API Gateway

2. AWS API Gateway

2.1 **Overview**

AWS API Gateway is a **fully managed service** used to create, publish, secure, and monitor APIs. It acts as a **front door** for applications to access backend services like Lambda, EC2, or other AWS services.

An API (Application Programming Interface) is a set of rules that lets one application request data or services from another application.

API Gateway acts as a front door that lets applications access backend services like Lambda, EC2, or web applications.

API Gateway creates RESTful APIs that:

- Are HTTP-based.
- Enable stateless client-server communication.
- Implement standard HTTP methods such as GET, POST, PUT, PATCH, and DELETE.

API Gateway creates WebSocket APIs that:

- Adhere to the WebSocket protocol, which enables stateful, full-duplex communication between client and server.
- Route incoming messages based on message content.

2.2 **Key Features**

- Supports REST APIs, HTTP APIs, and WebSocket APIs: Supports different API types for standard requests, lightweight APIs, and real-time communication.

- Authentication and authorization: Controls who can access your API using IAM, Cognito, or JWT tokens.

- Throttling and rate limiting: Limits the number of requests to protect backend services from overload.

- Monitoring and logging: Tracks API usage, errors, and performance using CloudWatch.

- Caching support: Stores API responses temporarily to reduce backend load and improve response time.

API Gateway

2.3 **Use-cases**

- Web & Mobile Backend: Acts as the backend entry point for web and mobile applications.

- Expose Lambda as an API: Used to create serverless REST or HTTP APIs backed by Lambda functions.

- Third-Party API Integration: Connects external services (payment gateways, CRMs, webhooks) to AWS backends.

- Real-Time Applications: Supports WebSocket APIs for chat apps, live notifications, and dashboards.

- Multi-Region API Frontend: Acts as a global API endpoint while backend runs in multiple regions.

- Data Validation Layer: Validates request format and parameters before reaching backend logic.

- Proxy for EC2 or On-Prem Apps: Provides a public API interface for applications running on EC2 or on-prem servers.

2.4 **Architecture Flow**

- Example: API request handling

- Client sends HTTP request

- API Gateway receives request

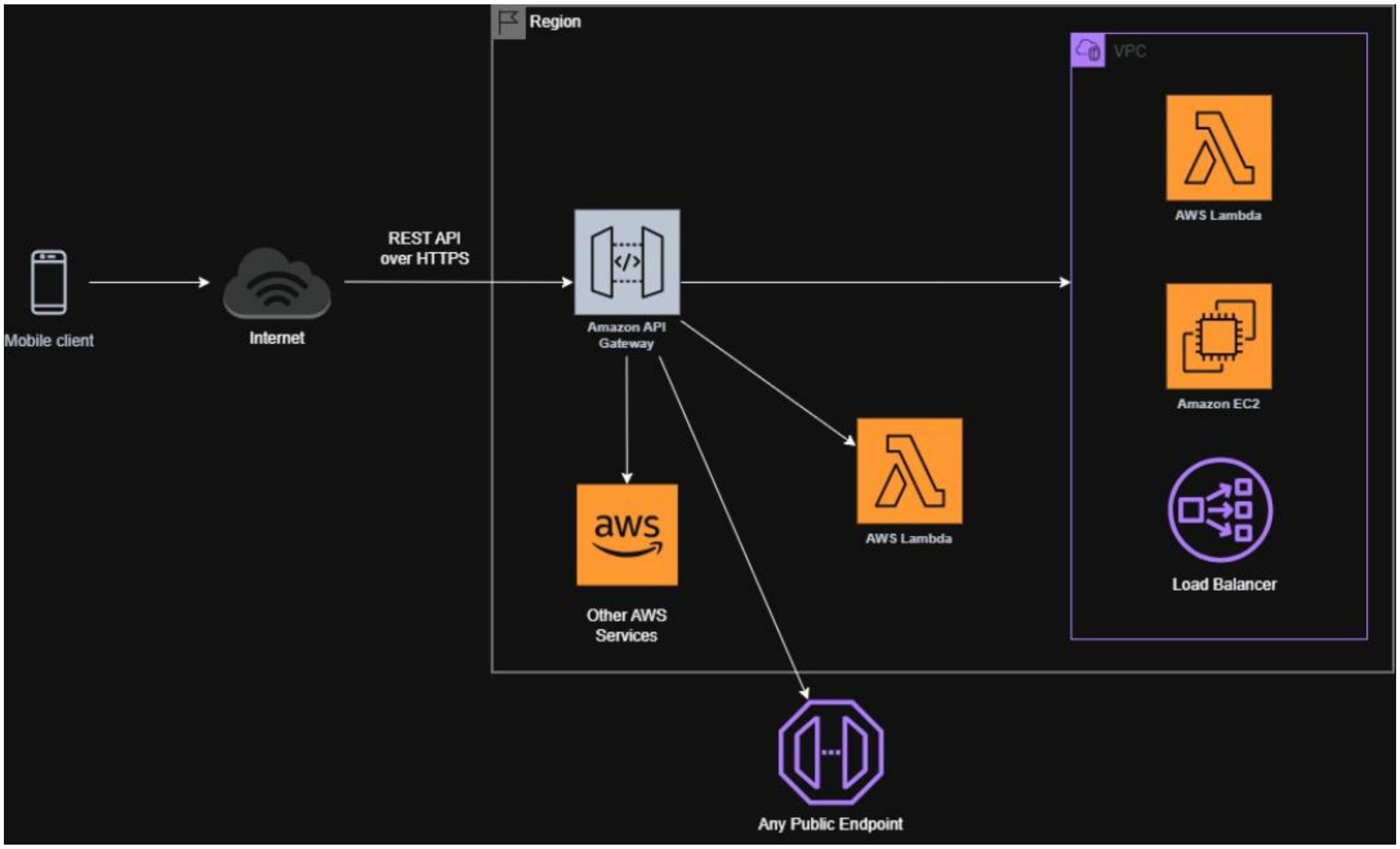
- API Gateway invokes Lambda

- Lambda processes logic

- Response sent back to client

- Flow: Client → API Gateway → Lambda → Client

API Gateway



2.5 **Execution Steps**

- Step 1: Launch an EC2 Instance.

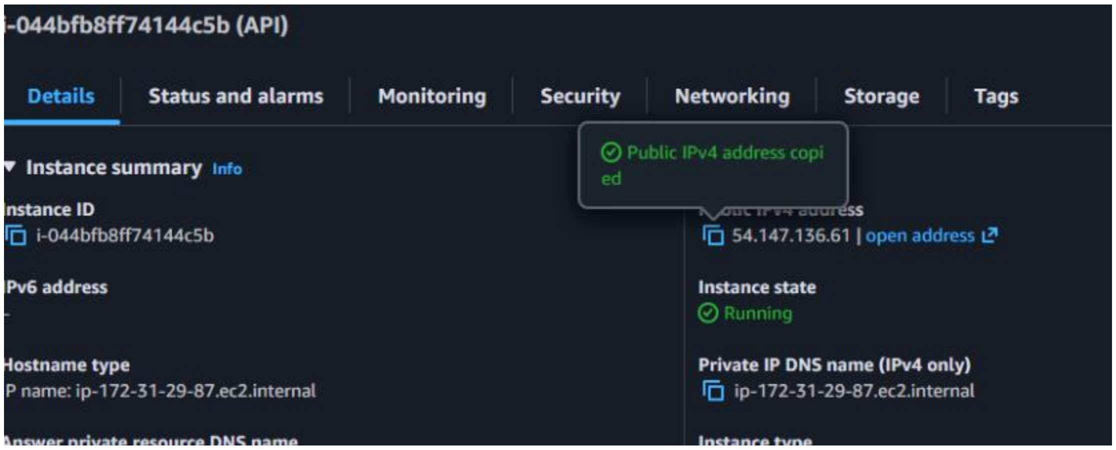
 - Go to **EC2 → Launch Instance**

 - Choose Amazon Linux

 - Instance type: t2.micro

 - Allow inbound traffic: HTTP (80) or custom app port (eg., 8080).

API Gateway


i-044bfb8ff74144c5b (API)

Details | Status and alarms | Monitoring | Security | Networking | Storage | Tags

Public IPv4 address copied

Instance summary Info

Instance ID
i-044bfb8ff74144c5b

Public IPv4 address
54.147.136.61 | open address

IPv6 address
–

Instance state
Running

Hostname type
IP name: ip-172-31-29-87.ec2.internal

Private IP DNS name (IPv4 only)
ip-172-31-29-87.ec2.internal

Answer private resource DNS name

Instance type

- Step 2: Install Web Server / App on EC2.

 - SSH into EC2 and run:

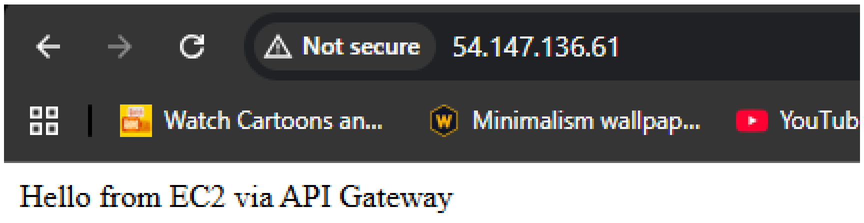
```
sudo dnf update -y

sudo dnf install httpd -y

sudo systemctl start httpd
```

- Create a test page:

```
echo "Hello from EC2 via API Gateway" | sudo tee /var/www/html/index.html
```

- Test in Browser:

 http://EC2-Public-IP


Not secure 54.147.136.61

Hello from EC2 via API Gateway

- Step 3: Create API Gateway.

 - Go to API Gateway → Create API

 - Choose REST API or HTTP API

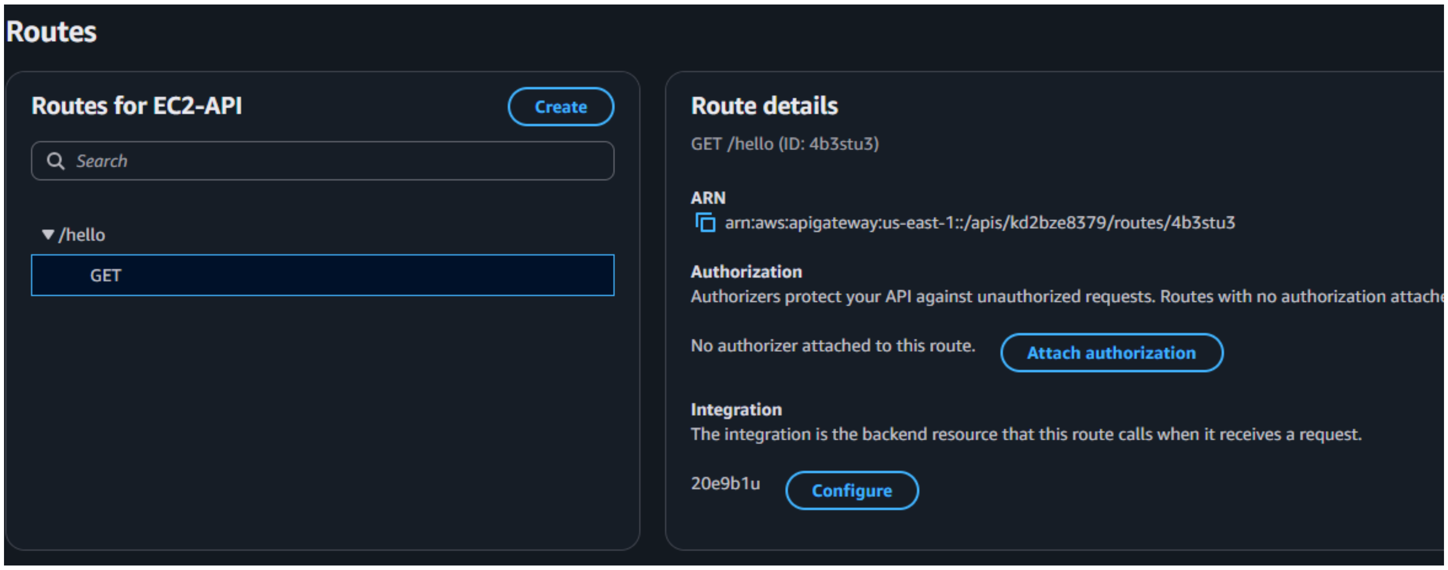
 - API name: `EC2-API`

API Gateway

- Step 4: Create Resource and Method.

- Create resource: /

- Create method: GET



Routes

Routes for EC2-API

▼/hello

GET

Route details

GET /hello (ID: 4b3stu3)

ARN

arn:aws:apigateway:us-east-1::/apis/kd2bze8379/routes/4b3stu3

Authorization

Authorizers protect your API against unauthorized requests. Routes with no authorization attache

No authorizer attached to this route.

Integration

The integration is the backend resource that this route calls when it receives a request.

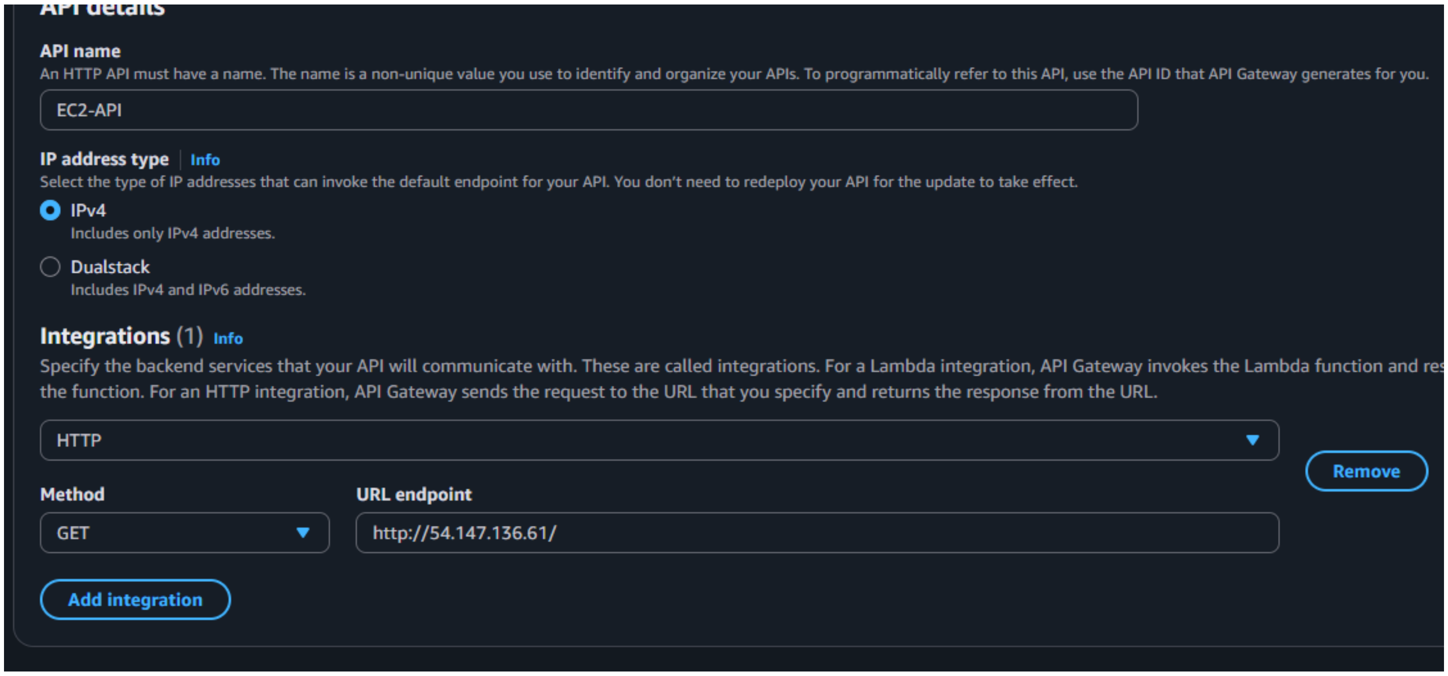
20e9b1u

- Step 5: Integrate API Gateway with EC2.

- Choose integration type:

- Integration type: HTTP

- Endpoint URL: http://EC2-Public-IP



API name

An HTTP API must have a name. The name is a non-unique value you use to identify and organize your APIs. To programmatically refer to this API, use the API ID that API Gateway generates for you.

EC2-API

IP address type Info

Select the type of IP addresses that can invoke the default endpoint for your API. You don't need to redeploy your API for the update to take effect.

[✓ IPv4] Includes only IPv4 addresses.

Dualstack Includes IPv4 and IPv6 addresses.

Integrations (1) Info

Specify the backend services that your API will communicate with. These are called integrations. For a Lambda integration, API Gateway invokes the Lambda function and res the function. For an HTTP integration, API Gateway sends the request to the URL that you specify and returns the response from the URL.

HTTP

Method: GET

URL endpoint: http://54.147.136.61/

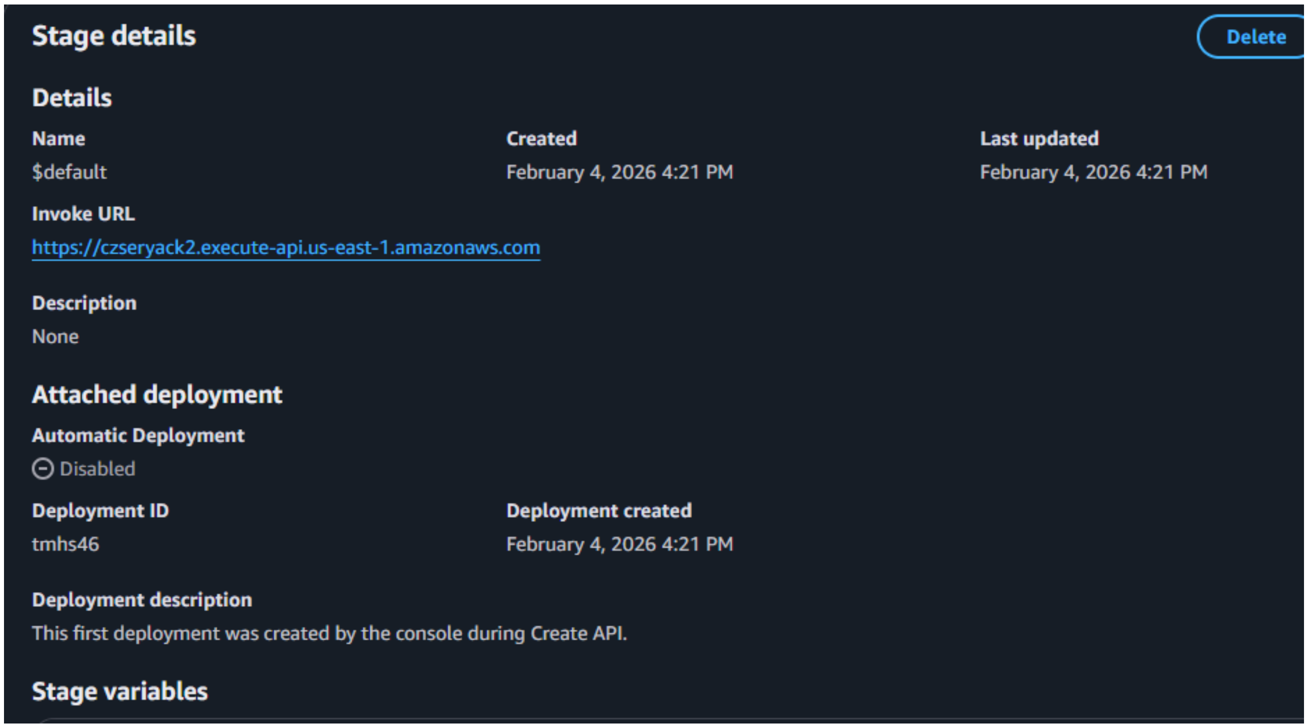
API Gateway

- Step 6: Deploy API.

  - Create stage: prod

  - Deploy API. We will get an Invoke URL like:

  https://abc123.execute-api.region.amazonaws.com/prod/hello



Stage details

Details

Name
$default

Created
February 4, 2026 4:21 PM

Last updated
February 4, 2026 4:21 PM

Invoke URL
https://czseryack2.execute-api.us-east-1.amazonaws.com

Description
None

Attached deployment

Automatic Deployment
Disabled

Deployment ID
tmhs46

Deployment created
February 4, 2026 4:21 PM

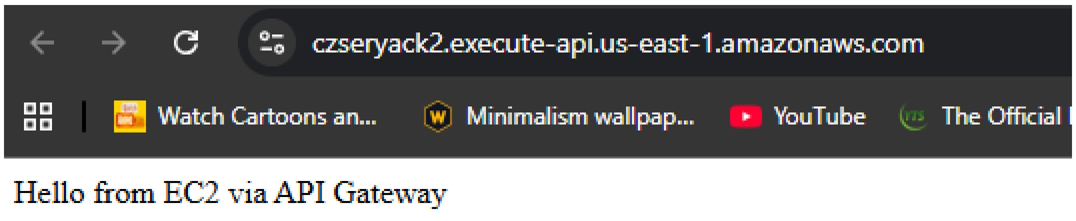
Deployment description
This first deployment was created by the console during Create API.

Stage variables

- Step 7: Test.

  - Open the invoke URL in browser.

  - Expected output: Hello from EC2 via API Gateway



Hello from EC2 via API Gateway

API Gateway

2.6 **Security and Best-Practices**

- Enable authentication (IAM / Cognito / JWT): Restrict access using IAM, Cognito, or JWT tokens.

- Enable throttling: Protect backend services from traffic spikes.

- Enable logging: Monitor API usage and troubleshoot issues.

- Use HTTPS only: Ensure data is encrypted in transit.

- Validate request parameters: Block invalid or malicious requests before reaching backend logic.

- Use custom domains and certificates: Provide secure and branded API endpoints.