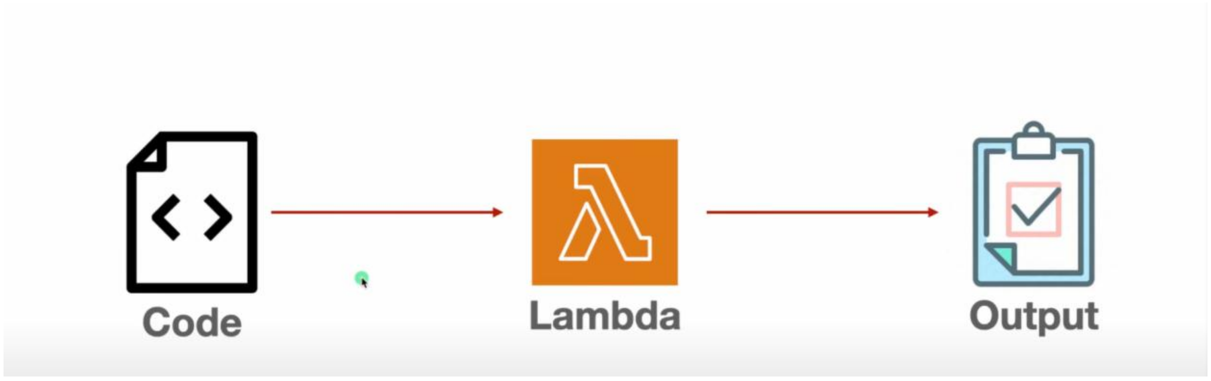
AWS Lambda

1]



Got it 👍 You want **very simple, step-by-step instructions** — starting from scratch to deploying a basic app on **AWS Lambda**. Let's take **Node.js** as example (Python is almost the same).

🚀 Steps to Deploy a Basic App on AWS Lambda

1. Create AWS account

- Go to AWS, sign up, and log in to the AWS Management Console.

---

2. Write your app (Lambda handler)

Make a file index.js:

```
exports.handler = async (event) => {
  return {
    statusCode: 200,
    body: "Hello from Lambda!"
  };
};
```

That's it — one function is enough.

---

3. Zip your code

In your project folder:

```
zip function.zip index.js
```

---

4. Create IAM role for Lambda

Lambda needs permission to log to CloudWatch.

- Go to **IAM → Roles → Create Role**.
- Choose **Lambda** as trusted entity.
- Attach policy **AWSLambdaBasicExecutionRole**.
- Name it lambda-exec-role.

---

5. Create Lambda function

- Go to **AWS Console → Lambda → Create function**.
- Choose **Author from scratch**.
- Name: helloLambda.
- Runtime: Node.js 22.x (or whichever you want).
- Execution Role: choose existing → lambda-exec-role.
- Create function.

---

6. Upload your code

Inside Lambda function page:

- Go to **Code → Upload from → .zip file**.
- Upload function.zip.
- Save.

---

7. Test the function

- In the Lambda console, click **Test**.
- Create a new test event (any name, leave default JSON).
- Run test → you should see Hello from Lambda!.

---

8. Expose via API Gateway (optional if you want a web URL)

- Go to **API Gateway → Create API → HTTP API**.

- Add integration → your Lambda function.
- Create route: GET /hello.
- Deploy → copy the **Invoke URL**.

Now open in browser:

https://<your-api-id>.execute-api.<region>.amazonaws.com/hello

✅ You'll see **Hello from Lambda!**

---

9. Logs

Go to **CloudWatch → Logs → Log groups → /aws/lambda/helloLambda** to see execution logs.

🎯 **Summary**

1. Write index.js with handler.
2. Zip file.
3. Create IAM role (basic execution).
4. Create Lambda function.
5. Upload zip.
6. Test in console.
7. (Optional) Add API Gateway for HTTP endpoint.

What is AWS IAM?

AWS **Identity and Access Management (IAM)** is the service that controls **who can access what** in your AWS account.

Think of it as the **security guard** at the AWS building:

- *Identity* → Who are you? (user, role, app).
- *Access* → What are you allowed to do? (read S3, write DynamoDB, invoke Lambda, etc.).

IAM Authentication = verifying *who* is making the request (user/role/federated), and IAM Authorization = checking *what* that identity can do based on policies.

🔹 What is AWS Lambda?

- Serverless compute service by AWS.
- Runs functions without needing to manage servers.
- Functions are self-contained code written in supported runtimes.
- Executes tasks like APIs, data processing, automation, event handling.
- "Serverless" = AWS manages servers, scaling, patching, etc.

🔹 How AWS Lambda works

- Each function runs inside a container.
- Lambda creates a container → allocates RAM & CPU → runs function.
- Charged = **Memory allocated × Execution time**.
- AWS manages all infra (servers, OS, networking).
- Supports **concurrent executions** (scales up or down automatically).

🔹 Why AWS Lambda in Serverless Architecture?

- Forms **compute layer** in serverless stack.
- Works with:
  - API Gateway (HTTP access)
  - DynamoDB / RDS (database)
  - S3 (storage)
- Supports many popular languages/runtimes.
- Ideal for **event-driven apps, APIs, automation**.

🔹 Common Use Cases

1. **Scalable APIs**
   - API Gateway routes HTTP requests → Lambda executes.
   - Auto-scales per endpoint load.
2. **Data Processing**
   - Integrates with DynamoDB, S3 events.

    - Examples: Notifications, counters, analytics.

3. **Task Automation**
    - Scheduled jobs (cron).
    - Infra cleanup, data transfer, workflow automation.

🔹 **Supported Languages & Runtimes (official)**

- Node.js (8.x, 10.x, 12.x, etc.)
- Python (2.7, 3.6–3.8)
- Ruby 2.5
- Java 8 & 11 (+ JVM languages like Scala, Clojure)
- Go 1.x
- C# (.NET Core 1.0, 2.1)
- PowerShell Core 6.0

📌 Pre-release (via AWS Labs / custom runtimes): **Rust, C++**.

🔹 **Benefits of AWS Lambda**

- **Pay per use** – charged only for compute time + memory + network.
- **Fully managed infra** – AWS handles scaling, patching, infra.
- **Automatic scaling** – creates instances on demand.
- **Tight AWS integration** – works seamlessly with S3, DynamoDB, API Gateway.

🔹 **Limitations of AWS Lambda**

1. **Cold Start Time**
    - Latency (5–10s) if function unused for ~15 mins.
    - Bad for latency-sensitive apps.
2. **Execution Time Limit**
    - Max = 15 minutes per invocation.
3. **Memory Limit**
    - 128 MB → 3008 MB (64 MB steps).

4. **Code Package Size**
    - Max 50 MB (zipped), 250 MB (unzipped).
5. **Concurrency Limit**
    - Default = 1,000 concurrent executions per account.
    - Extra → need to request AWS support.
6. **Payload Size** (API Gateway trigger)
    - Max = 10 MB per request.
7. **Cost Consideration**
    - Cost scales with load.
    - At high workloads, EC2/ECS may be cheaper.
8. **Limited Runtimes**
    - Custom runtimes possible but complex.