AWS Lambda Service

AWS Lambda is a Serverless compute Service that lets you run code without provisioning or managing servers. You simply upload your code, and Lambda takes care of everything required to run and scale it.

→ Only charging for the time your code runs.

Event-Driven Executions

- Lambda is an event-driven Service, meaning that it runs your code in response to certain triggers or events.
- These events can come from many different AWS Services like:
  - S3 (File uploads),
  - DynamoDB (database changes),
  - API Gateway (HTTP requests),
  - CloudWatch (Scheduled events) etc.

Key features

1. Event driven :- Executes automatically in response to trigger (S3 uploads, API calls etc).

2. Stateless :- Each function invocation is dependent. (means it doesn't remember anything about what happened in previous run.

3. Automatic Scaling :- Scales based on the number of events.

4. Multi-language :- Supports Python, NodeJs, Java, Go, C#, Ruby etc.

5. Short-lived tasks :- Max execution time : 15 minutes per invocation.

Pricing of AWS Lambda Service

- Free Tier : 1M requests & 400,000 GB-second per month.
- After free Tier :
  - $0.20 per 1 million requests.
  - $0.00001667 per GB-second

Steps-by-Steps (Basic Flow)

1. **Create a Lambda function**
   - Go to AWS console → lamba → Create function
   - Choose runtime (eg. Nodejs, python).
   - Author from scratch or zip/container

2. **Add a Trigger :-**
   - Choose an event source (eg. S3, API Gateway)

3. **Write code inline or upload :-**
   - Use the online Editer or upload .zip or container image from ECR (Elastic container Registry)

4. **Set permissions :**
   - USE IAM role to define what service Lambda can access.

5. **Test function :**
   - Use test events or real world triggers.

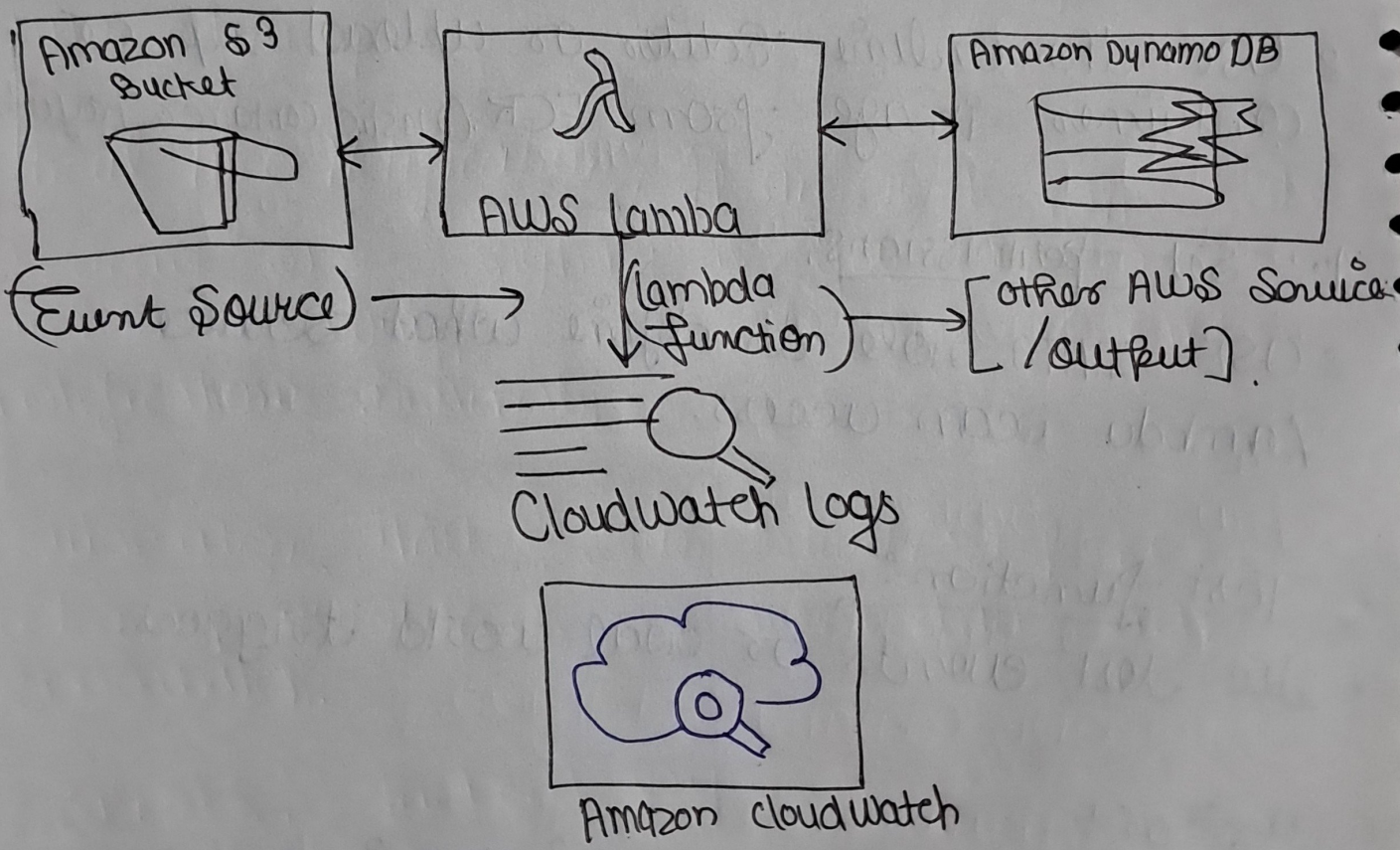
6. **Monitor :-**
   - Use CloudWatch for logs & logs & performances metrixs.

Note :- No need to add destination unless you want post processing routing.

Eg:- If in future you want to route lambda result to another service, then you can configure a lambda Destination. (Success / failure path).

→ But its' not need in the basic / most-used flow.

Lambda Architecture



≈ Event Source :- S3, Api Gateway, DynamoDB, Kinesis, CloudWatch Events, SNS etc.

≈ Lambda function :- Your code in a zip or container format.

- Destination :- Response to your or trigger other AWS Service.

AWS Lambda Limitation

1. Execution Time Limit :- Lambda functions can only run for a maximum of 15 minutes. If you need longer-running tasks, Lambda might not be the best choice.

2. Stateless :- Lambda function don't keep state b/w invocations, so they're best for tasks that don't require long-term memory.

3. Cold Start Delays :- If a Lambda function hasn't run in a while, there's sometimes a slight delay - called a 'cold-start' - when it startup. This can add a little latency, but AWS provides ways to mitigate it for critical functions

Use Cases

1. Image processing :- Lets' say users upload images to your app. you can set up lambda to automatically resize, compress, or even apply filters to each image as its uploaded."

2. Data Transformation :- If you need to clean up or process data before going storing it in a database, a lambda function can handle that transformation automatically.

3. Real Time Notification :- If an event happens - like a new user signing-up - you can use lambda to trigger an email, SMS, or other notifications instantly.

4. Automation Tasks :- Auto Start / Stop EC2, cleanup resources, scheduled tasks with Cloudwatch.

5. Chatbots :- Backend logic for Alexa or custom bots.

6. Streaming Analytics :- Process real-time data with Kinesis or DynamoDB Streams.

Automatic Scaling

- AWS Lamda automatically scales the execution of functions in response to the number of incoming requests.
- If a thousand requests come in at a same time, AWS Lamda will handle them in parallel, making it highly scalable without any configuration.

Pay-as-You Go feature

- Lambda uses a pay-as-you-go picking pricing model. You are billed based on the number of function executions and the duration of each function's runtime (measured in milisecond).

→ This is cost-effective as you only pay for the compute time that you use, and there's no charge for idel time.