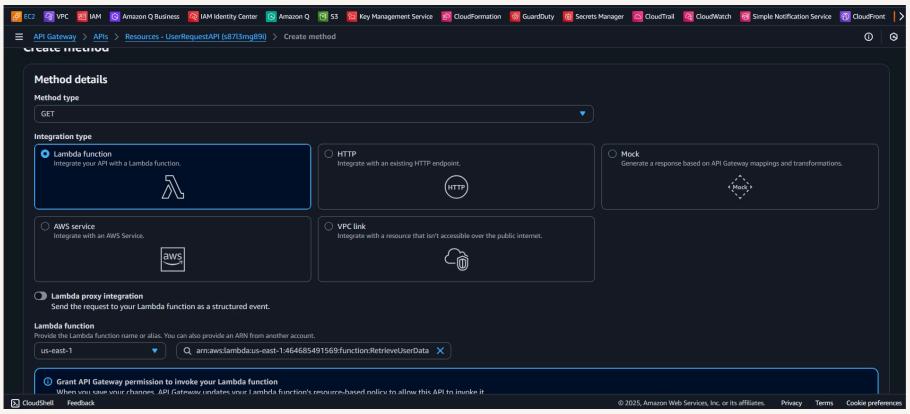
nextwork.org

APIs with Lambda + API Gateway

Chrispinus Jacob



Chrispinus Jacob
NextWork Student

Introducing Today's Project!

In this project, I will demonstrate how to set up an API using AWS Lambda and Amazon API Gateway. The goal is to deepen my understanding of how APIs work and to implement a scalable, serverless logic tier as part of a three-tier architecture. This project will focus on creating a clean, efficient API layer that connects the presentation layer (frontend) to the data tier (backend services), using AWS tools to ensure performance, reliability, and cost-effectiveness.

Tools and concepts

Services I used were Amazon API Gateway, AWS Lambda, and Amazon DynamoDB. Key concepts I learnt include Lambda functions as the logic layer for handling backend operations, API Gateway for exposing those functions to the web . I also gained hands-on experience with API methods, resource structuring, permission roles, deployment stages, and writing clear documentation to support API usability.

Project reflection

his project took me approximately 2 hours to complete, including setup, testing, and writing documentation. The most challenging part was configuring the correct IAM permissions to allow the Lambda function to access DynamoDB without triggering authorization errors. It was most rewarding to see the full flow work—from a user making a request in the browser to the Lambda function processing it and returning a response through API Gateway—because it demonstrated how serverless components can be combined to build a functional and scalable backend

Chrispinus Jacob
NextWork Student

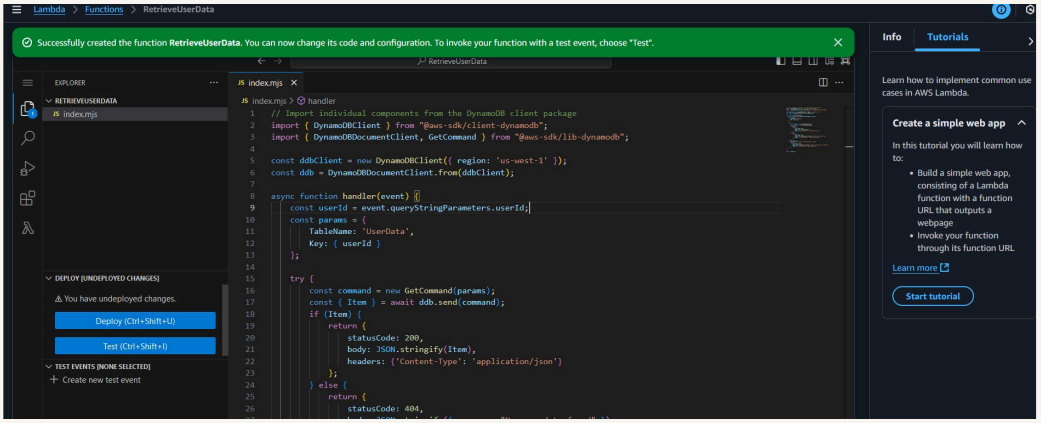
I did this project today to deepen my understanding of how serverless architecture works—specifically how AWS Lambda, API Gateway, and DynamoDB interact to build scalable APIs without managing traditional servers. Yes, this project met my goals. It helped me understand how to set up API endpoints, handle permissions, connect backend logic, and document the system properly. By the end, I had a working, real-world example of a serverless API that I can build on for future projects.

Lambda functions

AWS Lambda is a serverless compute service that lets you run code without provisioning or managing servers. I'm using Lambda in this project to handle the core logic of the application — specifically, to fetch data from the database and return it to the user through the API. This makes Lambda the “brains” of our system, enabling a clean, scalable, and event-driven logic tier within our three-tier architecture.

The code I added to my Lambda function will grab a user ID from the triggered event — typically submitted through a form or input field on a website. It then queries DynamoDB for a matching record based on that user ID. The function also includes error handling to ensure the system responds appropriately if the data isn't found or if something goes wrong during executio

Chrispinus Jacob
NextWork Student



Chrispinus Jacob
NextWork Student

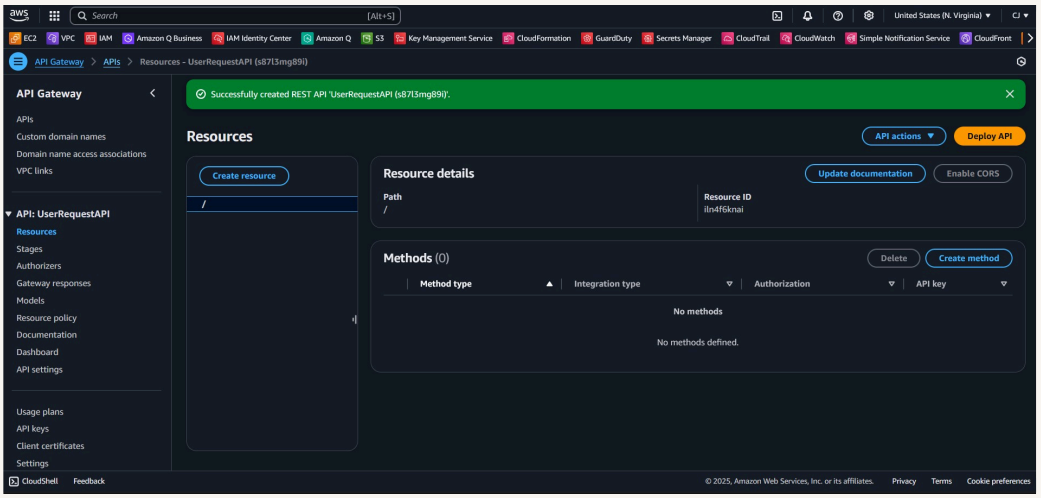
API Gateway

APIs are interfaces that allow different software systems to communicate with each other by sending and receiving data. There are different types of APIs, like REST, SOAP, GraphQL, and WebSocket APIs—each with its own structure and use cases. My API is a REST API, which uses standard HTTP methods like GET and POST to handle requests. It's designed to receive input from a user's browser, pass that data to a Lambda function for processing, and return a response—making it ideal for building scalable, serverless web applications.

Amazon API Gateway is a fully managed service that makes it easy to create, publish, and manage APIs at any scale. I'm using API Gateway in this project to act as the interface between the user's browser and the backend Lambda function. It receives HTTP requests from the client, forwards them to the Lambda function for processing, and then returns the function's response back to the user. This setup allows me to securely expose my Lambda logic to the web while maintaining control over routing, access, and data formats.

When a user makes a request—such as submitting a form or clicking a button on a website—API Gateway receives that HTTP request and routes it to the appropriate AWS Lambda function. The Lambda function then executes the backend logic, such as fetching data from a database, processing input, or performing calculations. Once the function finishes executing, it returns a response to API Gateway, which then sends that response back to the user’s browser. This seamless connection allows for serverless, scalable web applications without the need to manage traditional servers.

Chrispinus Jacob
NextWork Student



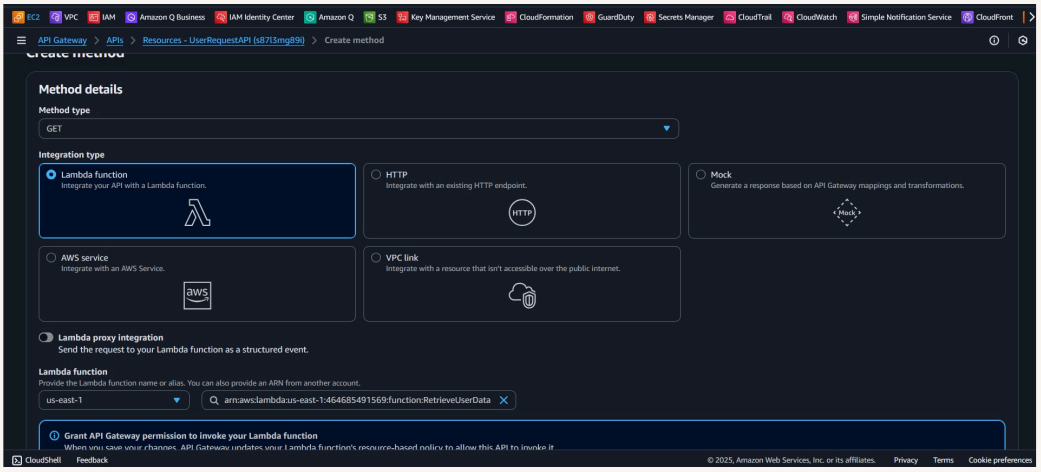
Chrispinus Jacob
NextWork Student

API Resources and Methods

An API is made up of resources, which represent specific paths or endpoints that define what actions the API can perform. Each resource typically corresponds to a particular part of your application, such as `/users`, `/products`, or `/orders`. Resources help organize and structure the API by grouping related operations under a common path. For example, a `/users` resource might support different HTTP methods like GET to fetch user data or POST to create a new user. This makes the API easier to understand, manage, and scale.

Each resource consists of methods, which are the specific HTTP operations—like GET, POST, PUT, or DELETE—that define what actions can be performed on that resource. Methods determine how the API should handle incoming requests and what type of response to return. For example, a GET method on the `/users` resource might fetch user data, while a POST method could create a new user entry. By setting up methods, we control the logic, permissions, and backend integrations (like linking to a Lambda function) that power each API operation.

I created a GET method that connects API Gateway with a Lambda function. This means that when a user makes a request to our API—for example, `api.com/users`—and it's a GET request, API Gateway knows to forward that request to the Lambda function. The Lambda function will then process the event, such as fetching user data, and return a response. This setup allows our API to respond dynamically based on the type of request received.

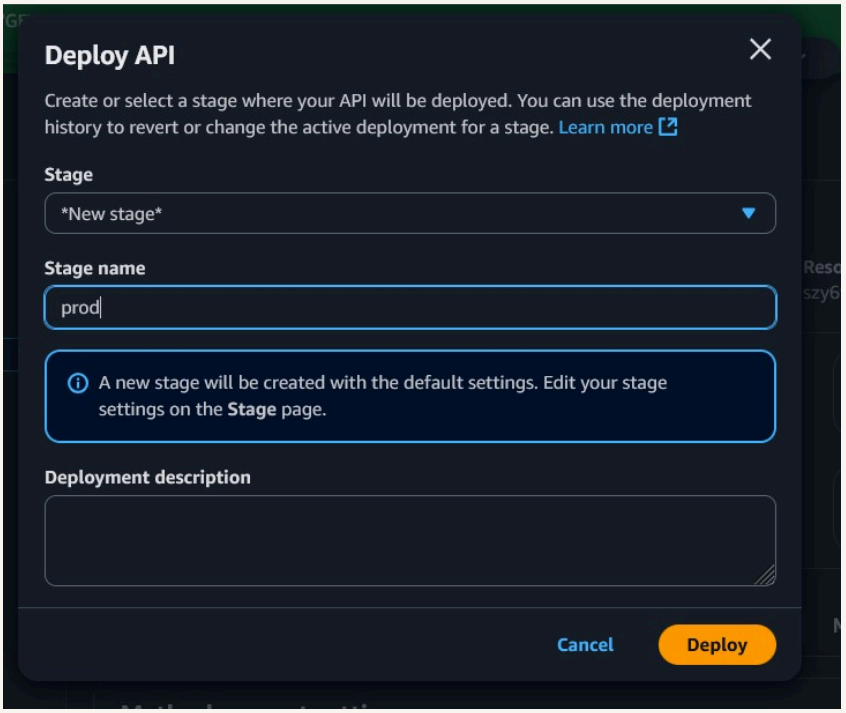

EC2 VPC IAM Amazon Q Business IAM Identity Center Amazon Q S3 Key Management Service CloudFormation GuardDuty Secrets Manager CloudTrail CloudWatch Simple Notification Service CloudFront
API Gateway > APIs > Resources - UserRequestAPI (s87l3mg89i) > Create method
Method details
Method type
GET
Integration type
Lambda function
Integrate your API with a Lambda function.
HTTP
Integrate with an existing HTTP endpoint.
Mock
Generate a response based on API Gateway mappings and transformations.
AWS service
Integrate with an AWS Service.
VPC link
Integrate with a resource that isn't accessible over the public internet.
Lambda proxy integration
Send the request to your Lambda function as a structured event.
Lambda function
Provide the Lambda function name or alias. You can also provide an ARN from another account.
us-east-1
arn:aws:lambda:us-east-1:464685491569:function:RetrieveUserData
Grant API Gateway permission to invoke your Lambda function
When you save your changes, API Gateway updates your Lambda function's resource-based policy to allow this API to invoke it.
CloudShell Feedback
© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Chrispinus Jacob
NextWork Student

API Deployment

When you deploy an API, you deploy it to a specific stage. A stage is a snapshot of your API at a specified time, representing a version that's ready to be accessed by users. I deployed to the `prod` stage, which is where the live version of the API resides—this is the version that real users interact with. Deploying to `prod` means the API is stable, tested, and publicly accessible via a URL endpoint.

To visit my API, I visited the invoke URL provided by API Gateway after deployment. The API displayed an error because it is connected to a Lambda function that doesn't have the necessary permissions to access DynamoDB. This lack of permissions can trigger an authentication or authorization issue, preventing the Lambda function from querying the database and returning a proper response. To fix this, I need to update the Lambda function's execution role to include the required DynamoDB access policies.


Deploy API
Create or select a stage where your API will be deployed. You can use the deployment history to revert or change the active deployment for a stage. Learn more
Stage
New stage
Stage name
prod
A new stage will be created with the default settings. Edit your stage settings on the Stage page.
Deployment description
Cancel
Deploy

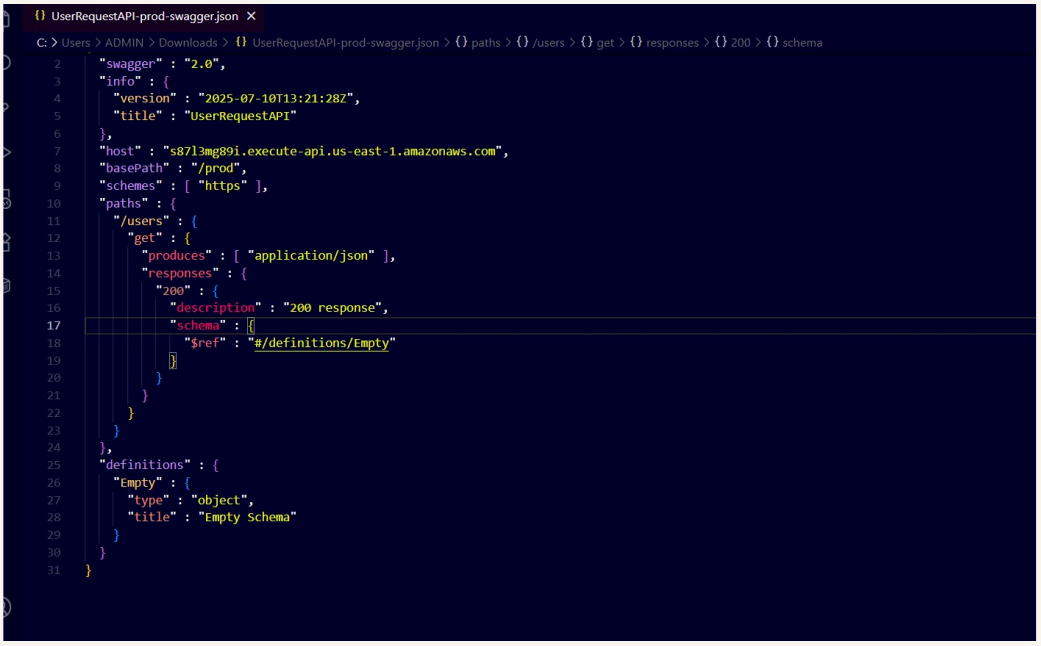
Chrispinus Jacob
NextWork Student

API Documentation

For my project's extension, I am writing API documentation because it helps others (and my future self) understand how to use the API, what each endpoint does, what data it expects, and what responses it returns. You can do this in various ways, such as creating a markdown file, using Swagger (OpenAPI), Postman documentation, or integrating tools like API Gateway's built-in documentation feature. Clear documentation ensures the API is easier to adopt, debug, and maintain over time.

Once I prepared my documentation, I can publish it to a specific stage in Amazon API Gateway, such as `dev`, `test`, or `prod`. You have to publish your API to a specific stage because each stage represents a versioned snapshot of your API, and only deployed stages are accessible to users. Publishing documentation to a stage ensures that developers can view accurate, stage-specific details—like available endpoints, request formats, and expected responses—directly from the API's live environment.

My published and downloaded documentation showed me a clear overview of my API's structure, including available resources, methods, request parameters, and example responses. It confirmed that each endpoint was correctly configured and helped identify any missing descriptions or inconsistencies. Having this documentation makes it easier to share the API with other developers and ensures that the implementation aligns with the intended design.


UserRequestAPI-prod-swagger.json
C: > Users > ADMIN > Downloads > {} UserRequestAPI-prod-swagger.json > {} paths > {} /users > {} get > {} responses > {} 200 > {} schema
{
"swagger" : "2.0",
"info" : {
"version" : "2025-07-10T13:21:28Z",
"title" : "UserRequestAPI"
},
"host" : "s87l3mg89i.execute-api.us-east-1.amazonaws.com",
"basePath" : "/prod",
"schemes" : ["https"],
"paths" : {
"/users" : {
"get" : {
"produces" : ["application/json"],
"responses" : {
"200" : {
"description" : "200 response",
"schema" : {
"$ref" : "#/definitions/Empty"
}
}
}
}
}
},
"definitions" : {
"Empty" : {
"type" : "object",
"title" : "Empty Schema"
}
}
}

The place to learn & showcase your skills

Check out nextwork.org for more projects