

# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangaman, Belagavi – 590018



**A Research Internship/ Industry Internship (21INT82) Report on**  
**“Secure Serverless API using AWS API Gateway, Lamda,**  
**WAF and IAM”**

*Submitted in partial fulfillment for the requirements for the award of degree*

*of*

**BACHELOR OF ENGINEERING**

**in**

**COMPUTER SCIENCE AND ENGINEERING**

**By**

**DIVYA LAKSHMI B**

**1EP21CS026**

**Under the guidance of**

**Mrs. Shilpa Patil**

**Asst. Professor,**

**Dept. of CSE, EPCET**



**Department of Computer Science and Engineering**

Approved by AICTE New Delhi | Affiliated to VTU, Belagavi,  
Virgo Nagar, Bengaluru-560049



2024-2025

## CERTIFICATE

This is to certify that the Research Internship/ Industry Internship(21INT82) “**Secure Serverless API using AWS API Gateway, Lamda, WAF and IAM**” is a Bonafide workcarried out by **DIVYA LAKSHMI B [1EP21CS026]** in the partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering of Visvesvaraya Technological University, Belagavi**, during the year **2024-2025**. It is certified that corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The report has been approved as it satisfies the academic requirements in respect of Internship work prescribed for the said degree.

---

**Internal Guide**

**Mrs. Shilpa Patil**

**Asst. Professor,**

**Dept. of CSE, EPCET,**

**Bengaluru**

---

**Signature of the HOD**

**Dr. I. Manimozhi**

**HOD, Dept. of CSE,**

**EPCET, Bengaluru**

---

**Signature of the Principal**

**Dr. Mrityunjaya V Latte**

**Principal,**

**EPCET, Bengaluru**

**Reviewers:**

**Reviewer 1**

**Name:**

**Signature with date:**

**Reviewer 2**

**Name:**

**Signature with date:**

**Examiners:**

**Examiner 1**

**Name:**

**Signature with date:**

**Examiner 2**

**Name:**

**Signature with date:**

## ACKNOWLEDGEMENT

First and foremost, I would like to express our sincere regards and thanks to **Mr. Promod Gowda** and **Mr. Rajiv Gowda**, CEOs East Point Group of Institutions, Bangalore, for providing necessary infrastructure and creating good environment.

I am grateful to **Dr. S Prakash, Senior Vice President, East Point Group of Institutions, Bengaluru** for his conscientious guidance and support that enabled us to complete this Internship.

I would like to express our gratitude to **Dr. Mrityunjaya V Latte, Principal, EPCET** for his constant support in completing the Internship successfully.

I would like to express our sincere thanks to **Dr. I. Manimozhi**, Professor and Head of the Department of Computer Science and Engineering, EPCET for her valuable suggestions and encouragement to do our best in this Internship.

I am grateful to acknowledge the guidance given to us by **Mrs. Shilpa Patil**, Asst. Professor, Department of CSE, Internship Coordinators **Prof. Madhushree**, Assistant Professor, and **Prof. Nithyananda C R**, Associate Professor, Dept. of CSE, EPCET, Bangalore, for rendering a valuable assistance.

I would like to thank **AWS**, Company name for providing with technical support and knowledge.

I would like to extend our thanks to all the faculty members of **CSE department** for their valuable inputs as reviewers during the course of the Internship.

Finally, I would like to thank our parents and friends for their support and encouragement in successful completion of this Internship.

**Divya Lakshmi B [1EP21CS026]**

# **ABSTRACT**

This project focuses on the implementation of a secure, serverless API architecture using Amazon Web Services (AWS). It demonstrates how to deploy a Lambda function integrated with API Gateway, while enforcing access control through AWS Identity and Access Management (IAM) and protecting the API from common threats using AWS Web Application Firewall (WAF). The goal is to provide a scalable, cost-effective, and secure solution for modern cloud-based applications. The project also emphasizes best practices in cloud security by simulating real-time API invocation, IAM-based authentication, and WAF-based traffic filtering, thereby enhancing the reliability and resilience of serverless APIs.

# CONTENTS

<b>Chapter No.</b>	<b>Description</b>	<b>Page No.</b>
<b>1</b>	<b>Company Profile</b>	<b>01</b>
<b>2</b>	<b>About the Company(AWS)</b>	<b>04</b>
<b>3</b>	<b>Task Performed</b>	<b>06</b>
	3.1 Introduction to Cloud Computing	<b>06</b>
	3.1.1 Core components of Cloud Computing	<b>06</b>
	3.2 Introduction to Topic	<b>07</b>
	3.2.1 Key components	<b>08</b>
	3.3 AWS Lamda – Serverless Compute Logic	<b>08</b>
	3.4 Amazon API Gateway – RESTful Interface	<b>09</b>
	3.5 IAM – Identity and Access Management	<b>10</b>
	3.6 AWS WAF – Web Application Firewall	<b>10</b>
	3.7 AWS CLI and SigV4 Script – Secure API Testing	<b>11</b>
<b>4</b>	<b>Reflection</b>	<b>13</b>
	4.1 Project Execution	<b>13</b>
<b>5</b>	<b>Conclusion</b>	<b>31</b>
	<b>References</b>	<b>32</b>

## CHAPTER 1

### COMPANY PROFILE

Amazon Web Services (AWS) is the world's most comprehensive and widely adopted cloud platform, offering over 200 fully featured services from data centers globally. AWS enables individuals, businesses, and governments to access computing power, storage, and a wide range of IT resources on a pay-as-you-go basis.

- Company Name: Amazon Web Services (AWS)
- Headquarters: Seattle, Washington, United States
- Parent Company: Amazon.com, Inc.
- Founded: Officially launched in March 2006
- Founder: Andy Jassy (Former CEO of AWS, now CEO of Amazon)
- CEO (as of 2025): Adam Selipsky

### Global Reach and Infrastructure

AWS operates across a global network of Availability Zones (AZs) within multiple geographic regions including North America, Europe, Asia-Pacific, South America, the Middle East, and Africa. Each region is made up of multiple AZs to ensure high availability, fault tolerance, and disaster recovery. As of 2025, AWS spans over 30 regions and 90+ Availability Zones, making it one of the largest and most reliable cloud infrastructures in the world. AWS data centers are designed for resilience and security. Each region is isolated and designed to be independent, while AZs within a region are connected with low-latency, high-throughput, and redundant networking. This robust infrastructure enables AWS to deliver services with 99.99% uptime and high reliability.

AWS offers over 200 services, categorized into areas like:

- Compute (EC2, Lambda, ECS)
- Storage (S3, EBS, Glacier)
- Databases (RDS, DynamoDB, Redshift)
- Networking & Content Delivery (VPC, Route 53, CloudFront)

- Machine Learning & AI (SageMaker, Rekognition, Comprehend)
- Security & Identity (IAM, KMS, Cognito, WAF)
- DevOps & Monitoring (CloudWatch, CloudTrail, CodeDeploy)
- Developer Tools and Serverless Computing

These services support workloads such as web hosting, mobile application backends, large-scale analytics, Internet of Things (IoT), data lakes, and more.

### **Clientele:**

AWS serves millions of customers globally, including startups, large enterprises, public sector organizations, and government agencies. Some notable clients include Netflix, NASA, Samsung, Adobe, and Airbnb.

### **Revenue and Market Share:**

As of 2023, AWS generated over \$85 billion in annual revenue and holds the largest share of the global cloud market (about 32%). It is a key revenue generator for Amazon, contributing significantly to its operating profit.

### **Key Milestones:**

- 2006: Launch of Amazon S3 and EC2
- 2011: Introduction of AWS GovCloud for U.S. government workloads
- 2014: Launch of AWS Lambda – ushering in serverless computing
- 2015: AWS becomes a \$7B business
- 2020–2023: Major expansion into AI/ML, Quantum Computing, and Sustainability-focused services
- 2021: Andy Jassy becomes CEO of Amazon

## **Workforce:**

As of 2024, Amazon employs over 1.5 million people worldwide, with a significant portion dedicated to AWS operations, development, and innovation. Technology Stack Used Internally:

AWS leverages and builds technologies using:

- Languages: Java, Python, Go, Rust, C++, Node.js
- Infrastructure: Kubernetes, Docker, Terraform
- Monitoring: CloudWatch, X-Ray
- Security: IAM, KMS, CloudTrail
- CI/CD: CodePipeline, Jenkins, GitHub Actions



## CHAPTER 2

### ABOUT COMPANY (AWS)

Amazon Web Services (AWS), a subsidiary of Amazon.com, is the world's leading cloud computing platform offering reliable, scalable, and inexpensive cloud computing services. It was officially launched in March 2006 and is headquartered in Seattle, Washington, USA. AWS was originally created by Andy Jassy, who later became the CEO of Amazon. As of 2025, AWS is led by Adam Selipsky, continuing its legacy of innovation and rapid expansion.

The core idea behind AWS was to allow businesses to rent computing power and infrastructure instead of maintaining costly, on-premises hardware. AWS introduced a new paradigm in IT infrastructure through on-demand, pay-as-you-go pricing and self-service provisioning, revolutionizing how applications are developed and deployed.

AWS has a strong global presence with data centers in over 30 regions and 90+ Availability Zones, offering low latency and high availability across the globe. It has become the backbone for thousands of businesses ranging from startups to large enterprises, public sector organizations, and academic institutions.

#### Service Portfolio

AWS offers a broad and deep set of more than 200 fully featured services, including:

- Compute Services: Amazon EC2, AWS Lambda, Elastic Beanstalk
- Storage Services: Amazon S3, Amazon EBS, Amazon Glacier
- Database Services: Amazon RDS, DynamoDB, Aurora
- Networking & CDN: Amazon VPC, Route 53, CloudFront
- Security & Identity: AWS IAM, AWS KMS, AWS WAF
- Analytics & Machine Learning: SageMaker, Athena, EMR

These services support a variety of use cases such as web applications, mobile backends, data analytics, serverless computing, DevOps automation, and artificial intelligence.

## **Security & Compliance**

Security is at the core of AWS, with services and policies designed to secure data, applications, and infrastructure. AWS follows a shared responsibility model, where AWS secures the infrastructure and customers secure the data they store and manage. To enhance security, AWS provides features like Identity and Access Management (IAM), Key Management Service (KMS), encryption, compliance monitoring, and threat protection using services like AWS WAF and AWS Shield. AWS complies with major industry and international security standards including SOC, ISO, GDPR, HIPAA, and FedRAMP, making it suitable for use in highly regulated environments.

## **Learning & Certification**

AWS also offers a comprehensive set of learning resources and certifications, which are widely recognized in the tech industry. Its certification tracks help professionals become proficient in various cloud roles, ranging from Cloud Practitioner to Solutions Architect, SysOps, and DevOps Engineers.

## **Innovation & Impact**

AWS continues to drive digital transformation for organizations across sectors, including media, healthcare, finance, education, and government. It has played a major role in the rise of remote work, scalable web applications, big data analytics, and AI/ML-driven solutions. Major companies like Netflix, Airbnb, NASA, Spotify, GE, and BMW rely on AWS infrastructure for their mission-critical applications.

## CHAPTER 3

### TASK PERFORMED

#### 3.1 Introduction to the Cloud Computing

Cloud computing has transformed the way organizations build, deploy, and scale applications. Among its most impactful innovations is the emergence of serverless computing, which allows developers to focus solely on writing code without managing the underlying infrastructure. Serverless architecture significantly reduces operational overhead, enables rapid development, and supports cost-efficient scaling, making it ideal for modern, event-driven applications.

Security, however, remains a primary concern in serverless environments. Applications built on services such as AWS Lambda, API Gateway, and Amazon S3 require robust configurations to ensure access control, data protection, and threat mitigation. Securing serverless APIs involves multiple layers, including authentication, authorization, network protections, and logging—all tailored to a stateless execution environment.

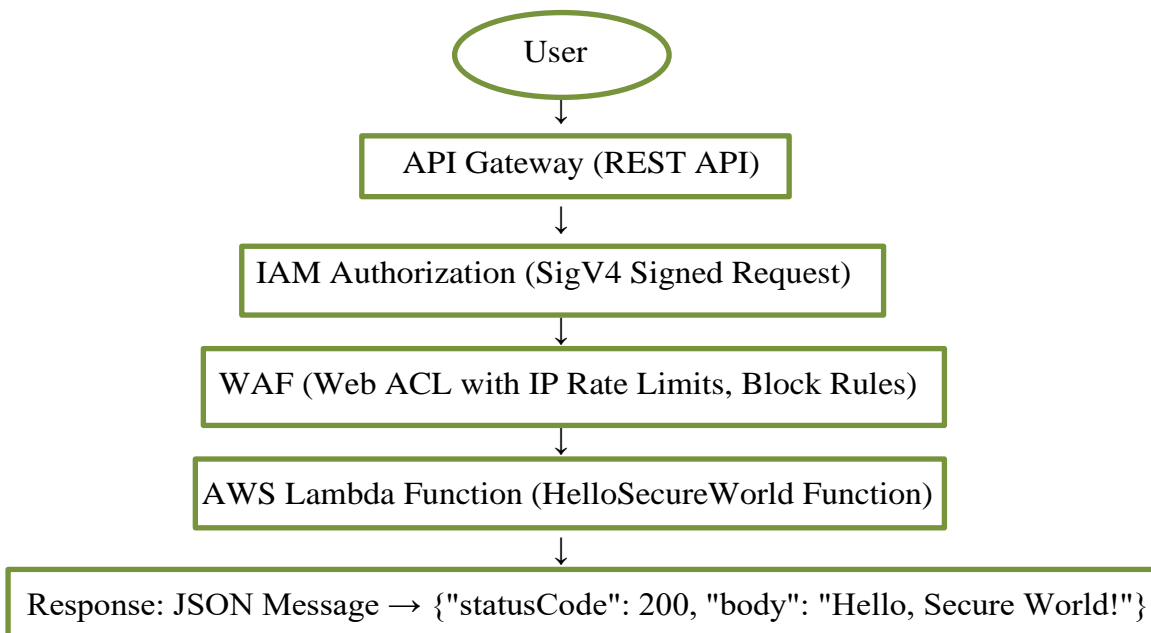
##### 3.1.1 Core Components of Cloud Computing:

- Cloud computing replaces traditional hardware setups with virtual services accessed through the internet.
- It enhances innovation by allowing rapid development, deployment, and scaling of applications.
- Offers a cost-effective model where users pay only for the resources they use.
- Includes robust built-in security features such as encryption, identity and access control, and compliance tools.
- Provides highly scalable infrastructure that adjusts dynamically to meet varying demand.
- Supports multiple service models:
  - Infrastructure as a Service (IaaS) – e.g., Amazon EC2
  - Platform as a Service (PaaS) – e.g., AWS Elastic Beanstalk

- Software as a Service (SaaS) – e.g., Office 365, Google Workspace
- Offers deployment flexibility with Public, Private, Hybrid, and Multi-Cloud options.
- Easily integrates with advanced technologies such as DevOps, machine learning, and data analytics.
- Leading cloud providers include Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP).

### 3.2 Introduction to the Topic

This project focuses on building and securing a Serverless API using AWS Lambda, API Gateway, IAM and WAF . The API is powered by AWS Lambda, which executes backend logic in a stateless compute service. Amazon API Gateway is used to expose this Lambda function as a RESTful API. Security is integrated using AWS Identity and Access Management (IAM) for authentication and AWS WAF (Web Application Firewall) to prevent malicious HTTP requests. By implementing a serverless architecture, the system benefits from auto-scaling, high availability, and cost-efficiency, while security services enforce access control and protect from common web attacks.



Secure Serverless API Architecture

### 3.2.1 Key Components

- **Cloud-Native Architecture:** Utilizes AWS Lambda and API Gateway to eliminate server management.
- **Enhanced Security:** Incorporates IAM for fine-grained access control and WAF for protection against common web threats.
- **Cost-Efficient Deployment:** Takes advantage of the pay-per-use model of serverless services, reducing overall infrastructure costs.
- **Real-World Application:** Reflects practical use-cases such as creating secure APIs for financial services, healthcare platforms, and e-commerce systems.
- **Scalable and Resilient:** Easily handles changes in traffic without manual intervention, ensuring high availability.

## 3.3 AWS Lambda – Serverless Compute Logic

### **Task Performed:**

Created a Lambda function in Python that returns a simple secure response to simulate backend logic.

### **What it Does:**

AWS Lambda allows running code without provisioning or managing servers. It scales automatically and charges only for the compute time used.

### **Advantages:**

- No server management
- Scales with demand
- Pay-per-use model

### **Disadvantages:**

- Cold start latency

- Limited execution time (15 minutes)
- Vendor lock-in to AWS-specific configurations

**Usage Justification:**

Lambda was chosen to keep the backend lightweight and serverless, which aligns with modern application architectures. It also simplifies deployment and maintenance.

### 3.4 Amazon API Gateway – RESTful Interface

**Task Performed:**

Configured a REST API in API Gateway, integrated it with Lambda, and exposed the endpoint /hello via the /dev stage.

**What it Does:**

API Gateway serves as a managed front door to access backend services like Lambda through secure HTTP endpoints.

**Advantages:**

- Fully managed, scalable API management
- Built-in throttling, caching, monitoring
- Supports IAM, Lambda Authorizers, and WAF

**Disadvantages:**

- Can become complex with multiple integrations
- Limited request/response transformation capability

**Usage Justification:**

API Gateway provided a secure and scalable way to expose Lambda logic to external users. It was essential to set up routes and stages cleanly for production-level behavior.

## 3.5 IAM – Identity and Access Management

### Task Performed:

Created a user named secure-api-user, assigned a policy to allow calling API Gateway actions, and used AWS Signature Version 4 to authenticate API requests.

### What it Does:

IAM manages access to AWS services and resources securely through users, roles, and policies.

### Advantages:

- Granular access control
- Supports role-based and user-based permissions
- Integration with all AWS services

### Disadvantages:

- Policy writing can be complex
- Requires careful testing to avoid over-permissive roles

### Usage Justification:

IAM ensured only authenticated users could access the API. Using SigV4 authentication also aligned with best practices for secure programmatic access.

## 3.6 AWS WAF – Web Application Firewall

### Task Performed:

Created a WebACL, defined rules (IP block, rate-based blocking), and attached it to the API Gateway stage.

### What it Does:

AWS WAF protects applications by filtering malicious HTTP requests using customizable rules.

**Advantages:**

- Custom rules for IPs, geolocation, headers, etc.
- Protects from SQLi, XSS, bot attacks
- Integrated logging and monitoring

**Disadvantages:**

- Requires good understanding of request patterns
- Misconfigured rules can block valid users

**Usage Justification:**

WAF added a crucial security layer at the API edge, blocking unwanted traffic and abuse based on IP behavior and request frequency.

### **3.7 AWS CLI and SigV4 Script – Secure API Testing**

**Task Performed:**

Tested API Gateway using Python scripts that generate AWS SigV4-signed HTTP requests with valid IAM credentials.

**What it Does:**

AWS SigV4 signs HTTP requests to ensure integrity and authenticity when interacting with AWS APIs.

**Advantages:**

- Ensures request origin authenticity
- Prevents man-in-the-middle attacks
- Follows AWS best practices for security

**Disadvantages:**

- More complex than standard API key usage



- Requires programmatic handling of credentials and headers

**Usage Justification:**

Using signed requests validated that only authenticated users with valid IAM credentials could successfully invoke the API, even when the endpoint was public.

## CHAPTER 4

# REFLECTION

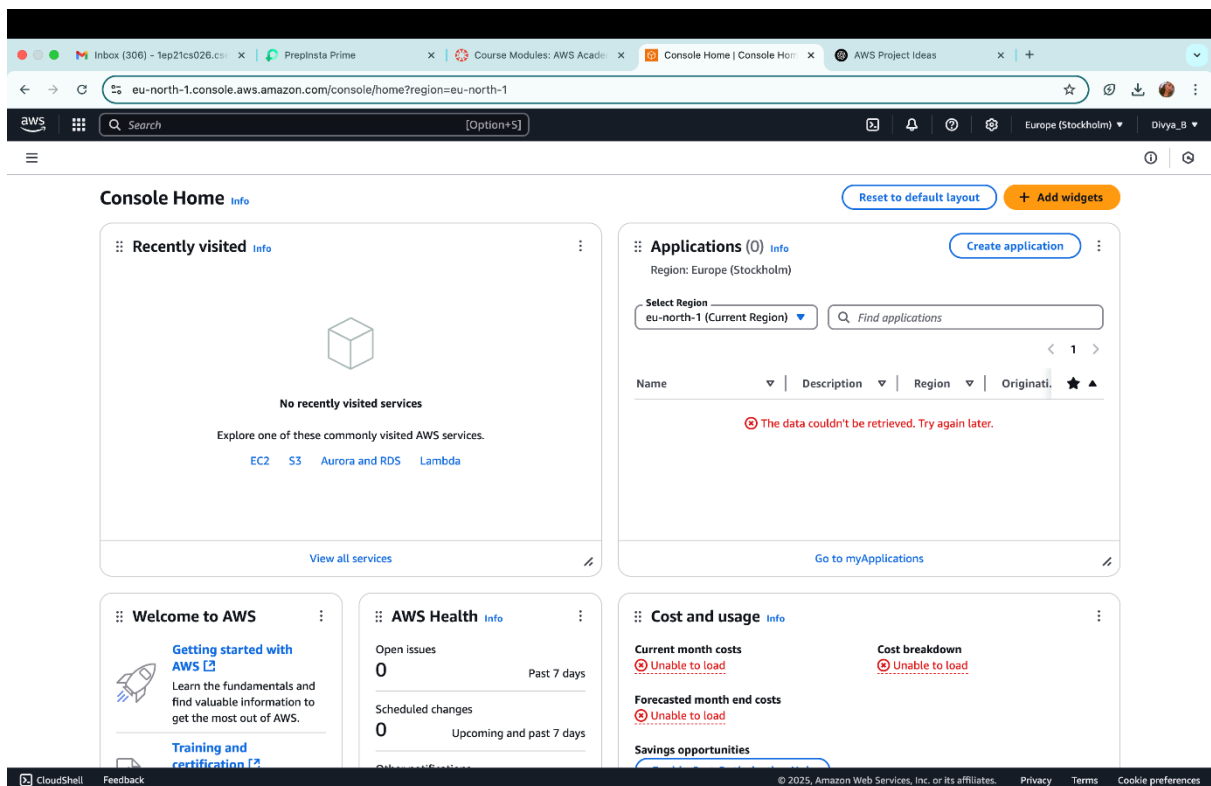
### 4.1 Project Execution

The execution of the Secure Serverless API project was carried out through a systematic series of tasks, leveraging AWS cloud services to build a highly secure, scalable, and fully serverless backend application. This project followed a modular architecture, beginning with the creation of the compute logic using AWS Lambda, exposing the service using API Gateway, and gradually adding advanced security layers such as IAM-based access control and AWS WAF protection. The steps below summarize the full project execution flow:

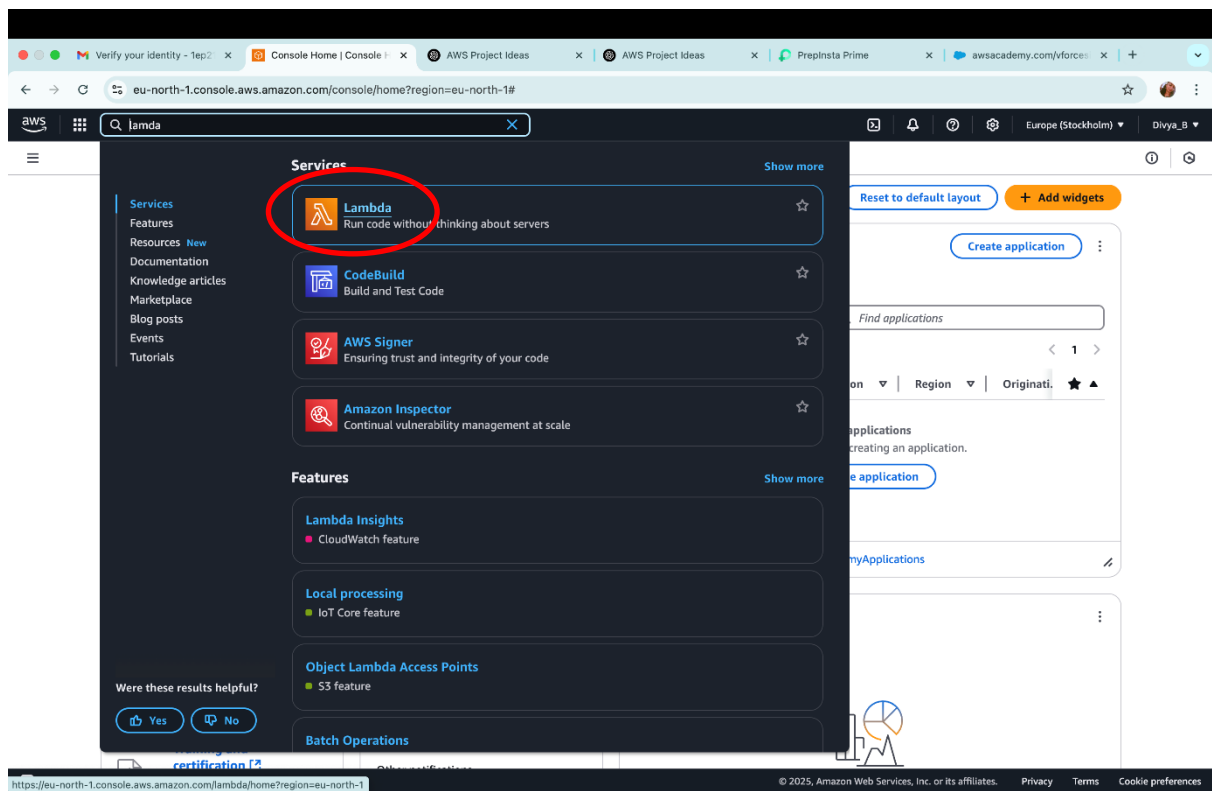
#### Step 1: Lambda Function Development

The first step in the execution was to create the backend logic for the API. This was done using AWS Lambda—a serverless compute service that runs code in response to HTTP triggers.

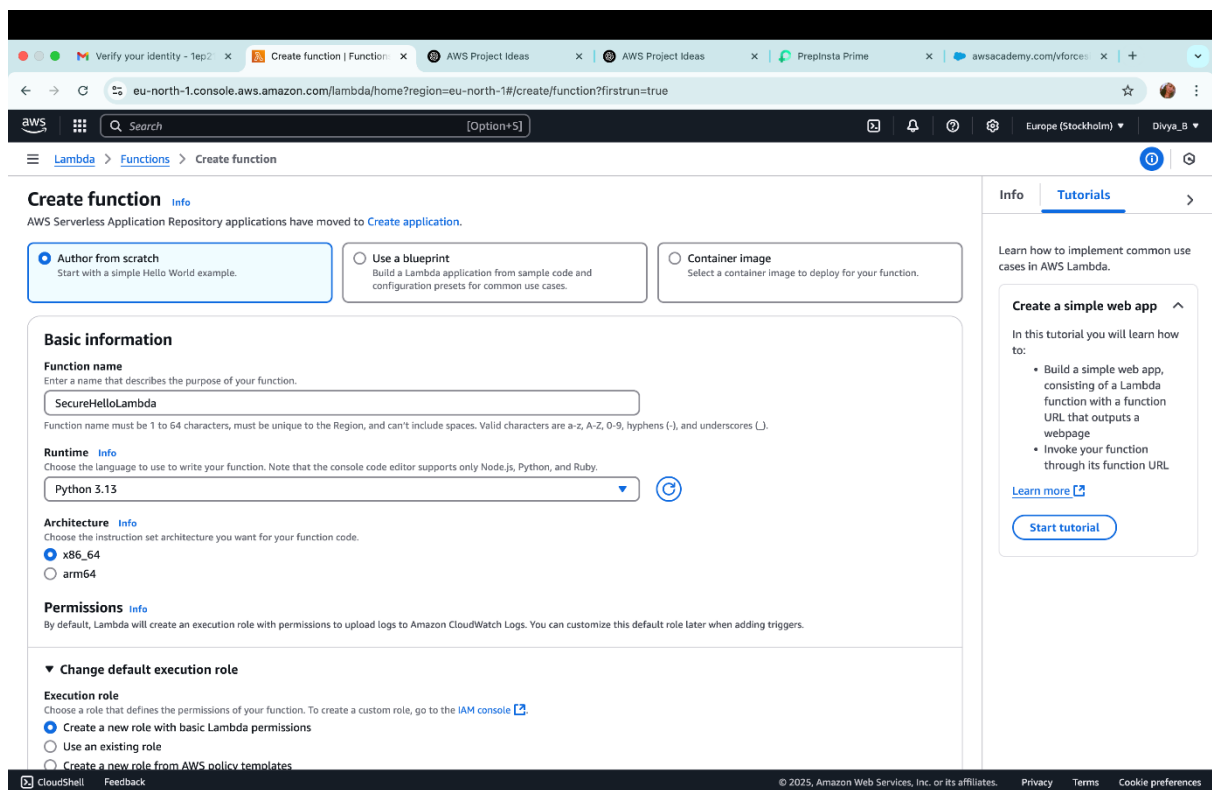
- Go to the AWS Management Console.



- Navigate to **Lambda** > Click **Create function**.



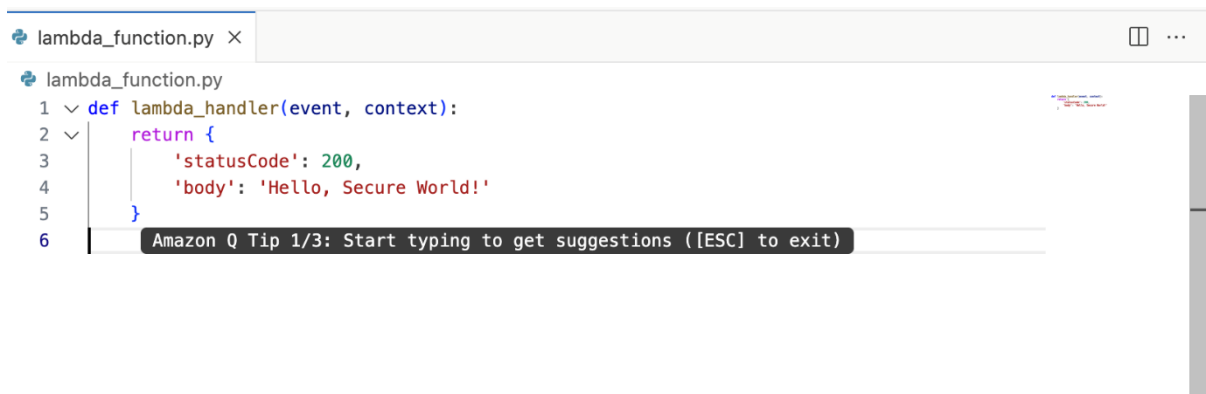
- Choose **Author from scratch**.



- Configure:
  - **Function name:** *secureHelloLambda*
  - **Runtime:** Python 3.12 (or Node.js, based on your implementation)
- Click **Create function**.

Create a function

- In the Function code editor, enter the code:



```
lambda_function.py x
lambda_function.py
1 def lambda_handler(event, context):
2     return {
3         'statusCode': 200,
4         'body': 'Hello, Secure World!'
5     }
6 Amazon Q Tip 1/3: Start typing to get suggestions ([ESC] to exit)
```

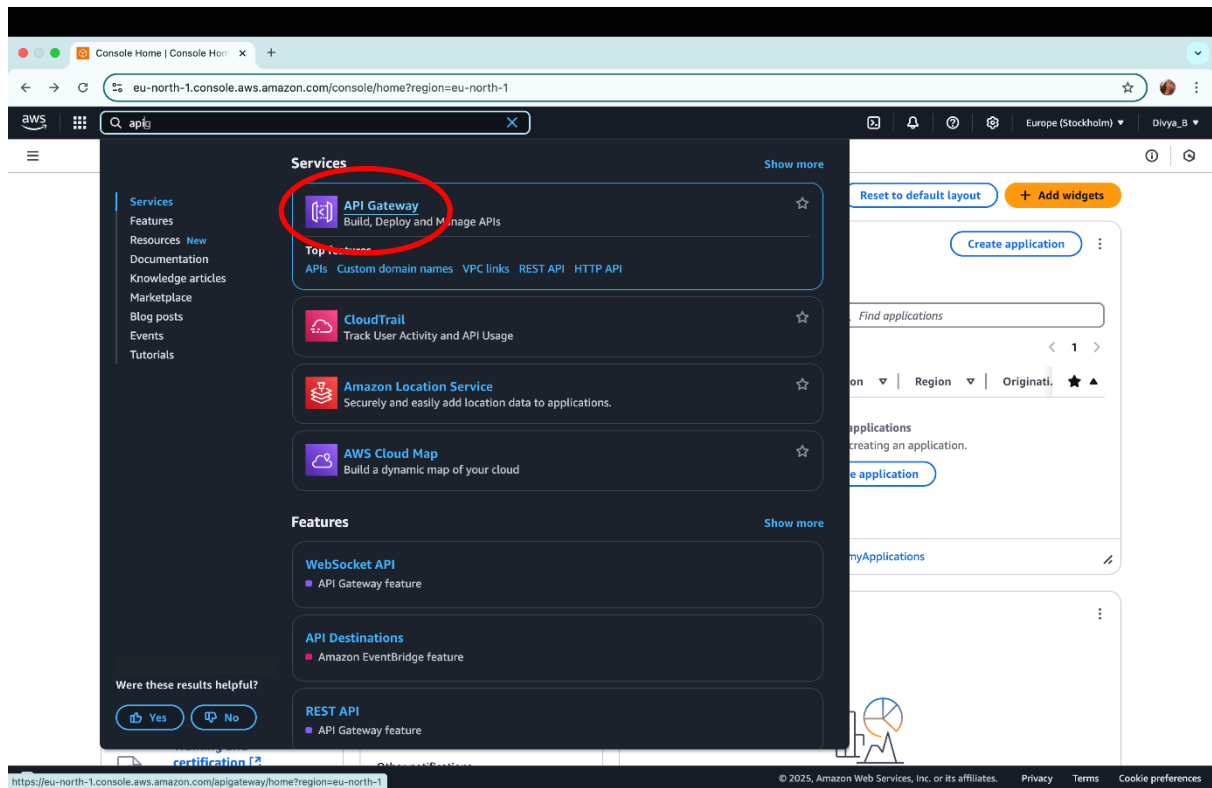
- Click **Deploy** to save the function.

Deploy (⇧⌘U)

## Step 2: API Gateway Setup (REST API)

Once the Lambda function was operational, Amazon API Gateway was configured to expose it via a RESTful endpoint.

- Navigate to **API Gateway** from the AWS Console.



- Choose **Create API** > **REST API** (Build).

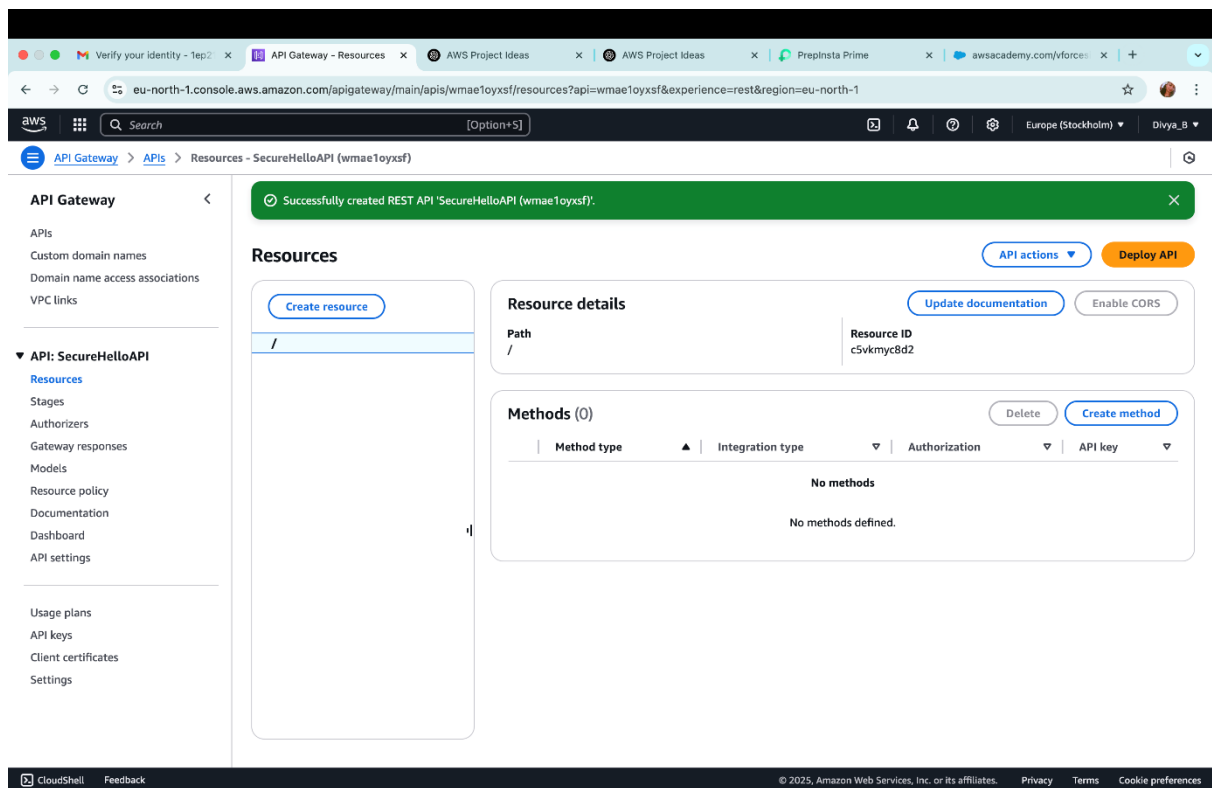
**REST API**

Develop a REST API where you gain complete control over the request and response along with API management capabilities.

**Works with the following:**  
Lambda, HTTP, AWS Services

[Import](#) [Build](#)

- Configure:
  - **API name:** *SecureHelloAPI*
  - Leave other settings as default.
- Click **Create API**.



### Step 3: Create Resource and Method

- Under your API, click Resources.

#### Resources

Stages

Authorizers

Gateway responses

Models

Resource policy

Documentation

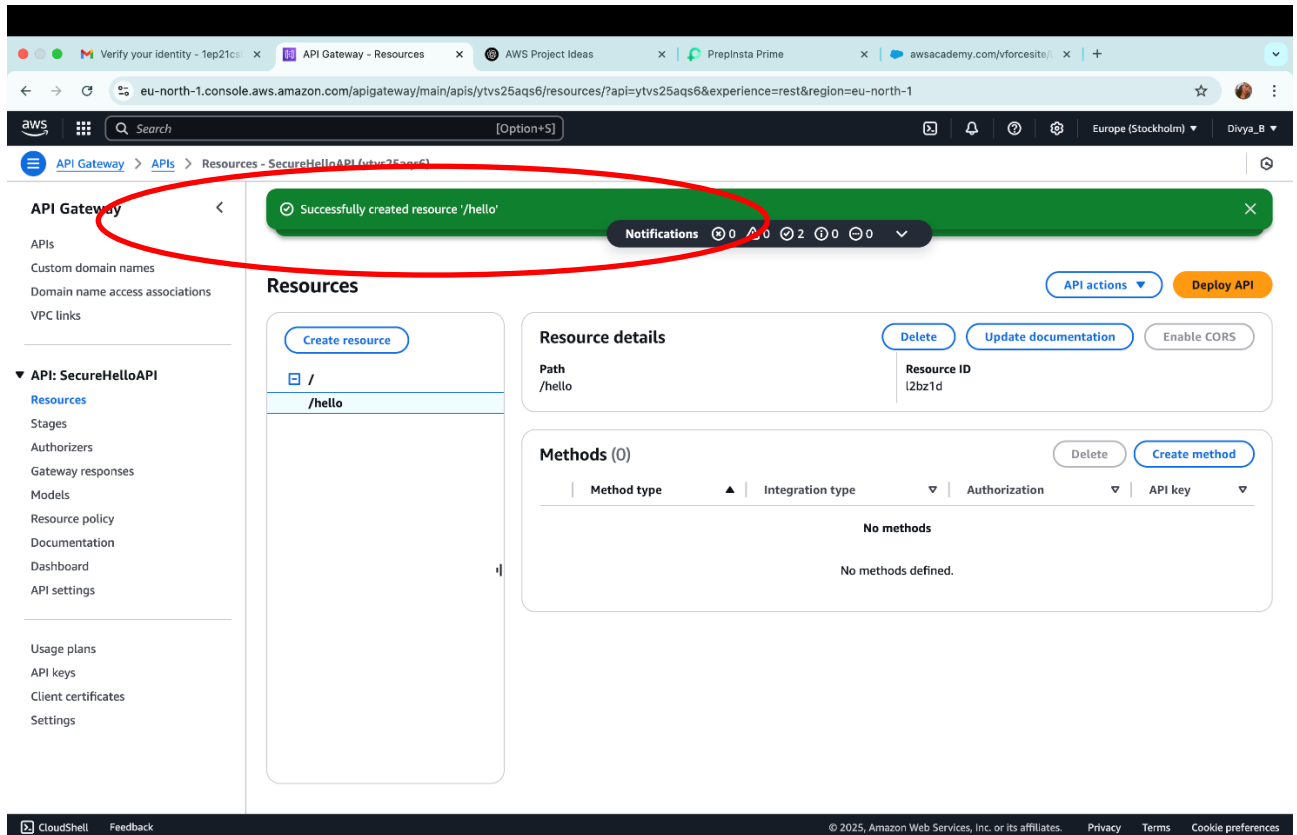
Dashboard

API settings

- Click Actions > Create Resource.

Create resource

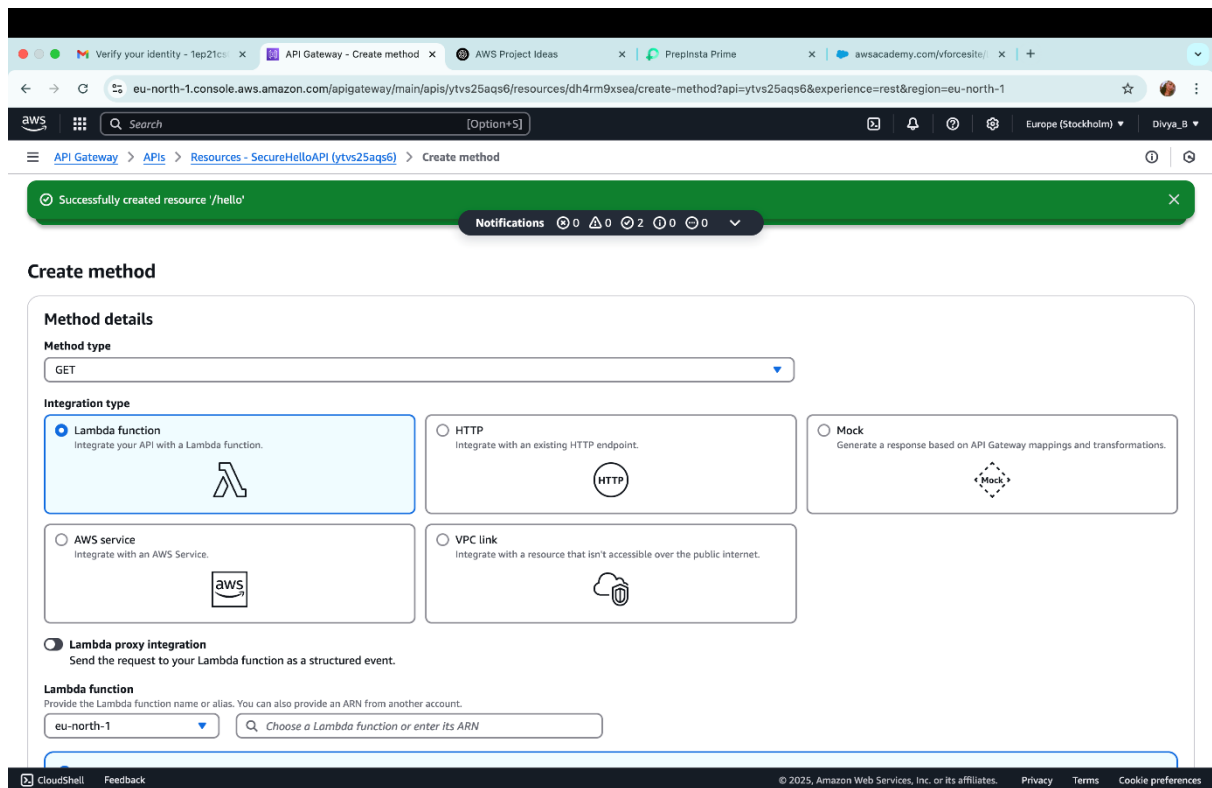
- Configure:
  - Resource Name: hello
  - Resource Path: /hello
- Click Create Resource.
- Select the newly created resource /hello.



- Click Actions > Create Method > Choose GET.

**Create method**

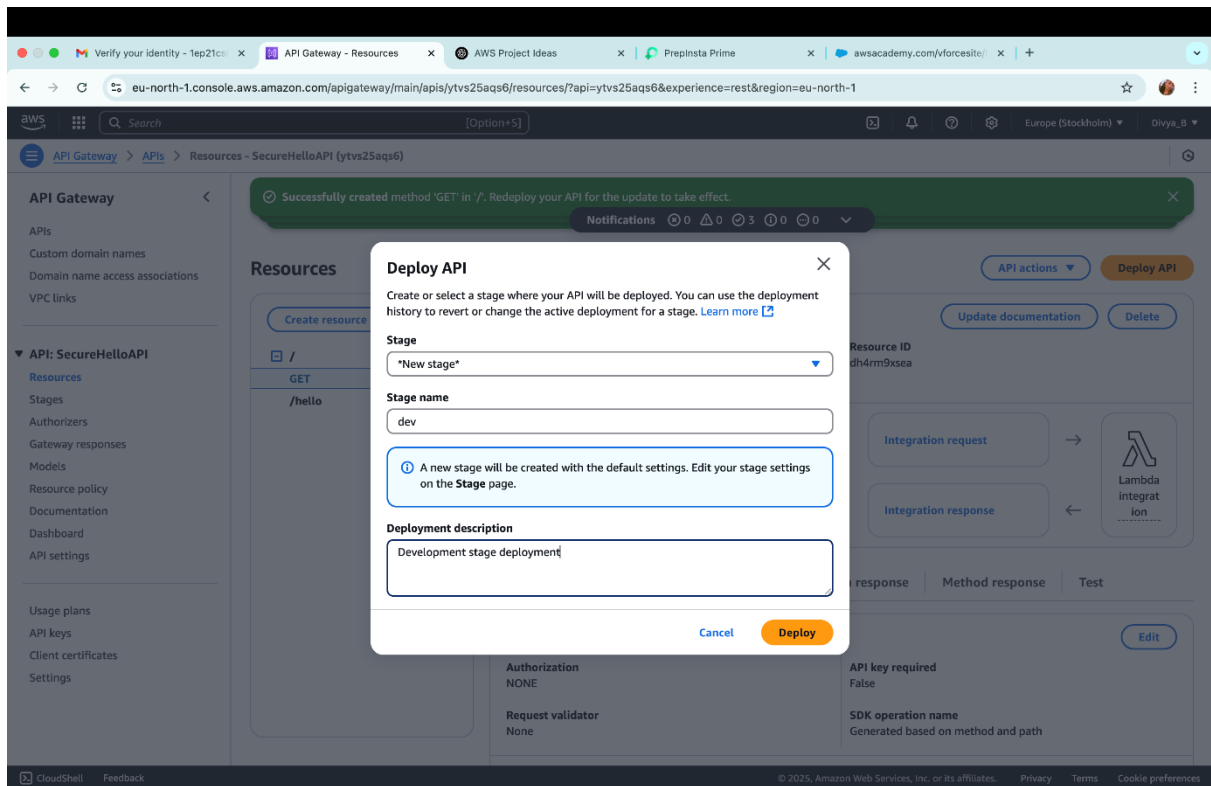
- Integration type: Lambda Function.
- Check Use Lambda Proxy integration.
- Enter Lambda function name: secureHelloLambda.
- Click Save, then OK to grant permissions.



### Step 4: Deploy the API

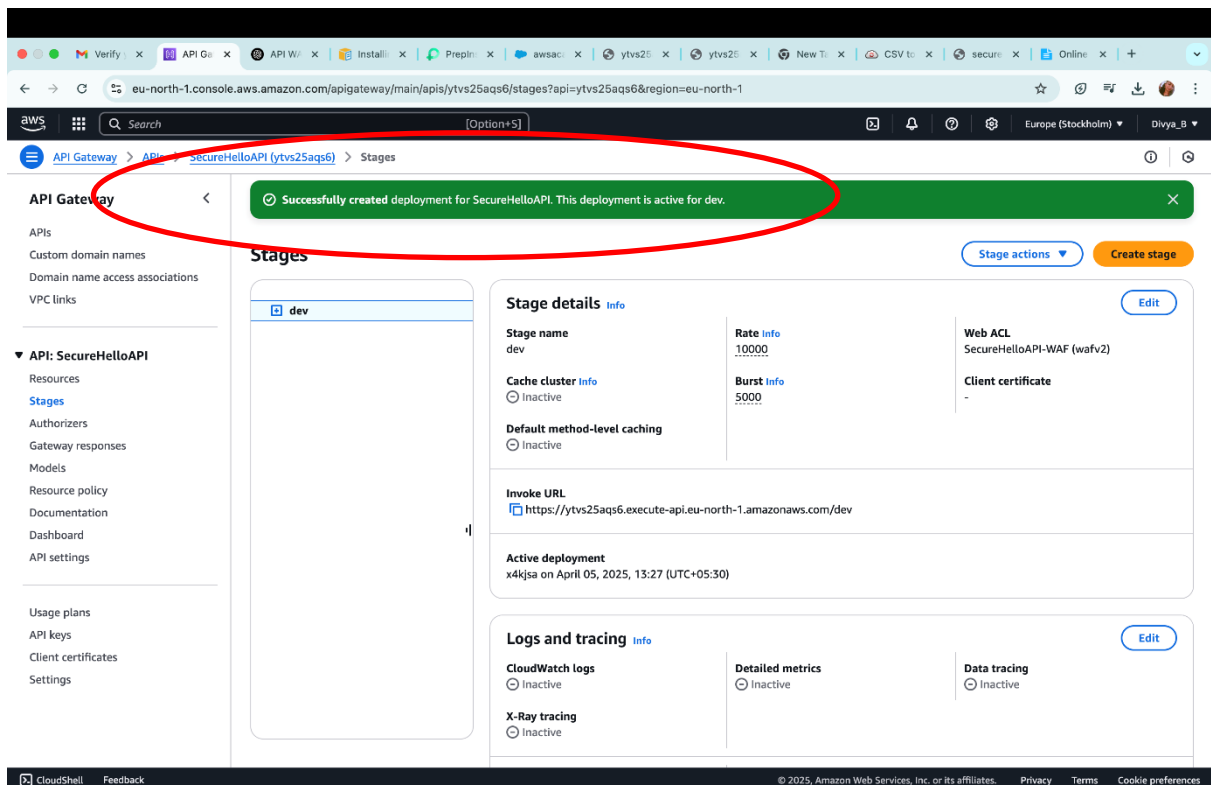
- Click **Actions > Deploy API**.
- Select:
  - **Deployment stage:** [New Stage]
  - **Stage name:** prod
- Click **Deploy**.
- Copy the **Invoke URL** shown at the top.



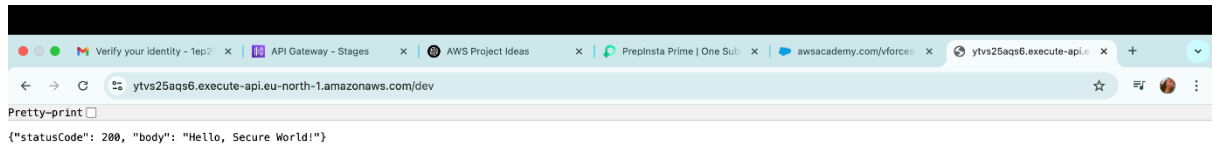


### Step 5: Test the API

1. Open the copied Invoke URL in a browser.

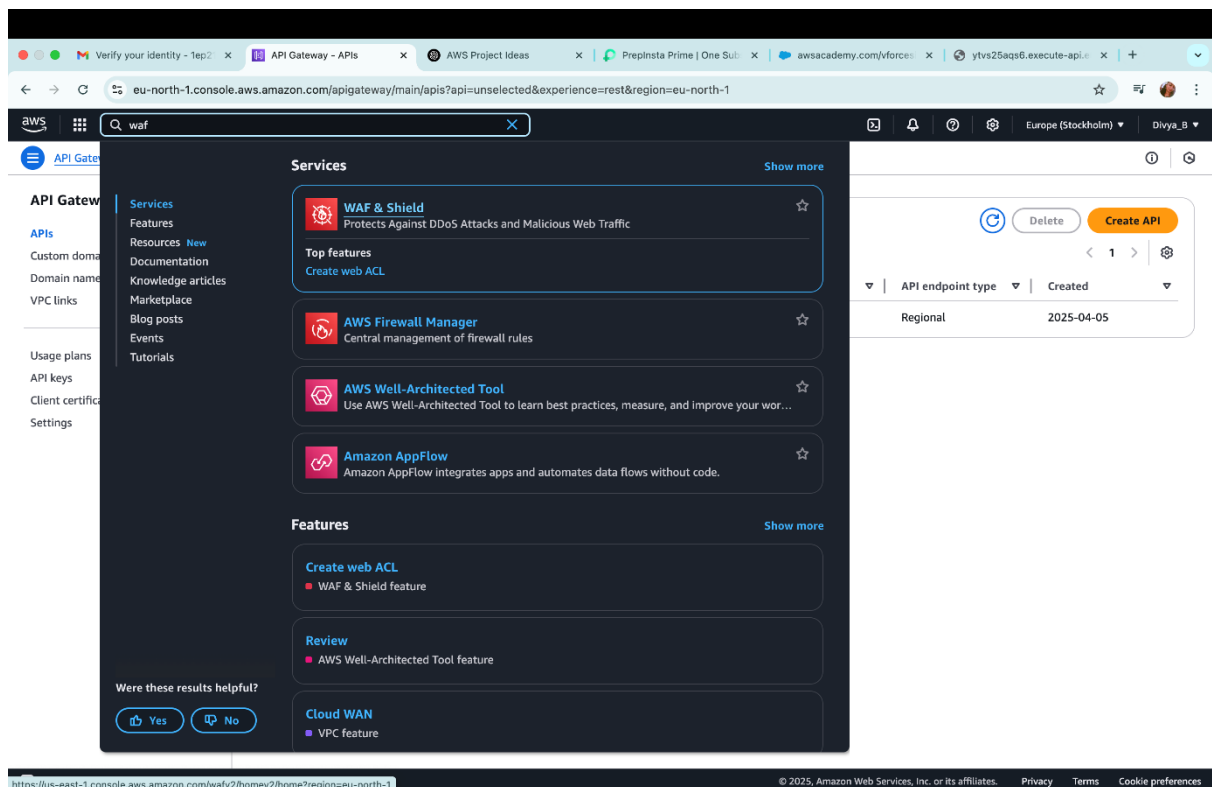


2. Append /hello to the URL.
3. Confirm you receive: Hello, Secure World!



### Step 6: Create WAF Web ACL

1. Go to **WAF & Shield** in AWS Console.



2. Click **Create web ACL**.

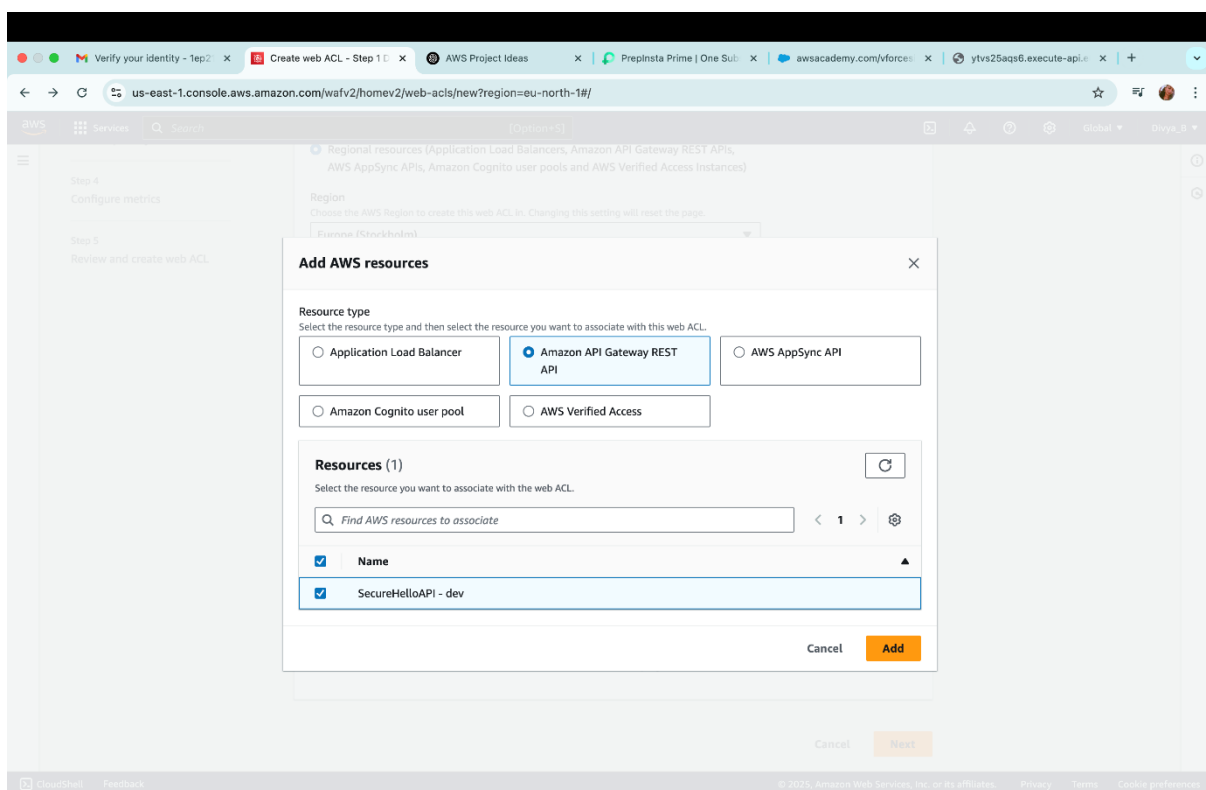


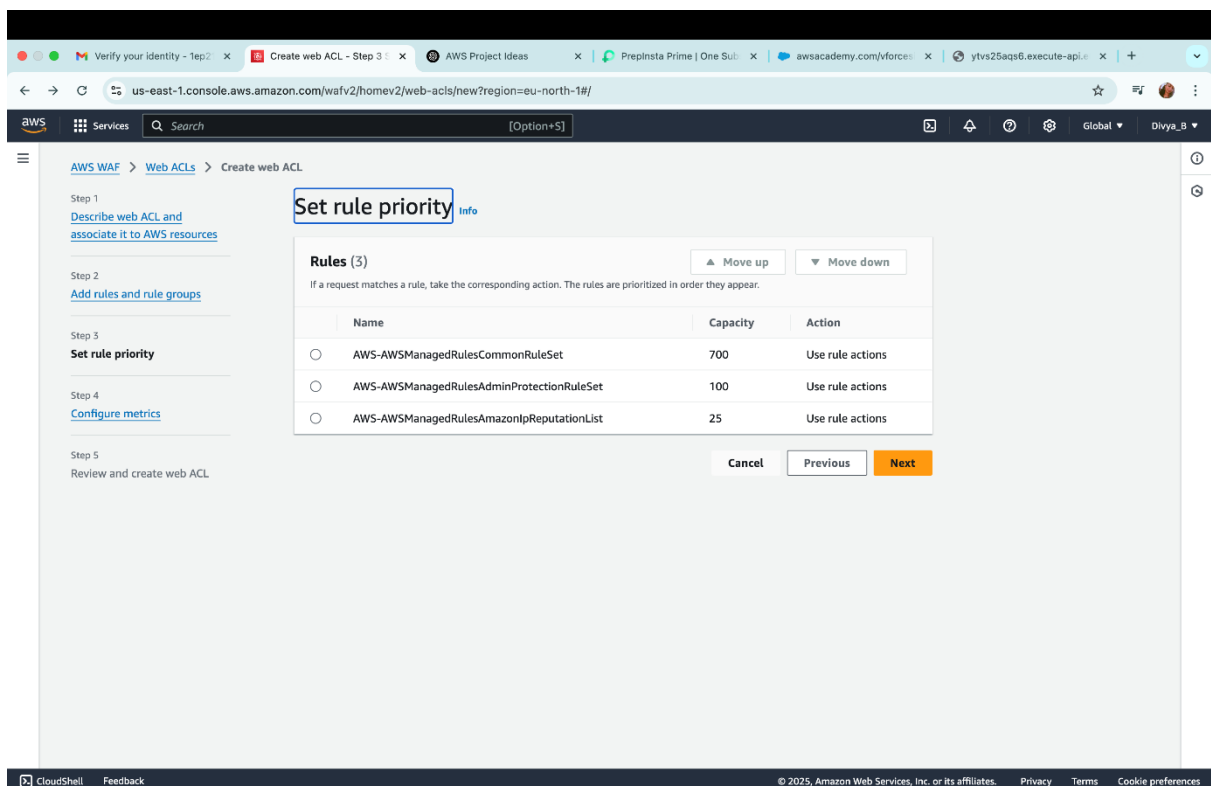
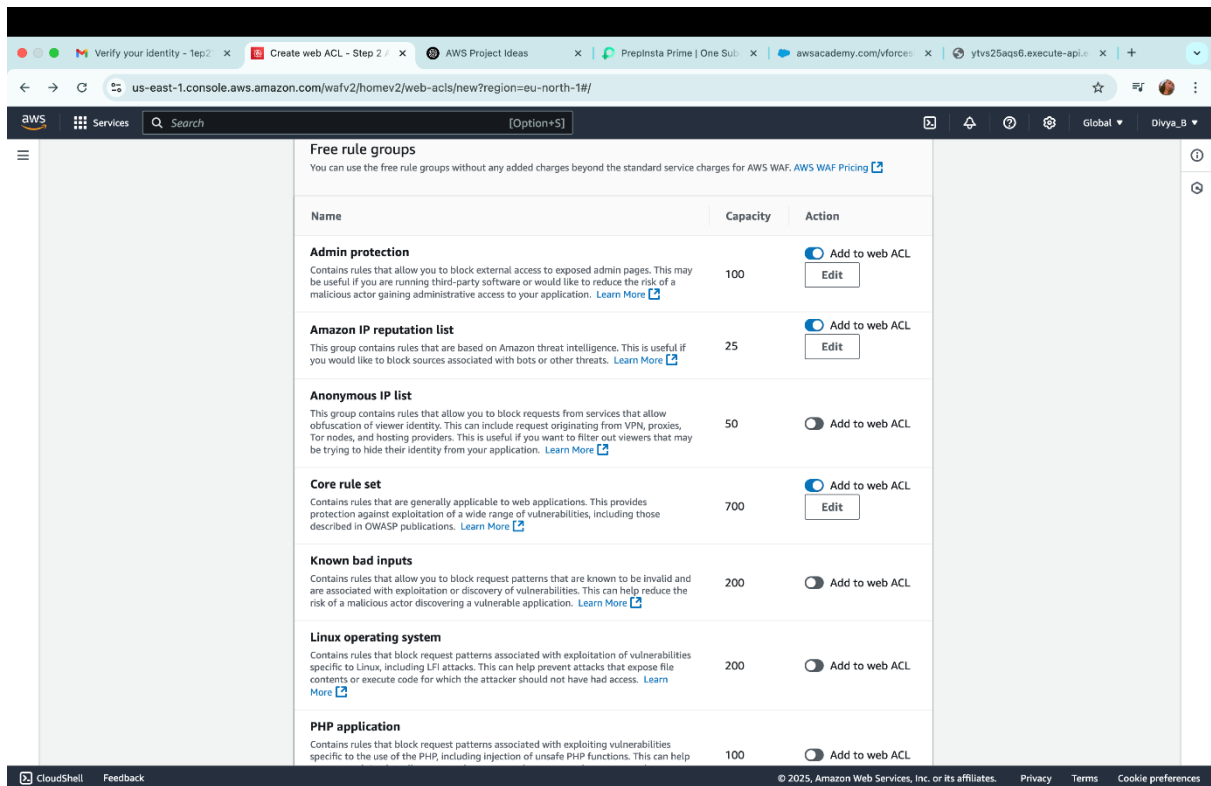
### 3. Configure:

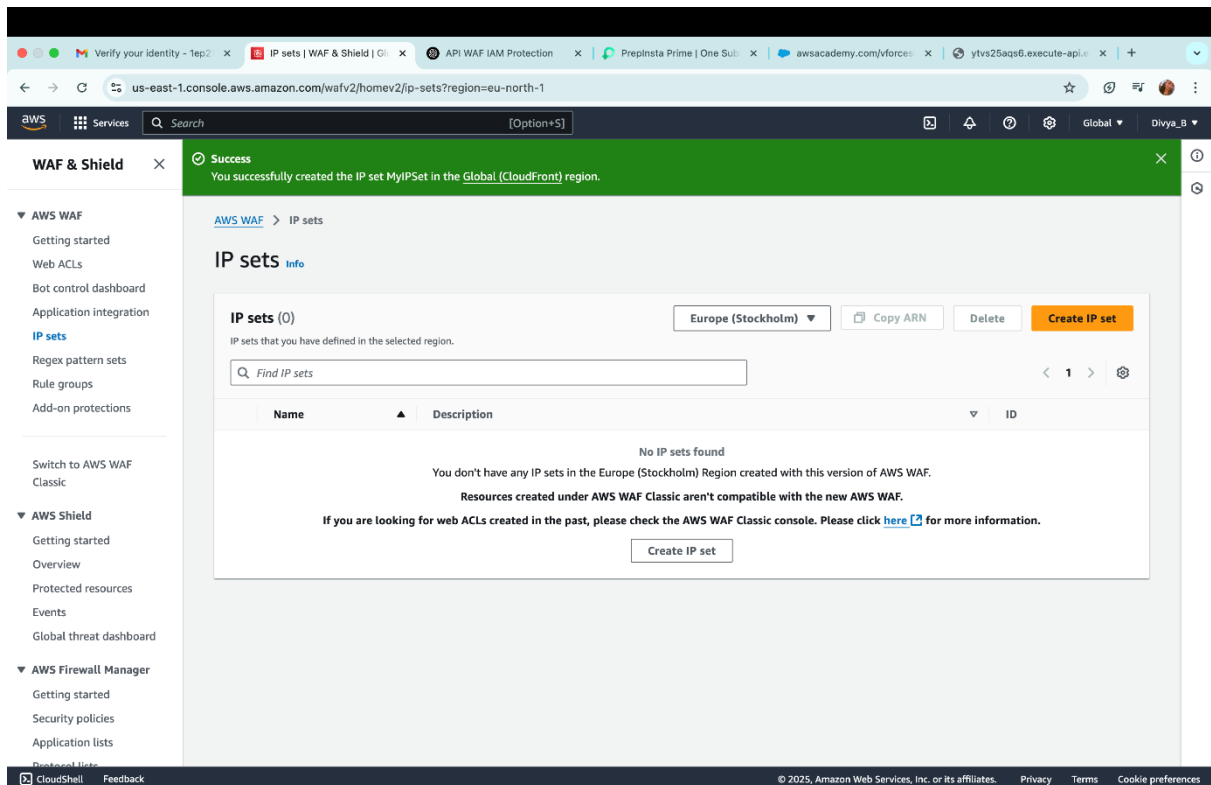
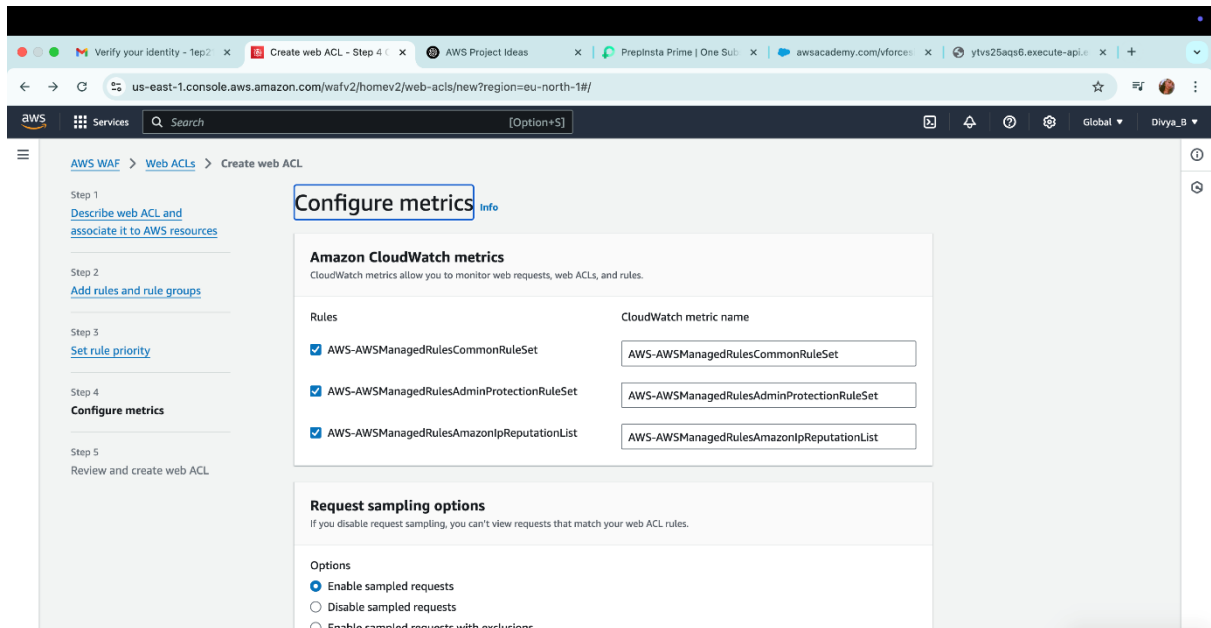
- **Name:** SecureHelloACL
- **Region:** Same as your API Gateway
- **Resource type:** Amazon API Gateway

### 4. Choose **Add AWS resource** > Select your API Gateway.

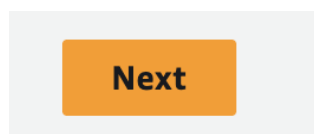
### 5. Add rules if desired (Rate limiting, IP block, etc.)

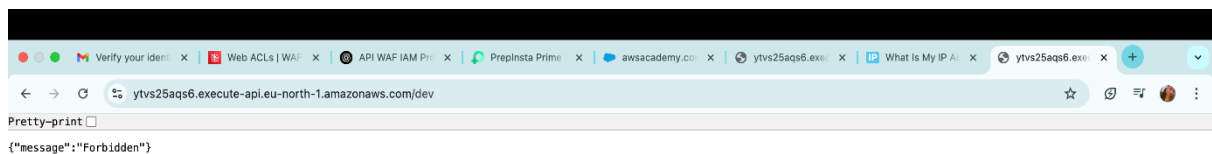
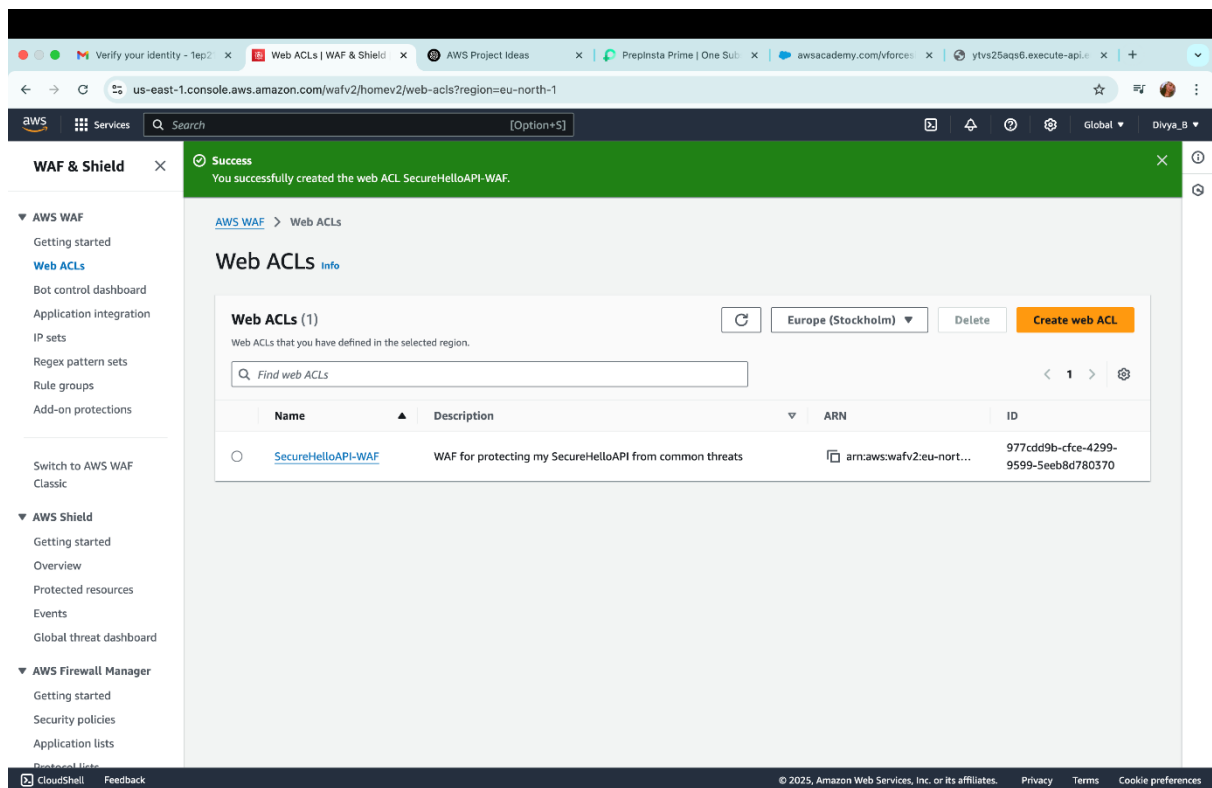






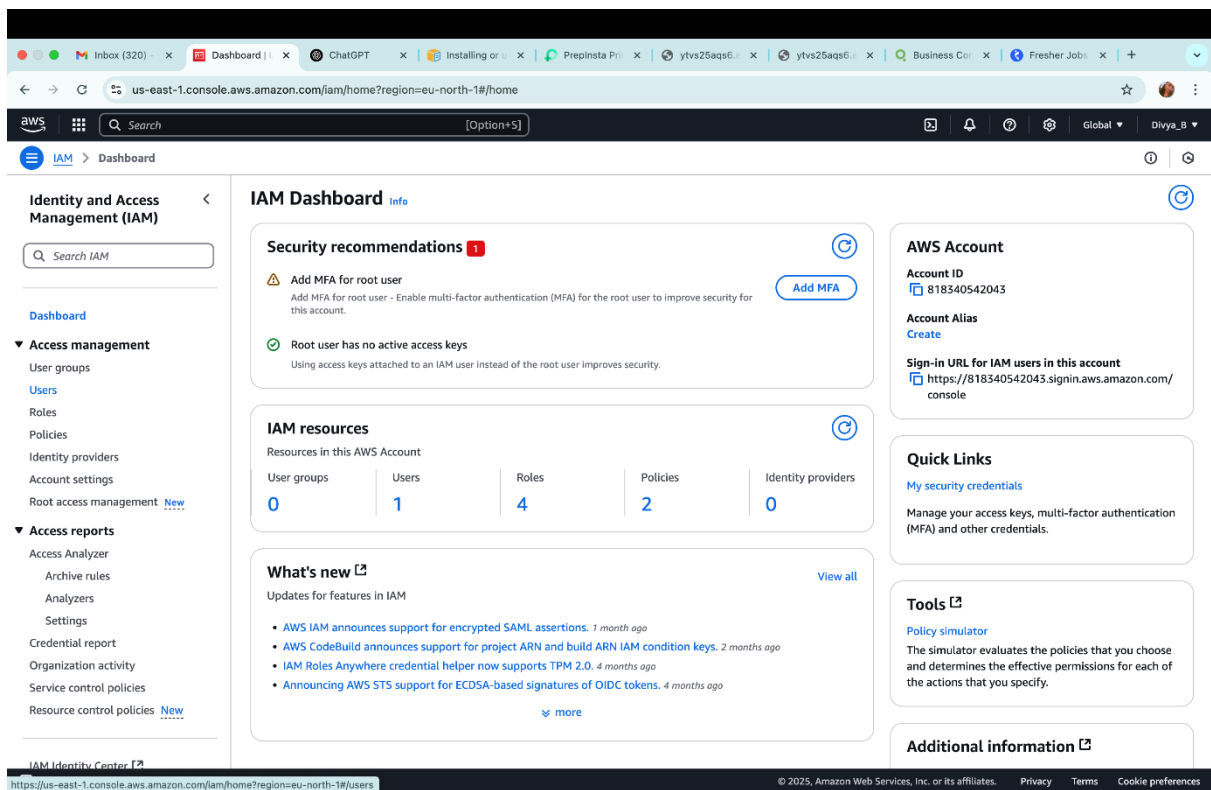
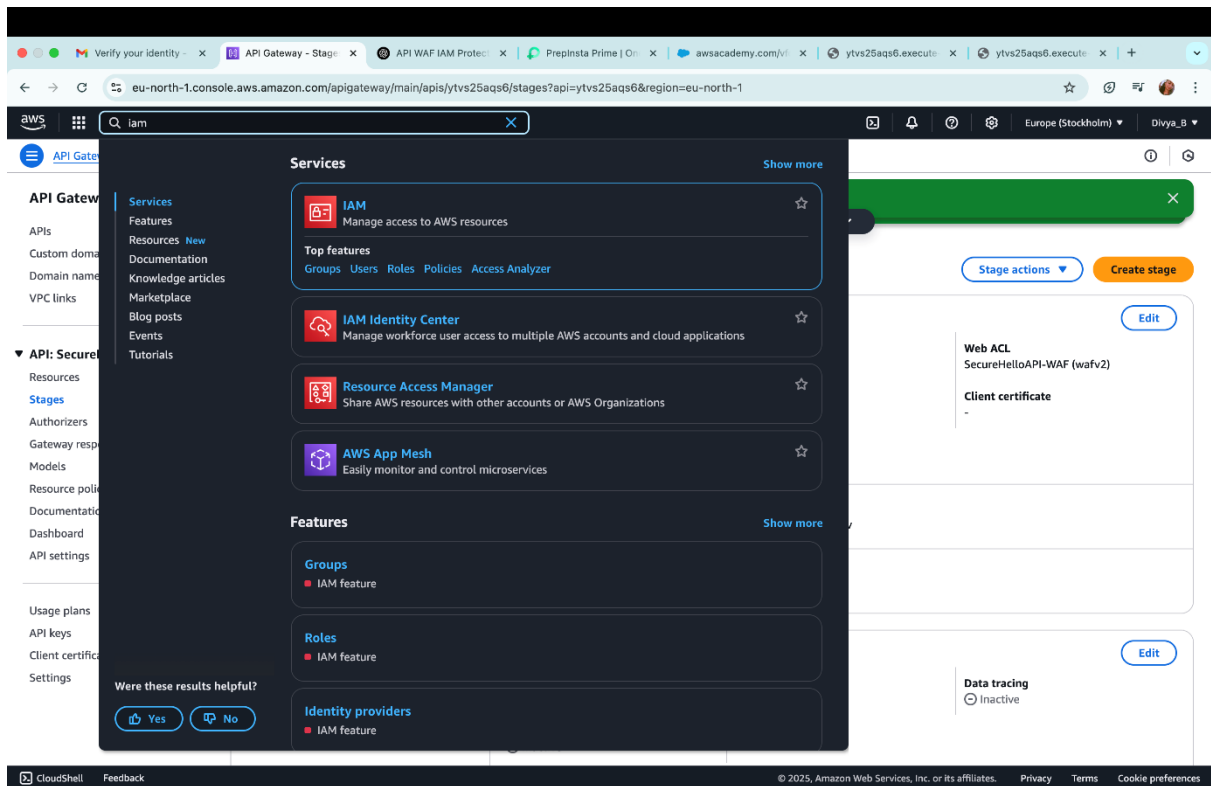
6. Click **Next** until you reach **Create Web ACL**.





### Step 7: Create IAM User for Programmatic Access

1. Navigate to **IAM > Users > Add User**.



Create user

2. Configure:

- **User name:** secure-api-user
- Enable **Programmatic access**



3. Attach policy:

- Use **AmazonAPIGatewayInvokeFullAccess** (or custom policy)

4. Click **Next** and **Create User**.



5. Save **Access Key ID** and **Secret Access Key**.

**Step 8: Enable IAM Authorization on GET Method**

1. Go to **API Gateway** > Your API > Resources > /hello > **GET**.
2. Click **Method Request**.
3. Under **Authorization**, choose **AWS\_IAM**.



The screenshot shows the AWS API Gateway console for the 'SecureHelloAPI' in the 'eu-north-1' region. The left sidebar shows the navigation menu with 'API Gateway' selected. The main content area shows the 'Resources' section for the 'GET' method. The 'Method request settings' are displayed, with 'Authorization' set to 'AWS\_IAM' (circled in red). The 'API key required' is set to 'False'. The 'Request paths' section shows a single path with a 'Caching' dropdown.

4. Click **Deploy API** again to apply changes.

The screenshot shows the AWS API Gateway console after the deployment. A green notification bar at the top states: 'Successfully edited method request for 'GET'. Redeploy your API for the update to take effect.' The 'Method request settings' are still displayed, with 'Authorization' set to 'AWS\_IAM' and 'API key required' set to 'False'. The 'Request paths' section shows a single path with a 'Caching' dropdown.

## Step 9: Test Using IAM Authorization (SigV4)

1. Use the AWS CLI or a Python script to sign the GET request.
2. Example Python script (using requests + botocore):
  - Script uses access keys to sign and invoke the API securely.
3. Confirm response: Hello, Secure World!

The screenshot shows a VS Code editor with a file named `call_secure_api.py` open. The script uses the `requests` library to make a GET request to an API endpoint, signing it with AWS credentials. The terminal output shows the script being executed multiple times, resulting in a 'Hello, Secure World!' response for successful requests and a 'Forbidden' message for failed ones.

```

16 credentials.access_key,
17 credentials.secret_key,
18 region,
19 service,
20 session_token=credentials.token
21 )
22
23 # Send GET request
24 response = requests.get(api_url, auth=auth)
25
26 # Output
27 print("Status Code:", response.status_code)
28 print("Response Body:", response.text)
29

```

```

Response Body: {"statusCode": 200, "body": "Hello, Secure World!"}
baparth@MacBook-Pro Desktop % python call_secure_api.py
Status Code: 403
Response Body: {"message": "Forbidden"}
baparth@MacBook-Pro Desktop % python call_secure_api.py
Status Code: 200
Response Body: {"statusCode": 200, "body": "Hello, Secure World!"}
baparth@MacBook-Pro Desktop % python call_secure_api.py
Status Code: 200
Response Body: {"statusCode": 200, "body": "Hello, Secure World!"}
baparth@MacBook-Pro Desktop % python call_secure_api.py
Status Code: 403
Response Body: {"message": "Forbidden"}
baparth@MacBook-Pro Desktop % python call_secure_api.py
Status Code: 403
Response Body: {"message": "Forbidden"}
baparth@MacBook-Pro Desktop % python call_secure_api.py

```

## Step 10 : Successfully the project is executed and got the Results

- Make directory : `cd ~/Desktop`
- Running the program: `python call_secure_api.py`

This screenshot is a close-up of the terminal window from the previous image, showing the execution of the `python call_secure_api.py` command. The output shows a successful response of 'Hello, Secure World!' for the first execution, followed by several 'Forbidden' responses, indicating that the script is being run repeatedly to test different scenarios.

```

Response Body: {"statusCode": 200, "body": "Hello, Secure World!"}
baparth@MacBook-Pro Desktop % python call_secure_api.py
Status Code: 403
Response Body: {"message": "Forbidden"}
baparth@MacBook-Pro Desktop % python call_secure_api.py
Status Code: 200
Response Body: {"statusCode": 200, "body": "Hello, Secure World!"}
baparth@MacBook-Pro Desktop % python call_secure_api.py
Status Code: 200
Response Body: {"statusCode": 200, "body": "Hello, Secure World!"}
baparth@MacBook-Pro Desktop % python call_secure_api.py
Status Code: 403
Response Body: {"message": "Forbidden"}
baparth@MacBook-Pro Desktop % python call_secure_api.py
Status Code: 403
Response Body: {"message": "Forbidden"}
baparth@MacBook-Pro Desktop % python call_secure_api.py

```

**Program used for Testing API:**

```
import boto3

from requests_aws4auth import AWS4Auth

import requests

# Configuration

region = 'eu-north-1'

service = 'execute-api'

api_url = 'https://ytvs25aqs6.execute-api.eu-north-1.amazonaws.com/dev'

# Get credentials from profile

session = boto3.Session(profile_name='secureapiuser')

credentials = session.get_credentials().get_frozen_credentials()

# Set up auth

auth = AWS4Auth(

    credentials.access_key,

    credentials.secret_key,

    region,

    service,

    session_token=credentials.token

)

# Send GET request

response = requests.get(api_url, auth=auth)

# Output

print("Status Code:", response.status_code)

print("Response Body:", response.text)
```

## CHAPTER 5

### CONCLUSION

This project successfully demonstrates how serverless technologies offered by AWS can be harnessed to build and secure a scalable API solution. By using AWS Lambda for compute, API Gateway for endpoint exposure, IAM for secure access control, and AWS WAF for threat protection, we implemented a robust and secure system that delivers dynamic responses without managing traditional infrastructure. Each component of the solution was carefully configured to adhere to best security practices, ensuring that only authenticated users can invoke the API and that potential web-based attacks are mitigated at the gateway level.

The project also deepened our understanding of cloud-native architectures and the AWS ecosystem. By going beyond basic deployments and integrating security features such as IAM Authorization and WAF, we learned how to combine multiple AWS services in a real-world use case. This hands-on implementation emphasized not only the ease and speed of deploying serverless applications but also the critical importance of securing them effectively. The experience gained through this internship project will be invaluable for future roles involving cloud architecture and secure software deployment.

.

## REFERENCE

- [1] Amazon Web Services Documentation - <https://docs.aws.amazon.com>
- [2] AWS Lambda Developer Guide  
<https://docs.aws.amazon.com/lambda/latest/dg/welcome.html>
- [3] API Gateway REST API Guide  
<https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-rest-api.html>
- [4] AWS WAF Developer Guide - <https://docs.aws.amazon.com/waf/latest/developerguide/>
- [5] IAM Policies and Best Practices -  
<https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html>
- [6] AWS Free Tier Usage - <https://aws.amazon.com/free>