AWS Lambda & Serverless Computing 🚀

🚀 What is Serverless?

Serverless doesn't mean there are no servers. It means **you don't manage** the servers!
Instead, the cloud provider (like AWS) handles the infrastructure so you can focus on writing code.

- You focus on writing code, while the cloud provider (AWS) handles infrastructure.
- No need to provision, scale, or maintain servers.
- Pay-per-use billing (you pay only for execution time).

☁️ Serverless in AWS (Popular Services)

Here are the main AWS services that support serverless architecture:

AWS Service	What It Does
🧠 **AWS Lambda**	Run code without managing servers
🌐 **Amazon API Gateway**	Create & manage RESTful APIs for serverless apps
🗃️ **Amazon DynamoDB**	NoSQL database that scales automatically
📦 **AWS Fargate**	Run containers without managing EC2
🗂️ **Amazon S3**	Serverless object storage
🔁 **Amazon EventBridge / CloudWatch Events**	Trigger functions on events
📨 **Amazon SNS / SQS**	Serverless messaging and notifications
🔐 **AWS Cognito**	Serverless user authentication and access control

🧠 AWS Lambda Overview

🚀 What is AWS Lambda?

AWS Lambda is a **serverless compute service**. It lets you run your code **without managing servers**. You just upload your code, and AWS runs it automatically **in response to events** — like an API call, file upload to S3, or a change in DynamoDB.

You only pay when your code is running — by **milliseconds**, not hours!

✅ Benefits of AWS Lambda

- ⚙️ **No server management** – You don't need to launch or maintain EC2 instances or patch operating systems.
- 📈 **Automatic scaling** – Lambda can handle one request or a million without you configuring anything.
- ⚡ **Event-driven** – Lambda can automatically trigger on events from S3, API Gateway, DynamoDB, CloudWatch, and more.
- 🚀 **Fast and easy to deploy** – Just write your code, upload, and it's ready to run.

🔹 Language Support

AWS Lambda supports:

- Node.js, Python, Java, C#, Go, Ruby, PowerShell
- Custom Runtimes (Bring your own language via Docker)

❓ Why AWS Lambda Instead of EC2?

Using EC2 means you manage the **whole virtual server**. You install the OS, handle patches, manage auto scaling, and monitor everything. You're charged **per second or hour**, even when your app is idle.

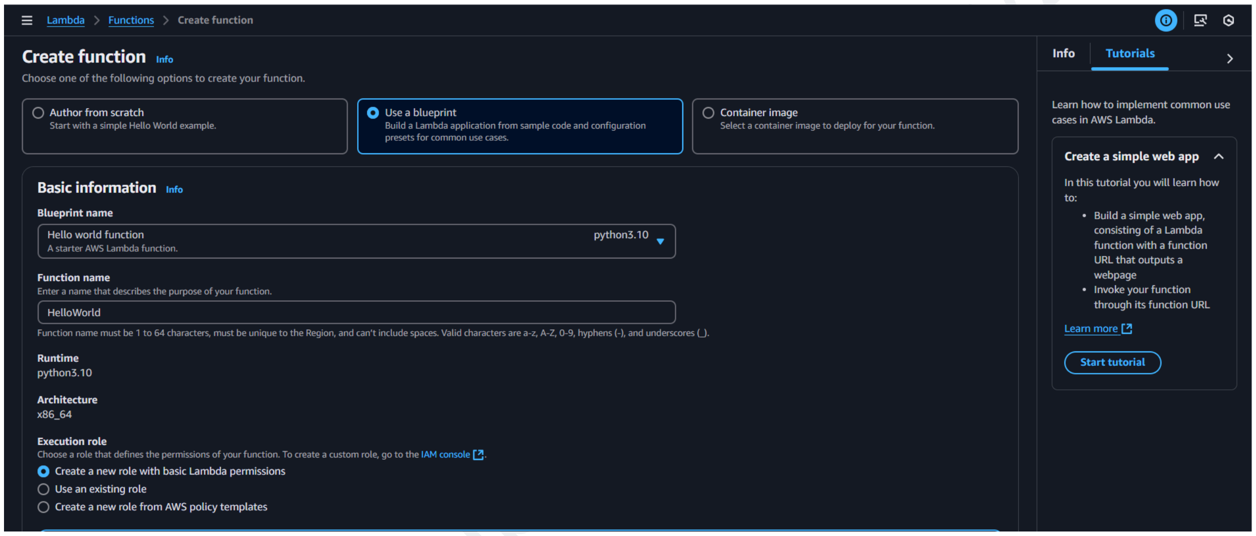
Lambda is different. You only write the **code**, and AWS runs it **automatically** when triggered. There's **no server to manage**, no need to worry about scaling, and you only **pay when your code runs**.

So, if you just need to run short tasks like handling an image upload, processing form data, or reacting to an S3 event — **Lambda is perfect**.

🧪 AWS Lambda Hands-On Guide

✅ Step 1: Create the Function

1. Go to **AWS Lambda** service.
2. Click **Create function**.
3. Select **Use a blueprint**.
4. Pick a **Blueprint name** that matches your use case (like `hello-world` or `s3-get-object`).
5. Give your function a **name**.
6. Click **Create function**.



Create function Info

Choose one of the following options to create your function.

- Author from scratch — Start with a simple Hello World example.
- Use a blueprint — Build a Lambda application from sample code and configuration presets for common use cases.
- Container image — Select a container image to deploy for your function.

Basic information Info

Blueprint name: Hello world function — A starter AWS Lambda function. python3.10

Function name: Enter a name that describes the purpose of your function. HelloWorld

Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

Runtime: python3.10

Architecture: x86_64

Execution role: Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.

- [✓ Create a new role with basic Lambda permissions]
- Use an existing role
- Create a new role from AWS policy templates

Learn how to implement common use cases in AWS Lambda.

Create a simple web app

In this tutorial you will learn how to:

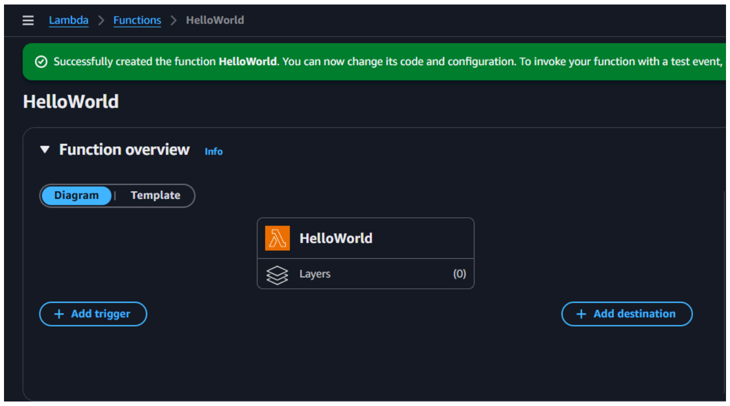
- Build a simple web app, consisting of a Lambda function with a function URL that outputs a webpage
- Invoke your function through its function URL

Learn more

Start tutorial

🔍 Step 2: View Your Function

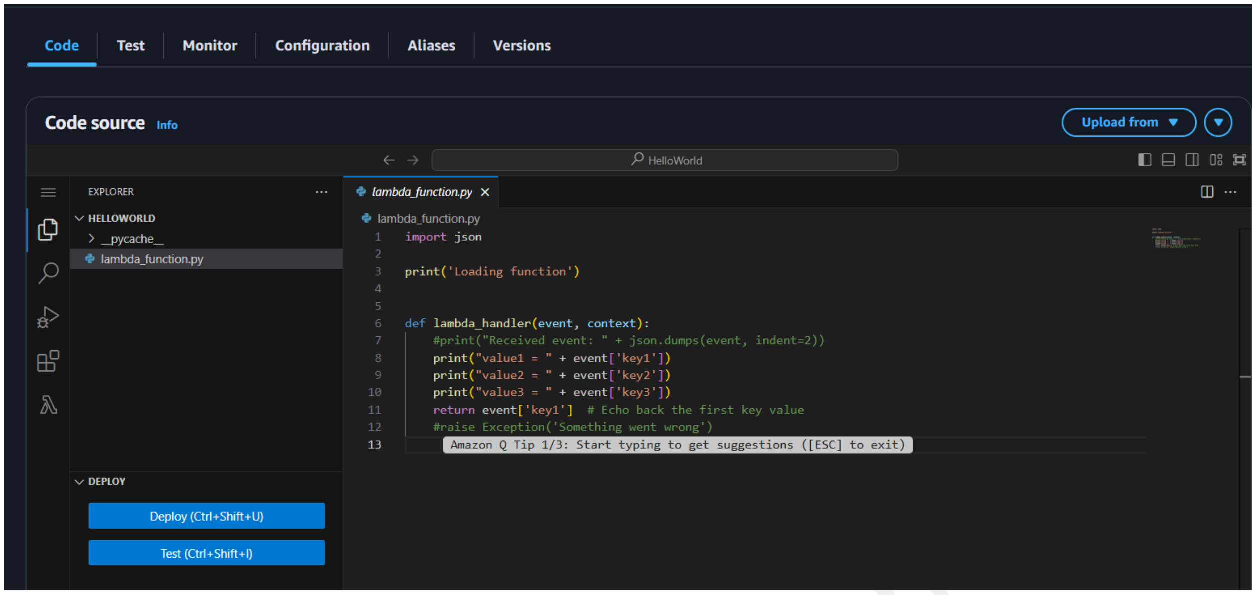
- After creation, you'll see:
  - **Function overview**
  - **Code source editor** to view/edit code.



Successfully created the function **HelloWorld**. You can now change its code and configuration. To invoke your function with a test event,

HelloWorld

Function overview Info



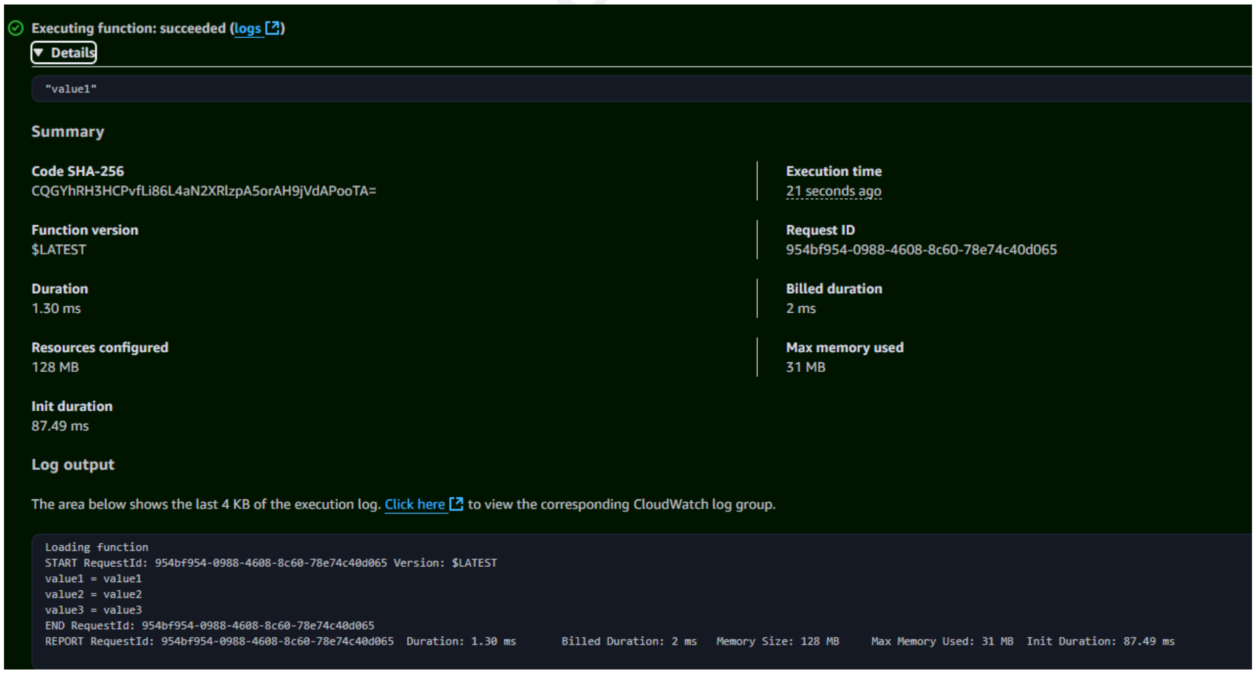
```
import json

print('Loading function')


def lambda_handler(event, context):
    #print("Received event: " + json.dumps(event, indent=2))
    print("value1 = " + event['key1'])
    print("value2 = " + event['key2'])
    print("value3 = " + event['key3'])
    return event['key1']  # Echo back the first key value
    #raise Exception('Something went wrong')
    Amazon Q Tip 1/3: Start typing to get suggestions ([ESC] to exit)
```

🧪 Step 3: Test the Function

1. Click on the **Test** button.
2. Give your test event a **name**.
3. Leave the default test JSON (or customize it if needed).
4. Click **Save**.
5. Then click **Test** again to run it.



Executing function: succeeded (logs)

Details

```
"value1"
```

Summary

Code SHA-256
CQGYhRH3HCPvfLi86L4aN2XRlzpA5orAH9jVdAPooTA=

Execution time
21 seconds ago

Function version
$LATEST

Request ID
954bf954-0988-4608-8c60-78e74c40d065

Duration
1.30 ms

Billed duration
2 ms

Resources configured
128 MB

Max memory used
31 MB

Init duration
87.49 ms

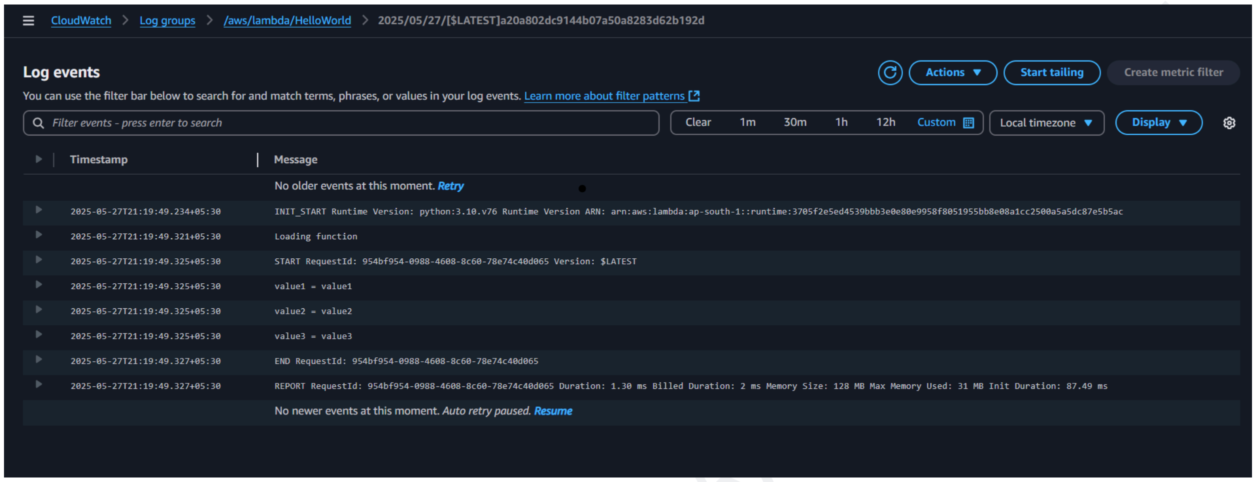
Log output

The area below shows the last 4 KB of the execution log. Click here to view the corresponding CloudWatch log group.

```
Loading function
START RequestId: 954bf954-0988-4608-8c60-78e74c40d065 Version: $LATEST
value1 = value1
value2 = value2
value3 = value3
END RequestId: 954bf954-0988-4608-8c60-78e74c40d065
REPORT RequestId: 954bf954-0988-4608-8c60-78e74c40d065  Duration: 1.30 ms      Billed Duration: 2 ms    Memory Size: 128 MB     Max Memory Used: 31 MB  Init Duration: 87.49 ms
```

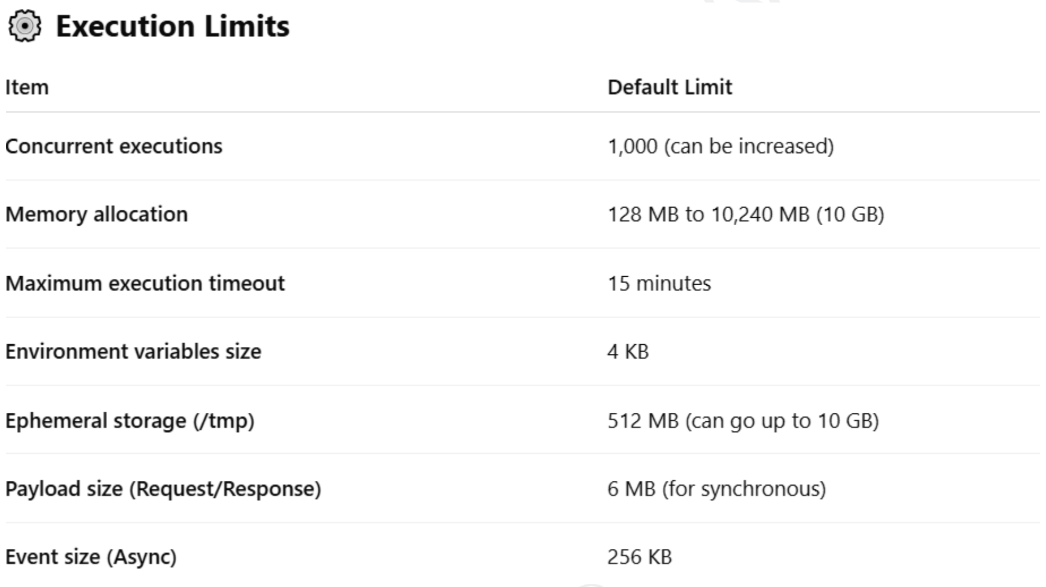
📊 Step 4: Monitor Logs

- Go to the **Monitoring** tab.
- Click **View logs in CloudWatch** to see:
  - Execution details
  - Logs and errors


CloudWatch > Log groups > /aws/lambda/HelloWorld > 2025/05/27/[$LATEST]a20a802dc9144b07a50a8283d62b192d
Log events
Actions
Start tailing
Create metric filter
Filter events - press enter to search
Clear
1m
30m
1h
12h
Custom
Local timezone
Display

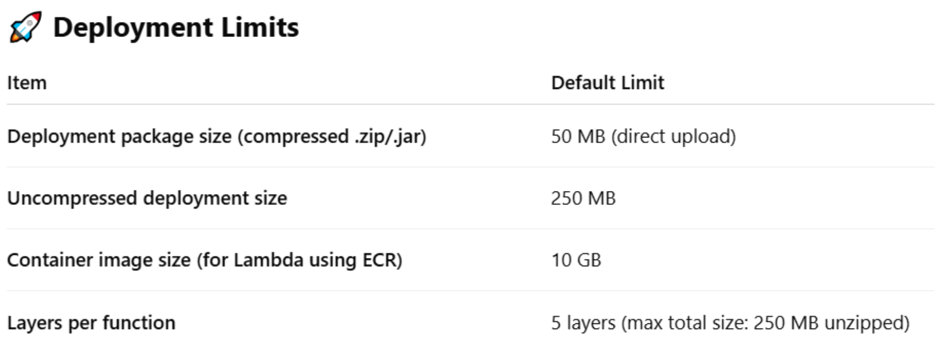
You can use the filter bar below to search for and match terms, phrases, or values in your log events. Learn more about filter patterns

```
Timestamp                        Message
                                 No older events at this moment. Retry
2025-05-27T21:19:49.234+05:30    INIT_START Runtime Version: python:3.10.v76 Runtime Version ARN: arn:aws:lambda:ap-south-1::runtime:3705f2e5ed4539bbb3e0e80e9958f8051955bb8e08a1cc2500a5a5dc87e5b5ac
2025-05-27T21:19:49.321+05:30    Loading function
2025-05-27T21:19:49.325+05:30    START RequestId: 954bf954-0988-4608-8c60-78e74c40d065 Version: $LATEST
2025-05-27T21:19:49.325+05:30    value1 = value1
2025-05-27T21:19:49.325+05:30    value2 = value2
2025-05-27T21:19:49.325+05:30    value3 = value3
2025-05-27T21:19:49.327+05:30    END RequestId: 954bf954-0988-4608-8c60-78e74c40d065
2025-05-27T21:19:49.327+05:30    REPORT RequestId: 954bf954-0988-4608-8c60-78e74c40d065 Duration: 1.30 ms Billed Duration: 2 ms Memory Size: 128 MB Max Memory Used: 31 MB Init Duration: 87.49 ms
                                 No newer events at this moment. Auto retry paused. Resume
```



⚙️ Execution Limits

Item	Default Limit
Concurrent executions	1,000 (can be increased)
Memory allocation	128 MB to 10,240 MB (10 GB)
Maximum execution timeout	15 minutes
Environment variables size	4 KB
Ephemeral storage (/tmp)	512 MB (can go up to 10 GB)
Payload size (Request/Response)	6 MB (for synchronous)
Event size (Async)	256 KB



🚀 Deployment Limits

Item	Default Limit
Deployment package size (compressed .zip/.jar)	50 MB (direct upload)
Uncompressed deployment size	250 MB
Container image size (for Lambda using ECR)	10 GB
Layers per function	5 layers (max total size: 250 MB unzipped)

🔄 What is Concurrency?

Concurrency means:
💡 *How many instances of a function can run at the same time.*

Example:

If 5 users call your Lambda function at the same time ⏱️, and your concurrency is 5, AWS will run 5 **parallel** instances of the function — one for each user ✅.

⚙️ Lambda Concurrency

In AWS Lambda, **Concurrency** =
Number of Lambda executions running at the same time in a region.

Types of Concurrency:

Type	Description
🟢 **Unreserved concurrency**	Shared across all functions in the region (default: 1,000).
🛡️ **Reserved concurrency**	You can reserve a fixed number for a specific function.
🔐 **Provisioned concurrency**	Pre-warms your function (always ready = no cold starts). Great for low-latency needs.

🚫 What is Throttle?

Throttle means:
💥 *Blocking or rejecting extra requests when limits are hit.*

❌ Lambda Throttling

Lambda **throttles** function invocations when:

1. 😵 **Concurrent executions exceed the limit** (e.g., 1,000).
2. 🔐 You **reserved concurrency too low** and it's fully used.
3. 🧊 **Provisioned concurrency is full** and new requests arrive.

What happens when throttled?

- **Synchronous calls** → Get a `429 - TooManyRequestsException` ❌
- **Asynchronous calls** → Automatically retried **2 times** with delays (if retry is enabled) 🔁

🧊💥 What's a Cold Start?

When Lambda needs to spin up a new instance of your function for the **first time**, it:

1. **Downloads your code**
2. **Initializes dependencies**
3. Then runs your code.

This takes time — especially for Java. That delay is called a **cold start**.

⚡ AWS Lambda SnapStart

🔍 What is SnapStart?

SnapStart is an AWS Lambda feature that 📸 **takes a snapshot of your function after it's initialized**, so it can **start much faster** later!

It reduces **cold start time** ⏱️, especially for **Java** functions — which are known for slow startup.

🚀 How SnapStart Works:

1. 📦 During **deployment**, AWS runs your function **once**, initializes it, and **takes a snapshot**.
2. 🧊 When a new instance is needed (cold start), Lambda just **restores from the snapshot** — super fast!

🌐 What is Customization at the Edge?

Customization at the Edge means running code **closer to users**, in **AWS Edge Locations** around the world — instead of in centralized servers or regions.

This helps with:

- ⚡ Faster response times
- 🔐 Early request filtering or redirects
- 🛠️ Customizing content per user/location/device

AWS offers two main services for this:

- ✨ **CloudFront Functions**
- 🧠 **Lambda@Edge**

⚡ CloudFront Functions

CloudFront Functions are **lightweight JavaScript functions** that run **at the edge**, right inside Amazon CloudFront.

✅ Use Cases:

- URL rewrites or redirects
- Header manipulation
- Access control (block bots, check geolocation)
- Simple A/B testing

💡 Key Features:

- Super **fast** (runs in microseconds)
- **Low-cost** and **highly scalable**
- Runs **only at the viewer request/response** stage

🧠 Lambda@Edge

Lambda@Edge lets you run **AWS Lambda functions** at CloudFront edge locations.

✅ Use Cases:

- Dynamic content generation at the edge
- Authentication & authorization
- Device-based content delivery (e.g., mobile vs desktop)
- Modify requests/responses deeply (e.g., cookies, user-agent logic)

💡 Key Features:

- Supports **Node.js and Python**
- Can run at **4 stages**:
    1. Viewer Request
    2. Origin Request
    3. Origin Response
    4. Viewer Response
- Can access the **full request and response**
- More **powerful** than CloudFront Functions, but with **slightly higher latency**

🛡️ Lambda in VPC

By default, **AWS Lambda runs in its own secure environment** — it doesn't have access to your **VPC resources** like RDS databases, EC2 instances, or private subnets.

But you can configure Lambda to **connect to your VPC**, allowing it to access internal resources (like private DBs) securely.

🏗️ Why Put Lambda in a VPC?

You need it when your Lambda function needs to:

- 🔒 Connect to **private subnets**
- 🔗 Access **RDS, ElastiCache**, or **EC2** in a private subnet
- 🔐 Use custom **security groups** or **network ACLs**

🔧 How It Works:

When you configure Lambda to run inside a VPC, it:

1. Uses **Elastic Network Interfaces (ENIs)** to connect to your VPC.
2. Attaches to the **subnets** and **security groups** you specify.
3. Can access private resources — **but loses default internet access** unless you configure a **NAT Gateway** or **NAT instance**!

📌 AWS Certification Exam Tips:

- Lambda can run in a VPC to access **private resources**.
- Once in a VPC, it **loses default internet access** — unless you configure a **NAT Gateway**.
- Lambda uses **ENIs** to connect to your VPC.
- You must provide **subnet IDs** and **security group IDs** during configuration.