git -y
```

```
git clone https://github.com/your-repository/web-app.git
cd web-app
```

✅ **Step 2:** Move files to Apache's Web Root Directory:

```
sudo mv * /var/www/html/
```

✅ **Step 3:** Restart Apache to Apply Changes:

```
sudo systemctl restart apache2
```

📌 **Verify Deployment:**

- Visit http://your-ec2-public-ip
- Your web application should now be accessible.

📌 **Step 5: Configure a Database (Optional - MySQL Setup)**

5.1 Install & Configure MySQL

✅ **Step 1:** Install MySQL Server

```
sudo apt install mysql-server -y
sudo systemctl start mysql
sudo systemctl enable mysql
```

✅ **Step 2:** Secure MySQL Installation

```
sudo mysql_secure_installation
```

✅ **Step 3:** Create a Database & User

```
sudo mysql -u root -p
CREATE DATABASE webapp_db;
```

```
CREATE USER 'webapp_user'@'localhost' IDENTIFIED BY 'securepassword';

GRANT ALL PRIVILEGES ON webapp_db.* TO 'webapp_user'@'localhost';

FLUSH PRIVILEGES;

EXIT;
```

📌 **Update the Web Application Configuration** to use **MySQL credentials**.

📌 Step 6: Enable a Custom Domain & SSL (Optional)

6.1 Configure a Domain with AWS Route 53

✅ **Step 1:** Purchase a domain from **Route 53** or use an existing one.
✅ **Step 2:** Create an **A Record** pointing to your **EC2 Public IP**.

6.2 Install Let's Encrypt SSL (HTTPS)

✅ **Step 1:** Install Certbot SSL:

```
sudo apt install certbot python3-certbot-apache -y
```

✅ **Step 2:** Generate an SSL Certificate:

```
sudo certbot --apache -d yourdomain.com
```

✅ **Step 3:** Auto-renew SSL:

```
sudo crontab -e
```

Add the following line to renew SSL every 90 days:

```
0 0 * * * certbot renew --quiet
```

📌 **Your web application is now secured with HTTPS!**

📌 Step 7: Scaling & Monitoring the Application

7.1 Enable Auto-Scaling & Load Balancing

✅ **Step 1:** Go to **AWS EC2** → **Load Balancers** → Create a **Load Balancer**.
✅ **Step 2:** Attach **EC2 Instances** to distribute traffic.
✅ **Step 3:** Enable **Auto Scaling** to handle traffic surges.

7.2 Monitor Logs & Performance

✅ **Enable CloudWatch Logs:**

sudo apt install awslogs -y

sudo systemctl start awslogsd

📌 **Monitor logs via AWS CloudWatch Dashboard.**

📌 Conclusion: Successfully Deployed Web Application

🚀 **Final Outcome:**
✅ A web application is **deployed on AWS EC2**.
✅ Apache web server is **configured to serve the application**.
✅ A **database (MySQL)** is set up for storing application data.
✅ Domain name & SSL certificate are **configured for security**.
✅ Auto-scaling & monitoring tools are **enabled for high availability**.

🔹 **By following this step-by-step guide, you have successfully deployed a web application on AWS Cloud!** 🚀

📌 Submission Guidelines

📌 **Format:**

✅ Submit a report in **Word (DOCX) or PDF format**.

✅ Include **screenshots of EC2, Apache setup, and application deployment**.

📌 **Word Limit:** 2000-2500 words

📌 **Deadline:** (To be provided by the instructor)

🚀 **Deploy your first cloud-based web application today!** 🚀