Independent Skill Development Mission

Independent Skill Development Mission NXT
EMPOWERING FUTURES, SHAPING TOMORROW

ISDM (INDEPENDENT SKILL DEVELOPMENT MISSION

AWS DATABASE SERVICES

AMAZON RDS (RELATIONAL DATABASE SERVICE) – STUDY MATERIAL

INTRODUCTION TO AMAZON RDS

What is Amazon RDS?

Amazon **RDS (Relational Database Service)** is a **fully managed relational database service** that simplifies the setup, operation, and scaling of databases in the AWS cloud. It automates **database management tasks**, including **provisioning, backups, patching, and scaling**, reducing the burden on database administrators.

Key Features of Amazon RDS

✓ **Fully Managed** – AWS handles database maintenance, backups, and patching.
✓ **High Availability** – Supports **Multi-AZ deployments** for failover protection.
✓ **Automatic Backups** – Enables **point-in-time recovery** for data protection.
✓ **Scalability** – Easily scale storage and compute resources as needed.
✓ **Security** – Supports **VPC, IAM, KMS encryption, and network**

isolation.

✓ **Monitoring & Performance Optimization** – Integrated with **CloudWatch, Performance Insights, and Enhanced Monitoring**.

✓ **Multi-Region Replication** – Supports **read replicas** for performance and disaster recovery.

CHAPTER 1: SUPPORTED DATABASE ENGINES IN AMAZON RDS

Amazon RDS supports **multiple database engines** to accommodate different application needs:

Database Engine	Best For	Features
Amazon Aurora	High-performance cloud-native database	5x faster than MySQL, automatic failover, auto-scaling
MySQL	Web applications, startups, open-source projects	Replication, high availability, cost-effective
PostgreSQL	Advanced analytics, enterprise workloads	Supports JSON, GIS, and complex queries
MariaDB	MySQL-compatible, open-source applications	Replication, clustering, high performance
Microsoft SQL Server	Enterprise applications, .NET workloads	Windows authentication, SQL Server features

Oracle	Enterprise applications requiring Oracle compatibility	Supports Oracle Data Guard, licensing flexibility

📌 **Example:**
A **startup** uses **Amazon Aurora** for its **high-performance SaaS application,** benefiting from **automatic scaling and fault tolerance**.

CHAPTER 2: HOW AMAZON RDS WORKS

1. RDS Architecture Overview

- **Primary Instance** – Main database handling read/write operations.
- **Multi-AZ Replica** – Standby instance for failover protection.
- **Read Replicas** – Additional replicas to handle high-read traffic.
- **Storage** – Uses **GP3, IO1, or Magnetic storage** for different performance needs.

2. Amazon RDS Deployment Models

- **Single-AZ Deployment** – Runs in one Availability Zone (**cheaper, but no failover**).
- **Multi-AZ Deployment** – Creates **standby instance in another AZ** for high availability.
- **Read Replicas** – Supports **read scaling** and **cross-region replication**.

📌 **Example:**
A **financial company** uses **Multi-AZ RDS for PostgreSQL** to ensure **database reliability and automatic failover**.

CHAPTER 3: SETTING UP AN AMAZON RDS DATABASE

Step 1: Create an Amazon RDS Instance

1. **Go to AWS Console** → Open **Amazon RDS**.
2. Click **"Create Database"**.
3. Select **Standard Create**.

Step 2: Choose Database Engine

1. Choose a database engine (e.g., **MySQL, PostgreSQL, or Aurora**).
2. Select **Engine Version** (latest stable version recommended).

Step 3: Configure Database Instance

1. **DB Instance Identifier:** my-rds-instance.
2. **Username:** admin.
3. **Password:** Set a strong password.
4. Choose **DB Instance Class** (e.g., db.t3.micro for small workloads).

Step 4: Storage and Connectivity

1. **Storage Type:** Choose **GP3** for general workloads.
2. **Allocate Storage:** Start with **20GB** (auto-scale enabled).
3. **VPC & Subnet:** Choose an existing **VPC and Public/Private Subnet**.
4. **Enable Multi-AZ Deployment** (Recommended for production).

Step 5: Configure Security and Monitoring

1. **Enable Encryption** (for data security).

2. **Set up IAM authentication** for secure access.
3. **Enable CloudWatch Monitoring**.

Step 6: Create and Connect to RDS Instance

1. Click **"Create Database"**.
2. Once the instance is **available,** copy the **endpoint URL**.
3. **Connect to RDS using MySQL CLI**:
4. mysql -h my-rds-instance.cndpjzabc123.us-east-1.rds.amazonaws.com -u admin -p

📌 **Expected Outcome:** Successfully created and connected to an RDS database.

CHAPTER 4: MANAGING AMAZON RDS (BACKUPS, SCALING, SECURITY)

1. Automated Backups & Snapshots

- **Automated Backups:** AWS **automatically backs up** data and transaction logs for **point-in-time recovery**.
- **Manual Snapshots:** Users can create **on-demand backups** and restore from snapshots.
- **Restore from Backup:**
- aws rds restore-db-instance-from-db-snapshot --db-instance-identifier my-restored-db --db-snapshot-identifier my-db-snapshot

2. Scaling Amazon RDS

- **Vertical Scaling:** Increase **CPU, RAM, and storage** by modifying the instance size.

- **Horizontal Scaling:** Use **Read Replicas** for **read-heavy workloads**.

3. Security Best Practices for Amazon RDS

✓ **Enable IAM authentication** instead of traditional credentials.
✓ **Restrict access with Security Groups and VPC settings**.
✓ **Encrypt data at rest using AWS KMS**.
✓ **Monitor database performance with Amazon CloudWatch**.

📌 **Example:**
A **healthcare application** stores **patient records** in **Amazon RDS with KMS encryption** to meet **HIPAA compliance**.

CHAPTER 5: MONITORING & TROUBLESHOOTING AMAZON RDS

1. Monitoring with Amazon CloudWatch

CloudWatch provides key RDS metrics, including:
✓ **CPU Utilization** – Tracks database performance.
✓ **Free Storage Space** – Prevents storage exhaustion.
✓ **Read/Write IOPS** – Monitors database query performance.
✓ **Database Connections** – Tracks concurrent user sessions.

📌 **Example:**
An e-commerce platform **sets up CloudWatch Alarms** to receive alerts **if CPU usage exceeds 80%**.

2. Troubleshooting Common Issues

Issue	Solution
High CPU Utilization	Scale up the instance, optimize queries, or use read replicas

Slow Query Performance	Enable **Performance Insights**, use indexing
Database Connectivity Issues	Check security groups, VPC settings, and IAM policies
Storage Full	Enable auto-scaling or manually increase storage

📌 **Example:**
A **data analytics company** uses **Amazon RDS Performance Insights** to detect **slow SQL queries** and optimize database performance.

Chapter 6: Amazon RDS Use Cases

✓ **E-Commerce Websites** – Stores **product catalogs, customer orders, and transaction data**.

✓ **SaaS Applications** – Uses **multi-tenant RDS databases** for scalable web apps.

✓ **Financial Systems** – Implements **Multi-AZ RDS for high availability**.

✓ **IoT & Big Data Analytics** – Stores **real-time sensor data** in RDS.

📌 **Case Study: Netflix Uses Amazon RDS**
Problem: Needed **scalable, fault-tolerant databases**.
Solution: Implemented **Amazon Aurora** with read replicas.
Outcome: 99.99% uptime and cost efficiency.

Conclusion: Mastering Amazon RDS

By using **Amazon RDS**, businesses can:

✅ **Deploy fully managed relational databases** with ease.

✅ **Ensure high availability with Multi-AZ.**
✅ **Scale seamlessly based on demand.**
✅ **Secure and monitor databases with AWS tools.**

FINAL EXERCISE:

1. **Create an RDS MySQL database,** connect to it, and create a **test table**.
2. **Set up a read replica** and measure query performance improvements.
3. **Implement a CloudWatch alarm** for high CPU utilization on an RDS instance.

Amazon DynamoDB (NoSQL Database) – Study Material

Introduction to Amazon DynamoDB

What is Amazon DynamoDB?

Amazon **DynamoDB** is a **fully managed NoSQL database service** designed for high availability, scalability, and low-latency performance. It provides **key-value and document-based storage** with **automatic scaling**, making it ideal for modern applications requiring real-time performance.

Key Features of Amazon DynamoDB

✓ **Fully Managed** – No need to manage servers, backups, or scaling.
✓ **High Availability & Durability** – Data is **automatically replicated** across multiple Availability Zones.
✓ **Performance at Scale** – Handles **millions of requests per second** with **single-digit millisecond latency**.
✓ **Flexible Data Model** – Supports **key-value** and **document-based storage**.
✓ **On-Demand & Provisioned Capacity** – Choose between **pay-per-request or fixed throughput**.
✓ **Security & Compliance** – Supports **encryption, IAM authentication, and VPC integration**.
✓ **Event-Driven Architecture** – Integrates with **AWS Lambda, Kinesis, and S3** for real-time data processing.

Chapter 1: DynamoDB Architecture & Core Concepts

1. How DynamoDB Works

- **Tables:** Collections of data with **no fixed schema**.

- **Items:** Similar to rows in SQL databases.
- **Attributes:** Individual fields within an item.
- **Primary Keys:** Unique identifiers for items (either **Partition Key** or **Partition + Sort Key**).

2. DynamoDB Key Design

- **Partition Key (Hash Key):** A **unique identifier** that determines **which partition the item is stored in**.
- **Composite Key (Partition + Sort Key):** Enables **querying on multiple attributes**.

📌 **Example:**
A **customer database** uses:

- CustomerID (Partition Key) to uniquely identify customers.
- OrderID (Sort Key) to store multiple orders under one customer.

3. Data Consistency in DynamoDB

- **Eventually Consistent Reads:** Faster reads with possible slight delays in data propagation.
- **Strongly Consistent Reads:** Ensures the most up-to-date data but may increase latency.

CHAPTER 2: CREATING A DYNAMODB TABLE

Step 1: Navigate to DynamoDB Console

1. Open **AWS Management Console**.
2. Search for **DynamoDB** and open the service.
3. Click **"Create Table"**.

Step 2: Define Table Schema

1. **Table Name:** CustomerOrders.
2. **Primary Key:**
    - **Partition Key:** CustomerID (String).
    - **Sort Key (Optional):** OrderID (String).
3. Choose **"On-Demand Capacity Mode"** (auto-scaling).
4. Click **"Create Table"**.

Step 3: Insert Sample Data

1. Click **"Explore Items"** → **"Create Item"**.
2. Add sample data:
3. {
4. "CustomerID": "CUST1001",
5. "OrderID": "ORD5001",
6. "ProductName": "Laptop",
7. "OrderDate": "2025-02-20"
8. }
9. Click **"Save Item"**.

📌 **Expected Outcome:** The item is successfully stored in **DynamoDB**.

CHAPTER 3: QUERYING & SCANNING DATA IN DYNAMODB

1. Querying Items Using Primary Key

- Use **Partition Key** and optionally **Sort Key** to retrieve items efficiently.
- Example query: Find **all orders for CustomerID = "CUST1001"**.

2. Scanning Items (Full Table Search)

- Scans **every item in the table** (less efficient than queries).
- Use **filters to refine results** (e.g., ProductName = "Laptop").

📌 **Example:**

Retrieve all orders for CUST1001 using **AWS CLI**:

```
aws dynamodb query --table-name CustomerOrders \
 --key-condition-expression "CustomerID = :cust" \
 --expression-attribute-values '{":cust":{"S":"CUST1001"}}'
```

CHAPTER 4: INDEXING & PERFORMANCE OPTIMIZATION

1. Global Secondary Index (GSI)

- Allows querying on **attributes other than the primary key**.
- Example: Create a GSI on ProductName to **find all orders for a specific product**.

2. Local Secondary Index (LSI)

- Uses the same **Partition Key** but a **different Sort Key** for faster lookups.
- Example: Store OrderDate as an LSI to **retrieve recent orders for a customer**.

3. Choosing Between GSIs & LSIs

Feature	Global Secondary Index (GSI)	Local Secondary Index (LSI)
Partition Key	Different from the base table	Same as the base table
Sort Key	Can be any attribute	Must be a different attribute from the base table
Use Case	Querying data by alternative fields	Sorting within a partition

📌 **Example:**
A **retail store** creates a **GSI on "ProductName"** to quickly **find all orders for a specific product**.

Chapter 5: DynamoDB Security & Best Practices

1. Security Best Practices

✓ **Enable Encryption at Rest** using AWS KMS.
✓ **Use IAM Roles & Policies** to restrict access.
✓ **Enable VPC Endpoints** for **private connectivity**.

2. Cost Optimization Best Practices

✓ Use **On-Demand Mode** for unpredictable workloads.
✓ Use **Provisioned Mode** for **steady traffic** with **auto-scaling enabled**.
✓ Delete **unused Global Secondary Indexes (GSIs)** to reduce costs.

📌 **Example:**
A **fintech company** encrypts DynamoDB **with AWS KMS** and limits access via **IAM roles** to protect sensitive data.

Chapter 6: DynamoDB Streams & Event-Driven Architectures

1. What is DynamoDB Streams?

- Captures **data changes in real-time**.
- Integrates with **AWS Lambda for event-driven applications**.

2. Use Case: Serverless Order Processing

1. **New order is inserted into DynamoDB**.
2. **DynamoDB Streams triggers AWS Lambda**.
3. **Lambda processes the order & updates inventory**.

📌 **Example:**
An **e-commerce store** uses **DynamoDB Streams + Lambda** to **automatically process orders** in real-time.

Chapter 7: Real-World Use Cases of DynamoDB

✓ **E-Commerce Applications** – Store **product catalogs, orders, and user sessions**.
✓ **Gaming Leaderboards** – Track **real-time player scores**.
✓ **IoT Data Storage** – Store **sensor data for real-time analytics**.
✓ **Social Media Apps** – Handle **user posts, likes, and comments** at scale.

📌 Case Study: Amazon Uses DynamoDB

Problem: Amazon needed a **high-speed NoSQL database** to handle **Black Friday sales traffic**.
Solution: Used DynamoDB with **auto-scaling & caching**.

Outcome: Reduced latency & handled millions of transactions per second.

Conclusion: Mastering Amazon DynamoDB

By using **Amazon DynamoDB,** businesses can:

✅ **Store & retrieve data at scale with high availability.**

✅ **Ensure low-latency performance with GSIs & caching.**

✅ **Implement event-driven architectures with DynamoDB Streams.**

Final Exercise:

1. **Create a DynamoDB table,** add **sample data,** and query using **AWS CLI**.
2. Implement **a Global Secondary Index (GSI)** to optimize queries.
3. Use **DynamoDB Streams with AWS Lambda** to **automate order processing**.

AMAZON AURORA – STUDY MATERIAL

INTRODUCTION TO AMAZON AURORA

What is Amazon Aurora?

Amazon **Aurora** is a **fully managed, high-performance relational database service** designed for **enterprise applications, SaaS solutions, and mission-critical workloads**. Aurora is **compatible with MySQL and PostgreSQL** while providing **up to 5x the performance of standard MySQL and 3x the performance of standard PostgreSQL**.

Key Features of Amazon Aurora

✓ **High Performance** – Delivers **low-latency, high-throughput performance**.
✓ **Fault Tolerance & High Availability** – Data is **automatically replicated across multiple Availability Zones (Multi-AZ)**.
✓ **Auto-Scaling Storage** – Expands **from 10GB to 128TB** automatically.
✓ **Fully Managed by AWS** – Automates **backups, patching, and monitoring**.
✓ **Security & Compliance** – Supports **encryption, IAM authentication, and VPC isolation**.
✓ **Global Database Support** – Enables **multi-region replication** for disaster recovery.
✓ **Serverless Option (Aurora Serverless)** – Scales automatically **based on demand**.

CHAPTER 1: AMAZON AURORA VS. OTHER DATABASES

1. Aurora vs. RDS MySQL & PostgreSQL

Feature	Amazon Aurora	RDS MySQL	RDS PostgreSQL
Performance	5x MySQL, 3x PostgreSQL	Standard MySQL	Standard PostgreSQL
Replication	Multi-AZ, Multi-Region	Read Replicas	Read Replicas
Scalability	Auto-scales up to 128TB	Manual scaling	Manual scaling
Failover Time	<30 seconds	Minutes	Minutes
Backup	Continuous, automated	Scheduled	Scheduled

2. Aurora vs. Traditional Databases (Oracle, SQL Server)

✓ **No licensing costs** – Uses **open-source MySQL/PostgreSQL**.
✓ **Fully managed** – No need for **manual backups or clustering**.
✓ **Better scalability** – Grows **dynamically without downtime**.

📌 **Example:**
A **large e-commerce website** migrates from **Oracle to Aurora PostgreSQL** for **cost savings and better performance**.

CHAPTER 2: AMAZON AURORA ARCHITECTURE & COMPONENTS

1. Aurora Cluster Components

- **Primary Instance** – Handles **read and write** operations.
- **Read Replicas** – Up to **15 read replicas** for load balancing.
- **Aurora Storage Layer** – Automatically scales from **10GB to 128TB**.

2. High Availability & Replication

- **Multi-AZ Replication** – Replicates **six copies of data across three Availability Zones**.
- **Aurora Global Database** – Allows **low-latency reads across multiple regions**.
- **Failover Support** – Automatically switches to a **read replica** in case of primary failure.

📌 **Example:**
A **fintech company** uses **Aurora Global Database** to replicate **customer transactions across the US & Europe**.

CHAPTER 3: SETTING UP AMAZON AURORA

Step 1: Create an Aurora Database Cluster

1. Open **AWS Console** → Go to **RDS Dashboard**.
2. Click **"Create Database"**.
3. Choose **Amazon Aurora** as the database engine.
4. Select **"Aurora MySQL"** or **"Aurora PostgreSQL"**.
5. Choose **"Production" or "Dev/Test"** environment.

Step 2: Configure the Cluster

1. **DB Cluster Identifier:** my-aurora-cluster.
2. **Username:** admin**, Password:** mypassword123.
3. Choose **Instance Class:**
    - db.t3.medium for development.
    - db.r5.large for production.

Step 3: Enable High Availability & Security

1. Select **Multi-AZ Deployment**.
2. Enable **Encryption (AWS KMS)**.
3. Set **Public Access = No** (for security).
4. Configure **Security Groups** to allow **only specific IPs**.

Step 4: Launch & Connect to Aurora

1. Click **"Create Database"**.
2. Once available, copy the **Endpoint URL**.
3. Connect using MySQL CLI:
4. mysql -h my-aurora-cluster.cluster-abc123.us-east-1.rds.amazonaws.com -u admin -p

📌 **Expected Outcome:** Successfully **created and connected** to an **Aurora database cluster**.

CHAPTER 4: SCALING, PERFORMANCE, AND OPTIMIZATION

1. Auto-Scaling Aurora Storage & Compute

✓ **Storage Auto-Scaling** – Expands **from 10GB to 128TB** automatically.
✓ **Aurora Auto-Scaling Read Replicas** – Automatically adds or removes read replicas **based on traffic**.

2. Performance Insights & Optimization

- **Enable Performance Insights** to analyze slow queries.
- Use **Aurora Query Plan Caching** to speed up repetitive queries.

- Implement **Connection Pooling** for high-traffic workloads.

📌 **Example:**
A **gaming company** uses **Aurora Auto-Scaling** to **handle traffic spikes during peak hours**.

Chapter 5: Aurora Serverless – On-Demand Scaling

1. What is Aurora Serverless?

- **Auto-scales compute capacity** based on demand.
- **No need for provisioning** fixed instances.
- **Cost-efficient for intermittent workloads**.

2. Setting Up Aurora Serverless

1. In the RDS Console, click **"Create Database"**.
2. Choose **Amazon Aurora** → Select **Aurora Serverless**.
3. Configure **Auto-Scaling Minimum & Maximum Capacity** (e.g., 1–10 ACUs).
4. Click **"Create Database"**.

📌 **Example:**
A **start-up with unpredictable traffic** uses **Aurora Serverless** to **reduce costs by automatically scaling**.

Chapter 6: Security & Backup Strategies in Aurora

1. Security Best Practices

✓ **Enable IAM Authentication** for passwordless access.
✓ **Use Security Groups & VPC Peering** for private access.
✓ **Enable Multi-Factor Authentication (MFA)** for database access.

2. Backup & Disaster Recovery

- **Automated Backups** – Retains up to **35 days of backups**.
- **Manual Snapshots** – Take snapshots before updates.
- **Aurora Global Database** – Replicates data across regions for DR.

📌 **Example:**
A **healthcare company** uses **Aurora Global Database** to **store patient records across continents securely**.

CHAPTER 7: AMAZON AURORA USE CASES & CASE STUDY

✓ **E-Commerce Applications** – Handles **large-scale transactions and analytics**.
✓ **Financial & Banking Services** – Supports **secure and high-speed processing**.
✓ **SaaS Applications** – Powers **multi-tenant applications efficiently**.
✓ **AI/ML & Analytics** – Works with **AWS Glue, Redshift, and S3**.

📌 CASE STUDY: NETFLIX USES AURORA

Problem: Needed a **highly scalable relational database** for content personalization.
Solution: Migrated from **traditional MySQL to Aurora MySQL**.
Outcome: Improved performance, reduced costs, and better scalability.

Conclusion: Mastering Amazon Aurora

By using **Amazon Aurora,** businesses can:

✅ **Improve performance compared to traditional RDS databases**.

✅ **Scale dynamically with Aurora Auto-Scaling & Serverless**.

✅ **Ensure high availability with Multi-AZ & Global Replication**.

✅ **Enhance security with IAM authentication and KMS encryption**.

Final Exercise:

1. **Create an Amazon Aurora MySQL database,** connect to it, and insert test data.
2. **Enable Auto-Scaling Read Replicas** and measure performance improvements.
3. **Set up a global Aurora cluster** and test cross-region replication.

AWS Security Essentials

AWS Shared Responsibility Model – Study Material

Introduction to the AWS Shared Responsibility Model

What is the AWS Shared Responsibility Model?

The **AWS Shared Responsibility Model** defines **who is responsible** for securing different aspects of the cloud environment:

- **AWS is responsible for the security of the cloud** (infrastructure, hardware, networking).
- **Customers are responsible for security in the cloud** (data, applications, IAM, encryption).

This model ensures that both **AWS and customers** play a role in **maintaining security and compliance**.

Key Features of the AWS Shared Responsibility Model

✓ **Clear separation of responsibilities** between AWS and customers.
✓ **AWS secures the global infrastructure** (data centers, networking, hypervisors).
✓ **Customers control and manage their cloud environment** (data, IAM, OS security).
✓ **Compliance with industry standards** (ISO, SOC, PCI DSS, HIPAA).

Chapter 1: Understanding the AWS Shared Responsibility Model

1. AWS Responsibility: Security OF the Cloud

AWS ensures the security of:

✅ **Physical Security** – Data centers, hardware, access controls.

✅ **Networking Security** – Firewalls, DDoS protection, AWS Shield.

✅ **Storage & Compute Infrastructure** – EC2, S3, RDS, Lambda security.

✅ **Compliance & Certifications** – GDPR, ISO 27001, SOC 2, HIPAA.

2. Customer Responsibility: Security IN the Cloud

Customers must secure:

✅ **Data & Application Security** – Encrypt data at rest and in transit.

✅ **IAM & Access Control** – Manage permissions with IAM roles & policies.

✅ **Operating System & Patching** – Keep OS and applications updated.

✅ **Network Configuration** – Secure VPC, Security Groups, NACLs.

📌 **Example:**
If a company hosts a **website on AWS EC2**, AWS is responsible for **server infrastructure**, but the company must **secure the OS, data, and user permissions**.

Chapter 2: Shared Responsibility Across AWS Services

1. Infrastructure Services (EC2, S3, VPC, RDS, Lambda)

AWS Responsibility	Customer Responsibility

Physical security of data centers	Configure IAM roles & permissions
Securing hardware, storage, and networking	Secure OS, applications, and databases
Ensuring availability & redundancy	Encrypt data and manage backup policies

📌 **Example:**

- **EC2 Instance Security** → AWS secures the **hardware**, but users must **install security patches** and **configure firewalls**.
- **S3 Bucket Security** → AWS manages the **storage infrastructure**, but users must **enable encryption and restrict public access**.

---

2. AWS Managed Services (Lambda, DynamoDB, RDS, ECS, CloudFront)

AWS Responsibility	Customer Responsibility
Securing the runtime environment	Secure application code & API permissions
Automated patching & maintenance	Manage user access & authentication
Ensuring availability & scaling	Monitor logs for anomalies (CloudWatch, GuardDuty)

📌 **Example:**

In **AWS Lambda**, AWS manages the **underlying servers**, but the customer must ensure **secure API permissions & data encryption**.

Chapter 3: Security Best Practices for Customers

1. Identity & Access Management (IAM)

✓ Use **IAM roles** instead of root credentials.
✓ Enable **Multi-Factor Authentication (MFA)** for users.
✓ Apply **least privilege access** (grant only necessary permissions).

2. Data Protection

✓ **Encrypt S3 buckets, databases, and EC2 storage** (AWS KMS).
✓ Enable **automatic backups and versioning** for data recovery.
✓ Use **AWS Secrets Manager** to store API keys & passwords.

3. Network Security

✓ Restrict access using **VPC Security Groups and NACLs**.
✓ Enable **AWS Shield & WAF** to protect against DDoS attacks.
✓ Implement **CloudTrail & GuardDuty** to monitor security events.

📌 **Example:**
A **finance company using AWS RDS** should enable **encryption, restrict public access, and use IAM authentication**.

Chapter 4: AWS Compliance and Shared Responsibility

1. AWS Compliance Certifications

AWS maintains **global security and compliance** certifications, including:

✅ **ISO 27001, SOC 1/2/3** – Cloud security and operational excellence.
✅ **HIPAA** – Protects healthcare data.
✅ **PCI DSS** – Secures payment processing systems.
✅ **GDPR** – Ensures data privacy for European users.

2. Customer Responsibilities for Compliance

✓ **Enable AWS Config & Audit Logs** to track changes.
✓ **Use AWS Organizations** to enforce security policies across accounts.
✓ **Follow AWS Well-Architected Security Framework** for compliance.

📌 **Example:**
A **hospital storing patient records in AWS** must **encrypt data, control IAM access, and follow HIPAA guidelines**.

Chapter 5: Real-World Use Cases of AWS Shared Responsibility Model

1. E-Commerce Website Security (Using EC2, S3, RDS, CloudFront)

✓ **AWS Responsibilities:**

- Secures data center infrastructure & networking.
- Ensures **high availability of EC2, S3, and CloudFront**.

✓ **Customer Responsibilities:**

- **Encrypt S3 bucket** storing user orders.
- **Secure EC2 instance** with a firewall (Security Groups).
- **Implement IAM roles** for developers and support teams.

---

2. Banking & Financial Application Security (Using RDS, Lambda, S3, VPC)

✓ **AWS Responsibilities:**

- Secures **RDS database infrastructure** with automated failover.
- Provides **DDoS protection** via AWS Shield.

✓ **Customer Responsibilities:**

- **Restrict database access using IAM roles & Security Groups**.
- **Enable data encryption (KMS) for S3 backups**.
- **Monitor transactions with AWS CloudTrail**.

📌 **Example:**
A **bank uses Amazon RDS for financial transactions** but must **enforce strong IAM policies and encryption**.

---

3. Healthcare Application (HIPAA-Compliant AWS Setup)

✓ **AWS Responsibilities:**

- Provides **HIPAA-compliant infrastructure**.
- Ensures **data encryption and compliance certifications**.

✓ **Customer Responsibilities:**

- **Encrypt patient data in Amazon RDS** using **AWS KMS**.
- **Restrict public access to sensitive information** using IAM.
- **Enable logging & monitoring (AWS CloudWatch, GuardDuty)**.

📌 **Example:**
A **telemedicine company** storing patient records in AWS must **encrypt data and follow HIPAA security policies**.

Conclusion: Mastering the AWS Shared Responsibility Model

By understanding the **AWS Shared Responsibility Model**, businesses can:

✅ **Ensure a secure cloud environment by managing responsibilities effectively**.

✅ **Encrypt and protect their own data while AWS secures the infrastructure**.

✅ **Follow compliance standards like HIPAA, PCI DSS, and GDPR**.

✅ **Implement best security practices using IAM, encryption, and monitoring tools**.

Final Exercise:

1. **List five AWS responsibilities and five customer responsibilities** for securing an EC2 instance.
2. **Configure an S3 bucket** with **encryption and IAM permissions** to ensure security.
3. **Enable AWS CloudTrail and CloudWatch** to monitor API activity in your AWS account.

IAM Policies and Permissions – Study Material

Introduction to AWS IAM (Identity and Access Management)

What is AWS IAM?

AWS **Identity and Access Management (IAM)** is a security service that enables you to **manage users, groups, and roles**, and **control their permissions** to AWS resources. It follows the **principle of least privilege**, ensuring that users only have the permissions necessary for their tasks.

Key Features of AWS IAM

✓ **Fine-Grained Access Control** – Assign permissions to specific AWS resources.
✓ **User Authentication & Authorization** – Securely manage users and roles.
✓ **Multi-Factor Authentication (MFA)** – Adds an extra layer of security.
✓ **Federated Access** – Allows SSO integration with external identity providers.
✓ **Policy-Based Permissions** – Uses JSON-based policies to define permissions.
✓ **AWS Organizations Integration** – Enforce security policies across multiple AWS accounts.

Chapter 1: IAM Policies and Permissions – Core Concepts

1. IAM Users, Groups, and Roles

- **IAM User** – Represents an individual user (e.g., developer01).

- **IAM Group** – Collection of users with shared permissions (e.g., Developers).
- **IAM Role** – A temporary access identity used by AWS services or external users.

2. IAM Policies – Defining Permissions

IAM **policies** are **JSON-based permission rules** that define **who can access what** in AWS.

📌 **Example:**

An **S3 read-only policy** allows users to view (but not modify) files in an S3 bucket.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::my-bucket/*"
    }
  ]
}
```

CHAPTER 2: TYPES OF IAM POLICIES

1. AWS-Managed Policies (Predefined by AWS)

AWS provides **predefined policies** for common use cases.

Policy Name	Purpose
AdministratorAccess	Grants full access to AWS
PowerUserAccess	Full access except IAM & security settings
AmazonS3ReadOnlyAccess	Allows users to **read but not modify S3 objects**
AmazonEC2FullAccess	Grants full control over EC2 instances

2. Customer-Managed Policies (Custom Policies)

✓ Created by AWS customers for **specific business needs**.
✓ Provides **more control over permissions** than AWS-managed policies.
✓ Useful for **restricting access to specific AWS resources**.

📌 **Example:**
A company creates a **custom IAM policy** to allow **developers to start and stop EC2 instances** but not delete them.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:StartInstances",
        "ec2:StopInstances"
      ],
```

```
      "Resource": "arn:aws:ec2:region:account-id:instance/*"
    },
    {
      "Effect": "Deny",
      "Action": "ec2:TerminateInstances",
      "Resource": "arn:aws:ec2:region:account-id:instance/*"
    }
  ]
}
```

3. Inline Policies (Attached Directly to Users, Groups, or Roles)

✓ Inline policies are **attached directly to an IAM entity (user, group, or role)**.
✓ Unlike managed policies, inline policies **cannot be reused** across multiple users or groups.
✓ Best for **one-time or highly specific permissions**.

CHAPTER 3: IAM POLICY STRUCTURE AND ELEMENTS

IAM policies consist of **five key elements**:

1. Version

Specifies the **policy language version** (always set to "2012-10-17").

2. Statement

Defines **permissions within a policy**.

3. Effect

- "Allow" → Grants permissions.

- "Deny" → Explicitly denies access.

4. Action

Defines the **specific AWS service actions** (e.g., s3:PutObject, ec2:StartInstances).

5. Resource

Specifies **which AWS resources** are affected (e.g., "arn:aws:s3:::my-bucket/*").

📌 **Example Policy:** Full access to EC2 but **denies terminating instances**.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "ec2:*",
      "Resource": "*"
    },
    {
      "Effect": "Deny",
      "Action": "ec2:TerminateInstances",
      "Resource": "*"
    }
  ]
```

}

Chapter 4: Applying IAM Policies to AWS Resources

1. Assigning a Policy to a User

1. Open **AWS IAM Console** → Click **"Users"**.
2. Select a user → Click **"Permissions"**.
3. Click **"Attach Policy"** → Select a policy (e.g., AmazonS3ReadOnlyAccess).
4. Click **"Apply Changes"**.

2. Assigning a Policy to a Role

1. Go to **IAM Console** → Click **"Roles"**.
2. Select an **existing role** or **create a new one**.
3. Attach a policy (e.g., AmazonEC2FullAccess).
4. Save changes and **assign the role to an EC2 instance or Lambda function**.

Chapter 5: IAM Best Practices

✓ **Follow the Principle of Least Privilege** – Grant **only necessary permissions**.

✓ **Enable Multi-Factor Authentication (MFA)** for all users.

✓ **Use IAM Roles Instead of IAM Users** for EC2, Lambda, and other AWS services.

✓ **Rotate IAM Credentials Regularly** – Avoid long-term access keys.

✓ **Use AWS Organizations** to apply policies at the **account level**.

✓ **Monitor IAM Activity with AWS CloudTrail**.

📌 **Example:**

A **finance company** enables **MFA for all IAM users** and uses **AWS CloudTrail** to monitor login attempts.

CHAPTER 6: REAL-WORLD USE CASES OF IAM POLICIES

1. Restricting Developers to Read-Only Access

A company allows **developers to read AWS resources** but prevents them from modifying or deleting them.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "ec2:Describe*",
      "Resource": "*"
    },
    {
      "Effect": "Deny",
      "Action": ["ec2:StopInstances", "ec2:TerminateInstances"],
      "Resource": "*"
    }
  ]
}
```

2. Granting Read/Write Access to a Specific S3 Bucket

A **marketing team** needs **access to only one S3 bucket**.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:GetObject", "s3:PutObject"],
      "Resource": "arn:aws:s3:::marketing-bucket/*"
    }
  ]
}
```

3. Temporary Access for Contractors Using IAM Roles

A company allows **external contractors** to temporarily **access AWS Lambda**.

- Uses **IAM Roles** instead of IAM Users.
- Sets **session duration limit** for security.

📌 **Example:**

A contractor accesses AWS **for 12 hours using an IAM role** but cannot modify policies.

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Allow",
      "Action": "lambda:InvokeFunction",
      "Resource": "*"
    }
  ]
}
```

Conclusion: Mastering IAM Policies and Permissions

By using **AWS IAM,** businesses can:

✅ **Secure AWS resources with fine-grained access control.**
✅ **Protect user identities and enforce security best practices.**
✅ **Monitor and audit IAM activity for compliance.**
✅ **Ensure least-privilege access across AWS accounts.**

Final Exercise:

1. **Create an IAM user** and attach **AmazonS3ReadOnlyAccess**.
2. **Create a custom policy** to allow **only EC2 start/stop actions**.
3. **Monitor IAM access logs** using **AWS CloudTrail**.

AWS KMS (Key Management Service) – Study Material

Introduction to AWS KMS (Key Management Service)

What is AWS KMS?

AWS **Key Management Service (KMS)** is a **managed encryption service** that enables users to **create, store, and manage cryptographic keys** securely. It helps **encrypt and decrypt data** across AWS services and applications while maintaining **security, compliance, and auditing** capabilities.

Key Features of AWS KMS

✓ **Fully Managed Key Storage** – AWS securely stores and manages encryption keys.
✓ **Seamless Integration** – Works with **S3, RDS, DynamoDB, Lambda, EBS, and other AWS services**.
✓ **Key Rotation & Lifecycle Management** – Automatically rotates keys for security.
✓ **Access Control & IAM Integration** – Restricts key usage via IAM policies.
✓ **Audit Logging with CloudTrail** – Logs key usage and access history.
✓ **FIPS 140-2 Compliance** – Meets stringent security standards.
✓ **Multi-Region Keys** – Enables cross-region encryption and replication.

Chapter 1: Understanding AWS KMS Concepts

1. AWS KMS Key Types

AWS KMS supports different types of encryption keys:

Key Type	Description
Customer Managed Keys (CMK)	User-created keys with full control over permissions.
AWS Managed Keys	Automatically created and managed by AWS for specific services.
AWS Owned Keys	Used internally by AWS for encryption across services.
KMS Multi-Region Keys	Enable cross-region encryption for disaster recovery.

📌 **Example:**
A company **encrypts sensitive customer data** in **Amazon S3 using a Customer Managed Key (CMK)** to control access.

---

2. Symmetric vs. Asymmetric Keys in AWS KMS

Key Type	Use Case
Symmetric Keys	Encrypt & decrypt with the **same key** (used in most AWS services).
Asymmetric Keys	Uses a **public-private key pair** for digital signatures & encryption.

📌 **Example:**

- **Symmetric keys** encrypt S3 files, while
- **Asymmetric keys** verify digital signatures in blockchain applications.

CHAPTER 2: SETTING UP AWS KMS FOR ENCRYPTION

Step 1: Create a Customer Managed Key (CMK)

1. Open **AWS Console** → Navigate to **KMS (Key Management Service)**.
2. Click **"Create a Key"**.
3. Select **"Symmetric Key"** for general encryption use.
4. Choose **"Customer Managed Key (CMK)"**.
5. Define a **Key Alias** (e.g., MyAppEncryptionKey).
6. Set **Key Administrators** (IAM users who can manage the key).
7. Set **Key Users** (IAM roles that can use the key for encryption).
8. Enable **Automatic Key Rotation** (Recommended).
9. Click **"Create Key"**.

📌 **Expected Outcome:** A **CMK is created** and ready to be used for encryption.

Step 2: Encrypt and Decrypt Data Using AWS KMS

1. Encrypting Data with AWS KMS (CLI Command)

```
aws kms encrypt --key-id alias/MyAppEncryptionKey \
--plaintext "SensitiveData" --output text --query CiphertextBlob
```

2. Decrypting Data with AWS KMS (CLI Command)

```
aws kms decrypt --ciphertext-blob fileb://encryptedfile \
--output text --query Plaintext
```

📌 **Example:**
An **IoT device encrypts sensor data using KMS before sending it to an AWS database**.

Step 3: Enable S3 Bucket Encryption with AWS KMS

1. Open **Amazon S3 Console**.
2. Select a **Bucket** → Click **Properties**.
3. Under **Default Encryption,** select **AWS KMS**.
4. Choose the **Customer Managed Key (CMK)** created earlier.
5. Click **Save**.

📌 **Expected Outcome:** All **new objects in the S3 bucket are automatically encrypted** using AWS KMS.

CHAPTER 3: USING AWS KMS WITH OTHER AWS SERVICES

1. AWS KMS with Amazon RDS (Encrypting Databases)

- AWS KMS encrypts **RDS databases and snapshots**.
- Can be enabled **during RDS instance creation**.

📌 **Example:**
A **banking application encrypts its PostgreSQL RDS database** with a KMS CMK to protect customer transactions.

---

2. AWS KMS with Amazon DynamoDB (Encrypting NoSQL Data)

- AWS KMS encrypts **DynamoDB tables at rest**.
- Uses **either AWS Managed Keys or CMK**.

📌 **Example:**
A **social media platform encrypts user profile data** in DynamoDB using AWS KMS for compliance.

---

3. AWS KMS with Amazon Lambda (Securing API Keys & Secrets)

- AWS Lambda can **decrypt KMS-encrypted environment variables**.

📌 **Example:**
A **serverless application decrypts database credentials** before connecting to an RDS instance.

Chapter 4: AWS KMS Security Best Practices

✓ **Use Customer Managed Keys (CMKs)** for full control over encryption policies.
✓ **Enable Key Rotation** to automatically generate new keys.
✓ **Use IAM Policies & Key Policies** to restrict access.
✓ **Monitor Key Usage with AWS CloudTrail**.
✓ **Use Multi-Region Keys for Global Applications**.

📌 **Example:**
A **healthcare company follows HIPAA compliance** by restricting access to encryption keys using IAM policies.

Chapter 5: Monitoring and Auditing AWS KMS with CloudTrail

AWS CloudTrail records **all AWS KMS API requests,** allowing users to monitor:
✓ **Who accessed encryption keys**.

✓ **When decryption events occurred**.

✓ **Which services used AWS KMS**.

📌 **Example:**

A **cybersecurity team detects unauthorized decryption attempts** in CloudTrail logs and revokes IAM permissions.

Chapter 6: AWS KMS Multi-Region Keys (Cross-Region Encryption)

- **Multi-Region Keys enable disaster recovery and global applications.**
- Allows **encrypting data in one AWS region and decrypting in another**.
- Useful for **multi-region SaaS applications**.

📌 **Example:**

A **global financial app** encrypts data in **US-East-1** and **decrypts in Europe for compliance**.

Chapter 7: Real-World Use Cases of AWS KMS

✓ **E-Commerce & Payment Processing** – Encrypts **credit card details** before storage.

✓ **Healthcare & Compliance (HIPAA, GDPR)** – Encrypts **medical records & patient data**.

✓ **SaaS & Enterprise Applications** – Protects **customer credentials & API keys**.

✓ **Government & Defense** – Ensures **secure encryption for classified information**.

📌 Case Study: Amazon Uses AWS KMS

Problem: Needed **secure key management** for S3 & DynamoDB encryption.
Solution: Implemented **AWS KMS with automated key rotation**.
Outcome: Improved **security, compliance, and scalability**.

Conclusion: Mastering AWS KMS

By using **AWS KMS**, businesses can:

- ✅ **Securely encrypt sensitive data across AWS services**.
- ✅ **Manage cryptographic keys easily with full control**.
- ✅ **Meet compliance standards like PCI DSS, HIPAA, and GDPR**.
- ✅ **Monitor and audit key usage for security best practices**.

Final Exercise:

1. **Create a Customer Managed Key (CMK) in AWS KMS**.
2. **Encrypt a file using AWS KMS** and **decrypt it using CLI**.
3. **Enable S3 Bucket Encryption** using a KMS key and upload a test file.

AWS Shield & AWS WAF (Web Application Firewall) – Study Material

Introduction to AWS Shield & AWS WAF

What is AWS Shield?

AWS **Shield** is a **managed Distributed Denial of Service (DDoS) protection service** that safeguards applications running on AWS. It provides **automatic attack mitigation,** ensuring application availability and security.

AWS Shield comes in two versions:

1. **AWS Shield Standard** – Free, always-on protection against common DDoS attacks.
2. **AWS Shield Advanced** – Premium protection with **real-time monitoring, cost protection, and attack response support**.

What is AWS WAF (Web Application Firewall)?

AWS **WAF (Web Application Firewall)** protects **web applications from malicious traffic, SQL injection, cross-site scripting (XSS), and bot attacks**. It **monitors, filters, and blocks** HTTP(S) requests based on predefined security rules.

Key Features of AWS Shield & AWS WAF

✓ **Real-Time DDoS Protection** – Detects & mitigates volumetric, state-exhaustion, and application-layer attacks.
✓ **Managed Rules for Web Security** – Predefined WAF rules to block threats like SQL injection & XSS.
✓ **Integration with AWS Services** – Works with **CloudFront, ALB, API Gateway, and Route 53**.

✓ **Custom Security Rules** – Define rules to block specific IPs, countries, or bot patterns.

✓ **Bot Control** – Mitigates automated bot traffic & scraper attacks.

✓ **Threat Intelligence from AWS** – Advanced security insights with **AWS Shield Advanced**.

CHAPTER 1: AWS SHIELD – DDOS PROTECTION

1. How AWS Shield Works

AWS Shield provides **always-on DDoS protection** to mitigate attacks targeting:

- **Network Layer (Layer 3 & 4)** – Protects against **UDP flood, SYN flood, and reflection attacks**.
- **Application Layer (Layer 7)** – Prevents **HTTP floods and bot-based attacks**.

AWS Shield Standard	AWS Shield Advanced
Free tier for all AWS customers	Paid service for **enterprise-level DDoS protection**
Protects against common volumetric DDoS attacks	Real-time attack visibility & dedicated AWS response team
Automatic attack mitigation	Cost protection against unexpected DDoS-related expenses
Works with CloudFront, ALB, Route 53	Works with AWS WAF for advanced threat intelligence

📌 **Example:**
An **e-commerce website** uses **AWS Shield Standard** to prevent **DDoS attacks** from affecting checkout pages.

2. Setting Up AWS Shield Advanced

1. Open **AWS Shield Console** → Click **"Enable AWS Shield Advanced"**.
2. Select AWS resources to **protect (ALB, CloudFront, API Gateway, Route 53)**.
3. Enable **AWS WAF integration** for additional filtering.
4. Configure **AWS DDoS Response Team (DRT) support** (for assistance during attacks).
5. Click **"Enable"**.

📌 **Expected Outcome:**
AWS Shield Advanced **monitors and mitigates DDoS threats** in real-time.

CHAPTER 2: AWS WAF (WEB APPLICATION FIREWALL) OVERVIEW

1. How AWS WAF Works

AWS WAF **filters HTTP(S) traffic** based on:
✓ **Rate-Based Rules** – Limits requests per IP.
✓ **IP Blocking Rules** – Blocks traffic from specific IPs or countries.
✓ **SQL Injection & XSS Protection** – Prevents **web application vulnerabilities**.
✓ **Bot Control** – Blocks malicious bots and scrapers.

📌 **Example:**
A **news website** uses AWS WAF **rate-based rules** to prevent excessive traffic from bots scraping their content.

---

2. AWS WAF Rule Types

Rule Type	Purpose
IP Match Rule	Blocks or allows traffic from specific IPs or IP ranges.
Geo Match Rule	Restricts access from specific countries or regions.
Rate-Based Rule	Limits the number of requests per IP over time.
SQL Injection Rule	Detects and blocks SQL injection attacks.
Cross-Site Scripting (XSS) Rule	Prevents cross-site scripting attacks.

📌 **Example:**
A **financial application** blocks all traffic **except from the US** using **Geo Match Rules**.

Chapter 3: Setting Up AWS WAF for Web Protection

Step 1: Create a Web ACL (Access Control List)

1. Open **AWS WAF Console** → Click **"Create Web ACL"**.
2. Select **Resource Type** (CloudFront, ALB, API Gateway, AppSync).
3. Enter **Web ACL Name** (MyWebAppWAF).

Step 2: Add Security Rules

1. Click **"Add Rules"** → Choose **AWS Managed Rules**.

2. Select **"SQL Injection Rule Set" & "XSS Rule Set"**.
3. Add **Rate-Based Rule** (Limit 1000 requests per 5 minutes).
4. Add **Geo Restriction Rule** (Allow only US, Canada).

Step 3: Associate WAF with AWS Services

1. Select **CloudFront Distribution or Application Load Balancer (ALB)**.
2. Click **"Associate Web ACL"**.
3. Save and **Deploy WAF Rules**.

📌 **Expected Outcome:**
AWS WAF **blocks malicious requests** while allowing legitimate traffic.

CHAPTER 4: ADVANCED AWS SHIELD & WAF CONFIGURATIONS

1. AWS WAF Logging & Monitoring

✓ Enable **AWS CloudWatch Metrics** to track **blocked and allowed requests**.
✓ Use **AWS WAF Logs** to analyze **malicious request patterns**.
✓ Integrate with **AWS Security Hub & GuardDuty** for advanced threat detection.

📌 **Example:**
A **healthcare website** logs **WAF events to CloudWatch** and triggers **alerts for unusual activity**.

---

2. AWS Shield Advanced Cost Protection

✓ AWS **reimburses DDoS-related AWS charges** under Shield Advanced.

✓ Ensures **unexpected scaling costs from DDoS attacks are covered**.

✓ Monitors **attack traffic in real-time** via AWS Security Hub.

📌 **Example:**

A **stock trading platform** prevents **DDoS-related auto-scaling costs** using **Shield Advanced**.

Chapter 5: Best Practices for AWS Shield & WAF

✓ **Use AWS Shield Standard for basic DDoS protection** (free tier).

✓ **Enable AWS Shield Advanced for mission-critical applications**.

✓ **Use AWS WAF rate-limiting rules** to prevent excessive bot traffic.

✓ **Enable multi-layered security** (AWS WAF + Shield + CloudFront).

✓ **Regularly monitor WAF logs** for suspicious activity.

📌 **Example:**

A **government website** implements **Shield Advanced + AWS WAF + CloudFront** to **protect against DDoS and bot traffic**.

Chapter 6: Real-World Use Cases of AWS Shield & WAF

1. Protecting an E-Commerce Website from DDoS Attacks

✓ **Shield Standard** prevents **volumetric DDoS attacks**.

✓ **AWS WAF** blocks **malicious bots scraping product prices**.

✓ **Rate-Based Rules** prevent excessive checkout requests from fraudulent users.

---

2. Securing a Banking API from Injection Attacks

✓ **AWS WAF SQL Injection Rule** blocks attacks on login & payment APIs.
✓ **Geo Restriction Rule** allows access **only from trusted regions**.
✓ **Shield Advanced** protects against **DDoS attacks targeting online banking**.

---

3. Preventing Fake Reviews on a Social Media Platform

✓ **AWS WAF Bot Control** blocks fake users from posting spam comments.
✓ **Rate-Based Rules** limit the number of reviews per user per hour.
✓ **CloudWatch Logs + WAF Monitoring** detects **unusual login patterns**.

Conclusion: Mastering AWS Shield & WAF

By using **AWS Shield & AWS WAF**, businesses can:

✅ **Prevent DDoS attacks and ensure uptime**.
✅ **Block SQL injection, XSS, and bot attacks**.
✅ **Enforce rate-based limits to prevent API abuse**.
✅ **Monitor and audit security threats in real-time**.

Final Exercise:

1. **Enable AWS Shield Standard** for an ALB or CloudFront distribution.
2. **Create a Web ACL in AWS WAF** with **rate-limiting and SQL injection protection**.

3. **Analyze WAF logs** in CloudWatch and identify **potential threats**.

IDENTITY & COMPLIANCE

AWS COGNITO FOR USER AUTHENTICATION – STUDY MATERIAL

INTRODUCTION TO AWS COGNITO

What is AWS Cognito?

AWS **Cognito** is a **fully managed authentication and user management service** that enables developers to add **sign-up, sign-in, and access control** to web and mobile applications. It allows users to **authenticate using social, enterprise, or federated identity providers**.

Key Features of AWS Cognito

✓ **User Pool & Identity Pool** – Supports user authentication and federated access.
✓ **Multi-Factor Authentication (MFA)** – Enhances security with OTP-based login.
✓ **OAuth 2.0, SAML & OpenID Connect** – Enables single sign-on (SSO).
✓ **Federated Identity** – Supports **Google, Facebook, Apple, and Amazon login**.
✓ **Adaptive Authentication** – Uses AI to detect suspicious login attempts.
✓ **Secure Access to AWS Services** – Grants **temporary AWS credentials** via Identity Pools.

CHAPTER 1: AWS COGNITO COMPONENTS

AWS Cognito consists of **two main components**:

1. User Pools (Authentication & User Management)

- **Manages user registration, sign-in, and account recovery**.
- Supports **email/phone-based authentication**.
- Integrates with **social login providers (Google, Facebook, Apple)**.
- Provides **OAuth 2.0 authentication tokens** (ID, Access, and Refresh tokens).

📌 **Example:**
A **mobile banking app** uses Cognito User Pools for **secure user authentication** and password resets.

---

2. Identity Pools (Authorization & AWS Resource Access)

- Provides **temporary AWS credentials** to authenticated users.
- Grants users access to **S3, DynamoDB, Lambda, and other AWS services**.
- Works with **User Pools or third-party identity providers**.

📌 **Example:**
An **e-learning platform** uses Cognito Identity Pools to **grant temporary access to S3-hosted video content**.

CHAPTER 2: SETTING UP AWS COGNITO USER POOL

Step 1: Create a Cognito User Pool

1. Open **AWS Cognito Console** → Click **"Create a User Pool"**.
2. Enter a **Pool Name** (MyAppUserPool).

3. Choose **Sign-in options**:
    - Email, Phone, or Username-based authentication.
    - Enable **Multi-Factor Authentication (MFA)** (Optional).

Step 2: Configure Security Policies

1. Set **Password Policy** (minimum length, special characters).
2. Enable **Multi-Factor Authentication (MFA)** via **SMS or Authenticator App**.
3. Define **User Attributes** (e.g., email, phone_number).

Step 3: Create an App Client (OAuth2 Token Issuer)

1. Click **"Add App Client"**.
2. Enter an **App Name** (e.g., MyWebAppClient).
3. Enable **OAuth 2.0** scopes (openid, profile, email).
4. Disable **Client Secret (for mobile apps)**.
5. Click **Create App Client**.

📌 **Expected Outcome:**
A **User Pool is created**, ready for authentication.

CHAPTER 3: INTEGRATING AWS COGNITO WITH WEB/MOBILE APPS

1. Using Cognito Hosted UI for Authentication

AWS Cognito provides a **pre-built Hosted UI** for authentication.

Steps to Use Hosted UI

1. Open **Cognito Console** → Select **User Pool** → Click **"App Integration"**.

2. Copy the **Hosted UI URL** (e.g., https://myapp.auth.us-east-1.amazoncognito.com).
3. Configure **Allowed Callback URLs** (e.g., https://myapp.com/login).
4. Redirect users to the Hosted UI to **authenticate via Google, Facebook, or AWS Cognito**.

📌 **Example:**
A **React web app** redirects users to the Cognito Hosted UI for **sign-in and registration**.

---

2. Authenticating via AWS SDK (Using Amplify)

AWS **Amplify SDK** simplifies Cognito integration in web and mobile apps.

Install Amplify in a React App

```
npm install aws-amplify @aws-amplify/auth
```

Configure Cognito in React App

```
import Amplify from 'aws-amplify';
import awsconfig from './aws-exports';

Amplify.configure(awsconfig);
```

Sign Up a New User

```
import { Auth } from 'aws-amplify';

Auth.signUp({
```

```
  username: 'john_doe',
  password: 'StrongPass123!',
  attributes: { email: 'johndoe@example.com' }
})
.then(data => console.log("User signed up!", data))
.catch(err => console.log("Error signing up", err));
```

Sign In an Existing User

```
Auth.signIn('john_doe', 'StrongPass123!')
.then(user => console.log("User signed in!", user))
.catch(err => console.log("Error signing in", err));
```

📌 **Expected Outcome:**
Users can **sign up and sign in** using **Cognito User Pool authentication**.

CHAPTER 4: USING COGNITO IDENTITY POOLS FOR AWS ACCESS

1. Granting AWS Access to Authenticated Users

Cognito **Identity Pools** allow users to **access AWS resources securely**.

Steps to Create an Identity Pool

1. Open **AWS Cognito Console** → Click **"Create Identity Pool"**.
2. Enable **Authenticated and Unauthenticated Users**.
3. Choose **IAM Roles** for user access (CognitoAuthenticatedRole).
4. Click **"Create Pool"** → Attach IAM policies for AWS access.

Example IAM Policy for S3 Access

```
{
 "Effect": "Allow",
 "Action": ["s3:GetObject"],
 "Resource": "arn:aws:s3:::my-secure-bucket/*"
}
```

📌 **Example:**
A **mobile photo-sharing app** allows authenticated users to **upload images to S3 using temporary credentials**.

CHAPTER 5: AWS COGNITO SECURITY BEST PRACTICES

✓ **Enforce Strong Password Policies** – Minimum length, special characters, expiration.
✓ **Enable Multi-Factor Authentication (MFA)** – Protects against password theft.
✓ **Use Least Privilege Access** – Restrict IAM roles with minimal permissions.
✓ **Enable Adaptive Authentication** – Blocks suspicious login attempts.
✓ **Monitor Cognito Logs with CloudTrail** – Detect unauthorized access.

📌 **Example:**
A **financial services app enforces MFA and logs all Cognito activity** in **AWS CloudTrail**.

CHAPTER 6: AWS COGNITO REAL-WORLD USE CASES

1. User Authentication for E-Commerce Applications

✓ **Cognito User Pools** manage customer logins.
✓ **Social login (Google, Facebook, Apple) for seamless onboarding**.
✓ **Cognito Identity Pools** grant temporary AWS credentials for accessing order history.

📌 **Example:**
An **online store** lets users **sign in via Google and access their purchase history in DynamoDB**.

---

2. Secure API Authentication with Cognito & API Gateway

✓ Cognito **issues JWT tokens** for API authentication.
✓ AWS API Gateway **validates Cognito tokens** before processing requests.
✓ API Gateway **allows or denies access based on Cognito groups**.

📌 **Example:**
A **ride-sharing app** uses Cognito **to authenticate drivers and passengers before calling APIs**.

---

3. Mobile Banking Security with Cognito MFA

✓ **SMS-based MFA** for customer logins.
✓ **Cognito Identity Pools** for **temporary AWS resource access**.
✓ **Adaptive Authentication** blocks login attempts from unusual locations.

📌 **Example:**
A **mobile banking app** requires **MFA and adaptive authentication** for fraud detection.

Conclusion: Mastering AWS Cognito

By using **AWS Cognito,** businesses can:

✅ **Provide secure user authentication with social and enterprise logins**.

✅ **Implement role-based access control using IAM roles**.

✅ **Protect user data with MFA and adaptive authentication**.

✅ **Ensure scalability for millions of users with AWS's secure infrastructure**.

Final Exercise:

1. **Create a Cognito User Pool** and authenticate a test user.
2. **Integrate Cognito authentication into a React or mobile app** using Amplify.
3. **Set up Cognito Identity Pools** and grant AWS S3 access for authenticated users.

AWS Organizations and Access Control – Study Material

Introduction to AWS Organizations and Access Control

What is AWS Organizations?

AWS **Organizations** is a service that helps businesses **manage multiple AWS accounts centrally**. It allows administrators to apply **security policies, consolidated billing, access controls, and service permissions** across accounts in an **organization hierarchy**.

Key Features of AWS Organizations

✓ **Centralized Account Management** – Create and manage multiple AWS accounts under a single organization.
✓ **Service Control Policies (SCPs)** – Enforce fine-grained security controls across accounts.
✓ **Consolidated Billing** – Simplifies cost management by grouping billing under one account.
✓ **Organizational Units (OUs)** – Group AWS accounts for easy policy management.
✓ **Integration with AWS IAM** – Apply policies to control IAM access across multiple accounts.
✓ **Cross-Account Access Management** – Enables secure resource sharing between AWS accounts.

Chapter 1: Understanding AWS Organizations Structure

1. AWS Organizations Hierarchy

AWS Organizations consists of the following key components:

Component	Description

Management Account	The root AWS account that controls all member accounts.
Member Accounts	AWS accounts managed under AWS Organizations.
Organizational Units (OUs)	Logical groups of AWS accounts that can have common policies.
Service Control Policies (SCPs)	Security policies that restrict permissions across member accounts.

📌 **Example:**

A **company with multiple teams** (Finance, Development, Security) creates **OUs for each team** to enforce different security and billing policies.

CHAPTER 2: SETTING UP AWS ORGANIZATIONS

Step 1: Enable AWS Organizations

1. Open **AWS Organizations Console**.
2. Click **"Create an Organization"** → Select **"Enable All Features"**.
3. Choose **"Create Organization"**.

Step 2: Add Member Accounts to an Organization

1. Click **"Add Account"** → Choose **"Create an AWS Account"** or **"Invite an Existing AWS Account"**.
2. Enter the **Email ID** of the account to invite.
3. Click **"Send Invitation"**.

📌 **Expected Outcome:**
The invited account **joins the AWS Organization,** and policies can be applied centrally.

Step 3: Create Organizational Units (OUs)

1. Open **AWS Organizations Console** → Click **"Organizational Units"**.
2. Click **"Create Organizational Unit"**.
3. Enter **OU Name** (e.g., Security-Team).
4. Drag and drop **AWS accounts into the new OU**.
5. Click **"Save"**.

📌 **Expected Outcome:**
AWS accounts are **grouped into OUs for better management**.

CHAPTER 3: IMPLEMENTING SERVICE CONTROL POLICIES (SCPS) IN AWS ORGANIZATIONS

1. What are SCPs (Service Control Policies)?

SCPs are **organization-wide permission policies** that enforce security best practices.

Policy Name	Description
Deny All Except Billing	Restricts all actions except billing.
Prevent IAM User Creation	Blocks creation of new IAM users.
Restrict S3 Public Access	Denies public access to S3 buckets.

Policy Name	Description
Restrict Root Account Usage	Prevents root user from performing operations.

📌 **Example:**
A **company applies an SCP** to **prevent unauthorized users from creating IAM roles** in production accounts.

---

2. Creating an SCP to Restrict Specific AWS Services

1. Open **AWS Organizations Console** → Click **"Policies"**.
2. Click **"Create Policy"** → Enter **Policy Name** (Deny-S3-Public-Access).
3. Enter the following **SCP JSON Policy**:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "s3:PutBucketPublicAccessBlock",
          "Resource": "*"
    }
        ]
}
```

14. Click **"Attach Policy"** → Select the **OU or Account**.

📌 **Expected Outcome:**
All AWS accounts **under this OU are restricted from making S3 buckets public**.

Chapter 4: AWS Access Control and IAM Integration

1. IAM vs. AWS Organizations

Feature	IAM	AWS Organizations
Scope	User and role permissions inside an AWS account	Organization-wide security and service restrictions
Policy Type	IAM Policies	Service Control Policies (SCPs)
Management	Managed within a single AWS account	Centrally applied across multiple accounts

📌 **Example:**
A **company applies IAM policies** to limit S3 access **within an account**, while **AWS Organizations applies SCPs** to **all accounts** in an OU.

---

2. Cross-Account Access with AWS Organizations

AWS Organizations enables **secure access between AWS accounts** using:

✓ **IAM Roles** – Allows temporary access to another AWS account.

✓ **Resource Sharing via AWS Resource Access Manager (RAM)** – Shares AWS resources securely.

Example: Setting Up Cross-Account IAM Roles

1. **In Account A (Sender)** – Create an IAM Role:
2. {
3. "Effect": "Allow",
4. "Principal": { "AWS": "arn:aws:iam::123456789012:root" },
5. "Action": "sts:AssumeRole"
6. }
7. **In Account B (Receiver)** – Assume Role using AWS CLI:
8. aws sts assume-role --role-arn arn:aws:iam::123456789012:role/CrossAccountRole

📌 **Expected Outcome:**
Users from **Account B can securely access resources in Account A** using a temporary role.

CHAPTER 5: COST MANAGEMENT WITH AWS ORGANIZATIONS

1. Consolidated Billing in AWS Organizations

✓ **Single Payment Method** – All AWS accounts are billed under the **Management Account**.
✓ **Cost Savings with Reserved Instances** – Discounts apply across multiple accounts.
✓ **Detailed Billing Reports** – View costs per **OU, service, or individual account**.

📌 **Example:**
A **company centralizes billing for multiple AWS accounts,** simplifying **cost tracking and optimization**.

Chapter 6: AWS Organizations Best Practices

✓ **Use OUs to Separate Production, Development, and Security Accounts**.

✓ **Apply SCPs to Restrict Unauthorized Actions Across AWS Accounts**.

✓ **Enable MFA for All Accounts and IAM Users**.

✓ **Use IAM Roles for Secure Cross-Account Access**.

✓ **Monitor AWS Accounts Using AWS Security Hub and CloudTrail**.

📌 **Example:**
A **large enterprise sets up AWS Organizations** with:

- **Separate OUs for production and development**.
- **SCPs to restrict unauthorized changes**.
- **Consolidated billing for cost efficiency**.

Chapter 7: Real-World Use Cases of AWS Organizations & Access Control

1. Enterprise Security and Compliance Enforcement

✓ **OUs group accounts by department** (HR, Finance, IT).
✓ **SCPs restrict IAM modifications in security accounts**.
✓ **CloudTrail logs all security events for auditing**.

📌 **Example:**
A **healthcare company** enforces **HIPAA security policies across all AWS accounts** using SCPs.

---

2. Multi-Account Cost Optimization for SaaS Companies

✓ **Consolidated billing across multiple AWS accounts**.
✓ **Reserved instance savings applied globally**.
✓ **OUs separate customer environments**.

📌 **Example:**
A **SaaS company** optimizes costs by **grouping accounts under AWS Organizations**.

Conclusion: Mastering AWS Organizations and Access Control

By using **AWS Organizations,** businesses can:
✅ **Centrally manage multiple AWS accounts with security policies**.
✅ **Apply Service Control Policies (SCPs) for organization-wide restrictions**.
✅ **Enable cross-account access securely using IAM roles**.
✅ **Optimize costs through consolidated billing**.

Final Exercise:

1. **Create an AWS Organization and add member accounts**.
2. **Set up an Organizational Unit (OU) and apply an SCP**.
3. **Configure cross-account IAM roles** and test secure access.

Data Encryption & Compliance Frameworks – Study Material

Introduction to Data Encryption & Compliance Frameworks

What is Data Encryption?

Data encryption is the **process of converting data into an unreadable format** (ciphertext) to prevent unauthorized access. Only users with the correct **decryption key** can access the original data. Encryption ensures **data confidentiality, integrity, and compliance** with security regulations.

Key Features of Data Encryption

✓ **Protects sensitive data** at rest and in transit.
✓ **Uses cryptographic algorithms** (AES, RSA, SHA).
✓ **Prevents unauthorized access and data breaches**.
✓ **Ensures compliance with regulatory standards (HIPAA, GDPR, PCI DSS)**.

What are Compliance Frameworks?

Compliance frameworks are **industry-specific regulations and standards** that define how organizations should **store, process, and protect data** to ensure security and privacy.

Key Compliance Frameworks for Data Protection

✓ **GDPR (General Data Protection Regulation)** – Protects personal data in the EU.
✓ **HIPAA (Health Insurance Portability and Accountability Act)** – Secures healthcare information.
✓ **PCI DSS (Payment Card Industry Data Security Standard)** – Safeguards credit card transactions.

✓ **ISO 27001 (Information Security Standard)** – Provides a global framework for data security.

Chapter 1: Understanding Data Encryption

1. Types of Data Encryption

Encryption Type	Description	Use Cases
Symmetric Encryption	Uses a single key for encryption & decryption.	Secure database storage, file encryption.
Asymmetric Encryption	Uses a public key for encryption and a private key for decryption.	Digital signatures, secure key exchange.
Hashing	Converts data into a fixed-length hash value (irreversible).	Password storage, data integrity verification.

📌 **Example:**
A **financial institution** encrypts customer bank records using **AES-256 encryption** before storing them in AWS S3.

---

2. Encryption Algorithms & Standards

Algorithm	Type	Key Length	Use Cases
AES (Advanced Encryption Standard)	Symmetric	128, 192, 256-bit	Secure file storage, cloud encryption.

RSA (Rivest-Shamir-Adleman)	Asymmetric	1024, 2048, 4096-bit	Digital signatures, SSL/TLS encryption.
SHA (Secure Hash Algorithm)	Hashing	256, 512-bit	Password hashing, blockchain security.

📌 **Example:**
A **web application** encrypts API traffic using **RSA 2048-bit encryption with TLS 1.3**.

---

3. Data Encryption in AWS Cloud

AWS provides several services for **encrypting data at rest and in transit**:

✓ **AWS Key Management Service (KMS)** – Manages encryption keys securely.
✓ **AWS Secrets Manager** – Stores & rotates API keys and credentials.
✓ **AWS Certificate Manager (ACM)** – Issues and manages SSL/TLS certificates.
✓ **AWS CloudHSM (Hardware Security Module)** – Provides dedicated encryption hardware.

📌 **Example:**
A **SaaS company** encrypts **Amazon RDS databases using AWS KMS Customer Managed Keys (CMKs)**.

CHAPTER 2: COMPLIANCE FRAMEWORKS & REGULATIONS

1. GDPR (General Data Protection Regulation)

✓ Enforced by the **European Union** to protect user data privacy.
✓ Requires **data encryption, user consent, and breach notifications**.
✓ Grants users the **"Right to be Forgotten"** (data deletion upon request).

📌 **Example:**
An **e-commerce company encrypts customer payment details** to comply with **GDPR Article 32** on data security.

---

2. HIPAA (Health Insurance Portability and Accountability Act)

✓ Regulates **healthcare data (ePHI - Electronic Protected Health Information)**.
✓ Requires **encryption of patient records, audit logging, and access controls**.
✓ Enforces **Business Associate Agreements (BAAs) for third-party service providers**.

📌 **Example:**
A **hospital encrypts patient medical records in Amazon S3 using AWS KMS** to comply with **HIPAA encryption guidelines**.

---

3. PCI DSS (Payment Card Industry Data Security Standard)

✓ Governs **credit card transactions and payment processing**.
✓ Requires **encryption of cardholder data (CHD) and secure network policies**.
✓ Enforces **regular security audits and penetration testing**.

📌 **Example:**
A **payment gateway encrypts credit card transactions using AES-256** and **stores keys securely in AWS CloudHSM**.

---

4. ISO 27001 (International Information Security Standard)

✓ Global **standard for managing information security risks**.
✓ Requires **encryption, risk assessment, and continuous monitoring**.
✓ Enforces **security policies across organizations**.

📌 **Example:**
A **tech company implements ISO 27001 standards** by encrypting **all user passwords with SHA-256 hashing**.

CHAPTER 3: IMPLEMENTING ENCRYPTION IN AWS

1. Enabling AWS KMS for Data Encryption

1. Open **AWS KMS Console** → Click **"Create Key"**.
2. Choose **Key Type** (Symmetric or Asymmetric).
3. Set **Key Alias** (e.g., MyAppEncryptionKey).
4. Assign **IAM Users/Roles** for encryption permissions.
5. Click **"Create Key"**.

📌 **Expected Outcome:**
A **Customer Managed Key (CMK) is created** and can be used to encrypt AWS services.

---

2. Encrypting Data in Amazon S3

1. Open **S3 Console** → Select a **Bucket** → Click **Properties**.
2. Under **Default Encryption,** choose **AWS KMS**.
3. Select **an existing KMS key** or create a new one.
4. Click **"Save Changes"**.

📌 **Expected Outcome:**
All **new objects in the S3 bucket are encrypted** with **AWS KMS**.

---

3. Enabling Transparent Data Encryption (TDE) in RDS

1. Open **AWS RDS Console** → Select a **Database Instance**.
2. Click **Modify** → Enable **Encryption** under **Additional Configurations**.
3. Select **AWS KMS Key**.
4. Click **"Apply Changes"**.

📌 **Expected Outcome:**
Amazon **RDS encrypts all stored data using AWS KMS**.

CHAPTER 4: AWS COMPLIANCE & AUDITING TOOLS

AWS provides tools to **monitor and ensure compliance**:

✓ **AWS CloudTrail** – Logs all API calls for security audits.
✓ **AWS Config** – Monitors resource configurations for compliance violations.
✓ **AWS Security Hub** – Centralized security monitoring and compliance reporting.
✓ **AWS Audit Manager** – Automates compliance assessments for frameworks like **GDPR, HIPAA, PCI DSS**.

📌 **Example:**
A **financial company enables AWS Security Hub** to monitor **PCI DSS compliance across AWS accounts**.

Chapter 5: Best Practices for Secure Data Encryption

✓ **Use AWS KMS Customer Managed Keys (CMKs) for full control over encryption**.
✓ **Enable S3 Bucket Encryption and Block Public Access**.
✓ **Rotate Encryption Keys Regularly** for better security.
✓ **Use IAM Policies to Restrict Encryption Key Usage**.
✓ **Monitor Encryption Events with AWS CloudTrail**.

📌 **Example:**
A **government agency follows encryption best practices** by implementing **AWS CloudHSM for secure key storage**.

Chapter 6: Real-World Use Cases of Data Encryption & Compliance

1. Financial Data Protection for Banks

✓ **Encrypts customer transaction data** using AWS KMS.
✓ **Uses CloudTrail for auditing all data access attempts**.
✓ **Implements IAM least privilege access policies**.

📌 **Example:**
A **bank ensures PCI DSS compliance** by encrypting **credit card transactions with AES-256**.

---

2. Healthcare Data Security for Hospitals

✓ **Stores patient medical records in encrypted RDS databases**.
✓ **Enforces HIPAA-compliant access controls** with IAM roles.
✓ **Logs all API calls in AWS CloudTrail for security audits**.

📌 **Example:**
A **hospital encrypts medical imaging files in Amazon S3** to comply with **HIPAA security standards**.

Conclusion: Mastering Data Encryption & Compliance

By implementing **data encryption & compliance frameworks**, businesses can:

✅ **Secure sensitive data across AWS services**.
✅ **Ensure compliance with GDPR, HIPAA, PCI DSS, and ISO 27001**.
✅ **Use AWS KMS and CloudHSM for key management**.
✅ **Automate security audits using AWS Security Hub & Audit Manager**.

Final Exercise:

1. **Encrypt an S3 bucket using AWS KMS**.
2. **Enable PCI DSS compliance monitoring in AWS Security Hub**.
3. **Use AWS CloudTrail to audit data access events**.

ASSIGNMENT

DEPLOY A MYSQL DATABASE ON RDS AND CONNECT TO AN APPLICATION

SOLUTION: DEPLOY A MYSQL DATABASE ON AWS RDS AND CONNECT TO AN APPLICATION

This step-by-step guide will walk you through deploying a **MySQL database on Amazon RDS** and connecting it to a **web application** (e.g., a PHP or Node.js app).

Step 1: Log in to AWS and Navigate to RDS

1. Open **AWS Console** → Search for **RDS**.
2. Click **"Create database"**.

Step 2: Create an RDS MySQL Database

1. Choose Database Creation Method

- Select **Standard create**.
- Under **Engine Options,** choose **MySQL**.
- Select **MySQL Version 8.0** (or the latest stable version).

2. Configure Database Instance

- **DB Instance Identifier:** my-mysql-db.
- **Master Username:** admin.
- **Master Password:** YourSecurePassword123! (Store it securely).

3. Choose Instance Type and Storage

- **DB Instance Class:** db.t3.micro (for free tier) or db.t3.medium (for production).

- **Storage Type:** General Purpose (SSD).
- **Allocated Storage:** 20 GB (Auto-scaling enabled).

4. Configure Connectivity

- **VPC:** Choose **default VPC** or create a new one.
- **Subnet Group:** Select **default subnet group**.
- **Public Access: Enable** (for testing purposes; disable in production).
- **Security Group:** Choose **"Create new security group"**.

📌 **Modify Security Group to Allow Access**:

- Go to **EC2** → **Security Groups**.
- Select the RDS security group → Click **Inbound Rules** → **Edit**.
- **Add Rule:**
  - **Type:** MySQL/Aurora
  - **Port Range:** 3306
  - **Source:** 0.0.0.0/0 (for testing, restrict to your IP in production).
  - Click **Save Changes**.

5. Configure Database Settings

- **Initial Database Name:** myappdb.
- **Enable Automated Backups**: **Yes** (Backup retention: 7 days).
- **Enable Performance Insights**: **Yes** (for monitoring).

6. Create the RDS Instance

Click **"Create Database"** and wait for it to be **Available** (may take a few minutes).

📌 **Find RDS Endpoint**:

1. Navigate to **RDS Console → Databases**.
2. Click **"my-mysql-db"**.
3. Copy **"Endpoint"** (e.g., my-mysql-db.xyz123.us-east-1.rds.amazonaws.com).

Step 3: Connect to RDS MySQL Database

1. Connect Using MySQL CLI

1. Install **MySQL Client** on your local machine:
   - **Linux/Mac:**
   - sudo apt install mysql-client -y # Ubuntu/Debian
   - sudo yum install mysql -y # Amazon Linux
   - **Windows:** Install **MySQL Workbench** from MySQL Website.
2. Connect to RDS:
3. mysql -h my-mysql-db.xyz123.us-east-1.rds.amazonaws.com -u admin -p

Enter your **password** when prompted.

📌 **Expected Outcome:**
You are now connected to your **AWS RDS MySQL database**.

---

2. Create a Sample Table

Once connected, run:

```
CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(100),
  email VARCHAR(100) UNIQUE,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

📌 **Expected Outcome:**
The users table is created.

Step 4: Connect a Web Application to MySQL RDS

1. Connecting a PHP Application

Install **MySQL PDO extension** and update your **PHP app's database configuration**:

```
<?php
$host = "my-mysql-db.xyz123.us-east-1.rds.amazonaws.com";
$dbname = "myappdb";
$username = "admin";
$password = "YourSecurePassword123!";

try {
  $pdo = new PDO("mysql:host=$host;dbname=$dbname", $username, $password);
```

```
    $pdo->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
    echo "Connected successfully!";
} catch (PDOException $e) {
    die("Connection failed: " . $e->getMessage());
}
?>
```

📌 **Expected Outcome:**
If connected successfully, the script prints **"Connected successfully!"**.

---

2. Connecting a Node.js Application

1. Install MySQL package:
2. npm install mysql2
3. Create a **db.js** file in your Node.js project:
4. const mysql = require('mysql2');
5.
6. const connection = mysql.createConnection({
7. host: "my-mysql-db.xyz123.us-east-1.rds.amazonaws.com",
8. user: "admin",
9. password: "YourSecurePassword123!",
10. database: "myappdb"
11. });
12.

```
13.connection.connect(err => {
14.         if (err) {
15.     console.error("Database connection failed: " + err.stack);
16.           return;
17.  }
18.         console.log("Connected to MySQL database.");
19.       });
20.
21.       module.exports = connection;
22.       Run your Node.js app:
23.       node db.js
```

📌 **Expected Outcome:**
Terminal prints **"Connected to MySQL database."**

Step 5: Secure & Optimize Your RDS MySQL Database

1. Restrict Public Access (Production Only)

1. Go to **RDS Console** → **Select your Database**.
2. Click **Modify** → Set **Public Access = No**.
3. Apply changes and **restart the instance**.

2. Enable Multi-AZ Deployment (For High Availability)

1. Open **RDS Console** → Click **Modify**.
2. Enable **Multi-AZ Deployment** for **automatic failover**.
3. Click **Save Changes**.

3. Enable IAM Authentication for Secure Login

1. Open **AWS RDS Console** → Select **MySQL Database**.
2. Click **Modify** → Enable **IAM Database Authentication**.
3. Use **IAM roles** instead of passwords for authentication.

Step 6: Monitoring & Scaling MySQL Database

1. Monitor RDS Performance with Amazon CloudWatch

✓ Open **AWS CloudWatch** → **Metrics** → **RDS** to monitor:

- **CPU Utilization**
- **Read/Write IOPS**
- **Database Connections**

📌 **Example:** Set up a CloudWatch Alarm for **CPU Usage > 75%**.

---

2. Scale RDS MySQL Instance (Vertical Scaling)

1. Open **AWS RDS Console** → Click **Modify**.
2. Change **Instance Class** (e.g., db.t3.medium → db.m5.large).
3. Click **Apply Changes** (Apply immediately or during a maintenance window).

📌 **Expected Outcome:**
RDS scales **vertically** to handle higher loads.

---

3. Scale RDS MySQL with Read Replicas (Horizontal Scaling)

1. Open **RDS Console** → Click **Create Read Replica**.

2. Choose a **region** and **instance class**.
3. Click **Create** → Use replica for **read-heavy workloads**.

📌 **Expected Outcome:**
The **read replica** helps offload database read operations.

Conclusion: Successfully Deployed MySQL RDS & Connected to an Application

By following this guide, you have:

✅ **Created an AWS RDS MySQL Database.**
✅ **Configured security & networking settings.**
✅ **Connected the RDS MySQL database to a web application.**
✅ **Enabled monitoring & performance optimization.**
✅ **Secured the database for production use.**

Final Exercise:

1. **Create an IAM-based authentication for MySQL RDS.**
2. **Implement automatic backup retention for 7 days.**
3. **Deploy an AWS Lambda function to fetch data from RDS.**

IMPLEMENT IAM POLICIES AND TEST ROLE-BASED ACCESS

SOLUTION: IMPLEMENT IAM POLICIES AND TEST ROLE-BASED ACCESS

This step-by-step guide will walk you through **creating an IAM policy, assigning it to a role, and testing role-based access** in AWS.

Step 1: Log in to AWS IAM Console

1. Open the **AWS Management Console**.
2. Navigate to **IAM (Identity and Access Management)**.

Step 2: Create a Custom IAM Policy

1. Navigate to Policies

1. In the **IAM Console**, click **"Policies"** in the left menu.
2. Click **"Create Policy"**.

2. Define the Policy Permissions

- Select **"JSON"** and paste the following policy that grants **read-only access to Amazon S3**:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
```

```
            "s3:ListBucket",
            "s3:GetObject"
        ],
        "Resource": [
            "arn:aws:s3:::my-secure-bucket",
            "arn:aws:s3:::my-secure-bucket/*"
        ]
    }
  ]
}
```

📌 **Explanation:**
✓ **Allows listing the bucket and reading objects in "my-secure-bucket".**
✓ **Does NOT allow modifying or deleting objects.**

3. Name and Create the Policy

1. Click **"Next: Tags"** → (Optional: Add tags).
2. Click **"Next: Review"**.
3. **Policy Name:** S3ReadOnlyAccess.
4. Click **"Create Policy"**.

📌 **Expected Outcome:** A **custom IAM policy** is created successfully.

Step 3: Create an IAM Role and Attach the Policy

1. Navigate to IAM Roles

1. Click **"Roles"** → **"Create Role"**.
2. **Choose Trusted Entity:** Select **AWS service**.
3. **Select Use Case:** Choose **EC2** (if assigning to an instance).
4. Click **"Next"**.

2. Attach the Custom Policy

1. Search for S3ReadOnlyAccess (the policy created earlier).
2. Select it → Click **"Next"**.

3. Name and Create the Role

1. **Role Name:** S3ReadOnlyRole.
2. **Description:** "Grants read-only access to S3 bucket".
3. Click **"Create Role"**.

📌 **Expected Outcome:**
The **IAM role is created and ready to be assigned** to users or services.

Step 4: Assign the IAM Role to an EC2 Instance (Optional)

1. Navigate to **EC2 Console** → Click **"Instances"**.
2. Select an EC2 instance → Click **"Actions"** → **"Security"** → **"Modify IAM Role"**.
3. Select **S3ReadOnlyRole** → Click **"Update IAM Role"**.

📌 **Expected Outcome:**
The **EC2 instance now has read-only access to the S3 bucket**.

Step 5: Testing Role-Based Access (AWS CLI or Console)

1. Testing S3 Access via AWS CLI

1. SSH into the EC2 instance:
2. ssh -i my-key.pem ec2-user@your-ec2-public-ip
3. Run the following AWS CLI command to **list S3 buckets**:
4. aws s3 ls
5. Try to **list objects in the bucket**:
6. aws s3 ls s3://my-secure-bucket
7. Attempt to **upload a file (which should fail)**:
8. echo "Test file" > test.txt
9. aws s3 cp test.txt s3://my-secure-bucket/

📌 **Expected Outcome:**
✓ **Listing objects should work** ✅.
❌ **Uploading files should fail (due to read-only access)** ❌.

Step 6: Modify IAM Policy to Grant Full Access (Optional)

If you need to grant **full access** instead of read-only, modify the policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
        "Action": "s3:*",
        "Resource": [
          "arn:aws:s3:::my-secure-bucket",
          "arn:aws:s3:::my-secure-bucket/*"
        ]
      }
    ]
}
```

📌 **Expected Outcome:**
Users with this role can now **read, write, and delete objects** in S3.

Step 7: Monitor IAM Role Activity with AWS CloudTrail

1. Navigate to **AWS CloudTrail Console**.
2. Click **"Event History"** → Filter by **IAM Role Activity**.
3. Check if unauthorized access attempts appear (e.g., trying to delete objects).

📌 **Expected Outcome:**
IAM access logs show **successful and failed API calls**, helping in **security audits**.

CONCLUSION: SUCCESSFULLY IMPLEMENTED IAM POLICIES & ROLE-BASED ACCESS

By following this guide, you have:

✅ **Created a custom IAM policy** for S3 access.

- ✅ **Assigned the policy to an IAM role.**
- ✅ **Attached the IAM role to an EC2 instance.**
- ✅ **Tested role-based access using AWS CLI.**
- ✅ **Monitored IAM activity using AWS CloudTrail.**

FINAL EXERCISE:

1. **Modify the IAM policy** to grant write access to a specific folder in S3 (my-secure-bucket/logs/).
2. **Create an IAM policy that restricts EC2 instance termination.**
3. **Enable IAM MFA (Multi-Factor Authentication) for IAM users.**

SECURE A WEB APPLICATION USING AWS WAF

SOLUTION: SECURE A WEB APPLICATION USING AWS WAF (WEB APPLICATION FIREWALL)

This step-by-step guide will walk you through **securing a web application using AWS WAF** by creating **a Web ACL,** setting up **WAF rules,** and integrating it with **Amazon CloudFront or an Application Load Balancer (ALB)**.

Step 1: Log in to AWS and Navigate to AWS WAF Console

1. Open the **AWS Management Console**.
2. Search for **"AWS WAF & Shield"** in the **Services** search bar.
3. Click **"AWS WAF"** to open the console.

Step 2: Create a Web ACL (Access Control List)

1. Click **"Create Web ACL"**.
2. Enter **Web ACL Name** (MyWebAppWAF).
3. Select **Resource Type** (Choose where you want to apply WAF):
    - **CloudFront** (Recommended for global applications).
    - **Application Load Balancer (ALB)** (Best for backend services).
    - **API Gateway** (Secures REST APIs).
4. Click **"Next"**.

📌 **Expected Outcome:** A **Web ACL** is created and ready for rules.

Step 3: Configure Security Rules in AWS WAF

1. Block SQL Injection Attacks

1. Click **"Add Rules"** → Select **"Add Managed Rule Groups"**.
2. Search for **"AWS WAF SQL Injection Rule Set"**.
3. Select it and click **"Add Rules"**.

2. Block Cross-Site Scripting (XSS) Attacks

1. Click **"Add Rules"** → Select **"AWS WAF XSS Rule Set"**.
2. Click **"Add Rules"**.

3. Restrict Traffic from Specific Countries (Geo-Blocking)

1. Click **"Add Rules"** → Select **"Add My Own Rules and Rule Groups"**.
2. Click **"Create Rule"**.
3. **Rule Name:** BlockCountryRule.
4. Choose **"Geographic Match"** → Select countries to block (e.g., China, Russia).
5. Click **"Save Rule"** → **"Add to Web ACL"**.

4. Rate-Limit Requests to Prevent DDoS & Bots

1. Click **"Add Rules"** → Choose **"Rate-Based Rule"**.
2. Enter **Rule Name:** RateLimitRequests.
3. Set **Rate Limit** (e.g., 1000 requests per 5 minutes).
4. Click **"Save Rule"** → **"Add to Web ACL"**.

5. Allow Only Specific IPs (Whitelist)

1. Click **"Add Rules"** → Select **"IP Set"**.
2. Enter **Rule Name:** AllowedIPs.
3. Add **specific IP addresses** (e.g., 192.168.1.10/32).
4. Click **"Save Rule"** → **"Add to Web ACL"**.

📌 **Expected Outcome:** The **Web ACL now includes rules to prevent SQL Injection, XSS, Geo-blocking, Rate Limiting, and IP-based filtering**.

Step 4: Associate WAF with AWS Resources

1. Associate WAF with CloudFront (Recommended for Global Protection)

1. Go to **AWS WAF Console** → **Web ACLs**.
2. Select **"MyWebAppWAF"** → Click **"Add AWS Resources"**.
3. Select **"CloudFront Distribution"** → Click **"Associate"**.

2. Associate WAF with an Application Load Balancer (ALB)

1. Open **AWS EC2 Console** → Click **"Load Balancers"**.
2. Select your **ALB** → Click **"Edit Web ACL Association"**.
3. Choose **"MyWebAppWAF"** → Click **"Associate"**.

📌 **Expected Outcome:** WAF is **protecting incoming traffic** to the **web application**.

Step 5: Test and Monitor AWS WAF Rules

1. Test with a SQL Injection Attack

1. Open a browser → Enter the following URL:

2. https://your-web-app.com/?id=1' OR '1'='1'
3. If WAF is working, the request should be **blocked**.

2. Test with a DDoS Attack Simulation

1. Run an HTTP load testing tool (like Apache Bench or ab command):
2. ab -n 1000 -c 50 https://your-web-app.com/
3. If **rate-limiting is applied**, requests beyond the set limit should be **blocked**.

3. Monitor WAF Logs in CloudWatch

1. Open **AWS CloudWatch Console** → Click **"Logs"**.
2. Search for **AWS WAF logs**.
3. Check blocked requests and rule matches.

📌 **Expected Outcome:**
You should see **blocked SQL injection attempts, geo-blocked IPs, and rate-limited requests** in WAF logs.

Step 6: Enable AWS Shield for DDoS Protection (Optional but Recommended)

1. Navigate to **AWS Shield Console**.
2. Click **"Enable AWS Shield Advanced"** (requires AWS Enterprise Support).
3. Select **CloudFront or ALB** as protected resources.
4. Click **"Enable Protection"**.

📌 **Expected Outcome:** Your **web application is now protected against Layer 3/4 DDoS attacks**.

Step 7: Enable AWS Security Hub for Continuous Monitoring

1. Open **AWS Security Hub** → Click **"Enable Security Hub"**.
2. Enable **AWS WAF integration**.
3. Monitor **security threats** and **WAF rule violations**.

📌 **Expected Outcome:**
AWS Security Hub provides **detailed insights into web security threats**.

Conclusion: Successfully Secured a Web Application Using AWS WAF

By following this guide, you have:

✅ **Created a Web ACL in AWS WAF**.
✅ **Added security rules for SQL Injection, XSS, Geo-Blocking, and Rate Limiting**.
✅ **Associated WAF with CloudFront or ALB for web application protection**.
✅ **Tested WAF by simulating attacks**.
✅ **Monitored security events using AWS CloudWatch & Security Hub**.

Final Exercise:

1. **Create a custom AWS WAF rule to allow traffic only from your corporate IP range**.

2. **Enable bot control rules to block automated traffic.**
3. **Set up a CloudWatch alarm to trigger when WAF blocks more than 100 requests in an hour.**