AMAZON API GATEWAY

A Hands-On Lab Report on Building a FAQ Microservice with AWS Lambda and Amazon API Gateway

Modern software development is increasingly moving toward serverless architectures and microservices. Instead of maintaining large monolithic applications that are hard to scale and manage, organizations are breaking applications down into smaller, self-contained services that can run independently, scale automatically, and respond quickly to changing demands.

One of the most powerful tools in the AWS ecosystem for implementing this approach is Amazon API Gateway. API Gateway provides a fully managed service for creating, deploying, and maintaining APIs. Combined with AWS Lambda, which allows developers to run code without provisioning or managing servers, API Gateway enables developers to design powerful, flexible, and cost-effective serverless applications.

This lab, i approached as a hands-on exercise to gain practical experience with microservices and serverless development. The central task is to build a simple FAQ microservice that returns a random question-and-answer pair when a user makes a request through an API endpoint. While the use case is intentionally simple, the skills learned are fundamental to real-world cloud development.

In this report, we dive into every aspect of the project: the architecture, the services used, step-by-step execution, debugging, and future extensions. Along the way, we'll explain key concepts such as microservices, RESTful APIs, JSON, monitoring, and best practices for API design.

By the end of this report, you'll have not only a clear understanding of how to build and deploy an API with API Gateway and Lambda but also a strong foundation for building more complex serverless applications in the future.

Lab Overview

This lab simulates the development of a microservice architecture on AWS. The specific service created is a FAQ (Frequently Asked Questions) microservice, which provides users with random question-and-answer pairs stored in a JSON object.

Here's how the lab works step by step:

A user sends a HTTP GET request to an API endpoint hosted by Amazon API Gateway.

API Gateway is the first AWS service to receive the request. It transforms the request into JSON and forwards it through a VPC endpoint to AWS Lambda.

The Lambda function executes, randomly selects a FAQ from the JSON object defined in its code, and returns a JSON-formatted response.

The response flows back through the VPC endpoint to API Gateway, which converts it into an HTTP response and sends it back to the user.

The result is that the user sees a random FAQ in their browser or API client.

Lab Objectives

By completing this lab, you will learn to:

Create an AWS Lambda function.

Deploy and configure Amazon API Gateway endpoints.

Debug API Gateway and Lambda using Amazon CloudWatch logs.

Prerequisites

Familiarity with basic programming concepts.

Completion of an introductory AWS Lambda lab (recommended).

Basic understanding of JSON and APIs.

Duration

The lab takes about 100 minutes to complete, depending on prior familiarity with the AWS Management Console.

Key AWS Services Used

Amazon API Gateway: To receive and route API requests.

AWS Lambda: To process the logic of the microservice.

Amazon CloudWatch: To monitor and debug logs.

AWS Identity and Access Management (IAM): For access control.

Technical Concepts

The Microservices architecture involves breaking an application into a set of small, independent services that communicate via lightweight APIs. Each microservice is responsible for one functionality and can be developed, deployed, and scaled independently.

In this lab, our FAQ service is a microservice: a small, independent service that could be part of a larger system, like a customer support portal.

Benefits of microservices include:

Easier scaling of specific components.

Independent deployments.

Technology flexibility (different services can use different languages or databases).

Better fault isolation.

Serverless computing also allows developers to build and run applications without managing servers. AWS takes care of provisioning, scaling, and managing infrastructure. With AWS Lambda, you only pay for execution time with no idle costs.

Our FAQ microservice is entirely serverless: the API is hosted on API Gateway, the logic runs on Lambda, and monitoring happens via CloudWatch.

Representational State Transfer (REST) is an architectural style for designing APIs. RESTful APIs follow principles such as:

Client-server model: Clients send requests; servers handle responses.

Statelessness: Each request contains all the information needed; no client state is stored on the server.

Uniform interface: APIs follow consistent patterns like GET /questions.

Layered system: APIs can be composed of multiple layers (client → API Gateway → Lambda).

Our FAQ service is a RESTful API with a GET method that returns a JSON response.

JSON is a lightweight format for exchanging data. It's widely used in APIs because it's human-readable and machine-friendly.

Example JSON response from our FAQ microservice:

```
{
  "q": "What is AWS Lambda?",
  "a": "AWS Lambda lets you run code without provisioning or managing servers..."
}
```

Step-by-Step Lab Execution

Let’s walk through the tasks in detail, as performed in the lab.

Task 1: Create a Lambda Function

Step 1.1: Create the Lambda Function

In the AWS Console, search for Lambda.

Choose Create function.

Select Author from scratch.

Configure:

Function name: FAQ

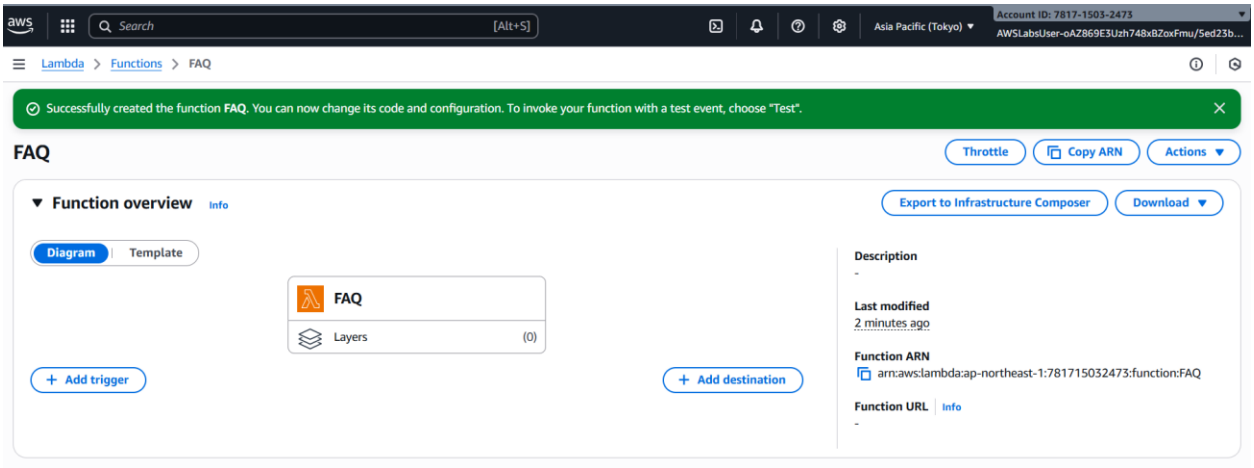
Runtime: Node.js 18.x

Execution role: lambda-basic-execution (pre-created for this lab)

Attach the function to the provided VPC, subnets, and security group.

Click Create function.

This step sets up the Lambda environment, which will host our FAQ logic.



Step 1.2: Add FAQ Code

In the Code tab, I replaced the default code in index.js with the following code snippet.

Key features of the code:

Defines a JSON object with FAQs.

Selects a random FAQ using Math.random().

Returns the selected FAQ as a JSON response.

var json = { "questions": [{...}, {...}, {...}] };

export const handler = function(event, context) {

    var rand = Math.floor(Math.random() * json.questions.length);

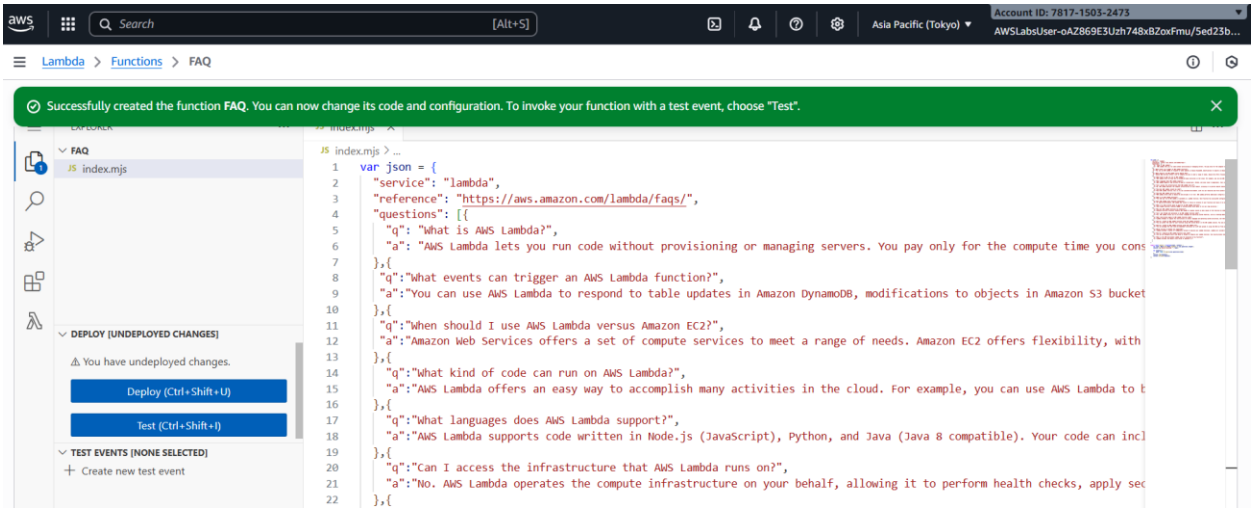
    var response = { body: JSON.stringify(json.questions[rand]) };

    context.succeed(response);

};

After pasting the code, click Deploy.

Result: You now have a working Lambda function that returns a random FAQ when invoked.



Task 2: Create an API Gateway Endpoint

In the Lambda function console, under Function overview, choose Add trigger.

Configure:

Source: API Gateway.

Intent: Create a new API.

API type: REST API.

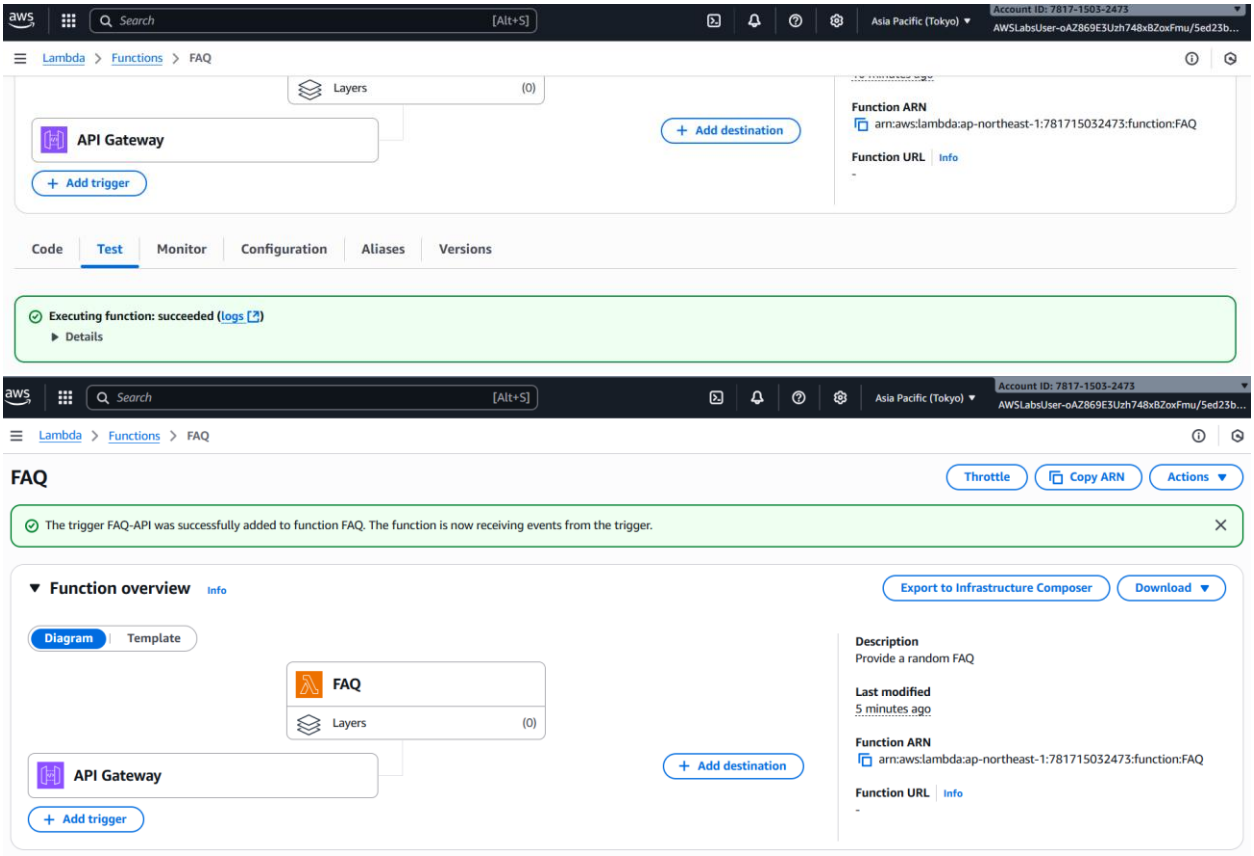
Security: Open (for simplicity in this lab).

API name: FAQ-API.

Deployment stage: myDeployment.

Click Add.

Result: This creates a new API Gateway REST endpoint that invokes the Lambda function whenever it receives a request.



Task 3: Test the Lambda Function

Step 3.1: Test via Browser

In Triggers, locate your API Gateway endpoint.

Copy the URL and paste it into a browser.

Refresh multiple times — each request returns a random FAQ.

Example output:

{

"q": "What languages does AWS Lambda support?",

"a": "AWS Lambda supports code written in Node.js, Python, and Java..."

}

Step 3.2: Test in Lambda Console

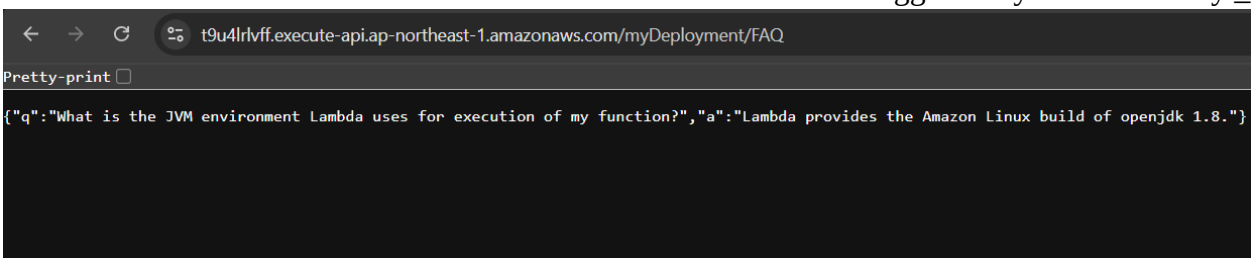
In the Lambda console, open the Test tab.

Create a test event named BasicTest with empty JSON {}.

Run the test.

Check the Execution result and logs.

This confirms the Lambda works both in isolation and when triggered by API Gateway._



Task 4: Debug with CloudWatch

Go to the Monitor tab in Lambda.

Select View logs in CloudWatch.

Open a log stream.

Here you'll see log entries like:

START RequestId: xxx Version: $LATEST

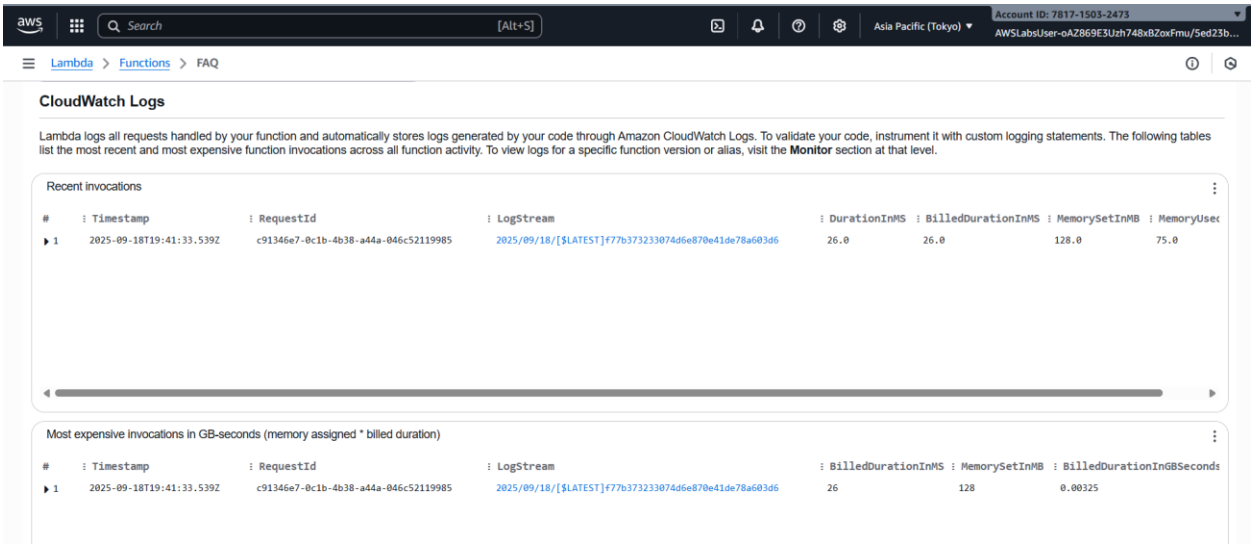
Quote selected: 4

Response: { body: "..." }

END RequestId: xxx

REPORT Duration: 3 ms Billed Duration: 100 ms Memory Size: 128 MB

CloudWatch provides detailed insights into execution times, resource usage, and errors.



5. Deep Dive into AWS Services

5.1 Amazon API Gateway

API Gateway is a managed service to create, publish, maintain, and secure APIs. Key features:

Request and response transformation.

Access control with IAM.

API keys and usage plans.

CloudWatch integration for monitoring.

Caching with Amazon CloudFront.

Multiple stages (dev, test, prod).

Custom domains.

5.2 AWS Lambda

Lambda is AWS's serverless compute engine. Key features:

No servers to manage.

Pay-per-use billing.

Triggers from 200+ services.

Automatic scaling.

Security integration with IAM.

Logging and monitoring.

5.3 Amazon CloudWatch

CloudWatch provides observability for AWS resources. In this lab, it’s used for:

Lambda execution logs.

Debugging API Gateway integration.

Monitoring latency and errors.

6. Use Cases and Extensions

While this lab creates a simple FAQ service, the same pattern can be extended to real-world applications.

Potential Extensions

Store FAQs in Amazon DynamoDB instead of hardcoding.

Add authentication with IAM or Cognito.

Add a POST method to let users contribute FAQs.

Add versioning via multiple stages (dev, prod).

Deploy with frameworks like Serverless Framework or AWS SAM.

Enterprise Use Cases

Customer support APIs.

Product catalogs.

Real-time notification services.

IoT backends.

Chatbot integrations.

7. Best Practices for RESTful APIs

When designing APIs, follow these principles:

Use clear and consistent endpoints (e.g., /questions/17).

Follow HTTP method conventions (GET, POST, PUT, DELETE).

Implement error handling with meaningful status codes.

Use versioning (/v1/questions).

Secure APIs with IAM, Cognito, or API keys.

Enable CloudWatch logging and metrics.

8. Next Steps

I will consider:

Extending the FAQ service with DynamoDB storage.

Adding authentication via IAM or Cognito.

Exploring API Gateway caching and throttling.

Deploying APIs with the Serverless Framework.

Exploring advanced API Gateway features like WebSockets and private APIs.