AWS Lambda

1. **AWS Lambda**

1.1 **Overview**

It is a **serverless compute service** that allows us to run code without provisioning or managing servers. We can only upload our code, and AWS will automatically run it in response to events such as HTTP requests, file uploads, or database updates. We pay only for the compute time that we consume — there is no charge when our code is not running. So, we are basically paying for number of requests and execution time.

The servers are fully managed by the cloud provider — meaning they handle provisioning, scaling, security, and isolation. This allows developers to focus on their applications without the overhead of managing infrastructure.

It supports multiple programming languages using runtimes. A runtime provides a language-specific environment that manages the relay of invocation events, context information, and responses between Lambda and our function code. We can choose from the runtimes provided by AWS, or, if our application requires a different environment, we have the flexibility to create custom runtimes.

AWS Lambda runs your code inside small virtual machines (microVMs) created using Firecracker. Each request runs in its own isolated environment for security. AWS automatically runs many copies of your function on different servers to handle load and provide high availability.

Lambda worker servers have a limited lifetime (about 14 hours). When a worker is about to expire, AWS stops sending new requests to it, safely shuts down the running microVMs, and then terminates the worker. AWS continuously monitors this process to keep the service running smoothly.

Firecracker is AWS's technology for running small, fast, and secure virtual machines used by Lambda and containers.

AWS Lambda

1.2 **Lambda Invocation Types**

AWS Lambda supports different ways to run functions based on application needs. An invocation happens every time AWS Lambda runs your function in response to an event. Lambda functions can be invoked in two primary ways, synchronously or asynchronously. Each invocation type is optimized for performance and scalability.

- Synchronous Invocation:
  - Used when an **immediate response is needed**.
  - Common for APIs and web applications.
  - Example: API Gateway calls Lambda → Lambda queries a database → returns results to user.
- Asynchronous Invocation:
  - Used when **no immediate response is required**.
  - The event is placed in an internal queue and processed later by Lambda.
  - Example: File uploaded to S3 triggers Lambda to process it.
- Event Source Mapping
  - Used for **streaming and continuous data sources** like Kinesis or DynamoDB Streams.
  - Lambda polls the source and processes data in batches.
  - Example: New records in DynamoDB Stream trigger Lambda.

At the heart of each Lambda invocation is the Frontend Service, which is the entry point for Lambda functions. When a Lambda function is invoked, the Frontend Service manages the request and directs it to the appropriate data plane services, initiating the execution process.

Synchronous Invocation: In synchronous invocations, the Frontend Service directly routes the request to a MicroVM for immediate processing.

Asynchronous Invocation: For asynchronous invocations, the Frontend Service places the request into an internal queue within Lambda. This internal queuing mechanism efficiently handles the distribution of queued events to available MicroVMs. The queue enables Lambda to maintain a balanced load and optimize performance, particularly during high-traffic periods or demand spikes, by ensuring smooth event distribution.

AWS Lambda

1.3 **Key Features**

- No server management

- Auto-scaling

- High availability

- Supports multiple languages (Python, Node.js, Java, Go, etc.)

- Integrated with many AWS services like SNS and Step Function.

1.4 **Architectural Workflow**

- Example: If a user is uploading a file.

- The user uploads a file to S3.

- S3 triggers Lambda function.

- Lambda processes the file.

- Output stored in database or another S3 bucket.

- **Flow:** Client → S3 → Lambda → Database / Storage

1.5 **Common Use-Cases**

- Web and Mobile Backend - Handle login, signup, and user profile logic. Also process form submissions.

AWS Lambda

- File & Media Processing - Image thumbnail generation. PDF or document processing.

- Chatbots & AI-Integration – Backend for Chatbots. Also call AI models.

- Database Operations – Automatically update records. Validate DB entries.

- DevOps & Infrastructure Automation – Start/Stop EC2 Instances, Auto-tag resources and monitor resource usage.

- Gaming Backends - Player Authentication, Matchmaking logic, Event rewards.

- Big Data & Analytics – Trigger data pipeline jobs, generate dashboards, pre-process data for Athena.

1.6 **Execution steps**

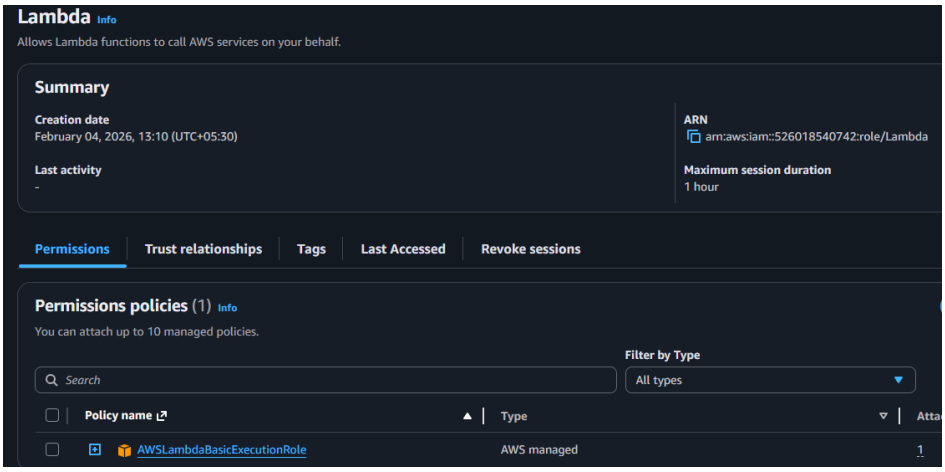
- Step 1: Create an IAM Role.

- Go to IAM → Roles → Create a role.

- Choose: Lambda.

- Attach policy: `AWSLambdaBasicExecutionRole.`

- Create a role.



- Step 2: Create Lambda Function.

- Go to AWS Lambda → Create function.

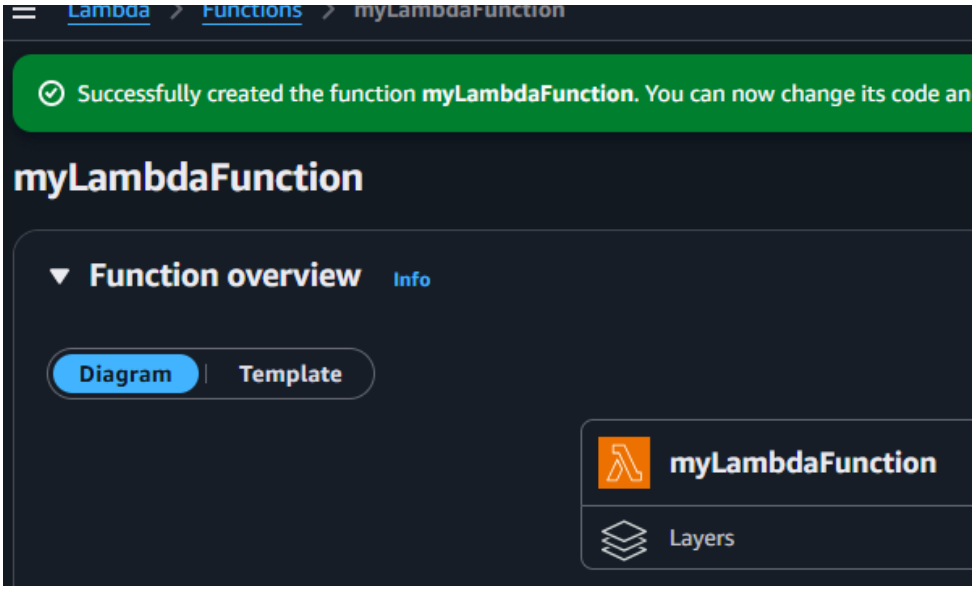
AWS Lambda

- Choose: Author from scratch.

- Function name: `myLambdaFunction.`

- Runtime: Python 3.12 (or Node.js).

- Role: Choose existing role.



- Step 3: Add the Code.

```
def lambda_handler(event, context):
   return {
      "statusCode": 200,
      "body": "Hello from AWS Lambda"
   }
```

AWS Lambda

lambda_function.py

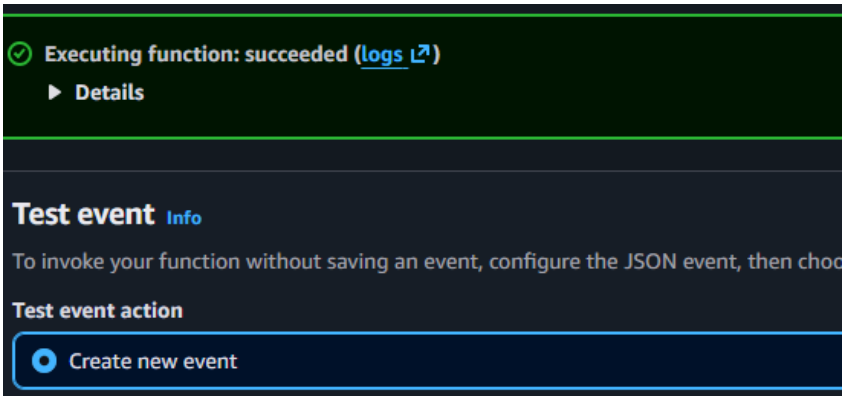
```python
import json

def lambda_handler(event, context):
    return {
        "statusCode": 200,
        "body": "Hello from AWS Lambda"
    }

```

- Step 4: Test

- Click Test.

- Create a test event.

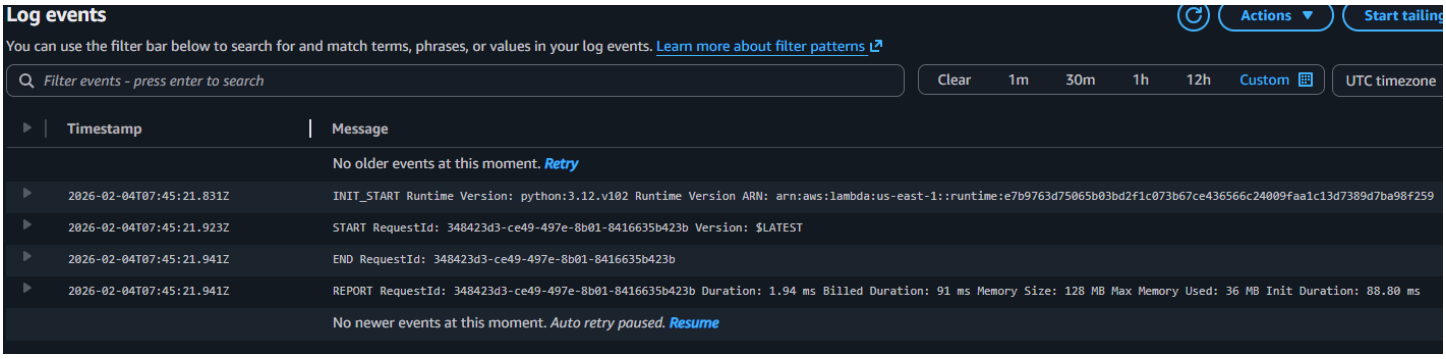
- Execute and verify the output.



- Step 5: Monitor

- Use CloudWatch Logs to view logs.

- Metrics: Invocations, Errors, Duration.



AWS Lambda

1.7 **Security and Best-Practices**

- Enable logging and monitoring: Track errors and behavior using CloudWatch.

- Set memory and timeout properly: Prevent runaway executions and control costs.

- Least-privilege IAM roles: Give Lambda only the permissions it actually needs.

- Small, focused functions: Keep each Lambda doing one task to improve security and maintainability.