☁️ AWS (Amazon Web Services) - Full Developer-Friendly Guide

🗺️ AWS Roadmap for Developers (Only Required Topics)

Here's a roadmap tailored just for **backend/full-stack developers and DevOps engineers** — we'll cover only **useful and interview-worthy services**:

✅ Core AWS Services to Learn

#	Topic	Description
1	**What is AWS?**	Intro, benefits, pricing
2	**EC2 (Elastic Compute Cloud)**	Launch Linux/Windows servers
3	**S3 (Simple Storage Service)**	Store files/images/videos
4	**IAM (Identity & Access Mgmt)**	Secure users, roles, permissions
5	**RDS (Relational Database Service)**	Use MySQL/PostgreSQL on cloud
6	**VPC (Virtual Private Cloud)**	Network control & IP ranges
7	**Elastic Load Balancer (ELB)**	Load balance traffic to servers
8	**Auto Scaling**	Automatically increase/decrease instances
9	**CloudWatch**	Monitoring logs, alarms, metrics
10	**Route 53**	DNS hosting, domain name management
11	**Lambda**	Serverless function execution
12	**ECS & EKS**	Container (Docker/K8s) management
13	**CloudFormation**	Infrastructure as Code (IaC)
14	**S3 + CloudFront**	Host static websites with CDN
15	**CodePipeline + CodeDeploy**	CI/CD automation on AWS
16	**Secrets Manager & Parameter Store**	Securely store credentials/keys

✅ Topic 1: What is AWS? (Amazon Web Services)

🔍 Definition:

AWS (Amazon Web Services) is a **cloud computing platform** by Amazon that provides **on-demand services** like computing power, storage, databases, networking, security, and many more — over the internet.

Think of it as **renting computers and tools from Amazon**, instead of buying and managing them yourself.

🎯 Why Use AWS?

Reason	Explanation
✅ **No upfront cost**	Pay only for what you use
🌍 **Global availability**	Data centers in 245+ countries
📈 **Scalable**	Easily handle traffic spikes
🔒 **Highly Secure**	Used by Netflix, NASA, Facebook
⚙️ **Automation Ready**	Works with CI/CD, Docker, Jenkins
🧰 **Wide Service Range**	200+ services for every use case

🧠 How AWS Works (Simple Explanation)

Imagine this:

You want to launch a website. Normally, you would need:

- A physical server (expensive!)
- A network setup
- A storage disk
- Security & power backup

With AWS:

- You create a **virtual server (EC2)**

- Store your files in **S3**
- Use **RDS** for databases
- Secure access via **IAM**
- Automatically scale and load-balance

⚡ All with **a few clicks or commands** – and no need to manage physical hardware!

💡 **Real-Life Example:**

👷 **You're building a student registration app**:

- Backend API: Hosted on **EC2**
- Student photos: Stored in **S3**
- Database: MySQL on **RDS**
- Access secured via **IAM**
- Logs and errors monitored via **CloudWatch**
- URL served via **Route 53**

This is exactly how real-world production apps work on AWS.

📜 **AWS Pricing (Simple View)**

Plan	Description
Free Tier	12-months free for many services (EC2, S3, RDS)
Pay-as-you-go	Billed by usage (per second/minute/GB)
Reserved & Spot	Discounts for long-term or spare capacity

💸 **Tip**: Always use **Free Tier** while learning.

📋 **Common Use Cases for Developers**

Use Case	AWS Service
Host web app	EC2 + RDS

Use Case	AWS Service
Store images/videos	S3
CI/CD deployment	CodePipeline + CodeDeploy
Serverless app	Lambda
Monitor logs	CloudWatch
DNS/domain	Route 53
Host Docker apps	ECS

💉 Interview Questions from Topic 1

1. What is AWS?
2. What are the benefits of AWS?
3. How is AWS cost-effective?
4. Name some common AWS services.
5. Difference between EC2 and Lambda?
6. What is S3 used for?
7. What is meant by cloud computing?

✅ Summary (Fast Recap)

- **AWS** = Rent compute, storage, and services via cloud
- **Why?** = Easy, fast, scalable, secure, pay-as-you-go
- **Popular services** = EC2, S3, RDS, IAM, Lambda, VPC
- **Used by** = Startups to large enterprises

✅ Topic 2: EC2 (Elastic Compute Cloud)

🔍 What is EC2?

Amazon EC2 (Elastic Compute Cloud) is a **virtual server** in the cloud that you can launch to run your applications (websites, APIs, databases, etc.).

Think of EC2 as **renting a computer from Amazon**, where:

- You choose the OS (Linux/Windows),
- Set CPU, memory, storage size,
- And control it remotely.

🔧 Why Use EC2?

Feature	Reason
🖥️ Virtual Servers	Run anything just like a physical server
⚙️ Highly Configurable	Choose RAM, CPU, storage as needed
📦 OS Flexibility	Use Ubuntu, Windows, Amazon Linux, etc.
🌍 Global	Available in all AWS regions
🛠️ Integrates with AWS	Works with S3, RDS, IAM, CloudWatch
💰 Cost-efficient	Pay per second/minute/hour

⚙️ How EC2 Works (Step-by-Step)

1. **Create an Instance**:
    - Choose an **Amazon Machine Image (AMI)** (OS like Ubuntu)
    - Select an **Instance Type** (e.g., t2.micro)
    - Configure network, storage, etc.
2. **Key Pair**:
    - Generate a key pair (used to SSH into your EC2)
    - Download the .pem file (important!)

3. **Security Group**:
    - Set inbound/outbound rules (e.g., allow HTTP, SSH)
4. **Launch**:
    - Click **Launch Instance**
    - After 1-2 mins, it's ready 🎉
5. **Connect**:
    - SSH from terminal using:

bash

CopyEdit

ssh -i keyname.pem ec2-user@<public-ip>

💡 **Real-Life Example:**

You're building a student registration app backend:

- You launch an **EC2** with Ubuntu
- Install Java, Spring Boot, MySQL
- Deploy your .jar or .war file
- Your REST API is now live on EC2's public IP

🧰 **EC2 Important Terms**

Term	Meaning
AMI	Amazon Machine Image (OS template)
Instance	The actual running virtual server
Instance Type	Hardware config (e.g., t2.micro)
Key Pair	Private key to connect via SSH
Security Group	Acts like a firewall for EC2
Elastic IP	Static IP address for EC2

Term	Meaning
EBS	Storage volume for your EC2

🧪 Interview Questions from Topic 2

1. What is Amazon EC2?
2. What is an AMI?
3. What is a security group in EC2?
4. Difference between Elastic IP and public IP?
5. What are key pairs used for?
6. How can you connect to an EC2 instance?
7. What happens if EC2 instance stops?
8. Can EC2 store data permanently?

✅ Benefits

- Quick and scalable server setup
- Full control over software/OS
- Can integrate with Auto Scaling
- Ideal for hosting websites, APIs, backend services

❌ Drawbacks

- Needs manual setup (unless automated via scripts)
- Can lose data on restart unless using EBS
- Requires security management (e.g., SSH access)
- Pricing can grow if not monitored

🧠 Summary (Fast Recap)

- EC2 = Cloud virtual machine

- Used to host applications/websites
- Needs key-pair, security group
- Can SSH into EC2 and deploy apps
- Works well with RDS, S3, VPC, etc.

✅ Topic 3: S3 (Simple Storage Service)

🔍 What is S3?

Amazon S3 (Simple Storage Service) is an **object storage** service used to **store files/data** like:

- Images
- Videos
- PDFs
- Backups
- Static website files

📦 You can think of it as a **giant cloud hard drive**, where you can upload and access data anytime from anywhere via URL or API.

🎯 Why Use S3?

Feature	Benefit
☁️ Fully Managed	No server or infrastructure needed
💾 Unlimited Storage	Store any amount of data
🌍 Accessible Everywhere	Global access via HTTP/HTTPS
🔐 Secure	Built-in encryption and access control
💸 Cost-Effective	Pay only for what you store and transfer
⚙️ Easy Integration	Works with EC2, Lambda, CDN, etc.

🧠 How S3 Works

- **Data is stored in Buckets** (like folders)
- Inside each bucket, you can upload **Objects** (files)
- Every object has a **unique URL**

- You can **make files public**, restrict access, or generate temporary download links

📂 **S3 Structure:**

bash

CopyEdit

Bucket: student-app-files

├— /photos/student1.jpg

├— /resumes/student2.pdf

├— /certificates/student3.png

🧰 **Key S3 Concepts**

Term	Meaning
Bucket	Top-level container (like a drive/folder)
Object	Actual file (image, video, doc, etc.)
Key	Name of the object (like file path)
Region	Physical location of bucket data
ACL	Access control list (who can access what)
S3 URL	Web link to access the object
Storage Class	Data access types (Standard, IA, Glacier)

⚙️ **Common Use Cases**

Use Case	How S3 Helps
Upload profile pictures	Store them in a bucket and access via URL
Static website hosting	Host HTML, CSS, JS in public S3 bucket
File sharing	Upload & share via pre-signed URL
Backup & archive	Store app/database backups safely

Use Case	How S3 Helps
Big data analytics	Store input/output data for processing

🔐 Security in S3

- Use **IAM roles & policies** to limit access
- Enable **encryption** (SSE-S3, SSE-KMS)
- Use **Bucket Policy** to define access rules
- Block all public access by default (optional)

💡 Real-Life Example:

You're building a resume portal:

- Users upload resumes via your Spring Boot app
- Resumes are stored in **S3 bucket**
- App saves the S3 **file URL** in database
- Admin can view/download resumes anytime

🧪 Interview Questions from Topic 3

1. What is Amazon S3?
2. What is a Bucket in S3?
3. Difference between Bucket and Object?
4. Can S3 host a website?
5. How do you secure your S3 data?
6. What is a pre-signed URL?
7. What is an S3 Storage Class?
8. How is S3 different from EBS?

✅ Benefits

- Easy file upload/download
- Secure and durable (99.999999999% availability)
- Built-in versioning & logging
- Automatically scales with file size or count
- Can integrate with Lambda, EC2, RDS, CloudFront

❌ **Drawbacks**

- Not meant for real-time file systems (e.g., you can't "edit" files directly)
- Needs permissions setup (can be complex for beginners)
- File retrieval from Glacier (cold storage) takes time

📌 **Summary**

- **S3 = Cloud file storage**
- Stores any type/size of file as **objects inside buckets**
- Files can be private/public
- Integrated with almost every AWS service

✅ Topic 4: IAM (Identity and Access Management)

🔍 What is IAM?

IAM (Identity and Access Management) is an AWS service that helps you **securely control access** to AWS services and resources.

Think of IAM as the **security guard** of AWS — it controls **who** can access **what** and **how**.

👮 Why IAM is Used?

Use	Reason
👥 Create Users	Allow individual developers access
🔐 Assign Permissions	Limit what users can do (read-only, full access, etc.)
🏢 Manage Roles	Grant temporary permissions to apps or services
📜 Use Policies	Define access control rules
🔁 Enable MFA	Extra security for logins (like OTP)

🧰 IAM Key Components

Component	Description
Users	Individual AWS accounts (e.g., for a developer)
Groups	Collection of users with same permissions
Roles	Temporary access to AWS resources (used by services, EC2, Lambda)
Policies	JSON rules that define what is allowed or denied
MFA	Multi-Factor Authentication for stronger security

⚙️ How IAM Works (Step-by-Step)

1. **Create a User**:
    - You add users (developers, testers) and give them passwords or keys

2. **Assign Policies**:
    - Attach policies (e.g., S3 read-only access) to users or groups
3. **Create Roles**:
    - Used when an EC2/Lambda service needs access to S3/RDS
4. **Fine-Grained Access**:
    - Set exact rules for which actions are allowed

🛡️ **Real-Life Example**

Imagine you're running a Spring Boot app on EC2:

- You want the app to access **S3** to upload files
- Instead of storing AWS credentials in your code (unsafe!), you:
    - Create an **IAM Role** with S3 access
    - Attach that role to EC2
    - App can now access S3 **securely**

🔐 **IAM Policy Example (S3 Read Access)**

json

CopyEdit

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

✅ Benefits

- **Highly Secure**: No need to store AWS credentials in code
- **Fine-Grained Access Control**: Allow/deny specific actions
- **Temporary Credentials**: Useful for short-lived access
- **Centralized Access Management**: Manage all access from one place
- **Audit Logs**: IAM integrates with CloudTrail to track activity

---

❌ Drawbacks

- IAM policies are written in JSON — may be hard to write/understand for beginners
- Misconfiguration can lead to **security risks** (e.g., giving too much access)

🧪 Interview Questions from Topic 4

1. What is IAM in AWS?
2. What is the difference between a user, group, and role?
3. What are IAM policies?
4. How does IAM role differ from user credentials?
5. What is MFA in IAM?
6. How do you secure your AWS resources using IAM?
7. Can you give an example of a policy?
8. How can an EC2 instance access S3 securely?

🧠 Summary

- IAM is AWS's identity and access control system.
- Create users, assign roles, and write policies to allow/deny access.
- Never hardcode credentials — use roles for services like EC2 or Lambda.

✅ Topic 5: Amazon RDS (Relational Database Service)

🔍 What is RDS?

Amazon RDS is a **managed relational database service** from AWS. It helps you set up, operate, and scale a relational database (like MySQL, PostgreSQL, or Oracle) in the cloud — **without needing to manage servers**.

📌 Why Use RDS?

Feature	Benefit
🛠️ Fully Managed	No need to install, patch, or manage DB software
👷 Easy to Use	Quick launch via AWS Console or CLI
📈 Scalable	Easy to increase storage and compute
🔐 Secure	Supports encryption, VPC, IAM
☁️ Highly Available	Multi-AZ deployments ensure uptime
🧰 Backup & Restore	Automated backups, snapshots, and point-in-time restore

---

🧠 How RDS Works

1. You choose your **DB engine** (MySQL, PostgreSQL, etc.)
2. You configure DB size, storage, backup preferences
3. AWS handles:
    - Hardware provisioning
    - Software patching
    - Monitoring and maintenance
4. You connect your app to RDS using a JDBC/URL string

💡 Real-Life Example

You're building a Spring Boot Student Registration System:

- Backend API stores student data in a MySQL database
- You create an **RDS MySQL instance**
- Configure DB URL in your application.properties
- App connects to RDS and reads/writes student info securely

🧰 Supported RDS Engines

- **MySQL**
- **PostgreSQL**
- **MariaDB**
- **Oracle**
- **Microsoft SQL Server**
- **Amazon Aurora** (AWS-optimized DB)

🏗️ Key RDS Features

Feature	Purpose
Multi-AZ Deployment	For high availability
Read Replicas	For performance & scalability
Automated Backups	Daily snapshots + point-in-time restore
Monitoring	CloudWatch integration
Encryption	At-rest and in-transit encryption
VPC Support	Private network configuration

⚙️ Steps to Create RDS DB

1. Go to **AWS Console > RDS**
2. Choose **Create Database**
3. Select **Engine type** (MySQL, PostgreSQL, etc.)
4. Set instance size (t3.micro for free tier)

5. Set DB name, username, and password
6. Enable **automatic backups** and **Multi-AZ** (if needed)
7. Launch

➡️ AWS gives you a **host URL** to connect your app

🛡 **Security Best Practices**

- Don't open public access (avoid 0.0.0.0/0)
- Use **VPC security groups** to restrict traffic
- Enable **encryption** and **backups**
- Rotate credentials regularly

✅ **Benefits of Using RDS**

- No manual maintenance
- Automatically scales with traffic
- Reliable and secure
- Integrates well with EC2, Lambda, etc.
- Saves time and effort

❌ **Drawbacks**

- Slightly costlier than managing your own DB
- Limited control vs self-hosted DBs
- Write-heavy apps might need Aurora instead

🧪 **Interview Questions from Topic 5**

1. What is Amazon RDS?
2. What are the advantages of RDS over self-managed DB?
3. Which engines does RDS support?

4. How do you connect a Java app to RDS?
5. What is the use of Multi-AZ and Read Replica?
6. What is Amazon Aurora?
7. Can we back up and restore data in RDS?
8. How does RDS differ from DynamoDB?

🧠 **Summary**

- RDS = fully managed relational database in AWS
- Choose engine (MySQL, PostgreSQL, etc.)
- AWS handles backups, scaling, patching
- Connect app using the provided DB URL
- Secure it using VPC, IAM, and encryption

✅ Topic 6: AWS Elastic Beanstalk

🔍 What is Elastic Beanstalk?

Elastic Beanstalk is a **Platform-as-a-Service (PaaS)** by AWS that helps you **deploy and manage applications** (like Java, Spring Boot, Node.js, etc.) **without needing to manage the infrastructure**.

Think of it as:
👉 "Just upload your code — AWS takes care of everything else."

💡 Why Use Elastic Beanstalk?

Purpose	Benefit
⚙️ Simplifies Deployment	You don't need to set up EC2, load balancer, RDS, etc.
🔁 Auto-scaling	Automatically scales up/down your app
🧰 Pre-configured	Comes with Java, Tomcat, Node.js, Python environments
📊 Monitoring	Integrated with CloudWatch for metrics
🛡 Security	Integrates with IAM, VPC, and security groups

🚀 Supported Platforms

- Java (Spring Boot, JSP)
- Node.js
- .NET
- Python
- PHP
- Go
- Ruby
- Docker

⚙️ How Elastic Beanstalk Works (Step-by-Step)

1. You package your app (e.g., Spring Boot .jar)
2. You upload it to **Elastic Beanstalk**
3. It automatically:
    - Launches an **EC2 instance**
    - Sets up a **load balancer**
    - Configures **Auto Scaling**
    - Deploys your app
4. Provides a public URL for access

👷 Real-Life Example (Spring Boot Deployment)

1. Create a Spring Boot .jar
2. Go to AWS Elastic Beanstalk
3. Create an **application**
4. Choose **Java** platform
5. Upload .jar file
6. AWS provisions EC2, security groups, etc.
7. Your app is live at: http://your-app.elasticbeanstalk.com

📂 Deployment Options

Method	Description
AWS Console	Easy UI to upload app
AWS CLI	Use eb init, eb create, eb deploy
CI/CD	Integrate with GitHub, CodePipeline, Jenkins

⚙️ Configuration Options

- Instance type (e.g., t2.micro)

- Environment variables (e.g., DB URL)
- Auto-scaling rules
- Logs and health monitoring
- VPC settings

🛡 **Security in Elastic Beanstalk**

- Use **IAM roles** to control permissions
- Secure EC2 via **Security Groups**
- Add HTTPS using **SSL certificates**

✅ **Benefits of Elastic Beanstalk**

- 🎯 Focus on code, not infrastructure
- 🔁 Auto-scaling and load balancing included
- 📊 Health checks, logging, and metrics
- 🧩 Integrates with RDS, S3, IAM
- ⚙️ Supports versioning and rollback

❌ **Drawbacks**

- Less control over infrastructure compared to manual EC2 setup
- Not ideal for highly custom architecture
- Startup time may be slow for large apps
- Charges for underlying resources (EC2, S3, etc.)

💉 **Interview Questions from Topic 6**

1. What is AWS Elastic Beanstalk?
2. How does Elastic Beanstalk work internally?
3. How can you deploy a Spring Boot app on Elastic Beanstalk?

4. What are the advantages of using Elastic Beanstalk?
5. How does Elastic Beanstalk handle auto-scaling?
6. Can you customize EC2 instances in Beanstalk?
7. What are the alternatives to Elastic Beanstalk?
8. What are the limitations or drawbacks?

🧠 Summary

- Elastic Beanstalk helps you **deploy applications quickly** without managing infrastructure.
- Ideal for Java/Spring Boot apps when you want to **focus on development**, not infrastructure setup.
- Handles provisioning, deployment, scaling, and health checks.

✅ Topic 7: AWS S3 (Simple Storage Service)

🔍 What is S3?

Amazon S3 (Simple Storage Service) is a **cloud storage service** that lets you **store and retrieve any amount of data** at any time from anywhere on the web.

It's used to store:

- Images, videos, audio
- Backups
- Logs
- Static web content
- Files generated by your app (e.g., profile pictures, documents)

📌 Why Use S3?

Feature	Benefit
📁 Object Storage	Store files (called "objects") in "buckets"
🌍 Accessible via URL	Public/private access via HTTP
🔐 Secure	IAM policies, bucket policies, encryption
♻️ Durable	99.999999999% durability (11 9s!)
💰 Cheap	Pay only for storage used
🔁 Versioning	Restore deleted/overwritten files
🧠 Easy to Integrate	Works well with Java apps, Spring Boot, etc.

⚙️ Key Concepts

Term	Meaning
Bucket	Top-level container for storing objects (like a folder)
Object	The file (image, PDF, video, etc.) you store

Term	Meaning
Key	The unique name for each object in a bucket
Region	Location where bucket is stored (e.g., ap-south-1)

💡 **Real-Life Example (Spring Boot)**

You're building a property listing app. Users upload property images. You:

1. Upload the image file to S3 using the AWS SDK
2. Store the **S3 URL** in your database
3. Use the URL to show images in the app

📂 **How to Use S3 in Java (Spring Boot)**

1. Add AWS SDK dependency:

xml

CopyEdit

```
<dependency>
  <groupId>software.amazon.awssdk</groupId>
  <artifactId>s3</artifactId>
</dependency>
```

2. Configure AWS credentials:

- Use ~/.aws/credentials
- Or IAM Role (if on EC2/Lambda)

3. Sample Code:

java

CopyEdit

```
S3Client s3 = S3Client.builder().region(Region.of("ap-south-1")).build();
PutObjectRequest request = PutObjectRequest.builder()
  .bucket("your-bucket-name")
```

```
    .key("uploads/image.jpg")
    .build();
s3.putObject(request, RequestBody.fromFile(new File("image.jpg")));
```

🛡 Security Options

- **Bucket Policy** – control who can access your bucket
- **IAM Policy** – allow access to specific buckets via IAM users/roles
- **Pre-signed URLs** – share temporary access to private files

---

📊 Use Cases

- Store user uploads
- Backup/restore
- Store logs and reports
- Host static websites
- Integrate with CloudFront for CDN

📈 Storage Classes

Class	Use Case
Standard	General purpose
Intelligent-Tiering	Auto-moves to cheaper tiers
Infrequent Access	For rarely accessed data
Glacier	For long-term archival
Glacier Deep Archive	Cheapest, longest retrieval

✅ Benefits of S3

- Highly available and scalable
- Easy to integrate in Java apps

- Durable (11 9s)
- Secure
- Global access
- Backup and version control

❌ Drawbacks

- Complex access controls if not managed properly
- Public access misconfiguration risk (data leaks)
- Not for storing databases or running apps — only files

🧪 Interview Questions from Topic 7

1. What is Amazon S3?
2. What is the difference between a bucket and an object?
3. How do you upload a file to S3 in Java?
4. How do you secure files in S3?
5. What are storage classes in S3?
6. What are pre-signed URLs and why are they used?
7. What is the durability and availability of S3?
8. Can S3 host a static website?

🧠 Summary

- S3 is a powerful, scalable file storage system.
- Perfect for apps needing media storage or file uploads.
- Easy to use with Spring Boot or any Java backend.
- Security and access control is essential.

✅ Topic 8: AWS IAM (Identity and Access Management)

🔍 What is IAM?

IAM (Identity and Access Management) is a **security service** in AWS that helps you **control access** to AWS services and resources securely.

It allows you to:

- Create **users, roles, and groups**
- Assign **permissions**
- Use **policies** to control **who can access what**

💡 Why Do We Use IAM?

Need	How IAM Helps
🔐 Secure access to AWS	Control who can log in and what they can do
🔄 Programmatic access	Use IAM roles to allow EC2/Lambda to access services
📜 Audit & monitor	Track user actions via CloudTrail
👨‍💼 Team management	Create roles for developers, testers, admins, etc.

🧱 Key Components of IAM

Component	Description
User	Individual account with login credentials
Group	Collection of users with common permissions
Role	Set of permissions assumed by users or services (no credentials)
Policy	JSON document that defines permissions
Permissions	Actions allowed/denied (e.g., s3:PutObject)

⚙️ How IAM Works

1. **You create users or roles**
2. **Attach policies** that define access rules (Allow/Deny)
3. AWS checks IAM policies to decide if an action is allowed
4. Permissions are **evaluated before** any request is served

---

🧰 IAM Policy Example (JSON)

Allow S3 read access:

json

CopyEdit

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

🔐 Real-Life Use Cases

Use Case	IAM Role
EC2 needs to access S3	Assign IAM Role to EC2
Developer needs limited access	Create IAM user and assign read-only policy
Lambda writing logs	IAM Role with logs:PutLogEvents permission
S3 Bucket access control	Bucket policy + IAM user access

🚫 Types of Permissions

1. **Identity-based policies**: attached to users, groups, roles
2. **Resource-based policies**: attached directly to resources (like S3 buckets)
3. **Permissions boundaries**: limits what IAM roles can do
4. **Service Control Policies (SCP)**: for AWS Organizations (account-level restrictions)

📌 Best Practices for IAM

Practice	Why
🔐 Use least privilege	Minimize risk of misuse
🔁 Rotate access keys	Avoid long-term exposure
🚫 Don't use root user	Use it only for setup
🧩 Use IAM Roles	Better than hardcoding credentials
📊 Enable MFA	For strong authentication

✅ Benefits of IAM

- Centralized control over access
- Secure management of credentials
- Fine-grained permissions
- Auditable and trackable (CloudTrail)
- Free to use (only underlying services are billed)

❌ Drawbacks

- Complex policies can be hard to manage
- Misconfigured permissions may lead to security holes
- Requires understanding of JSON syntax

🧪 Interview Questions from Topic 8

1. What is IAM in AWS?
2. What is the difference between a user and a role?
3. What is a policy in IAM? Can you write an example?
4. How can you secure your AWS account using IAM?
5. What is the principle of least privilege?
6. What are the best practices for IAM?
7. How do IAM roles help secure access to AWS resources?
8. What are the types of IAM policies?

🧠 Summary

- IAM controls **who can access what** in AWS.
- You define **users, roles, and policies** to grant secure access.
- **IAM Roles** are crucial for services like EC2, Lambda, etc.
- Always follow **least privilege** and **rotate keys/MFA**.

✅ Topic 9: AWS RDS (Relational Database Service)

🔍 What is AWS RDS?

Amazon RDS (Relational Database Service) is a **fully managed cloud database service** that makes it easy to set up, operate, and scale **relational databases** like:

- MySQL
- PostgreSQL
- Oracle
- SQL Server
- MariaDB
- Amazon Aurora

💡 Why Do We Use RDS?

Problem	Solution
🛠️ Managing DB servers is hard	RDS automates backups, patching, and scaling
⏱️ Time-consuming setup	RDS launches DB instances in minutes
🔐 Security and access control	IAM, VPC, and encryption are built-in
⚖️ High availability	Multi-AZ deployment
📈 Auto scaling	Easily scale storage and performance

⚙️ How RDS Works

1. Choose a database engine (e.g., MySQL)
2. Set up an instance (RAM, CPU, storage)
3. Configure security (VPC, subnets, port, IAM)
4. Use JDBC/ODBC connection string in your Java app
5. AWS handles:
    - Backups

- Monitoring
- Failover
- Updates

🧠 Real-Life Java Example

1. You build a Spring Boot app.
2. You use **RDS MySQL** to store user data.
3. In application.properties, you use:

properties

CopyEdit

spring.datasource.url=jdbc:mysql://your-db-name.rds.amazonaws.com:3306/dbname

spring.datasource.username=admin

spring.datasource.password=yourPassword

4. Now your app connects securely to the RDS instance — no manual DB install or maintenance required.

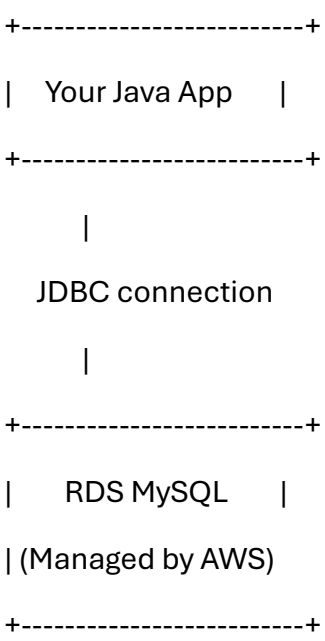
📦 Key RDS Features

Feature	Use
Multi-AZ	High availability and auto-failover
Read Replicas	Improve read performance
Snapshots	Manual or scheduled backups
Monitoring	CloudWatch metrics and alerts
Encryption	At rest and in transit (SSL/TLS)
VPC support	Launch inside a private network

⚙️ RDS Architecture (Simple)

pgsql

CopyEdit



🔐 Security

- Use **VPC security groups** to control access
- **IAM integration** for authentication
- **Encrypt DB** using KMS
- Use **SSL** for secure connection

✅ Benefits of RDS

- Fully managed DB
- Backup, patching, and failover handled by AWS
- Easily scalable storage and performance
- High availability and disaster recovery (Multi-AZ)
- Secure and easy integration with Java apps

❌ Drawbacks

- Less control than self-hosted DB
- Costs can grow with storage and traffic

- Write scaling is limited (horizontal write scaling is not supported)

💉 Interview Questions from Topic 9

1. What is Amazon RDS?
2. What DB engines are supported in RDS?
3. How do you connect Spring Boot with RDS?
4. What is Multi-AZ in RDS?
5. What is the difference between Read Replica and Multi-AZ?
6. How is RDS different from self-managed DB on EC2?
7. Can you encrypt RDS databases?
8. What monitoring tools are available for RDS?

🧠 Summary

- RDS makes it simple to use a reliable, scalable, and secure database without managing the server yourself.
- Ideal for **Java backend applications**, especially with Spring Boot.
- Supports failover, replication, and snapshots for production-grade apps.

✅ Topic 10: AWS Lambda (Serverless Compute Service)

🔍 What is AWS Lambda?

AWS Lambda is a **serverless compute service** that lets you run code **without provisioning or managing servers**. You simply upload your code, and AWS automatically handles everything required to run and scale it.

💡 Why Use Lambda?

Problem	Lambda Solves
⛏ Server setup overhead	No server management at all
📉 Wasted compute costs	You only pay when code runs
🚀 Need auto-scaling	Lambda scales automatically
🔁 Trigger-based tasks	Lambda runs in response to events (e.g., S3 file upload)

⚙️ How AWS Lambda Works

1. You write a **function** (e.g., in Java, Python, Node.js)
2. You deploy it to AWS Lambda
3. You define a **trigger** (e.g., S3 upload, API Gateway request)
4. Lambda automatically **executes the function** when the event happens
5. You are billed **only for execution time**

🧠 Real-Life Java Use Case

Problem:

You want to compress images when they are uploaded to S3.

Solution:

- Write a Java Lambda function that compresses images
- Set the function to **trigger when a file is uploaded to an S3 bucket**

Lambda Handler in Java:

java

CopyEdit

```
public class MyHandler implements RequestHandler<S3Event, String> {
  public String handleRequest(S3Event event, Context context) {
    // Access the uploaded file from S3 and process it
    return "Processed file";
  }
}
```

🔌 Common Lambda Triggers

Source	Example
API Gateway	Serverless REST API
S3	Image processing on upload
DynamoDB	Auto-processing of DB changes
SNS/SQS	Background jobs, notifications
EventBridge	Scheduled cron jobs

🧰 Key Features

Feature	Use
Auto-scaling	Handles 1 to 10,000+ requests instantly
Stateless	Each invocation is independent
Event-driven	Reacts to triggers from AWS services
Language support	Java, Python, Node.js, Go, etc.
Environment Variables	Pass config without hardcoding
IAM Permissions	Secure access to other AWS resources

📦 Architecture Example

arduino

CopyEdit

User uploads image → S3 Bucket → Triggers Lambda → Lambda compresses image → Saves to another bucket

---

📌 Java Lambda Deployment

You can deploy Lambda using:

- AWS Console (upload .zip or .jar)
- AWS CLI
- Infrastructure-as-code (CloudFormation, Terraform)
- AWS SAM (Serverless Application Model)
- Maven Plugin (for Java)

✅ Benefits

- No server maintenance
- Automatically scales
- High availability by default
- Cost-effective (pay-per-use)
- Easy integration with AWS services

❌ Drawbacks

Limitation	Details
🧠 Cold starts	Initial delay when function is idle
⏱️ Timeout	Max 15 minutes runtime
📦 Package limit	250MB package size (uncompressed)

Limitation	Details
💾 Stateless	No persistence between calls (use S3/DB for data)

🧪 Interview Questions from Topic 10

1. What is AWS Lambda?
2. How is Lambda different from EC2?
3. What languages does AWS Lambda support?
4. Can you trigger Lambda from S3 or API Gateway?
5. What are cold starts in Lambda?
6. How can Lambda be used in Java applications?
7. What are the benefits and limitations of Lambda?
8. What is the maximum execution time for a Lambda function?
9. Can Lambda functions access other AWS services?

🧠 Summary

- **AWS Lambda** lets you run backend logic **without provisioning servers**.
- Commonly used for microservices, automation, scheduled jobs, file processing, and real-time apps.
- Great for Java developers using **S3**, **API Gateway**, or **event-driven** apps.

✅ Topic 11: Amazon DynamoDB (NoSQL Database Service)

🔍 What is DynamoDB?

Amazon DynamoDB is a **fully managed NoSQL database** offered by AWS. It provides **fast and predictable performance with seamless scaling**.

Unlike traditional SQL databases, DynamoDB stores **data in key-value or document format**, which makes it ideal for applications requiring **low-latency data access at any scale**.

💡 Why Use DynamoDB?

Problem	Solution with DynamoDB
❌ Complex database scaling	✅ DynamoDB automatically scales read/write throughput
❌ Slow performance under load	✅ Consistent performance under any scale
❌ Manual infrastructure	✅ Fully serverless and managed by AWS
❌ Traditional RDBMS not suitable for high-speed, flexible schema	✅ NoSQL structure allows flexible, unstructured data

---

📐 Key Concepts

Term	Description
Table	Collection of items (like a table in SQL)
Item	A single row (JSON-style data)
Attribute	A field in the item (like a column)
Primary Key	Uniquely identifies an item (can be partition key or partition + sort key)
Partition Key	Used to distribute data

Term	Description
Sort Key	Optional; used for sorting items with same partition key
Secondary Index	Like a SQL index to allow different query patterns

---

⚙️ How It Works

1. You create a **table** with a **primary key**.
2. You insert **items** (JSON format).
3. You query or scan data using **keys or indexes**.
4. DynamoDB auto-scales throughput and storage.
5. You can enable features like:
    - TTL (auto-expire data)
    - Streams (track changes)
    - Global tables (multi-region replication)

🧠 Real-Life Java Use Case

Let's say you're building a **user login system** with Spring Boot.

- You use DynamoDB to store user profiles (id, name, email, password).
- You access the DB via **AWS SDK for Java**.

Sample Code (using AWS SDK v2):

java

CopyEdit

```
DynamoDbClient client = DynamoDbClient.create();


PutItemRequest request = PutItemRequest.builder()
  .tableName("Users")
  .item(Map.of(
    "userId", AttributeValue.fromS("101"),
```

```
    "email", AttributeValue.fromS("xyz@example.com")
  ))
  .build();

client.putItem(request);
```

🔐 Security in DynamoDB

- Use **IAM roles and policies** to control access.
- Supports **encryption at rest and in transit**.
- Can integrate with **VPC endpoints** for secure private access.

📦 Features of DynamoDB

Feature	Description
On-Demand / Provisioned Capacity	Choose automatic scaling or fixed throughput
Global Tables	Replicate across multiple AWS regions
Streams	Capture real-time changes
DAX	In-memory caching for faster reads
TTL (Time To Live)	Auto-delete expired data
Transactions	ACID-compliant transactions supported

✅ Benefits

- Fully managed (no server setup)
- High availability and durability
- Very low latency for reads/writes
- Horizontal scaling
- Serverless
- Ideal for mobile, IoT, gaming, and real-time apps

❌ Drawbacks

Limitation	Explanation
❗ No joins or complex queries	No SQL-like operations
❗ Size limit	Max 400KB per item
❗ Querying flexibility	Queries limited to key/index patterns
❗ Learning curve	Different from traditional RDBMS

📊 Use Cases

- User session data
- Shopping cart data
- Product catalogs
- IoT sensor data
- Leaderboards in games

🧪 Interview Questions from Topic 11

1. What is DynamoDB?
2. How is DynamoDB different from RDS?
3. What is the primary key in DynamoDB?
4. What's the difference between partition key and sort key?
5. What are the read/write capacity modes?
6. What are Global Secondary Indexes (GSI)?
7. What is DAX in DynamoDB?
8. What is DynamoDB Streams?
9. How can you ensure security in DynamoDB?
10. Can DynamoDB handle transactions?

🧠 Summary

- DynamoDB is a NoSQL database that is highly scalable, fast, and fully managed.
- Ideal for use cases where flexible schema, high-speed access, and horizontal scaling are needed.
- You can use it easily with Java via AWS SDK.

✅ Topic 12: Amazon S3 – Static Website Hosting & Versioning

🔍 What is Amazon S3?

Amazon S3 (Simple Storage Service) is an **object storage service** that lets you store and retrieve **any amount of data from anywhere**. It's widely used for:

- File backups
- Static website hosting
- Application assets
- Logs
- Data lakes

🎯 Use Case Focus: Static Website Hosting + Versioning

In this topic, we'll focus on:

1. How to **host a static website** on S3
2. How to **enable versioning** for backup and rollback of files

---

🧠 Static Website Hosting with S3

You can host **HTML, CSS, JS** files in S3 and make them publicly accessible as a **website**.

🔧 Steps:

1. **Create an S3 bucket**
    - Bucket name must be **globally unique**
    - Example: my-website-bucket
2. **Upload your website files**
    - Upload index.html, styles.css, etc.
3. **Enable Static Website Hosting**
    - Go to **Properties → Static website hosting**
    - Enable it and set:

        - Index document: index.html
        - Error document: error.html (optional)

4. **Make your files public**
    - Go to **Permissions → Bucket Policy** and add:

```
json

CopyEdit

{

 "Version": "2012-10-17",

 "Statement": [{

  "Sid": "PublicReadGetObject",

  "Effect": "Allow",

  "Principal": "*",

  "Action": "s3:GetObject",

  "Resource": "arn:aws:s3:::my-website-bucket/*"

 }]

}
```

5. **Access your website**
    - Use the website URL provided under static hosting settings.
    - Format: http://<bucket-name>.s3-website-<region>.amazonaws.com

---

🗂️ Versioning in S3

S3 Versioning allows you to **keep multiple versions of an object** in the same bucket. This protects against:

- Accidental overwrite
- Deletion of important files

🧪 How to Enable Versioning:

1. Go to your bucket → **Properties**

2. Under **Bucket Versioning**, click **Enable**

Now every time you upload a file with the same name, a **new version** is created instead of overwriting the old one.

🔁 Rollback a File

You can go to the file → **Versions tab** → **Choose older version** → **Download/restore**

👷 Real-Life Java Use Case

Suppose you're building a Java app that uploads user profile images.

- You enable versioning so users can restore older profile pictures.
- Java uses the AWS SDK to upload files:

java

CopyEdit

```
PutObjectRequest request = PutObjectRequest.builder()
  .bucket("my-versioned-bucket")
  .key("profile.jpg")
  .build();

s3Client.putObject(request, RequestBody.fromFile(new File("profile.jpg")));
```

🔐 Security Best Practices

- Use **IAM roles** to give secure access.
- Don’t make entire buckets public unless it's a website.
- Use **S3 bucket policies** and **CORS** if needed for web apps.

✅ Benefits

Benefit	Explanation
🎯 Easy hosting	Host static websites in minutes

Benefit	Explanation
🔄 Version control	Prevents accidental data loss
🧠 Simple setup	No backend or server needed
🌍 Global access	Reliable, scalable, global

❌ Drawbacks

Drawback	Details
❌ No backend logic	You can't run server code like PHP or Java
❌ Manual version cleanup	Old versions may pile up unless managed
❌ Public bucket risks	Exposing too much can lead to data leaks

---

🧪 Interview Questions from Topic 12

1. What is Amazon S3 used for?
2. How can S3 host static websites?
3. How do you make S3 files publicly accessible?
4. What is versioning in S3 and how does it work?
5. Can you roll back to an older version of a file in S3?
6. How do you secure S3 buckets?
7. What are the limitations of hosting a website on S3?
8. Can S3 store large files?
9. How is data organized in S3?
10. What are buckets, objects, and keys in S3?

🧠 Summary

- **S3 Static Hosting** is ideal for frontend websites (HTML, CSS, JS).
- **Versioning** helps maintain and restore previous versions of files.

- It's simple, cost-effective, and integrates well with other AWS services.

✅ Topic 13: Amazon CloudFront (CDN for S3 / Static Content)

🔍 What is Amazon CloudFront?

Amazon CloudFront is a **Content Delivery Network (CDN)** by AWS. It **distributes content (like HTML, CSS, JS, images, videos)** to users across the globe with **low latency and high speed**.

Instead of loading content directly from your S3 bucket or server (which may be located far from users), CloudFront caches content in **edge locations** close to your users.

💡 Why Use CloudFront?

Without CloudFront	With CloudFront
User in India accesses content from US	Content served from Mumbai edge location
High latency, slower load time	Faster load time due to caching
Heavy load on origin (S3/EC2)	CloudFront offloads this by caching

---

⚙️ How CloudFront Works

1. You store your **static content** in S3 (e.g., website images, videos, CSS, JS).
2. You create a **CloudFront distribution** and **connect it to S3** as the origin.
3. CloudFront **caches** content in 300+ **edge locations** worldwide.
4. When users request content:
    - CloudFront checks the nearest edge cache.
    - If available → serves from cache.
    - If not → fetches from S3 (origin), caches it, and then serves.

📐 Key Components

Component	Description
Origin	Source of content (S3, EC2, load balancer, etc.)
Edge Location	AWS data centers globally that cache content
Distribution	Configuration of CloudFront setup
Cache Behavior	Rules to control caching, request forwarding, etc.
TTL (Time-to-Live)	How long an object stays in cache
Invalidation	Force CloudFront to remove outdated content

👷 Real-Life Java Use Case

You host your static frontend (React app) in **S3** and serve it via **CloudFront**.

1. Upload build files to S3.
2. Set up CloudFront distribution pointing to the S3 bucket.
3. Java backend can serve dynamic data via API Gateway or EC2, while CloudFront handles the static part efficiently.

🔐 Security with CloudFront

Feature	Benefit
HTTPS support	Secure transmission of content
Signed URLs / Cookies	Grant temporary access to private content
Geo restriction	Block access from specific countries
Origin Access Control (OAC)	Securely connect to private S3 buckets (replacement of OAI)

📦 Key Features

Feature	Description
Low Latency	Serves content from nearest edge location
Caching	Stores content closer to users
HTTPS	Supports SSL/TLS encryption
Custom Domain + SSL	Serve from your own domain with custom certificate
Invalidations	Remove/refresh cache manually
Logging	Track requests via CloudFront access logs

✅ Benefits

- ⚡ Lightning-fast content delivery
- 🌍 Global edge location support
- 🔒 Enhanced security (HTTPS, geo-restrictions)
- 🧠 Load off backend (cost + speed efficient)
- 📊 Real-time performance with logging & monitoring

❌ Drawbacks

Drawback	Explanation
❗ Slight learning curve	Needs initial config understanding
❗ Invalidation cost	Too many invalidations can cost extra
❗ Caching issues	Might serve outdated content if not invalidated

---

🧪 Interview Questions from Topic 13

1. What is Amazon CloudFront?
2. How does CloudFront improve website performance?
3. What are edge locations?
4. Can CloudFront serve private content?

5. What is TTL in CloudFront?
6. How do you invalidate a file in CloudFront?
7. How is CloudFront used with S3?
8. How do you use CloudFront with custom domains?
9. What are signed URLs in CloudFront?
10. What is Origin Access Control (OAC) in CloudFront?

🧠 **Summary**

- **CloudFront is a CDN** used to distribute and cache content globally.
- Works perfectly with **S3** for static websites and **EC2/API Gateway** for dynamic content.
- Improves **speed, scalability, and security**.

✅ Topic 14: Amazon API Gateway – Build, Host & Secure REST APIs

---

🔍 What is Amazon API Gateway?

Amazon API Gateway is a **fully managed service** that lets you create, publish, maintain, monitor, and secure **REST, HTTP, and WebSocket APIs** at any scale.

It's often used to:

- Expose **backend services (Lambda, EC2, Microservices)** via **REST APIs**
- Secure APIs using **authentication, throttling, and logging**
- Handle traffic spikes automatically without manual scaling

💡 Why Use API Gateway?

Need	Solution
Expose backend logic via APIs	Use API Gateway to create REST endpoints
Secure APIs (JWT, IAM, etc.)	Use built-in auth layers
Monitor requests	CloudWatch integration
Limit traffic per user/IP	Use throttling and rate limiting

🧱 Key Components

Component	Description
Resource	Endpoint path (/users, /orders)
Method	HTTP methods (GET, POST, PUT, DELETE)
Integration	Connects to Lambda, EC2, or any backend
Stage	Environment (e.g., dev, prod)
Deployment	Publishes the API
API Key / Usage Plan	Limit access based on key and quota

⚙️ How It Works (Architecture)

pgsql

CopyEdit

User Request (HTTP)

↓

API Gateway

↓

+----------------------+

| Integration Options |

| - AWS Lambda |

| - HTTP (EC2/microservice) |

| - Mock / VPC |

+----------------------+

↓

Response

🧪 Example Use Case

A Java Spring Boot backend deployed on **EC2** or **Lambda** (via AWS Java SDK):

- API Gateway receives a POST /register
- Forwards it to your Lambda function or EC2 endpoint
- Returns the response to the client (browser/mobile)

🔐 Security Features

Feature	Purpose
IAM Auth	Allow only AWS IAM users
Cognito Auth	JWT/OAuth-based login (user pools)
API Keys	Identify and limit users

Feature	Purpose
Throttling	Prevent misuse (e.g., 1000 req/min)
Resource Policies	Restrict access to IP, VPC, etc.

📈 Monitoring Features

- Integrated with **CloudWatch** for:
  - Logging requests/responses
  - Metrics (latency, error rates, etc.)
- Can trigger **alarms** if usage or error spikes

🧠 Benefits

Benefit	Explanation
🧠 Serverless	No infra setup needed
🔒 Secure	Supports JWT, IAM, and API keys
🚀 Scalable	Handles millions of requests/sec
⚙️ Integration	Connects easily with Lambda, EC2, DynamoDB, etc.
📊 Monitoring	Full request logs & analytics

❌ Drawbacks

Drawback	Explanation
⚠️ Cold start	Slight delay if used with Lambda (first-time call)
💰 Pricing	Pay per call, can get costly at high volumes
🧩 Complexity	Setup of integration/stage/methods can confuse beginners

---

📘 Real Life Example

E-commerce site backend:

- /products → GET (EC2 service)
- /order → POST (AWS Lambda)
- /auth/login → POST (Cognito authentication)

All APIs are behind a single gateway with logging, rate limits, and monitoring enabled.

💉 Interview Questions

1. What is Amazon API Gateway?
2. How do you expose a Lambda function via API?
3. What is a Stage in API Gateway?
4. How does API Gateway help with security?
5. What is throttling in API Gateway?
6. Can you connect API Gateway with EC2?
7. What is the use of an API key?
8. How do you monitor requests in API Gateway?
9. What's the difference between proxy and non-proxy integration?
10. How does API Gateway work with Cognito?

🧠 Summary

- API Gateway acts as a **front door** for backend services.
- It adds **security**, **scalability**, and **monitoring** without managing servers.
- Can connect to almost any backend: **Lambda, EC2, containers, VPCs**.

✅ Topic 15: AWS Lambda – Serverless Compute for Java

🔍 What is AWS Lambda?

AWS Lambda is a **serverless compute service** that lets you **run code without provisioning or managing servers**. You just write your function, upload it, and AWS handles the rest (infrastructure, scaling, etc.).

Lambda supports **Java**, along with many other languages like Python, Node.js, Go, etc.

💡 Why Use Lambda?

Need	Solution
Run backend logic without server	Use Lambda
Auto-scale backend	Lambda scales automatically
Reduce server cost	Pay only when your function runs
React to AWS events (S3 upload, API Gateway call)	Use Lambda triggers

---

⚙️ How AWS Lambda Works

1. You write your **Java method** (or any supported language).
2. Package it as a .jar (or .zip for native runtimes).
3. Upload it to Lambda.
4. Set a **trigger** (e.g., API Gateway, S3, DynamoDB).
5. Lambda runs the function **only when triggered**, and shuts down after.

📦 Common Triggers for Lambda

Trigger	Use Case
API Gateway	Expose your function as a REST API
S3	Process files when uploaded (e.g., resize image)
DynamoDB Streams	React to DB insert/update/delete

Trigger	Use Case
SNS/SQS	Process messages from pub-sub/queue
CloudWatch Events	Schedule functions (like a cron job)

📁 Structure for Java Lambda

You create a handler class like this:

java

CopyEdit

```
public class MyHandler implements RequestHandler<String, String> {
  @Override
  public String handleRequest(String input, Context context) {
    return "Hello " + input;
  }
}
```

- String input/output can be replaced by custom DTOs.
- You upload it to Lambda as a **.jar file**.

🔐 Security

- Lambda runs inside a **VPC** or **public context** depending on setup.
- You attach **IAM Roles** to Lambda for permissions:
    - Example: Read from S3, write to DynamoDB.

📊 Monitoring & Logging

- Integrated with **CloudWatch Logs**.
- You can:
    - View logs (System.out.println() appears here)
    - Track number of invocations

- Set alarms

🧠 Benefits

Benefit	Explanation
☁️ Serverless	No need to manage infrastructure
💸 Pay per use	You're billed only for execution time
🚀 Auto-scaling	Instantly handles multiple concurrent calls
🔁 Event-driven	Connect with 200+ AWS triggers
🔒 IAM secured	Control access tightly with roles

❌ Drawbacks

Drawback	Explanation
❄️ Cold Starts	Java has higher startup time (compared to Node.js/Python)
🕐 Timeout Limit	Max 15 minutes per execution
🧱 Stateless	No session or persistent memory between invocations
🔧 Debugging	Can be harder to debug than regular apps

🤹 Real-Life Use Case

Image Processing:

- A user uploads a photo to **S3**.
- **S3 triggers a Lambda** function written in Java.
- Lambda resizes the image and saves it to another S3 folder.

API Backend:

- API Gateway exposes REST /register.
- Triggers Java Lambda.
- Saves user data in DynamoDB.

💉 Interview Questions

1. What is AWS Lambda?
2. How do you deploy a Java-based Lambda?
3. What is a cold start?
4. Can Lambda connect to a database?
5. What are the triggers available for Lambda?
6. How do you monitor Lambda functions?
7. What is the timeout limit for a Lambda?
8. How do IAM roles work with Lambda?
9. Can Lambda be used in VPC?
10. How does Lambda handle concurrency?

🧠 Summary

- **AWS Lambda** allows you to run Java (and other language) code **without managing any server**.
- It's perfect for **event-driven** applications and **microservices**.
- Combine with **API Gateway, S3, DynamoDB** for powerful backend workflows.

✅ Topic 16: Amazon DynamoDB – NoSQL Database for Serverless Applications

---

🔍 What is DynamoDB?

Amazon DynamoDB is a **fully managed NoSQL database** service that provides fast and predictable performance with seamless scalability. It is designed for applications that require **low-latency data access** and is commonly used with **serverless architectures**.

💡 Why Use DynamoDB?

Need	Solution
NoSQL database (key-value/document)	Use DynamoDB
High availability and fault tolerance	Built-in by AWS
Serverless and auto-scalable	No infra or manual scaling
Fast read/write for any scale	Sub-millisecond latency possible

---

⚙️ Core Concepts

Concept	Description
Table	Collection of data (like a SQL table)
Item	Single record (like a row)
Attribute	Field (like a column)
Partition Key	Uniquely identifies each item
Sort Key (optional)	Helps with data sorting within a partition
Primary Key	Partition key (or Partition + Sort key)

📁 Data Structure Example

json

CopyEdit

```
{
 "userId": "U001",      // Partition key
 "email": "john@example.com",
 "name": "John",
 "age": 29
}
```

🧱 Features

Feature	Description
⚡ Fast Performance	<10 ms latency
🌐 Global Tables	Multi-region replication
⏱️ TTL	Automatically expire old data
🔄 Streams	Real-time change data capture
🔐 IAM Integration	Secure access via roles
📊 Query/Scan	Search by key or filter across data

💻 Java SDK Code Sample

java

CopyEdit

```
DynamoDbClient client = DynamoDbClient.create();
PutItemRequest request = PutItemRequest.builder()
  .tableName("Users")
  .item(Map.of(
    "userId", AttributeValue.fromS("U001"),
    "name", AttributeValue.fromS("John")
  ))
```

```
    .build();
client.putItem(request);
```

🔐 Security and Access

Feature	Description
IAM Roles	Attach permissions to allow access
Fine-Grained Access	Row-level access control
Encryption	At rest and in transit

📊 Monitoring and Performance

- Integrated with **CloudWatch**
- Monitor:
    - Read/Write capacity
    - Throttled requests
    - Latency
- Set alarms for high usage or failures

🤹 Real-Life Use Cases

Use Case	How it Helps
E-commerce cart	Store cart data with fast access
IoT data ingestion	Store millions of events per day
User profile storage	Each item represents a user
Gaming leaderboards	Use sort keys for ranking

❌ Drawbacks

Drawback	Explanation
No joins	Only single-table access
Query limitations	Complex queries not supported like SQL joins
Size limits	400 KB per item
Pricing	Can be expensive if not optimized (e.g., provisioned write capacity)

🧠 Benefits

Benefit	Explanation
🚀 Serverless	Auto-scales up and down
🔒 Secure	Integrated with IAM
⚡ Fast	Millisecond performance at scale
📈 Scalable	Supports millions of requests per second
🛠️ Easy to integrate	Works well with Lambda, API Gateway, etc.

🧪 Interview Questions

1. What is DynamoDB?
2. How is DynamoDB different from RDS?
3. What is a Partition Key and Sort Key?
4. What is the maximum size of an item?
5. How do you monitor and scale DynamoDB?
6. What are DynamoDB Streams?
7. How does DynamoDB handle security?
8. Can you perform joins in DynamoDB?
9. What is TTL in DynamoDB?
10. Difference between **Query** and **Scan**?

🧠 Summary

- **DynamoDB** is ideal for **serverless**, **high-speed**, **scalable** data storage.
- Works perfectly with **AWS Lambda**, **API Gateway**, and other AWS services.
- Great for use cases like **user sessions**, **IoT**, **analytics**, and **gaming apps**.