Independent Skill Development Mission

Independent Skill Development Mission NXT
EMPOWERING FUTURES, SHAPING TOMORROW

ISDM (INDEPENDENT SKILL DEVELOPMENT MISSION)

AWS COMPUTE SERVICES

EC2 DEEP DIVE: AMIS, KEY PAIRS, AND SECURITY GROUPS

AMAZON EC2 (**ELASTIC COMPUTE CLOUD**) IS ONE OF THE **MOST POWERFUL COMPUTING SERVICES** OFFERED BY AWS, ENABLING BUSINESSES TO **LAUNCH, CONFIGURE, AND MANAGE VIRTUAL SERVERS (INSTANCES)** IN THE CLOUD. A DEEP UNDERSTANDING OF **AMIS, KEY PAIRS, AND SECURITY GROUPS** IS CRUCIAL FOR EFFECTIVELY DEPLOYING AND SECURING EC2 INSTANCES.

THIS STUDY MATERIAL PROVIDES AN IN-DEPTH **EXPLANATION, EXAMPLES, EXERCISES, AND BEST PRACTICES** FOR **AMAZON MACHINE IMAGES (AMIS), KEY PAIRS, AND SECURITY GROUPS**.

CHAPTER 1: AMAZON MACHINE IMAGES (AMIS) – THE FOUNDATION OF EC2 INSTANCES

WHAT IS AN AMI (AMAZON MACHINE IMAGE)?

AN **AMAZON MACHINE IMAGE (AMI)** IS A **PRE-CONFIGURED TEMPLATE** THAT CONTAINS EVERYTHING REQUIRED TO LAUNCH AN EC2 INSTANCE,

INCLUDING:

✓ **OPERATING SYSTEM (OS)** – LINUX, WINDOWS, OR CUSTOM OS VERSIONS.

✓ **SOFTWARE PACKAGES** – PRE-INSTALLED APPLICATIONS LIKE WEB SERVERS, DATABASES, AND SECURITY TOOLS.

✓ **CONFIGURATIONS & PERMISSIONS** – CUSTOMIZED SETTINGS AND PRE-DEFINED IAM ROLES.

TYPES OF AMIS IN AWS

AWS PROVIDES SEVERAL TYPES OF AMIS BASED ON **CUSTOMIZATION AND OWNERSHIP**:

1. **AWS-PROVIDED AMIS**
   - INCLUDES AMAZON LINUX, UBUNTU, WINDOWS SERVER, AND MORE.
   - REGULARLY UPDATED BY AWS FOR SECURITY AND PERFORMANCE.
2. **MARKETPLACE AMIS**
   - PRE-CONFIGURED AMIS FROM THIRD-PARTY VENDORS (E.G., RED HAT, SUSE, SQL SERVER).
   - AVAILABLE FOR PAY-AS-YOU-GO LICENSING.
3. **CUSTOM AMIS**
   - USERS CAN **CREATE THEIR OWN AMIS** BY CUSTOMIZING AN EC2 INSTANCE AND SAVING IT AS AN AMI.
   - USEFUL FOR **STANDARDIZING DEPLOYMENTS** ACROSS MULTIPLE INSTANCES.

EXAMPLE:

A COMPANY WANTS TO LAUNCH **MULTIPLE EC2 INSTANCES** WITH **PRE-INSTALLED WEB SERVERS (APACHE, PHP, AND MYSQL)**. INSTEAD OF CONFIGURING EACH INSTANCE MANUALLY, THEY **CREATE A CUSTOM AMI** AND USE IT FOR FUTURE DEPLOYMENTS.

EXERCISE:

1. LAUNCH AN EC2 INSTANCE USING AN **AWS-PROVIDED AMI** (AMAZON LINUX 2023).
2. CREATE A **CUSTOM AMI** FROM AN EXISTING EC2 INSTANCE.

CHAPTER 2: EC2 KEY PAIRS – SECURE SSH ACCESS

WHAT IS AN EC2 KEY PAIR?

AWS USES **KEY PAIRS (PUBLIC & PRIVATE KEYS)** FOR **SECURE AUTHENTICATION** TO EC2 INSTANCES. INSTEAD OF USING PASSWORDS, EC2 INSTANCES USE SSH KEYS FOR ACCESS.

✓ **PUBLIC KEY** – STORED IN THE EC2 INSTANCE.
✓ **PRIVATE KEY** – DOWNLOADED AND STORED SECURELY ON THE USER'S SYSTEM.

CREATING AND USING KEY PAIRS

1. CREATE AN EC2 KEY PAIR

1. OPEN **AWS MANAGEMENT CONSOLE** → NAVIGATE TO **EC2**.
2. CLICK **"KEY PAIRS"** → **"CREATE KEY PAIR"**.
3. ENTER A **KEY PAIR NAME** (MY-KEY-PAIR).
4. CHOOSE **RSA (DEFAULT) OR ED25519 (MORE SECURE)**.
5. CLICK **"CREATE KEY PAIR"** AND DOWNLOAD THE .PEM FILE.

2. Use Key Pair to Connect to EC2 Instance

For **Linux/macOS**: Open Terminal and use SSH:

```
ssh -i my-key-pair.pem ec2-user@your-ec2-public-ip
```

For **Windows**: Use **PuTTY** (convert .pem to .ppk format using PuTTYgen).

Best Practices for Key Pairs

✓ Store the **private key** securely; do not share it.
✓ Use **different key pairs for different environments** (dev, staging, production).
✓ Rotate key pairs periodically for **enhanced security**.

Example:

A developer creates an EC2 instance and **downloads a Key Pair**. They use the **private key** to SSH into the instance securely, without using a password.

Exercise:

1. Create an EC2 **Key Pair** and launch an EC2 instance using it.
2. **Connect to the EC2 instance via SSH** using the key pair.

Chapter 3: EC2 Security Groups – Firewall Rules for Instances

What is a Security Group?

A **Security Group** is an **AWS firewall that controls inbound and outbound traffic** to EC2 instances. It acts as a **virtual firewall** and determines:

✓ **Who can connect** to the instance.
✓ **Which ports and protocols** are allowed.

Security Group Rules

Security Groups consist of **Inbound and Outbound rules**:

- **Inbound Rules** – Define what type of traffic is **allowed into the instance** (e.g., SSH, HTTP).
- **Outbound Rules** – Define what type of traffic the **instance can send out** (by default, all traffic is allowed).

Configuring Security Group Rules

1. **Default Security Group** (Restrictive by Default)
    - Blocks all inbound traffic.
    - Allows all outbound traffic.
2. **Common Security Group Configurations:**

Rule Name	Protocol	Port Range	Source
SSH	TCP	22	My IP
HTTP	TCP	80	0.0.0.0/0 (All)
HTTPS	TCP	443	0.0.0.0/0 (All)
RDP	TCP	3389	My IP

Example:

A business hosts a **public website on EC2**. Their Security Group:
✓ Allows **HTTP (80) and HTTPS (443)** from anywhere.
✓ Allows **SSH (22) only from the administrator's IP**.
✓ Blocks all other traffic.

Exercise:

1. **Create a Security Group** that allows SSH (22) only from your IP.

2. **Modify an existing Security Group** to allow HTTP and HTTPS traffic.

Chapter 4: Best Practices for AMIs, Key Pairs, and Security Groups

1. AMI Best Practices

✓ **Use AWS-provided AMIs** for up-to-date, secure OS images.
✓ **Create custom AMIs** for frequently used application setups.
✓ **Enable AMI encryption** for sensitive data storage.

2. Key Pair Best Practices

✓ **Never share private keys** or store them in publicly accessible locations.
✓ **Use different key pairs** for different environments.
✓ **Rotate keys periodically** and create new ones if compromised.

3. Security Group Best Practices

✓ **Limit SSH/RDP access to specific IPs** (My IP instead of 0.0.0.0/0).
✓ **Use separate Security Groups** for different application tiers.
✓ **Restrict outbound traffic** where possible for better security.

Example:

A company deploying a **multi-tier web application** configures:
✓ **Web Server Security Group** → Allows HTTP & HTTPS but blocks SSH.
✓ **Database Server Security Group** → Allows connections only from the web server.
✓ **Admin Security Group** → Allows SSH from admin's IP only.

Conclusion: Mastering EC2 AMIs, Key Pairs, and Security Groups

By understanding **AMIs, Key Pairs, and Security Groups**, you can:

- ✅ **Efficiently deploy and configure EC2 instances.**
- ✅ **Ensure secure authentication** with **Key Pairs.**
- ✅ **Manage instance access using Security Groups.**

Final Exercise:

1. **Launch an EC2 instance** with a **Custom AMI** and a **new Key Pair.**
2. **Configure a Security Group** that only allows SSH and HTTP access.
3. **Test your setup** by connecting to the instance and accessing a web server.

Elastic Load Balancer (ELB) & Auto Scaling in AWS

In cloud computing, ensuring **high availability, fault tolerance, and scalability** is essential for applications. AWS provides **Elastic Load Balancer (ELB) and Auto Scaling** to help applications **handle traffic efficiently, prevent downtime, and optimize costs**.

This study material provides an in-depth understanding of **ELB and Auto Scaling**, covering **types, benefits, best practices, and real-world use cases**.

Chapter 1: Elastic Load Balancer (ELB) – Distributing Traffic Efficiently

What is an Elastic Load Balancer (ELB)?

An **Elastic Load Balancer (ELB)** is a **managed AWS service** that automatically distributes incoming traffic across multiple EC2 instances, ensuring that no single server is overwhelmed.

Key Features of ELB:

✓ **High Availability** – Ensures application uptime by routing traffic to healthy instances.
✓ **Scalability** – Works with Auto Scaling to adjust resources dynamically.
✓ **Security** – Supports **SSL/TLS encryption, DDoS protection, and AWS Shield**.
✓ **Health Checks** – Monitors instances and removes unhealthy ones from rotation.

Types of AWS Elastic Load Balancers

AWS PROVIDES **FOUR TYPES OF ELBS,** EACH DESIGNED FOR DIFFERENT USE CASES:

LOAD BALANCER TYPE	USE CASE
APPLICATION LOAD BALANCER (ALB)	ROUTES TRAFFIC BASED ON HTTP/HTTPS (LAYER 7)
NETWORK LOAD BALANCER (NLB)	HANDLES HIGH-PERFORMANCE, TCP/UDP TRAFFIC (LAYER 4)
GATEWAY LOAD BALANCER (GWLB)	USED FOR THIRD-PARTY SECURITY APPLIANCES (FIREWALLS, PROXIES)
CLASSIC LOAD BALANCER (CLB)	LEGACY LOAD BALANCER, NOT RECOMMENDED FOR NEW DEPLOYMENTS

1. APPLICATION LOAD BALANCER (ALB) – LAYER 7 ROUTING

- BEST FOR **WEB APPLICATIONS** THAT NEED **INTELLIGENT ROUTING**.
- SUPPORTS **HOST-BASED AND PATH-BASED ROUTING** (E.G., ROUTE /IMAGES TO ONE SERVER AND /VIDEOS TO ANOTHER).

2. NETWORK LOAD BALANCER (NLB) – LAYER 4 PERFORMANCE

- BEST FOR **HIGH-PERFORMANCE, LOW-LATENCY APPLICATIONS**.
- SUPPORTS **MILLIONS OF REQUESTS PER SECOND**.

3. CLASSIC LOAD BALANCER (CLB) – LEGACY SOLUTION

- SUPPORTS **BASIC LAYER 4 AND LAYER 7 ROUTING**.
- RECOMMENDED ONLY FOR OLDER APPLICATIONS.

EXAMPLE:

A **GLOBAL E-COMMERCE WEBSITE** USES **APPLICATION LOAD BALANCER (ALB)** TO DISTRIBUTE TRAFFIC AMONG **BACKEND WEB SERVERS,** ENSURING SEAMLESS USER EXPERIENCE.

EXERCISE:

1. CREATE AN **APPLICATION LOAD BALANCER** IN AWS.
2. ATTACH IT TO AN **EC2 AUTO SCALING GROUP** TO DISTRIBUTE TRAFFIC DYNAMICALLY.

CHAPTER 2: AUTO SCALING – DYNAMIC RESOURCE MANAGEMENT

WHAT IS AUTO SCALING?

AWS **AUTO SCALING** IS A FEATURE THAT **AUTOMATICALLY ADJUSTS THE NUMBER OF EC2 INSTANCES** BASED ON DEMAND. IT ENSURES **OPTIMAL PERFORMANCE AND COST-EFFICIENCY** BY ADDING OR REMOVING INSTANCES AS NEEDED.

KEY FEATURES OF AUTO SCALING:

✓ **IMPROVES PERFORMANCE** – AUTOMATICALLY INCREASES CAPACITY DURING PEAK LOADS.
✓ **OPTIMIZES COST** – REDUCES INSTANCES WHEN DEMAND IS LOW, SAVING MONEY.
✓ **FAULT TOLERANCE** – REPLACES FAILED INSTANCES TO MAINTAIN UPTIME.
✓ **FULLY MANAGED** – INTEGRATES WITH ELB FOR SEAMLESS TRAFFIC DISTRIBUTION.

AUTO SCALING COMPONENTS

Component	Description
Launch Template	Defines instance type, AMI, key pair, and user data
Auto Scaling Group (ASG)	Group of instances managed together
Scaling Policies	Rules for adding or removing instances
Health Checks	Determines if instances should be replaced

Types of Scaling Policies in Auto Scaling

1. **Dynamic Scaling** – Adjusts capacity **automatically based on demand**.
2. **Scheduled Scaling** – Adds/removes instances at **fixed times**.
3. **Predictive Scaling** – Uses **machine learning** to forecast demand.

Example:

A **news website** experiences **high traffic during breaking news events**. Auto Scaling automatically **adds EC2 instances** to handle traffic spikes and removes them when traffic decreases.

Exercise:

1. Create an **Auto Scaling Group** with a minimum of **2 instances**.
2. Set a **scaling policy** to add an instance if CPU usage exceeds **70%**.

Chapter 3: Setting Up ELB and Auto Scaling

Step 1: Create an Application Load Balancer (ALB)

1. Go to AWS Console → EC2 → Load Balancers.
2. Click "Create Load Balancer" → Select Application Load Balancer.
3. Name the ALB (e.g., my-app-load-balancer).
4. Choose "Internet-facing" for public access.
5. Select subnets across multiple Availability Zones.
6. Configure listener settings (e.g., HTTP/HTTPS).
7. Create a target group and register EC2 instances.
8. Click "Create Load Balancer".

Step 2: Create an Auto Scaling Group

1. Go to Auto Scaling Groups in AWS EC2.
2. Click "Create Auto Scaling Group".
3. Select an existing launch template or create a new one.
4. Choose minimum, maximum, and desired capacity (e.g., Min: 2, Max: 5).
5. Attach the ALB Target Group for traffic distribution.
6. Set a Scaling Policy (e.g., Add instance when CPU > 70%).
7. Click "Create Auto Scaling Group".

Chapter 4: ELB & Auto Scaling Best Practices

1. Best Practices for Elastic Load Balancing

✓ **Use multiple Availability Zones (AZs)** for high availability.
✓ **Enable HTTPS for security** and terminate SSL/TLS at the ELB.
✓ **Use path-based routing in ALB** for microservices architectures.

2. Best Practices for Auto Scaling

✓ **Monitor CPU & memory usage** to adjust scaling policies.
✓ **Use predictive scaling** for seasonal workloads.
✓ **Enable health checks** to replace failing instances.

Example:

A **social media platform** experiences **high usage spikes**. By using **ALB + Auto Scaling,** the app can **handle millions of users dynamically** without downtime.

Exercise:

1. Configure an **Auto Scaling Group** with a **scaling policy** to adjust instance count dynamically.
2. Set up **health checks** to replace failed instances.

Chapter 5: Case Study – Netflix's Use of ELB & Auto Scaling

Problem:

Netflix required **high availability and scalability** to handle **millions of concurrent users** worldwide.

Solution:

- **Application Load Balancers (ALB)** distribute video streaming traffic.
- **Auto Scaling dynamically adds EC2 instances** when traffic spikes.

- **CloudWatch monitors server load** and adjusts scaling policies.

Outcome:

✓ **99.99% uptime** for video streaming services.
✓ **Automatic scaling saved infrastructure costs** by shutting down unused instances.
✓ **Seamless user experience** even during peak streaming hours.

Conclusion: Mastering ELB & Auto Scaling

By implementing **Elastic Load Balancer and Auto Scaling,** businesses can:

✅ **Ensure high availability and fault tolerance.**
✅ **Handle traffic spikes efficiently.**
✅ **Reduce cloud costs with optimized scaling.**

Final Exercise:

1. Deploy a **highly available web application** using **ALB and Auto Scaling.**
2. Set up an **Auto Scaling policy** to add an instance when CPU usage **exceeds 75%.**
3. Write a report on **how Netflix benefits from Auto Scaling.**

AWS Lambda & Serverless Computing

AWS Lambda is a **serverless computing service** that allows developers to run code **without provisioning or managing servers**. It is an integral part of **serverless computing,** enabling **event-driven applications,** cost savings, and scalability.

This study material provides an **in-depth understanding** of **AWS Lambda and Serverless Computing,** covering **concepts, architecture, benefits, use cases, best practices, examples, and exercises.**

Chapter 1: Introduction to AWS Lambda

What is AWS Lambda?

AWS Lambda is a **serverless computing service** that executes code **in response to events**. Instead of managing servers, users upload their **function code,** and AWS Lambda **handles execution, scaling, and availability** automatically.

Key Features of AWS Lambda

✓ **Serverless Execution** – No need to manage servers or infrastructure.
✓ **Event-Driven** – Functions are triggered by **AWS events, API calls, or scheduled tasks.**
✓ **Scalability** – Automatically scales from **zero to thousands of concurrent executions.**
✓ **Pay-as-You-Go Pricing** – Charges apply only for the **execution time (in milliseconds).**
✓ **Supports Multiple Programming Languages** – Python, Node.js, Java, Go, C#, and Ruby.

How AWS Lambda Works

1. **Trigger:** An event (S3 upload, API Gateway request, DynamoDB change, etc.) invokes the function.
2. **Execution:** AWS Lambda automatically provisions compute resources.
3. **Response:** The function executes the logic and returns the result.
4. **Auto-Scaling:** AWS manages scaling **based on request load**.

Example:

A web application **triggers AWS Lambda** when a user **uploads an image to S3,** and Lambda **automatically resizes** the image for display.

Exercise:

1. Create an **AWS Lambda function** that prints "Hello, AWS Lambda!".
2. Test the function using **AWS Console** or **CLI**.

Chapter 2: Serverless Computing – The Future of Cloud Applications

What is Serverless Computing?

Serverless computing is a **cloud-native execution model** where **cloud providers manage the infrastructure,** allowing developers to focus on **writing and deploying code**.

Benefits of Serverless Computing

✓ **No Server Management** – AWS handles infrastructure provisioning.

✓ **Scales Automatically** – Resources adjust dynamically based on demand.

✓ **Reduced Costs** – Pay only for execution time, reducing idle resource costs.

✓ **Improved Performance** – Fast execution with **low-latency request handling**.

AWS Services Used in Serverless Architectures

- **AWS Lambda** – Function execution.
- **Amazon API Gateway** – Exposes APIs to trigger Lambda functions.
- **Amazon S3** – Stores static files (images, videos, backups).
- **Amazon DynamoDB** – Serverless NoSQL database.
- **AWS Step Functions** – Orchestrates workflows for complex tasks.

Example:

A **serverless chatbot** uses **AWS Lambda, DynamoDB, and API Gateway** to respond to user messages **without managing any servers**.

Exercise:

1. Research how **Netflix and Airbnb** use serverless computing.
2. Identify three **real-world applications** for serverless computing.

Chapter 3: Deploying an AWS Lambda Function

Step 1: Create a Lambda Function in AWS Console

1. Open **AWS Management Console** → Navigate to **AWS Lambda**.
2. Click **"Create Function"**.
3. Choose **"Author from Scratch"**.
4. **Enter Function Name:** MyFirstLambdaFunction.
5. Select **Runtime:** Python 3.9 (or any preferred language).
6. Click **"Create Function"**.

Step 2: Write and Deploy Code

1. In the **Function Code** editor, replace the default code with:
2. def lambda_handler(event, context):
3. return "Hello from AWS Lambda!"
4. Click **"Deploy"** to save changes.

Step 3: Test the Lambda Function

1. Click **"Test" → Create New Test Event**.
2. Set **Event Name** as TestEvent1.
3. Click **"Invoke"**, and verify the response.

Chapter 4: Integrating AWS Lambda with Other AWS Services

1. AWS Lambda with API Gateway (Creating REST APIs)

- API Gateway triggers Lambda functions when an **HTTP request is received**.
- Used for **RESTful API development** and **microservices**.

2. AWS Lambda with Amazon S3 (Event-Driven Processing)

- Lambda executes automatically when an **object is uploaded to S3**.
- Used for **image processing, video encoding, and backup automation**.

3. AWS Lambda with DynamoDB (Serverless Databases)

- Lambda triggers on **DynamoDB table updates**.
- Used for **real-time analytics, fraud detection, and log monitoring**.

Example:

An **e-commerce platform** uses Lambda + API Gateway to process **customer orders in a serverless environment**.

Exercise:

1. Create an **AWS Lambda function** that runs **whenever an S3 file is uploaded**.
2. Configure an **API Gateway** to invoke a Lambda function using an HTTP request.

Chapter 5: Best Practices for AWS Lambda

✓ **Optimize Function Execution Time** – Reduce processing logic to minimize latency.

✓ **Use Environment Variables** – Store configuration settings securely.

✓ **Limit Memory Usage** – Adjust memory settings for cost optimization.

✓ **Monitor & Debug Using AWS X-Ray** – Trace function

EXECUTION FOR PERFORMANCE ISSUES.

✓ **SECURE FUNCTIONS WITH IAM ROLES** – RESTRICT ACCESS TO ONLY NECESSARY AWS RESOURCES.

EXAMPLE:

A **PAYMENT PROCESSING SYSTEM** USES LAMBDA **WITH IAM ROLES** TO ENSURE ONLY AUTHORIZED SERVICES **CAN ACCESS CUSTOMER TRANSACTIONS**.

EXERCISE:

1. ENABLE **AWS X-RAY** FOR DEBUGGING LAMBDA FUNCTION PERFORMANCE.
2. MODIFY IAM ROLES TO **RESTRICT LAMBDA ACCESS** TO SPECIFIC AWS SERVICES.

CHAPTER 6: AWS LAMBDA USE CASES & CASE STUDY

REAL-WORLD USE CASES OF AWS LAMBDA

✓ **REAL-TIME DATA PROCESSING** – IOT DEVICE LOGS, FINANCIAL TRANSACTIONS, STOCK MARKET DATA.

✓ **CHATBOTS & AI INTEGRATION** – POWERING **SERVERLESS CHATBOTS** AND **MACHINE LEARNING APPLICATIONS**.

✓ **AUTOMATED BACKUPS & CLEANUP TASKS** – RUNNING SCHEDULED **DATABASE BACKUPS**.

✓ **CI/CD PIPELINE AUTOMATION** – AUTOMATING DEPLOYMENTS WITH **CODEPIPELINE + LAMBDA**.

CASE STUDY: AWS LAMBDA AT COCA-COLA

PROBLEM: COCA-COLA WANTED A **COST-EFFICIENT SOLUTION** TO PROCESS VENDING MACHINE TRANSACTIONS.

SOLUTION: THEY REPLACED TRADITIONAL SERVERS WITH **AWS LAMBDA**

AND API GATEWAY.
OUTCOME: 80% COST SAVINGS, HIGH AVAILABILITY, AND **FASTER TRANSACTION PROCESSING.**

CONCLUSION: MASTERING AWS LAMBDA & SERVERLESS COMPUTING

BY USING **AWS LAMBDA AND SERVERLESS COMPUTING,** DEVELOPERS CAN:

✅ **RUN APPLICATIONS WITHOUT MANAGING SERVERS.**
✅ **REDUCE COSTS** BY PAYING ONLY FOR EXECUTION TIME.
✅ **SCALE AUTOMATICALLY** BASED ON DEMAND.

FINAL EXERCISE:

1. CREATE A **LAMBDA FUNCTION** THAT INTEGRATES WITH **DYNAMODB.**
2. RESEARCH **HOW AWS LAMBDA IMPROVES DEVOPS AUTOMATION.**

BY MASTERING **AWS LAMBDA,** YOU CAN BUILD **SCALABLE, COST-EFFICIENT, AND HIGHLY AVAILABLE APPLICATIONS** IN THE CLOUD. 🚀

AWS STORAGE SERVICES

AMAZON S3 ADVANCED FEATURES : VERSIONING & LIFECYCLE POLICIES

Amazon S3 (Simple Storage Service) is a **scalable, durable, and secure** cloud storage service. While basic S3 features provide efficient data storage, **advanced features like Versioning and Lifecycle Policies** enhance **data protection, cost optimization, and automated data management**.

This study material provides an in-depth look at **Amazon S3 Versioning and Lifecycle Policies**, covering **concepts, benefits, configurations, examples, exercises, and best practices**.

CHAPTER 1: AMAZON S3 VERSIONING – PROTECTING DATA INTEGRITY

What is Amazon S3 Versioning?

S3 **Versioning** is a feature that allows S3 buckets to **retain multiple versions of an object**. It protects against **accidental deletions, overwrites, and unintended changes** by storing **each modification as a separate version**.

Key Features of S3 Versioning:

✓ **Automatic Object Versioning** – Keeps multiple versions of the same object.

✓ **Rollback & Recovery** – Restore previous versions of an object if a file is deleted or corrupted.

✓ **Protection from Accidental Deletion** – Deleted objects remain accessible as older versions.

✓ **Works with MFA (Multi-Factor Authentication) Delete** – Prevents unauthorized deletions.

How S3 Versioning Works

- When **Versioning is enabled,** every new upload of an object creates a **new version** with a **unique version ID**.
- If a user **deletes an object,** S3 does not remove the data but instead **creates a delete marker,** allowing recovery.

Example Scenario:

A **company manages important contracts in S3**. If an employee **accidentally deletes or modifies** a file**, S3 Versioning allows recovery** of the previous version.

Step 1: Enable Versioning in an S3 Bucket

1. **Go to AWS S3 Console** → Select a bucket.
2. Click **Properties** → Locate **Bucket Versioning**.
3. Click **Edit** → **Enable Versioning** → Click **Save Changes**.

Step 2: Upload and Modify a File in a Versioned Bucket

1. Upload a **file (e.g., report.pdf)**.
2. Upload the **same file again** (modified version).
3. AWS assigns **unique version IDs** to each upload.

Step 3: Retrieve an Older Version

1. Open the **S3 bucket** → Locate the file.
2. Click **"Show Versions"** to view all object versions.
3. Select and download the required version.

Managing Versioning with AWS CLI

1. Enable Versioning on a Bucket

aws s3api put-bucket-versioning --bucket my-versioned-bucket --versioning-configuration Status=Enabled

2. List Object Versions

aws s3api list-object-versions --bucket my-versioned-bucket

3. Retrieve a Specific Version

aws s3 cp s3://my-versioned-bucket/report.pdf --version-id <version_id> .

Exercise: Test Versioning

1. Enable **Versioning** on an S3 bucket.
2. Upload a file, modify it, and upload again.
3. Retrieve and restore a **previous version** of the file.

CHAPTER 2: AMAZON S3 LIFECYCLE POLICIES – AUTOMATED DATA MANAGEMENT

What are S3 Lifecycle Policies?

S3 **Lifecycle Policies** allow users to **automatically transition or delete objects** based on predefined rules. This helps in **cost optimization** and **data retention management**.

Key Features of Lifecycle Policies:

✓ **Move data to cheaper storage tiers** (e.g., S3 Standard → S3 Glacier).
✓ **Automatically delete old or unused objects**.
✓ **Reduce storage costs without manual intervention**.
✓ **Works with Versioning** to delete **old versions** while keeping recent ones.

Example Scenario:

A **media company stores videos in S3 Standard** but **moves older videos to S3 Glacier** after **90 days** to save costs.

STEP 1: CREATE AN S3 LIFECYCLE RULE

1. **Go to AWS S3 Console** → Select a bucket.
2. Click **Management** → **Create lifecycle rule**.
3. Enter **Rule Name** (e.g., MoveToGlacier).
4. **Define Filter** (apply to all objects or specific prefixes).
5. **Set Transition Actions**:
    - Move to **S3 Standard-IA after 30 days**.
    - Move to **S3 Glacier after 90 days**.
6. **Set Expiration Actions**:
    - Delete objects **older than 1 year**.
7. Click **Create Rule**.

Managing Lifecycle Policies with AWS CLI

Create a **JSON policy file** (lifecycle-policy.json):

```
{
 "Rules": [
  {
   "ID": "MoveToGlacier",
   "Prefix": "",
   "Status": "Enabled",
   "Transitions": [
    {
     "Days": 30,
     "StorageClass": "STANDARD_IA"
    },
    {
     "Days": 90,
     "StorageClass": "GLACIER"
    }
   ],
   "Expiration": {
    "Days": 365
   }
  }
 ]
}
```

Apply the policy to an S3 bucket:

aws s3api put-bucket-lifecycle-configuration --bucket my-lifecycle-bucket --lifecycle-configuration file://lifecycle-policy.json

---

Exercise: Configure Lifecycle Policies

1. Set up a **Lifecycle Policy** to move objects to **S3 Glacier after 90 days**.
2. Set a **rule to delete old versions of files after 180 days**.

CHAPTER 3: USING VERSIONING & LIFECYCLE TOGETHER

Combining Versioning & Lifecycle Policies

- **Use Case:** Retain **only the latest 3 versions** of files while deleting older versions.
- **Solution:** Apply a **Lifecycle Policy for non-current versions**.

Lifecycle Policy for Versioned Buckets:

1. Go to **S3 Management Console → Lifecycle Rules**.
2. Create a new **Lifecycle Rule** for **previous versions**.
3. Configure:
    - **Move non-current versions to S3 Glacier after 30 days**.
    - **Delete non-current versions after 180 days**.
4. Click **Save**.

Example:

A **financial company keeps the latest 3 document versions** while **deleting older versions automatically**.

Chapter 4: Best Practices for S3 Versioning & Lifecycle

1. Best Practices for Versioning

✓ **Enable versioning only for critical data** (as it increases storage costs).

✓ **Use MFA Delete** to prevent accidental deletions.

✓ **Regularly review object versions to avoid unnecessary storage usage**.

2. Best Practices for Lifecycle Policies

✓ **Use transition policies** to move objects to cost-effective storage tiers.

✓ **Define clear retention periods** to avoid unnecessary data storage.

✓ **Regularly audit and update lifecycle rules** based on data access needs.

Example:

A **healthcare company stores patient records in S3** and applies **Versioning with Lifecycle Rules** to comply with **data retention policies**.

Conclusion: Mastering S3 Advanced Features

By implementing **Amazon S3 Versioning and Lifecycle Policies,** businesses can:

✅ **Protect data from accidental deletion**.

✅ **Reduce storage costs by transitioning data automatically**.

✅ **Ensure compliance with long-term retention policies**.

FINAL EXERCISE:

1. **Enable versioning** on an S3 bucket and upload multiple versions of a file.
2. **Create a lifecycle rule** to move old versions to Glacier and delete them after 180 days.
3. Research **how financial institutions use S3 Lifecycle Policies** for compliance.

AMAZON EBS (ELASTIC BLOCK STORE) – STUDY MATERIAL

Amazon EBS (**Elastic Block Store**) is a **high-performance, durable, and scalable block storage service** for Amazon EC2 instances. It provides **persistent storage**, meaning data remains intact even after an EC2 instance is stopped or terminated.

This study material covers **EBS concepts, types, use cases, features, configuration steps, best practices, and real-world applications**.

CHAPTER 1: INTRODUCTION TO AMAZON EBS

What is Amazon EBS?

Amazon **EBS (Elastic Block Store)** is a **block storage service** designed for **high availability, durability, and performance**. Unlike **Amazon S3**, which is object storage, EBS provides **low-latency block storage** similar to traditional hard drives.

Key Features of EBS:

✓ **Persistent Storage** – Data remains intact even after EC2 instance stops or restarts.
✓ **Scalability** – Volumes can be resized **without downtime**.
✓ **High Performance** – Supports **SSD & HDD** storage for different workloads.
✓ **Snapshot & Backup Support** – Enables **automated backups and disaster recovery**.
✓ **Encryption & Security** – Supports **AWS-managed encryption keys (KMS)**.

How Amazon EBS Works?

1. **Create an EBS Volume** and attach it to an **EC2 instance**.

2. The EC2 instance **uses the volume** like a traditional hard drive.
3. **Store & retrieve data** from the attached volume.
4. **Take snapshots** to back up data and restore it when needed.

Example Use Case:

A **database server** uses **Amazon EBS SSD volumes** to store and process **transactional data** with **high-speed performance**.

Exercise: Create an EBS Volume & Attach to EC2

1. **Create an EC2 instance** (Amazon Linux or Ubuntu).
2. **Create an EBS volume** (10GB, gp3).
3. **Attach the volume** to the EC2 instance.
4. **Format & mount the EBS volume** inside the instance.

CHAPTER 2: TYPES OF AMAZON EBS VOLUMES

Amazon EBS provides **different volume types** optimized for **various workloads**.

SSD-Based Volumes (Best for High Performance & Databases)

Volume Type	Best For	IOPS	Throughput
gp3 (General Purpose SSD)	Balanced workloads	3,000–16,000	Up to 1,000 MB/s
gp2 (General Purpose SSD)	General workloads	3 IOPS/GB (max 16,000)	Up to 250 MB/s

Volume Type	Best For	IOPS	Throughput
io2 (Provisioned IOPS SSD)	High-performance databases	Up to 64,000	1,000 MB/s
io1 (Legacy PIOPS SSD)	Older high-speed apps	Up to 64,000	1,000 MB/s

HDD-Based Volumes (Best for Large Sequential Workloads & Logs)

Volume Type	Best For	IOPS	Throughput
st1 (Throughput Optimized HDD)	Big data, logs, streaming	500 IOPS	500 MB/s
sc1 (Cold HDD)	Archival data	250 IOPS	250 MB/s

Choosing the Right EBS Volume

- **Use gp3** for **cost-effective general workloads** (e.g., web apps, dev/test environments).
- **Use io2** for **high-performance databases and low-latency applications**.
- **Use st1/sc1** for **big data workloads** (e.g., Hadoop clusters, logs).

Example Use Case:

An **e-commerce website** uses **gp3 volumes** for web servers and **io2 volumes** for its **high-speed relational database**.

Exercise: Change EBS Volume Type

1. Create a **gp2 volume** and attach it to an EC2 instance.

2. Convert it to **gp3** using the **Modify Volume** option in AWS Console.

CHAPTER 3: MANAGING EBS VOLUMES (ATTACH, RESIZE, SNAPSHOT, ENCRYPTION)

1. Attaching and Mounting an EBS Volume

Once an **EBS volume is created**, it must be **attached and mounted** to an EC2 instance.

Steps to Attach & Mount an EBS Volume (Linux EC2)

1. **Create an EBS volume** in the **same availability zone** as the EC2 instance.
2. **Attach the volume** to the instance (/dev/xvdf).
3. Connect to EC2 via SSH and run:
4. sudo mkfs -t ext4 /dev/xvdf
5. sudo mkdir /data
6. sudo mount /dev/xvdf /data
7. Verify the volume is mounted:
8. df -h

---

2. Resizing an EBS Volume Without Downtime

Amazon EBS allows **dynamic resizing** without stopping the instance.

Steps to Resize EBS Volume

1. Open **EC2 Console** → Select **Volumes**.

2. Click **Modify Volume**, enter new size (e.g., 20GB), and save changes.
3. Connect to EC2 and expand the file system:
4. sudo resize2fs /dev/xvdf

---

3. Creating and Restoring EBS Snapshots (Backup & Recovery)

What is an EBS Snapshot?

An **EBS snapshot** is an **incremental backup** of an EBS volume. AWS only stores **changed blocks**, making snapshots **efficient and cost-effective**.

Steps to Create an EBS Snapshot

1. Open **EC2 Console** → Select **Volumes**.
2. Click **Actions** → **Create Snapshot**.
3. Enter a **description** and save the snapshot.

Restoring from a Snapshot

1. Open **EC2 Console** → Select **Snapshots**.
2. Click **Create Volume from Snapshot**.
3. Attach the new volume to an EC2 instance.

---

4. Enabling EBS Encryption for Security

EBS volumes **can be encrypted** for **data security and compliance** using **AWS Key Management Service (KMS)**.

How to Encrypt an EBS Volume

1. **Create an encrypted volume** by selecting **Enable Encryption**.

2. **Use AWS KMS keys** for encryption management.
3. **Enable default encryption** to apply encryption automatically to new volumes.

Exercise: Manage EBS Volumes

1. Attach and format an **EBS volume** in an EC2 instance.
2. **Resize the volume** without downtime.
3. Take an **EBS snapshot** and restore it to a new volume.

Chapter 4: Best Practices for Amazon EBS

✓ **Use gp3 over gp2** to reduce costs while maintaining performance.
✓ **Regularly take EBS snapshots** for backup and disaster recovery.
✓ **Monitor performance using CloudWatch** to detect slow or overloaded volumes.
✓ **Use RAID configurations** (RAID 0 for performance, RAID 1 for redundancy).
✓ **Detach unused EBS volumes** to **avoid unnecessary costs**.

Example:

A **financial application** stores transaction logs on **io2 volumes** and uses **automated snapshots** to **back up data every 6 hours**.

Conclusion: Mastering Amazon EBS

By using **Amazon EBS,** businesses can:
✅ **Ensure persistent, high-performance storage for EC2 instances**.
✅ **Dynamically scale storage without downtime**.

✅ **Secure and back up critical data with snapshots and encryption**.

FINAL EXERCISE:

1. **Create an EBS volume,** attach it to an **EC2 instance,** and mount it.
2. **Take an EBS snapshot** and restore it as a **new volume**.
3. Research **how Netflix and Amazon use EBS for high-performance storage**.

AWS EFS (Elastic File System) – Study Material

Introduction to AWS EFS

What is AWS EFS?

Amazon **EFS (Elastic File System)** is a **fully managed, scalable, and elastic file storage service** for use with AWS cloud services and on-premises resources. It is designed to provide **high availability, scalability, and shared access** to multiple EC2 instances.

Unlike **Amazon EBS (block storage)**, which is tied to a single EC2 instance, **EFS allows multiple instances to access the same file system concurrently**.

Chapter 1: Key Features of AWS EFS

✓ **Fully Managed & Serverless** – No need to provision or manage infrastructure.
✓ **Elastic Scaling** – Automatically grows and shrinks as files are added or removed.
✓ **Multiple EC2 Instance Access** – Supports concurrent access across multiple AZs.
✓ **High Performance & Low Latency** – Optimized for **high throughput applications**.
✓ **Multi-AZ Availability** – Ensures **fault tolerance and durability**.
✓ **NFS Protocol Support** – Compatible with **Linux-based workloads**.
✓ **Encryption & IAM Access Control** – Ensures **data security and compliance**.
✓ **Backup and Lifecycle Management** – Supports **automatic backups and cost optimization**.

AWS EFS vs. Amazon EBS vs. Amazon S3

Feature	Amazon EFS	Amazon EBS	Amazon S3
Storage Type	File Storage	Block Storage	Object Storage
Access by Multiple Instances	Yes	No (attached to one instance)	Yes
Scalability	Automatic	Requires manual resizing	Automatic
Use Case	Shared file storage	High-performance disk storage	Long-term data archiving
Performance	High throughput, low latency	High-speed block storage	Moderate speed

Example Use Case:

A **web hosting company** uses **AWS EFS to store website assets** (HTML, CSS, JavaScript files) that need to be **accessed by multiple EC2 instances**.

Exercise: Compare EFS, EBS, and S3

1. Research the **best use cases** for EFS, EBS, and S3.
2. Identify **two scenarios where EFS is better than EBS**.

Chapter 2: How AWS EFS Works

EFS Architecture Overview

1. **EFS File System** – Stores files in a managed service.
2. **Mount Targets** – EC2 instances use **NFS protocol** to access EFS.
3. **Storage Classes** – Files can be stored in **Standard** or **Infrequent Access (IA)** to optimize costs.
4. **Automatic Scaling** – Expands or shrinks as needed without manual intervention.

EFS Performance Modes

Performance Mode	Use Case
General Purpose Mode	Best for **web servers, DevOps, and general workloads**
Max I/O Mode	Best for **big data, analytics, and high-throughput applications**

EFS Storage Classes

Storage Class	Best For	Cost Savings
EFS Standard	Frequently accessed files	-
EFS Infrequent Access (IA)	Long-term storage with occasional access	Up to **92% cheaper** than Standard

---

Example Use Case:

A **video streaming platform** stores frequently watched videos in **EFS Standard**, while **archived videos** move to **EFS Infrequent Access (IA)** to save costs.

Chapter 3: Setting Up AWS EFS

Step 1: Create an EFS File System

1. **Go to AWS Console** → Open **EFS Dashboard**.
2. Click **"Create File System"**.
3. Select **VPC and Availability Zones** for deployment.
4. Enable **automatic backups** (recommended for production).
5. Click **"Create"**.

Step 2: Attach EFS to EC2 Instances

1. Select **your EC2 instances**.
2. Click **"Attach File System"** and copy the **mount command**.
3. SSH into your EC2 instance and run:
4. sudo mkdir /mnt/efs
5. sudo mount -t nfs4 file-system-id.efs.aws-region.amazonaws.com:/ /mnt/efs

Step 3: Verify Mount & Permissions

1. Check mounted file system:
2. df -h
3. Create a test file in EFS:
4. touch /mnt/efs/testfile.txt

Exercise: Deploy AWS EFS

1. Create an **AWS EFS file system** and mount it to an **EC2 instance**.

2. Create a **test file** and verify access from multiple instances.

Chapter 4: Managing AWS EFS (Performance & Security)

1. Optimizing AWS EFS Performance

✓ Choose **General Purpose Mode** for web applications.
✓ Use **Max I/O Mode** for high-throughput workloads.
✓ Enable **EFS Lifecycle Policies** to move old files to **Infrequent Access (IA)** storage.
✓ Use **EFS Burst Credits** for improved read/write performance.

2. Security Best Practices for AWS EFS

✓ **Use IAM Policies & Security Groups** to restrict access.
✓ **Enable Encryption at Rest** using AWS KMS.
✓ **Apply Network ACLs** to allow access only from trusted instances.

Example Use Case:

A **finance company** uses **EFS encryption & IAM policies** to ensure **secure storage** of sensitive customer documents.

Chapter 5: AWS EFS Backup & Lifecycle Policies

1. Enabling AWS EFS Backups

AWS Backup provides **automatic backup schedules** for EFS.

Enable Backups in AWS Console

1. Go to **AWS Backup Dashboard**.
2. Click **"Create Backup Plan"**.

3. Select **AWS EFS** as the backup source.
4. Choose backup frequency (daily, weekly, monthly).

2. Using Lifecycle Policies for Cost Savings

Lifecycle policies **move old or infrequently accessed files** to **EFS Infrequent Access (IA)**.

Enable Lifecycle Policies

1. Go to **EFS Dashboard** → Select **File System**.
2. Click **"Lifecycle Management"**.
3. Choose when to **move files to Infrequent Access (e.g., after 30 days of inactivity)**.

Exercise: Backup & Lifecycle Policies

1. Configure **AWS Backup** for EFS and take a snapshot.
2. Enable a **Lifecycle Policy** to move files to **IA storage**.

Chapter 6: Real-World Use Cases of AWS EFS

1. Web Applications

- Multiple web servers share **CSS, JS, images** in an **EFS file system**.
- Ensures **consistent access** across all instances.

2. Big Data & Machine Learning

- Large datasets are stored in **EFS for real-time processing**.
- **Auto-scaling** adjusts file storage dynamically.

3. DevOps & CI/CD Pipelines

- Dev teams store **build artifacts, logs, and scripts** in EFS.
- **Multi-user access** allows teams to collaborate.

Case Study: AWS EFS at Lyft

Problem: Lyft needed a **high-performance file system** for AI training models.
Solution: They deployed **AWS EFS with Max I/O Mode**.
Outcome: Improved **training speeds by 50%** while **reducing storage costs**.

Conclusion: Mastering AWS EFS

By implementing **AWS EFS,** businesses can:
✅ **Provide scalable, shared storage for multiple EC2 instances**.
✅ **Optimize storage costs with Infrequent Access (IA) class**.
✅ **Ensure high availability and performance** for critical applications.

Final Exercise:

1. Create an **EFS file system** and mount it to multiple EC2 instances.
2. Enable **Lifecycle Management** to move files to **IA storage**.
3. Research how **Netflix uses EFS for video processing**.

By mastering **AWS EFS**, you can build **scalable, resilient, and cost-efficient file storage solutions** for modern cloud applications! 🚀

CLOUD MONITORING & MANAGEMENT

AMAZON CLOUDWATCH FOR MONITORING – STUDY MATERIAL

Introduction to Amazon CloudWatch

What is Amazon CloudWatch?

Amazon **CloudWatch** is a **monitoring and observability service** that provides insights into AWS resources, applications, and services. It helps businesses **track performance, detect anomalies, set up alerts, and automate responses** to ensure operational efficiency.

Key Features of CloudWatch:

✓ **Real-Time Monitoring** – Collects metrics, logs, and events from AWS resources.
✓ **Custom Dashboards** – Visualize key performance indicators (KPIs).
✓ **Alarms & Notifications** – Triggers alerts based on thresholds.
✓ **Automated Actions** – Integrates with AWS Lambda, Auto Scaling, and SNS for automated responses.
✓ **Application Performance Monitoring (APM)** – Tracks API requests, latency, and errors.

CHAPTER 1: UNDERSTANDING CLOUDWATCH COMPONENTS

1. CloudWatch Metrics

CloudWatch **automatically collects** key performance metrics from AWS services.

Common CloudWatch Metrics

AWS Service	Monitored Metrics
EC2 Instances	CPU Utilization, Disk Read/Write, Network Traffic
RDS Databases	Database Connections, Read/Write Latency
S3 Buckets	Number of Requests, Data Transfer
Lambda Functions	Invocation Count, Execution Duration, Error Rate
Auto Scaling	Number of Instances, Scaling Events

📌 **Example:**
A DevOps team uses **CloudWatch Metrics** to track **EC2 CPU utilization** and trigger **Auto Scaling** when CPU usage exceeds **70%**.

---

2. CloudWatch Alarms

CloudWatch **Alarms** monitor metrics and **trigger actions** when thresholds are met.

How CloudWatch Alarms Work:

1. Select a **metric** (e.g., CPUUtilization for EC2).
2. Set a **threshold** (e.g., > 75% CPU usage).
3. Choose an **action** (e.g., send an alert via SNS or trigger Auto Scaling).
4. CloudWatch continuously evaluates data and **sends notifications** when conditions are met.

📌 **Example:**
A company **creates an alarm** for an EC2 instance that **sends an email notification** when CPU usage exceeds **80% for 5 minutes**.

3. CloudWatch Logs

CloudWatch **Logs** capture, store, and analyze logs from AWS resources, applications, and services.

Key Features of CloudWatch Logs:

✓ **Log Retention & Analysis** – Store logs for troubleshooting and compliance.
✓ **Log Filtering** – Search for specific events (e.g., failed login attempts).
✓ **Log Insights** – Analyze logs using SQL-like queries.
✓ **Integration with AWS Lambda & S3** – Automate log processing and archival.

📌 **Example:**
A **web application logs user login attempts** in CloudWatch Logs and **triggers an alert** if multiple failed login attempts are detected.

---

4. CloudWatch Events (Amazon EventBridge)

CloudWatch **Events** (now part of **EventBridge**) allow real-time monitoring of AWS resources and event-driven automation.

Use Cases of CloudWatch Events:

✓ **Trigger Lambda functions** based on system events (e.g., new file uploaded to S3).
✓ **Start/Stop EC2 instances** on a schedule.
✓ **Automate security responses** (e.g., lock user accounts after repeated failed login attempts).

📌 **Example:**
A startup uses **CloudWatch Events** to **automatically stop non-production EC2 instances at midnight** to reduce costs.

---

5. CloudWatch Dashboards

CloudWatch **Dashboards** provide a **centralized view** of AWS resources and application performance.

✓ **Customizable graphs & charts** – Display key metrics.
✓ **Multi-service integration** – Monitor **EC2, RDS, Lambda, S3** in one view.
✓ **Interactive UI** – Drill down into specific resources.

📌 **Example:**
A DevOps engineer **creates a dashboard** showing **CPU, memory, and disk usage** for all EC2 instances to track performance at a glance.

CHAPTER 2: SETTING UP CLOUDWATCH FOR EC2 MONITORING

Step 1: Enable CloudWatch Metrics for an EC2 Instance

1. Open **AWS Console** → Navigate to **EC2 Dashboard**.
2. Select an **EC2 instance** → Click **Monitoring**.
3. View **pre-configured CloudWatch Metrics** (e.g., CPU Utilization, Disk IO).

Step 2: Create a CloudWatch Alarm for EC2 CPU Usage

1. Open **CloudWatch Console** → Click **Alarms** → **Create Alarm**.
2. Select **Metric:** EC2 → **CPU Utilization**.

3. Set **Threshold:** Greater than **75% for 5 minutes**.
4. Configure **Actions:**
    - Notify via **SNS (email/SMS)**.
    - Trigger **Auto Scaling** (optional).
5. Click **Create Alarm**.

📌 **Expected Outcome:** If **CPU usage exceeds 75%,** AWS **sends an alert** via email or SMS.

Step 3: Enable CloudWatch Logs for EC2

1. Install the CloudWatch Agent:
2. sudo yum install amazon-cloudwatch-agent -y
3. sudo systemctl enable amazon-cloudwatch-agent
4. sudo systemctl start amazon-cloudwatch-agent
5. Configure the **CloudWatch Agent** to send logs to CloudWatch Logs.

📌 **Expected Outcome:** Application logs are **streamed to CloudWatch for analysis**.

Step 4: Create a CloudWatch Dashboard

1. Open **CloudWatch Console** → Click **Dashboards** → **Create Dashboard**.
2. Add **widgets** for EC2 CPU, RDS Queries, and Lambda Errors.
3. Customize time range & layout.
4. Click **Save Dashboard**.

📌 **Expected Outcome:** A real-time **visual dashboard** of AWS resources.

CHAPTER 3: BEST PRACTICES FOR AMAZON CLOUDWATCH

✓ **Enable Detailed Monitoring** for EC2 instances for **1-minute metrics** (default: 5 minutes).

✓ **Use CloudWatch Logs Insights** to filter logs and detect issues.

✓ **Set Up Alarms for Key Metrics** like CPU, memory, and request latency.

✓ **Optimize Costs** by setting **log retention policies** to delete old logs.

✓ **Use Anomaly Detection** to automatically detect unusual patterns in metrics.

📌 **Example:**
A security team **uses anomaly detection** to **detect abnormal traffic spikes** and **trigger an AWS Lambda function** to block malicious requests.

CHAPTER 4: REAL-WORLD USE CASES OF CLOUDWATCH

1. Infrastructure Monitoring

- Track **EC2 instance health, RDS connections,** and **EBS disk activity**.
- Set alarms for **CPU usage spikes or low available memory**.

2. Security & Compliance

- Monitor **failed login attempts** in IAM logs.
- Trigger **alerts for unauthorized API calls**.

3. Cost Optimization

- Auto-stop **unused EC2 instances** based on CloudWatch Events.
- Set alarms to **notify when AWS costs exceed budget**.

4. Application Performance Monitoring

- Detect **slow API responses** using **CloudWatch Metrics + X-Ray**.
- Monitor **database query performance** and **optimize SQL queries**.

📌 **Case Study: Netflix Uses CloudWatch**
Problem: Netflix needed a **real-time monitoring solution** for millions of streaming users.
Solution: They implemented **CloudWatch Dashboards, Logs, and Alarms**.
Outcome: Reduced **service downtime** and **improved content delivery performance**.

Conclusion: Mastering Amazon CloudWatch

By using **Amazon CloudWatch,** businesses can:
✅ **Monitor AWS infrastructure in real-time.**
✅ **Detect performance issues before they impact users.**
✅ **Automate responses to system anomalies.**
✅ **Optimize cloud costs & ensure security compliance.**

Final Exercise:

1. **Create a CloudWatch Alarm** for an **EC2 instance with a CPU threshold** of 70%.
2. **Analyze logs using CloudWatch Insights** to detect errors in an application.

3. **Set up a CloudWatch Dashboard** to monitor an entire AWS environment.

AWS CloudTrail for Auditing – Study Material

Introduction to AWS CloudTrail

What is AWS CloudTrail?

AWS **CloudTrail** is a **logging and monitoring service** that **records all API calls and actions** performed in an AWS account. It helps organizations with **security auditing, compliance tracking, and troubleshooting** by maintaining a **detailed history of AWS API activity**.

Key Features of AWS CloudTrail

✓ **Continuous Monitoring** – Tracks all AWS API activity across the account.
✓ **Event History** – Stores recent activity logs for **90 days** by default.
✓ **CloudTrail Insights** – Detects unusual API activity and potential security threats.
✓ **Integration with CloudWatch** – Enables real-time alerts for suspicious activities.
✓ **Compliance & Security Auditing** – Helps meet regulatory requirements (HIPAA, GDPR, SOC 2).

📌 **Example Use Case:**
A security team uses **CloudTrail logs** to **investigate unauthorized access attempts** on AWS resources.

Chapter 1: Understanding CloudTrail Components

1. CloudTrail Events

CloudTrail captures two types of events:

Event Type	Description	Example
Management Events	Track changes to AWS resources	IAM policy updates, S3 bucket creation
Data Events	Track read/write operations on AWS services	S3 file access, Lambda function execution
Insights Events	Detect unusual activity patterns	Multiple failed login attempts in a short period

📌 **Example:**
An administrator **deletes an EC2 instance**, and CloudTrail logs the action with details like **who performed it and when**.

---

2. CloudTrail Event Structure

CloudTrail logs events in **JSON format**, including:

```
{
 "eventTime": "2025-02-20T12:34:56Z",
 "eventSource": "ec2.amazonaws.com",
 "eventName": "TerminateInstances",
 "userIdentity": {
  "type": "IAMUser",
  "userName": "admin-user"
 },
 "sourceIPAddress": "203.0.113.42",
```

```
 "requestParameters": {
  "instanceId": "i-1234567890abcdef"
 }
}
```


📌 **Example Use Case:**
A **security analyst** checks CloudTrail logs to **identify which IAM user terminated an EC2 instance**.

Chapter 2: Setting Up AWS CloudTrail

Step 1: Enable CloudTrail

1. Open **AWS Console → CloudTrail Dashboard**.
2. Click **"Create Trail"**.
3. Enter a **trail name** (e.g., MySecurityTrail).
4. Choose **Apply trail to all regions** (recommended for global tracking).
5. Select **"Create a new S3 bucket"** to store logs.
6. Click **"Create"** to start logging AWS events.

📌 **Expected Outcome:**
All AWS API activity is now **logged and stored** in an **S3 bucket** for auditing.

Step 2: View CloudTrail Event History

1. Open **AWS Console → CloudTrail → Event History**.
2. Filter by:

- **Event Name** (e.g., DeleteBucket)
- **Username** (e.g., admin-user)
- **Source IP** (e.g., 192.168.1.1)

3. Click on an event to view **detailed JSON logs**.

📌 **Example Use Case:**
An admin filters CloudTrail logs to find **who accessed an S3 bucket** on a specific date.

Step 3: Enable CloudTrail Insights for Anomaly Detection

1. Open **CloudTrail Dashboard → Trails**.
2. Select an existing trail.
3. Click **"Edit Trail"** → Enable **CloudTrail Insights**.
4. Insights will now **detect and report anomalies** (e.g., multiple failed API calls).

📌 **Example:**
A security team **detects unauthorized login attempts** using CloudTrail Insights and takes action.

CHAPTER 3: AWS CLOUDTRAIL & SECURITY MONITORING

1. Integrating CloudTrail with CloudWatch for Alerts

To receive **real-time security alerts**, integrate **CloudTrail with CloudWatch**.

Steps to Set Up CloudTrail Alerts in CloudWatch:

1. Open **CloudWatch Console → Create Metric Filter**.

2. Select the **CloudTrail log group**.
3. Define a **filter pattern** (e.g., eventName = "DeleteBucket").
4. Create a **CloudWatch Alarm** to send **email/SNS alerts** when this event occurs.

📌 **Example:**
A DevOps team **receives an alert** when an **IAM policy is modified** unexpectedly.

---

2. Protecting CloudTrail Logs with IAM Policies

Ensure CloudTrail logs are **secure and cannot be tampered with**.

Best Practices:

✓ Enable **S3 Bucket Encryption** for logs.
✓ Use **IAM Policies** to restrict CloudTrail log access.
✓ Enable **MFA Delete** to prevent unauthorized deletions.

📌 **Example:**
A security administrator **configures IAM policies** so only the **CISO (Chief Information Security Officer)** can access CloudTrail logs.

CHAPTER 4: AWS CLOUDTRAIL USE CASES & COMPLIANCE

1. Security Auditing & Forensics

- Investigate **who modified IAM policies** or **deleted critical resources**.
- Detect **unauthorized API calls** from unknown IP addresses.

2. Compliance & Regulatory Reporting

- Helps meet compliance standards:
  ✓ **GDPR** (General Data Protection Regulation)
  ✓ **HIPAA** (Healthcare Security Standards)
  ✓ **SOC 2** (Security & Trust Controls)

📌 **Example:**
A financial institution uses **CloudTrail logs** to track **access to sensitive customer data** for **GDPR compliance**.

---

3. Incident Response & Threat Detection

- CloudTrail logs provide **detailed event history** for **post-incident investigations**.
- CloudTrail **Insights detect unusual login activity,** helping prevent security breaches.

📌 **Example:**
A company uses CloudTrail to **identify a compromised IAM user** and **immediately revoke access**.

CHAPTER 5: BEST PRACTICES FOR AWS CLOUDTRAIL

✓ **Enable Multi-Region Trails** – Ensures **complete tracking across AWS regions**.
✓ **Encrypt CloudTrail Logs** – Use **AWS KMS** for security.
✓ **Integrate with AWS Security Hub** – Gain a **centralized security overview**.
✓ **Monitor Logs with Amazon GuardDuty** – Detect **malicious activities**.
✓ **Set Up Retention Policies** – Archive old logs in **Amazon S3 Glacier**.

📌 **Example:**
A government agency **stores CloudTrail logs in Amazon S3 Glacier** for **5 years to meet compliance requirements**.

Chapter 6: Real-World Case Study: CloudTrail at Netflix

Problem:

Netflix needed **detailed security auditing** to track changes in **AWS IAM policies** and **S3 bucket permissions**.

Solution:

- **Enabled AWS CloudTrail** for real-time monitoring.
- **Integrated CloudTrail with CloudWatch** to trigger alerts.
- **Used AWS Lambda** to automatically **revert unauthorized IAM changes**.

Outcome:

✓ **Increased security transparency.**
✓ **Reduced response time to security threats.**
✓ **Improved compliance tracking** for audits.

Conclusion: Mastering AWS CloudTrail

By implementing **AWS CloudTrail,** organizations can:
✅ **Monitor AWS API calls & detect security threats.**
✅ **Improve compliance with regulatory requirements.**
✅ **Integrate with CloudWatch for real-time alerts.**
✅ **Use CloudTrail Insights to detect anomalies.**

FINAL EXERCISE:

1. **Enable AWS CloudTrail** and view event logs.
2. **Create an alert** to detect when an **IAM user logs in from an unknown IP**.
3. **Research how AWS CloudTrail improves cloud security monitoring**.

AWS Trusted Advisor – Study Material

Introduction to AWS Trusted Advisor

What is AWS Trusted Advisor?

AWS **Trusted Advisor** is an **AWS management tool** that provides **real-time guidance and recommendations** to help businesses **optimize AWS infrastructure, improve security, enhance performance, and reduce costs**.

Trusted Advisor evaluates AWS accounts and services against **best practices** in five key areas:

1. **Cost Optimization** – Identifies opportunities to reduce costs.
2. **Performance** – Improves application and system efficiency.
3. **Security** – Detects security vulnerabilities and misconfigurations.
4. **Fault Tolerance** – Enhances system reliability and disaster recovery.
5. **Service Limits** – Ensures AWS resource usage stays within safe limits.

Chapter 1: AWS Trusted Advisor Features & Benefits

1. Key Features of AWS Trusted Advisor

✓ **Personalized Recommendations** – Custom suggestions for AWS account improvements.
✓ **Cost Optimization Insights** – Identifies **unused or underutilized** resources.

✓ **Security Checks** – Finds **vulnerabilities, open ports, and IAM issues**.

✓ **Performance Enhancements** – Monitors **EC2 instances, RDS databases, and storage**.

✓ **Service Limits Monitoring** – Prevents hitting AWS **quota limits** before they impact workloads.

✓ **Automated Alerts & Reports** – Sends **notifications when issues arise**.

---

2. Benefits of Using AWS Trusted Advisor

Benefit	Description
Cost Savings	Identifies **unused instances, underutilized resources, and Reserved Instance opportunities**.
Enhanced Security	Detects **publicly accessible S3 buckets, open security groups, and IAM policy issues**.
Improved Performance	Recommends **load balancing, database optimizations, and caching strategies**.
Higher Availability	Suggests **disaster recovery strategies, backup recommendations, and multi-AZ deployments**.
Avoid Resource Limits	Warns about **approaching AWS service quotas (e.g., EC2 instance limits)**.

📌 **Example:**

A startup uses **AWS Trusted Advisor** to identify an **unused EC2 instance** running for months, saving **$500/month** by terminating it.

Exercise: Explore AWS Trusted Advisor Dashboard

1. Open the **AWS Console** → Navigate to **AWS Trusted Advisor**.
2. Review the **Security and Cost Optimization** checks.
3. Identify **one cost-saving recommendation** and **implement it**.

CHAPTER 2: AWS TRUSTED ADVISOR CHECK CATEGORIES

AWS Trusted Advisor provides **checks across five categories**:

1. Cost Optimization Checks

Helps reduce **unnecessary AWS expenses**.

✓ **Idle EC2 Instances** – Detects **low-utilization EC2 instances**.
✓ **Unassociated Elastic IPs** – Identifies **unused IP addresses** incurring costs.
✓ **Underutilized Amazon RDS Instances** – Finds **low-use databases**.
✓ **S3 Lifecycle Rules** – Suggests moving **old data to cheaper storage tiers**.
✓ **Reserved Instance (RI) Recommendations** – Recommends **converting On-Demand instances** to RIs for cost savings.

📌 **Example:**
A company reduces costs by **switching EC2 On-Demand instances to Reserved Instances,** saving **30% annually**.

---

2. Performance Checks

Ensures **AWS services run efficiently**.

✓ **High Utilization EC2 Instances** – Identifies **overloaded instances**.
✓ **Overutilized Amazon RDS Databases** – Suggests scaling up

databases.

✓ **CloudFront Content Delivery Optimization** – Recommends enabling **Amazon CloudFront caching**.

📌 **Example:**

An **e-commerce website** improves **page load speed** by enabling **CloudFront caching**.

---

3. Security Checks

Identifies **potential security risks** and **misconfigurations**.

✓ **Open Security Groups** – Detects unrestricted access (0.0.0.0/0).

✓ **IAM Access Keys Rotation** – Warns about **stale IAM keys**.

✓ **Public S3 Buckets** – Detects **exposed data in Amazon S3**.

✓ **Multi-Factor Authentication (MFA)** – Ensures **MFA is enabled for IAM users**.

📌 **Example:**

A **financial company** enables **IAM MFA** after Trusted Advisor flags **security vulnerabilities**.

---

4. Fault Tolerance Checks

Improves **AWS service reliability** and **disaster recovery**.

✓ **Multi-AZ Deployments** – Recommends using **multiple Availability Zones**.

✓ **EBS Snapshot Best Practices** – Ensures **regular backups** of EBS volumes.

✓ **Load Balancer Configuration** – Suggests using **Elastic Load Balancing (ELB)** for high availability.

📌 **Example:**
A **healthcare application** configures **multi-AZ RDS databases** to prevent **downtime during failures**.

---

5. Service Limits Checks

Warns when **AWS service usage** nears **quota limits**.

✓ **EC2 Instance Limits** – Alerts when **EC2 instances exceed the region limit**.
✓ **VPC & Elastic IP Limits** – Detects **approaching quota restrictions**.
✓ **IAM Policies & Role Limits** – Ensures IAM roles stay **within policy limits**.

📌 **Example:**
A company prevents **EC2 instance failures** by **requesting a limit increase** before hitting **AWS quotas**.

Exercise: Review AWS Trusted Advisor Checks

1. Open **AWS Trusted Advisor** and check **security vulnerabilities**.
2. Identify **one cost optimization check** and implement the recommendation.

CHAPTER 3: USING AWS TRUSTED ADVISOR EFFECTIVELY

1. Accessing AWS Trusted Advisor

- Open the **AWS Console** → Navigate to **Trusted Advisor**.
- Review the **dashboard recommendations**.

2. Setting Up Trusted Advisor Notifications

Enable **weekly email reports** to track AWS account health.
✓ Go to **Trusted Advisor Console** → Click **"Notification Settings"**.
✓ Enable **weekly email alerts** for your AWS team.

3. Automating Trusted Advisor Recommendations

- Use **AWS Lambda** to **automatically stop unused EC2 instances**.
- Configure **AWS CloudWatch Alarms** for **Trusted Advisor alerts**.

📌 **Example:**
A **DevOps team** automates **S3 Bucket Public Access checks** using **AWS Lambda**.

CHAPTER 4: AWS TRUSTED ADVISOR BEST PRACTICES

✓ **Check Trusted Advisor Weekly** – Regularly review security, cost, and performance suggestions.
✓ **Enable MFA for IAM Users** – Secure AWS accounts from unauthorized access.
✓ **Optimize EC2 & RDS Resources** – Stop or downsize underutilized instances.
✓ **Monitor AWS Service Limits** – Avoid hitting **quota limits**.
✓ **Automate Trusted Advisor Fixes** – Use **AWS Lambda** & **CloudWatch**.

📌 **Example:**
A **SaaS company** automates **EBS snapshot backups** after Trusted Advisor recommends **fault tolerance improvements**.

Conclusion: Mastering AWS Trusted Advisor

By using **AWS Trusted Advisor,** businesses can:

✅ **Optimize AWS infrastructure for cost savings.**
✅ **Enhance security and detect vulnerabilities.**
✅ **Improve AWS performance and availability.**
✅ **Prevent service disruptions due to AWS limits.**

Final Exercise:

1. Open **AWS Trusted Advisor** and **identify one security issue**.
2. **Implement a cost-saving recommendation** from Trusted Advisor.
3. Research **how AWS Trusted Advisor helps enterprises reduce cloud costs**.

ASSIGNMENT

CREATE AN AUTO SCALING GROUP WITH LOAD BALANCER

SOLUTION: CREATE AN AUTO SCALING GROUP WITH A LOAD BALANCER

AWS **Auto Scaling Groups (ASG)** ensure that your application can handle varying levels of traffic by **automatically adding or removing EC2 instances** based on demand. When combined with an **Elastic Load Balancer (ELB),** traffic is evenly distributed among instances, improving availability and fault tolerance.

This step-by-step guide walks you through **creating an Auto Scaling Group (ASG) with a Load Balancer (ALB - Application Load Balancer)**.

Step 1: Log in to AWS Management Console

1. Open the **AWS Management Console**.
2. In the **search bar,** type and select **"EC2"**.

Step 2: Create a Launch Template for EC2 Instances

Before creating an Auto Scaling Group, we need a **Launch Template,** which defines the instance configurations.

1. Navigate to Launch Templates

1. In the **EC2 Dashboard,** click **"Launch Templates"** (under "Instances").
2. Click **"Create Launch Template"**.

2. Configure Launch Template

1. **Launch Template Name:** MyAutoScalingTemplate.

2. **AMI (Amazon Machine Image):** Select an AMI (e.g., **Amazon Linux 2023** or **Ubuntu**).
3. **Instance Type:** Choose t2.micro (or any other instance type as per requirement).
4. **Key Pair:** Choose an existing key pair or create a new one.
5. **Network Settings:** Select **"Don't include in launch template"** (VPC will be configured in the Auto Scaling Group).
6. **Storage:** Keep the default **8GB gp3 SSD** (or modify as needed).
7. **Security Group:**
    - Click **"Create New Security Group"**.
    - Allow **HTTP (80)** for public web access.
    - Allow **SSH (22)** from your IP.
8. **User Data (Optional – to install a web server automatically):**
    - Paste the following script (for Amazon Linux) to install Apache:
    - #!/bin/bash
    - yum update -y
    - yum install -y httpd
    - systemctl start httpd
    - systemctl enable httpd
    - echo "Welcome to My Auto Scaling Website" > /var/www/html/index.html
9. Click **"Create Launch Template"**.

Step 3: Create an Application Load Balancer (ALB)

1. Navigate to Load Balancers

1. In the **EC2 Dashboard,** click **"Load Balancers"** (under "Load Balancing").
2. Click **"Create Load Balancer"**.

2. Configure Load Balancer

1. **Choose Load Balancer Type:** Select **Application Load Balancer (ALB)**.
2. **Name:** MyAppLoadBalancer.
3. **Scheme:** Choose **"Internet-facing"**.
4. **VPC:** Select the default VPC (or a custom VPC if available).
5. **Availability Zones:** Select at least **two Availability Zones**.

3. Configure Security Group

1. Create a new security group or select an existing one.
2. Allow **HTTP (80) traffic** from anywhere (0.0.0.0/0).

4. Create a Target Group

1. Under **Listeners and Routing,** click **"Create a Target Group"**.
2. **Target Type:** Select **"Instances"**.
3. **Name:** MyAppTargetGroup.
4. **Protocol & Port:** HTTP (80).
5. **VPC:** Choose the same VPC as the Load Balancer.
6. Click **"Create Target Group"**.

5. Finalize and Create ALB

1. Go back to the **Load Balancer page** and refresh the Target Groups.
2. Select MyAppTargetGroup and click **"Create Load Balancer"**.

Step 4: Create an Auto Scaling Group (ASG)

1. Navigate to Auto Scaling Groups

1. In the **EC2 Dashboard**, click **"Auto Scaling Groups"**.
2. Click **"Create Auto Scaling Group"**.

2. Configure Auto Scaling Group

1. **Name:** MyAutoScalingGroup.
2. **Launch Template:** Select MyAutoScalingTemplate.
3. **VPC and Subnets:** Choose the same VPC as the Load Balancer. Select **at least two subnets** (for high availability).

3. Attach the Auto Scaling Group to ALB

1. Click **"Attach to an existing load balancer"**.
2. Select **"Application Load Balancer (ALB)"**.
3. Choose **"MyAppLoadBalancer"**.
4. Under **Target Groups**, select MyAppTargetGroup.

4. Define Scaling Policies

1. **Desired Capacity:** 2 (Number of instances to maintain).
2. **Minimum Capacity:** 2 (Never scale below this).
3. **Maximum Capacity:** 5 (Set based on expected load).
4. Click **"Add Scaling Policy"** → Choose **Target Tracking Scaling**.

5. Select **CPU Utilization** and set the threshold to **50%**.

5. Create Auto Scaling Group

Click **"Create Auto Scaling Group"** and AWS will start launching instances.

Step 5: Verify and Test the Setup

1. Check Auto Scaling Group Status

1. Go to **EC2 Dashboard → Auto Scaling Groups**.
2. Click on MyAutoScalingGroup and verify that instances are running.

2. Test Load Balancer

1. Go to **EC2 Dashboard → Load Balancers**.
2. Copy the **Load Balancer's DNS Name**.
3. Paste it into a web browser.
4. If configured correctly, it should display **"Welcome to My Auto Scaling Website"**.

3. Test Auto Scaling by Increasing Load

1. SSH into one of the EC2 instances:
2. ssh -i my-key.pem ec2-user@your-instance-public-ip
3. Install a CPU stress tool (for Amazon Linux):
4. sudo yum install -y stress
5. stress --cpu 2 --timeout 300
6. If CPU usage exceeds **50%,** the Auto Scaling Group should launch **new instances** automatically.

7. Monitor scaling activity in **Auto Scaling** → **Activity History**.

Step 6: Modify or Delete Auto Scaling Group

Modify ASG Settings

1. Go to **EC2** → **Auto Scaling Groups**.
2. Select MyAutoScalingGroup.
3. Click **"Edit"** to change **scaling policies, instance count, or attached ALB**.

Delete ASG and ALB

1. **Delete Auto Scaling Group** – Go to **Auto Scaling Groups**, select MyAutoScalingGroup, and delete it.
2. **Delete Load Balancer** – Go to **Load Balancers**, select MyAppLoadBalancer, and delete it.
3. **Delete Target Group** – Go to **Target Groups**, select MyAppTargetGroup, and delete it.

Conclusion: Successfully Created an Auto Scaling Group with a Load Balancer

By following this guide, you have:
✅ **Created a Launch Template for EC2 instances.**
✅ **Configured an Application Load Balancer (ALB) to distribute traffic.**
✅ **Created an Auto Scaling Group (ASG) to dynamically scale instances.**
✅ **Set up scaling policies to add/remove instances automatically.**

✅ **Verified that the ALB is routing traffic correctly to EC2 instances.**

This setup **ensures high availability, load balancing, and cost optimization** for your AWS infrastructure. 🚀

FINAL EXERCISE:

1. Modify the **scaling policy** to trigger at **60% CPU utilization** instead of 50%.
2. Create another **Auto Scaling Group** for a **different EC2 instance type**.
3. Research **how AWS Auto Scaling integrates with AWS Lambda** for **serverless scaling**.

By mastering **Auto Scaling & Load Balancing**, you can **deploy highly available and cost-effective cloud architectures**! 🚀

IMPLEMENT CLOUDWATCH ALARMS FOR AN EC2 INSTANCE

SOLUTION: IMPLEMENT CLOUDWATCH ALARMS FOR AN EC2 INSTANCE

Amazon **CloudWatch Alarms** allow you to **monitor EC2 instances in real-time** and **trigger notifications or automated actions** based on predefined conditions. This guide will walk you through **creating a CloudWatch Alarm to monitor EC2 CPU utilization** and sending alerts via **Amazon SNS (Simple Notification Service).**

Step 1: Log in to AWS Management Console

1. Open your **web browser** and go to the AWS Management Console.
2. Sign in to your AWS account.

Step 2: Navigate to CloudWatch

1. In the **AWS search bar,** type **"CloudWatch"** and select **CloudWatch** from the results.
2. Click on **"Alarms"** in the left-hand menu.
3. Click **"Create Alarm"**.

Step 3: Choose a Metric for Monitoring

1. Click **"Select metric"**.
2. In the **Browse tab,** click **"EC2"** → **"Per-Instance Metrics"**.
3. Find and **select the EC2 instance** you want to monitor.
4. Choose the metric **CPUUtilization** (to monitor CPU usage).

5. Click **"Select metric"**.

Step 4: Define the Alarm Conditions

1. Under **Statistic**, select **"Average"**.
2. Under **Period**, choose **"5 minutes"** (or your preferred interval).
3. Under **Threshold type**, select **"Static"**.
4. Set **"Whenever CPU Utilization is greater than 70%"** (Adjust as needed).
5. Click **"Next"**.

Step 5: Configure Notifications (Optional but Recommended)

1. Under **Notification**, choose **"Create new SNS topic"**.
2. Enter a **Topic Name** (e.g., EC2-CPU-Alerts).
3. Enter an **email address** to receive alerts.
4. Click **"Create topic"**.
5. AWS **sends a confirmation email** to your inbox—**confirm it** before proceeding.
6. Click **"Next"**.

Step 6: Define Alarm Actions (Optional: Automate Response)

You can configure actions **when the alarm triggers**, such as stopping, restarting, or scaling the EC2 instance.

1. Click **"Add EC2 action"** (Optional).

2. Choose an **Action Type** (e.g., "Stop the instance" when CPU exceeds 90%).
3. Select the **EC2 instance** for the action.
4. Click **"Next"**.

Step 7: Set Alarm Name and Description

1. Enter an **Alarm Name** (e.g., High-CPU-Usage-Alarm).
2. Add a **description** (e.g., "Triggers when EC2 CPU usage exceeds 70% for 5 minutes").
3. Click **"Next"**.

Step 8: Review and Create Alarm

1. **Review** all configurations.
2. Click **"Create Alarm"**.
3. The **alarm status** will initially be "OK" until the threshold is met.

Step 9: Test the CloudWatch Alarm (Optional)

To manually test the alarm, create high CPU usage on the EC2 instance:

1. SSH into your EC2 instance:
2. ssh -i my-key.pem ec2-user@your-ec2-public-ip
3. Run a CPU stress test:
4. sudo yum install stress -y # For Amazon Linux

5. stress --cpu 4 --timeout 300
6. Monitor the alarm in **CloudWatch → Alarms**.
7. After 5 minutes, you should receive an **email alert** (if SNS is configured).

Step 10: Modify or Delete CloudWatch Alarm

1. Go to **CloudWatch → Alarms**.
2. Click on the alarm you want to modify.
3. Click **"Edit"** to change settings.
4. To delete, click **"Actions" → "Delete"**.

CONCLUSION: SUCCESSFULLY IMPLEMENTED CLOUDWATCH ALARM FOR EC2

By following this guide, you have:
✅ **Created a CloudWatch Alarm** to monitor EC2 CPU usage.
✅ **Configured an SNS Notification** to receive email alerts.
✅ **Set an automated action** (optional) to stop or restart the instance.
✅ **Tested the alarm** by generating CPU load.

This setup **helps prevent performance issues and ensures real-time monitoring of EC2 instances**. 🚀

FINAL EXERCISE:

1. **Modify the alarm** to trigger at 80% CPU utilization.

2. **Create another CloudWatch Alarm** to monitor **disk usage** (DiskReadOps).
3. Research **how to integrate CloudWatch Alarms with AWS Lambda for auto-scaling**.

DEVELOP A SERVERLESS FUNCTION USING AWS LAMBDA

SOLUTION: DEVELOP A SERVERLESS FUNCTION USING AWS LAMBDA (STEP-BY-STEP GUIDE)

AWS **Lambda** is a **serverless computing service** that lets you run code **without provisioning or managing servers**. This guide walks you through **creating, deploying, and testing an AWS Lambda function** using the **AWS Console** and **AWS CLI**.

Step 1: Log in to AWS Management Console

1. Open the AWS Management Console.
2. Search for **Lambda** in the AWS services search bar and select **AWS Lambda**.

Step 2: Create a New AWS Lambda Function

1. Click **"Create Function"**.
2. Choose **"Author from Scratch"**.
3. Enter a **Function Name** (e.g., HelloWorldLambda).
4. Select **Runtime** (e.g., **Python 3.9** or **Node.js 18.x**).
5. Select **Execution Role**:
    - Choose **"Create a new role with basic Lambda permissions"** (allows logging to CloudWatch).
6. Click **"Create Function"**.

Step 3: Write the Lambda Function Code

1. Using AWS Console

1. In the **Function Code Editor,** delete the default code.
2. Enter the following **Python code**:
3. import json
4.
5. def lambda_handler(event, context):
6. return {
7. 'statusCode': 200,
8. 'body': json.dumps('Hello from AWS Lambda!')
9. }
10. Click **"Deploy"** to save changes.

---

2. Using AWS CLI (Optional)

1. Create a local Python file:
2. mkdir lambda_project && cd lambda_project
3. echo 'def lambda_handler(event, context): return {"statusCode": 200, "body": "Hello from AWS Lambda!"}' > lambda_function.py
4. Zip the function:
5. zip lambda_function.zip lambda_function.py
6. Create the Lambda function via AWS CLI:
7. aws lambda create-function --function-name HelloWorldLambda \

8. --runtime python3.9 --role arn:aws:iam::your-account-id:role/lambda-role \
9. --handler lambda_function.lambda_handler --zip-file fileb://lambda_function.zip

---

Step 4: Create a Test Event

1. In the AWS Lambda console, click **"Test"**.
2. Click **"Create New Test Event"**.
3. Enter an **Event Name** (e.g., TestEvent1).
4. Replace the event JSON with:
5. {
6. "message": "Hello Lambda!"
7. }
8. Click **"Create"**.

Step 5: Run and Validate the Lambda Function

1. Click **"Test"** to invoke the function.
2. The output should be:
3. {
4. "statusCode": 200,
5. "body": "\"Hello from AWS Lambda!\""
6. }
7. If successful, the Lambda function is working! ✅

Step 6: Deploy AWS Lambda with API Gateway (Optional - Making it a Web API)

1. Go to **AWS API Gateway** → Click **"Create API"**.
2. Choose **"HTTP API"** → Click **"Build"**.
3. Click **"Add Integration"** → Select **AWS Lambda**.
4. Choose the **HelloWorldLambda** function.
5. Click **"Next"**, set **Route as /hello**, and click **"Next"**.
6. Click **"Create"**.
7. Copy the **Invoke URL** and test in a browser:
8. curl -X GET https://your-api-id.execute-api.region.amazonaws.com/hello
9. You should see:
10. {"statusCode":200,"body":"Hello from AWS Lambda!"}

Step 7: Monitor Lambda Logs in CloudWatch

1. Go to **CloudWatch** → **Log Groups**.
2. Click /aws/lambda/HelloWorldLambda.
3. Review logs for function executions.

Step 8: Modify or Delete Lambda Function

1. To update, modify the function code and **click "Deploy"**.

2. To delete, go to **Lambda Console → Actions → Delete Function**.

Conclusion: Successfully Developed a Serverless Function with AWS Lambda

✅ **Created an AWS Lambda function.**
✅ **Tested the function** using the AWS console.
✅ **Integrated Lambda with API Gateway** to create a **serverless REST API**.
✅ **Monitored logs in CloudWatch**.

This setup enables **cost-efficient, auto-scalable, and event-driven applications**. 🚀

Final Exercise:

1. Modify the Lambda function to **return the current timestamp**.
2. Deploy a Lambda function using **Node.js or Go**.
3. Research **how AWS Step Functions** automate multi-step Lambda workflows.