**Amazon API Gateway**

**What is it?**

* A fully managed service for creating, publishing, maintaining, and securing APIs at any scale.
* Acts as a "front door" for applications to access data, business logic, or functionality from backend services.
* Supports RESTful APIs, WebSocket APIs, containerized and serverless workloads, as well as web applications.

**Benefits:**

* **No infrastructure to manage:** API Gateway handles tasks like traffic management, authorization, and monitoring.
* **Supports various protocols and workloads:** Works with different protocols (REST, WebSocket) and backend services (Lambda, EC2, web applications).
* **Handles versioning and environments:** Manages different API versions and environments (dev, test, prod).
* **Security features:** Provides authentication, authorization, API key creation, and request throttling.
* **Easy API definition:** Allows import of Swagger/Open API specifications for quick API definition.
* **Request and response processing:** Enables transformation and validation of requests and responses.
* **SDK and API generation:** Generates SDKs and API specifications for easy integration.
* **Caching:** Can cache API responses for improved performance.

**Integrations:**

* **Lambda Functions:** Easiest way to expose a REST API backed by Lambda.
* **HTTP Endpoints:** Expose internal HTTP APIs or integrate with Application Load Balancers.
* **AWS Services:** Expose any AWS API through API Gateway (e.g., start Step Functions, send messages to SQS).

**Endpoint Types:**

* **Edge-Optimized (default):** Best for global clients, uses CloudFront Edge locations for low latency.
* **Regional:** For clients within the same region, can be combined with CloudFront for more caching control.
* **Private:** Only accessible from your VPC using an interface VPC endpoint (ENI).

**Security:**

* **User Authentication:**
  + IAM Roles (internal applications)
  + Cognito (external users, e.g., mobile apps)
  + Custom Authorizer (your own logic)
* **HTTPS Security:**
  + Integrates with AWS Certificate Manager (ACM)
  + Requires specific certificate locations based on endpoint type
  + Requires CNAME or A-alias record setup in Route 53

**Note:**

* Free tier available for the first 1 million API calls per month.
* Pay-as-you-go pricing for API calls, data transfer, and other services.

**API Gateway Hands-on**

**Choosing an API type:**

* **API Gateway** offers various API types:
  + **REST API:** Most common, used for structured data exchange (e.g., GET, POST, PUT, DELETE).
  + **WebSocket API:** Enables real-time, two-way communication between clients and servers.
  + **HTTP API:** Integrates with existing HTTP endpoints or services.
* **Public vs. Private:** APIs can be publicly accessible or private (accessible only from your VPC).

**Building an API:**

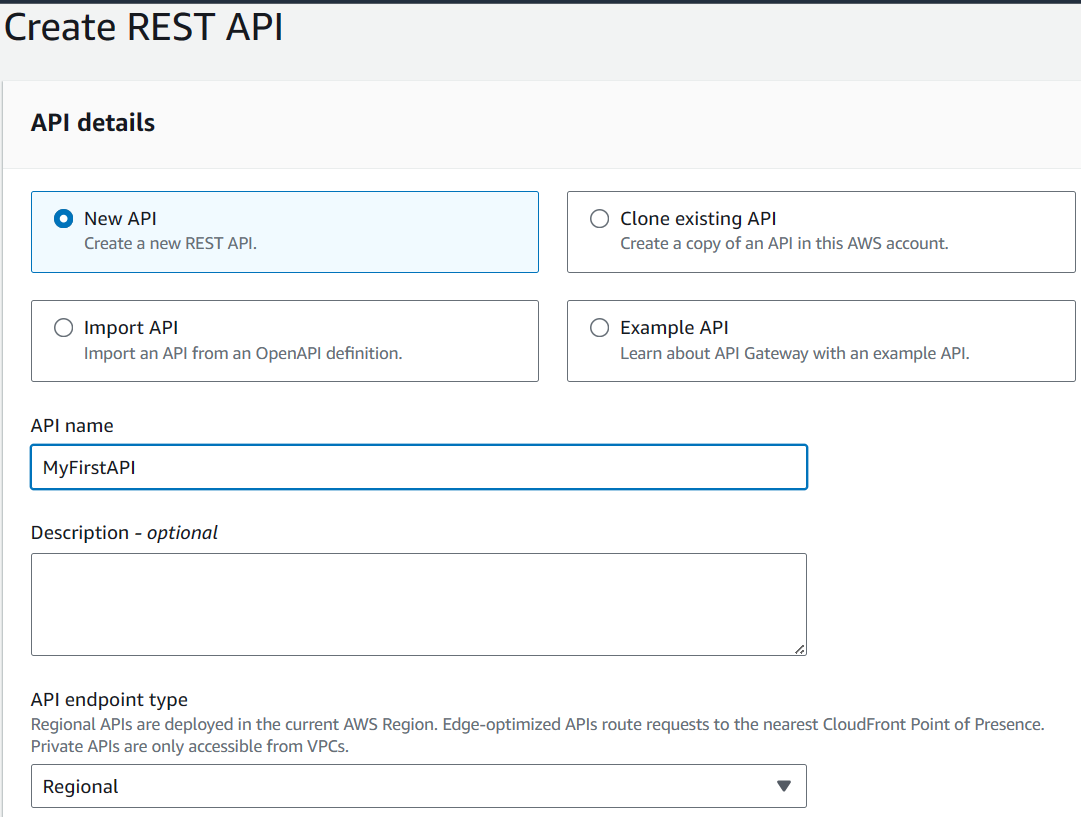
* **Create a new API:** Choose Rest API and name it as "MyFirstAPI".
* **Import from OpenAPI definition:** Define the API using a JSON/YAML file.
* **Clone an existing API:** Duplicate an existing API for reference.
* **Start from an example API:** Use a pre-built example for quick reference.

**API endpoint type:**

* **Regional:** Deployed in a single region (appropriate for regional clients).
* **Edge-optimized (default):** Deployed across multiple edge locations for global clients (better latency).
* **Private:** Accessible only from your Virtual Private Cloud (VPC).

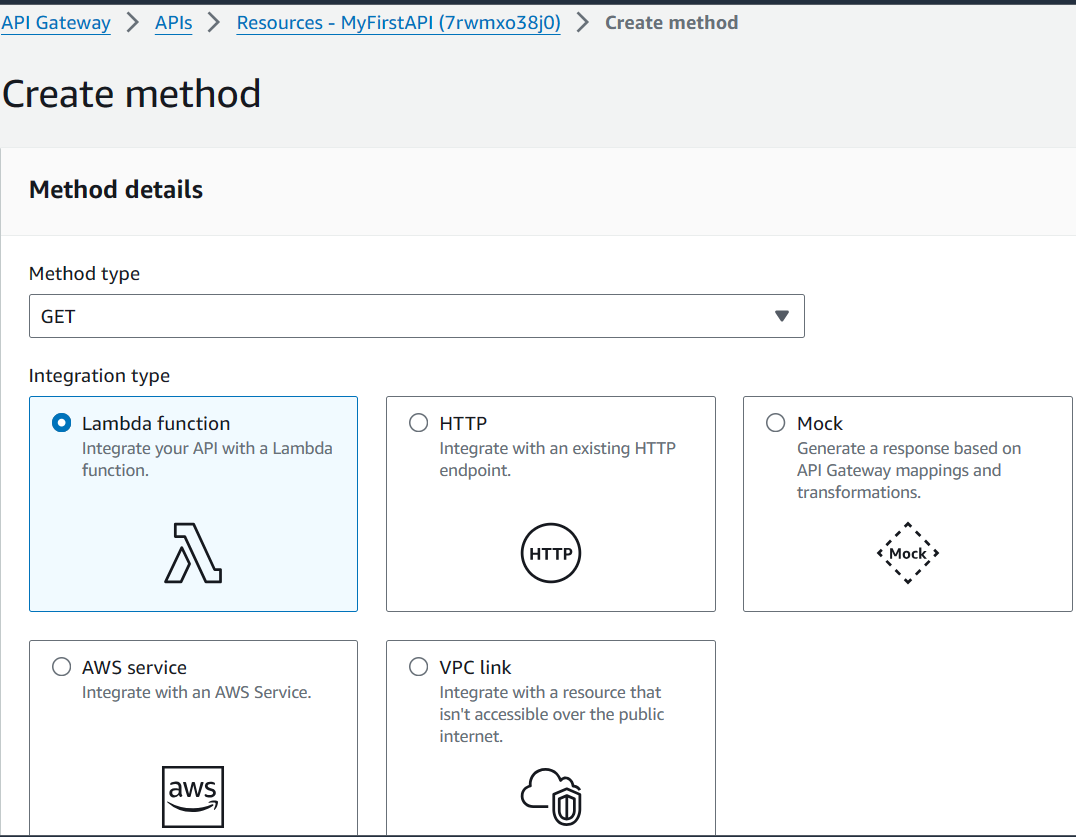
**Next steps:**

1. Choose "Regional" endpoint type.
2. Click "Create API" to proceed.



**Creating an API Method:**

1. Click "Create method" on your newly created API.
2. Choose "GET" as the method type.
3. Select "Lambda Function" as the integration type.



**Creating a Lambda Function:**

1. Go to the Lambda service and create a new function named "api-gateway-root-get".
2. Choose Python 3.12 as the runtime and create function.
3. Copy and paste the provided Python code into the function editor.

import json

def lambda\_handler(event, context):

    body = "Hello from Lambda!"

    statusCode = 200

    return {

        "statusCode": statusCode,

        "body": json.dumps(body),

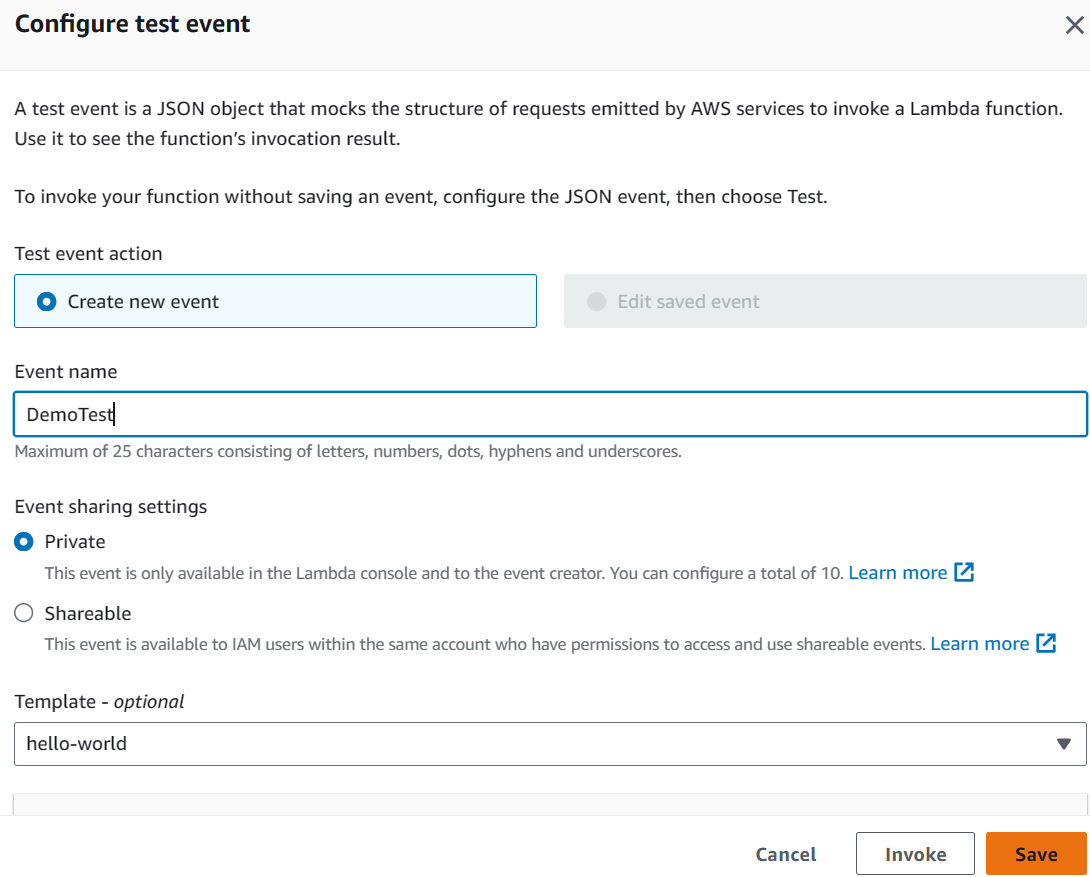
        "headers": {

            "Content-Type": "application/json"

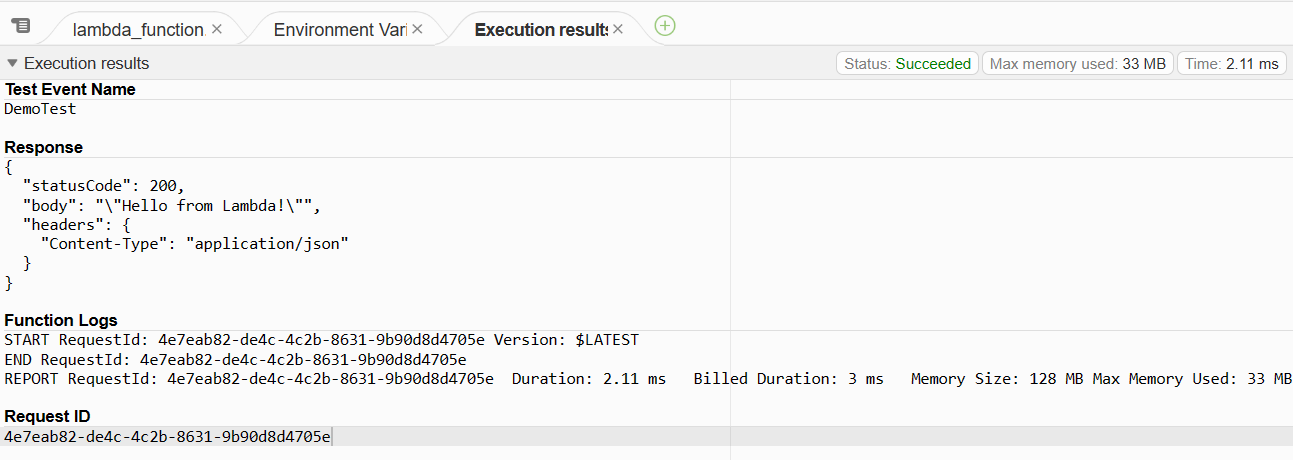
        }

    }

1. Click "Deploy" to deploy the Lambda function.
2. Create a test event called "DemoTest"

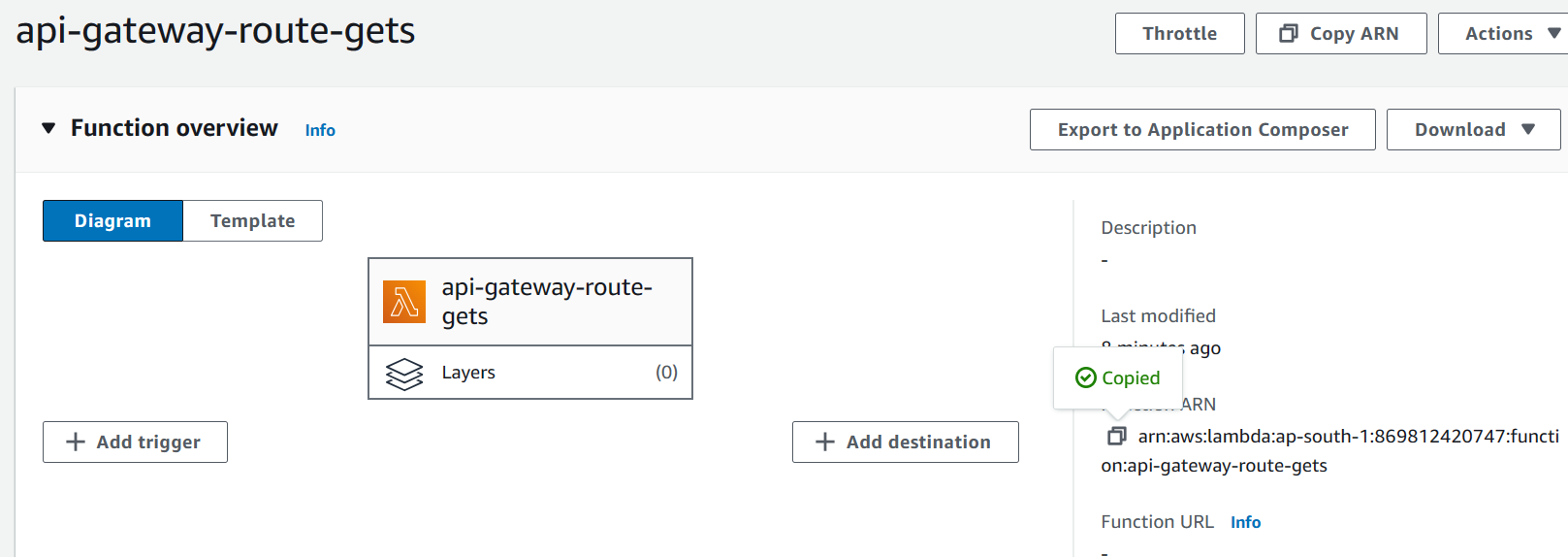


click "Test" to verify the function's output (should be "Hello from Lambda!" with status code 200).

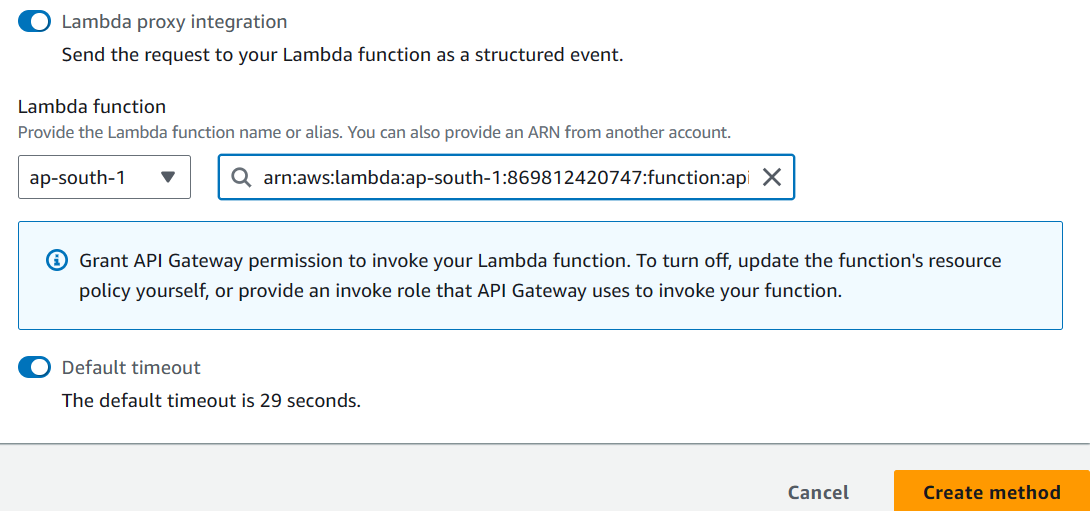


**Integrating Lambda with API Gateway:**

1. Copy the Lambda function's ARN (Amazon Resource Name).

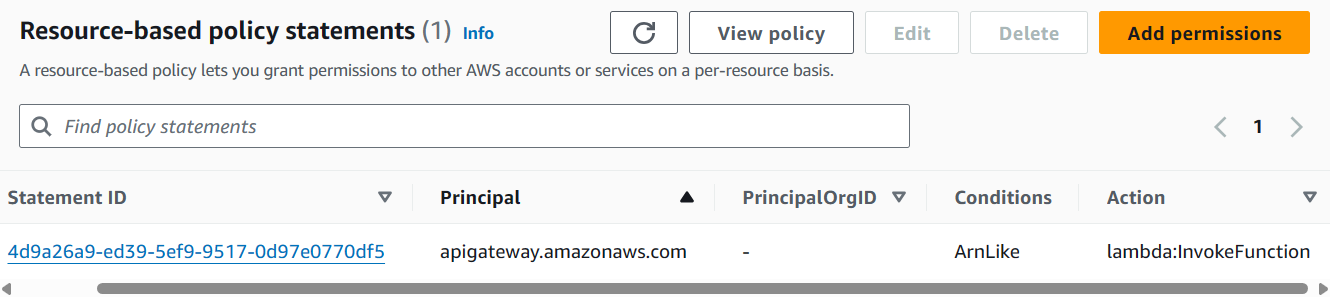


1. In API Gateway, paste the ARN into the integration field.
2. Customize the timeout limit if needed (default is 29 seconds).
3. Create the method, which automatically grants API Gateway permission to invoke the Lambda function.



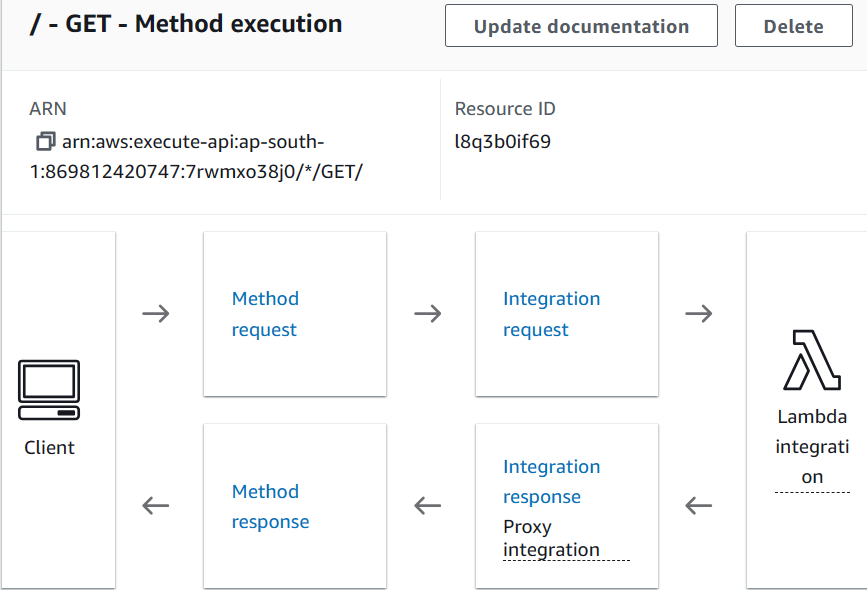
**Verifying Permissions:**

1. Refresh the Lambda function page.
2. Go to "Configuration" -> "Permissions".
3. View the resource-based policy statement to confirm that API Gateway has permission to invoke the Lambda function for the specified API and method (GET).

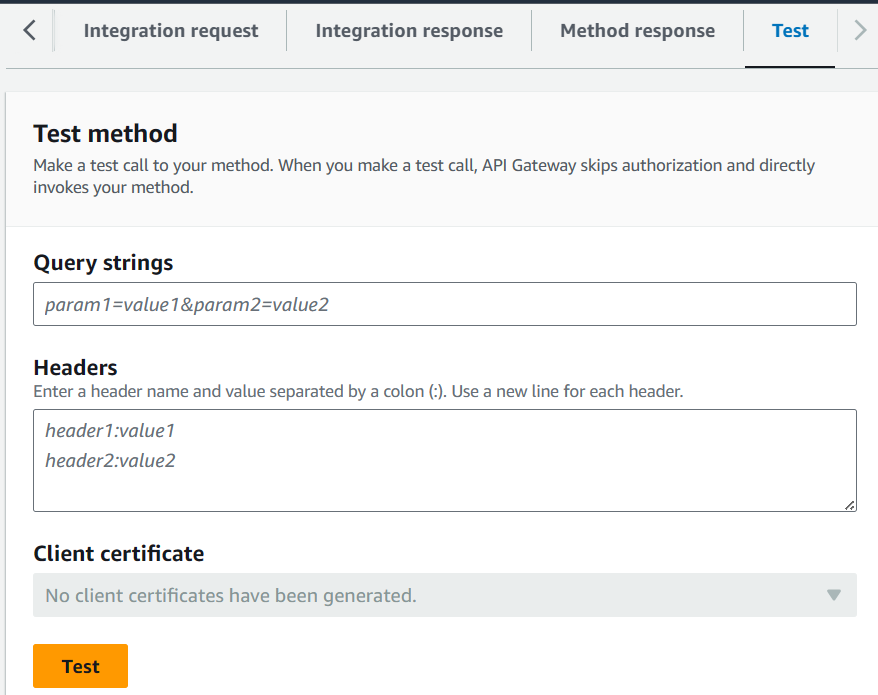


**Exploring an Existing API:**

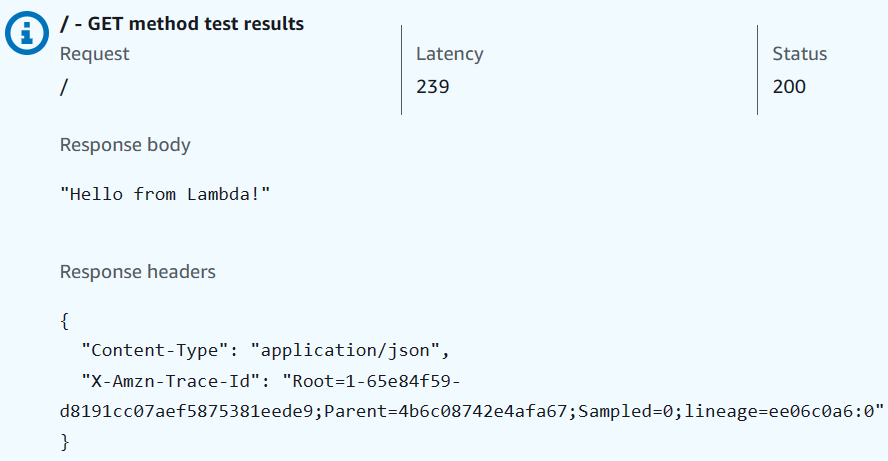
1. **Examine the API details:**
   * Integration type: Lambda function (proxy enabled).
   * Response type: Proxy integration (API Gateway parses Lambda output).
   * Method response: Looking for "application/json".



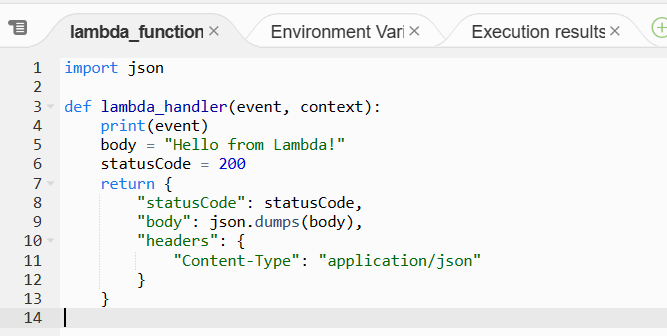
1. **Test the API:**
   * Use the "Test" feature with no arguments.



* + Observe the response:
    1. Status code: 200 (from Lambda code).
    2. Body: "Hello from Lambda" (from Lambda code).
    3. Headers: Content-Type: application/json.



1. **Debug Lambda Input:**
   * Modify the Lambda code by adding print(event).

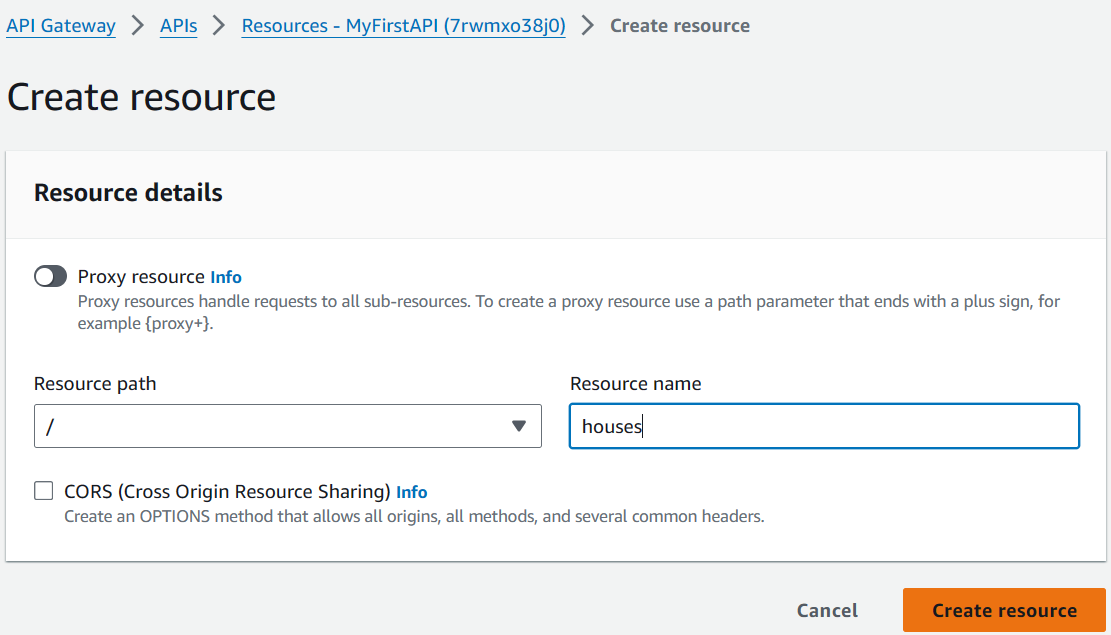


* + Redeploy the Lambda function.
  + Perform another test from API Gateway.
  + Check CloudWatch logs:
    1. Find the latest log stream.
    2. The printed event contains details like resource, path, method, headers, and query string parameters.

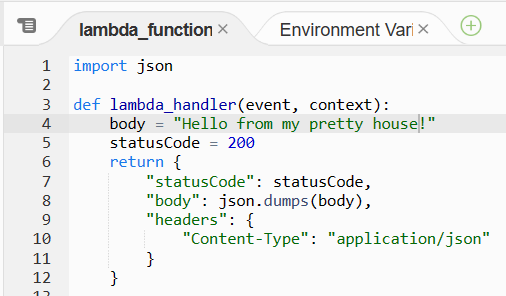


**Creating a New Resource:**

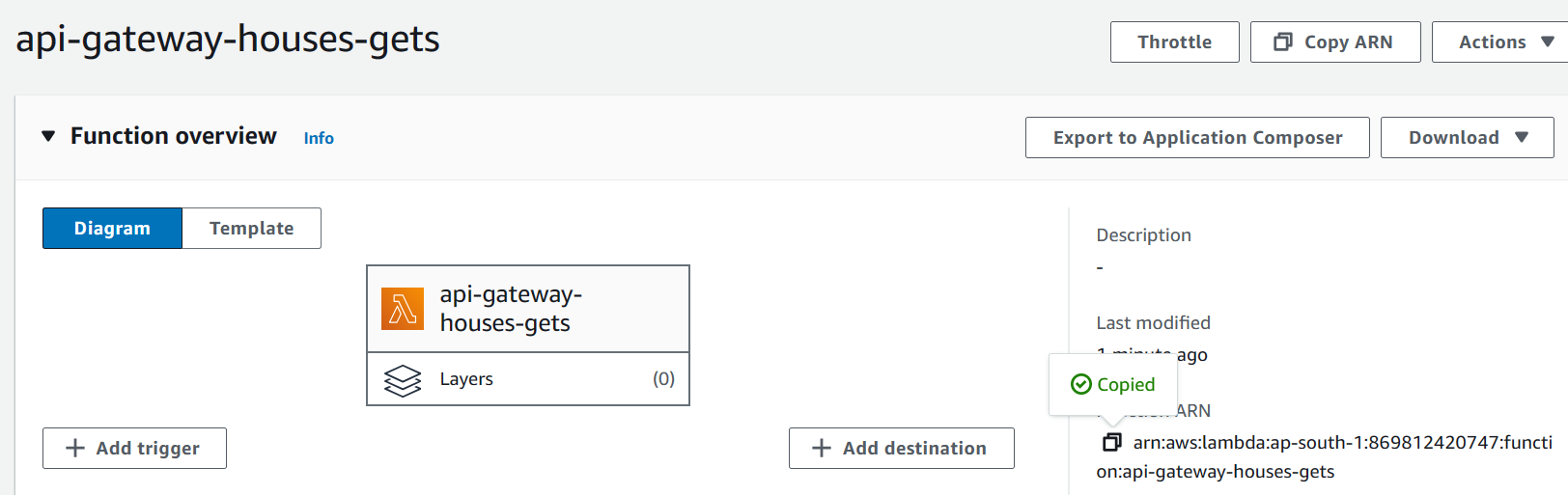
1. **Create a new resource named "houses".**



1. **Create a GET method on the "/houses" resource:**
   * Use Lambda function integration with proxy integration.
2. **Create a Lambda function named "api-gateway-houses-get":**
   * Use Python 3.12 runtime.
   * Copy and paste the previous code, modifying the message to "Hello from my pretty house".
   * Deploy the function.

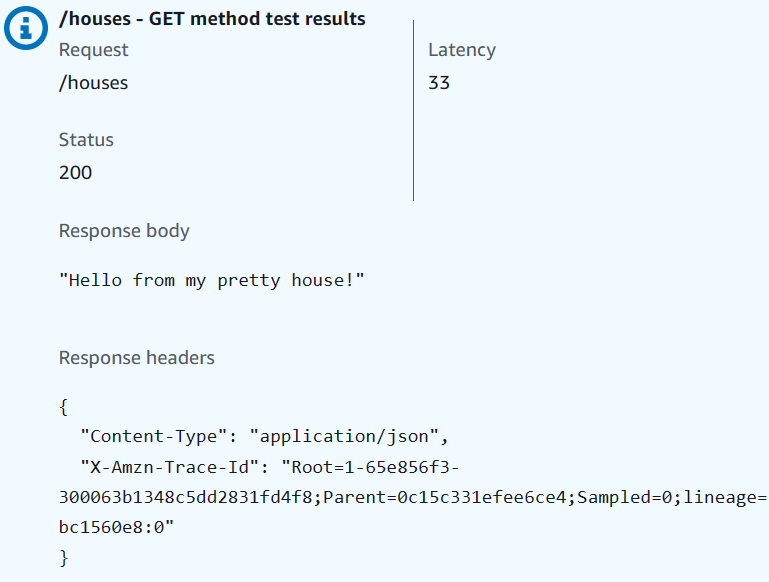


1. **Integrate the Lambda function with the API method:**
   * Copy the function ARN and paste it into the API Gateway integration field.
   * Create the method.

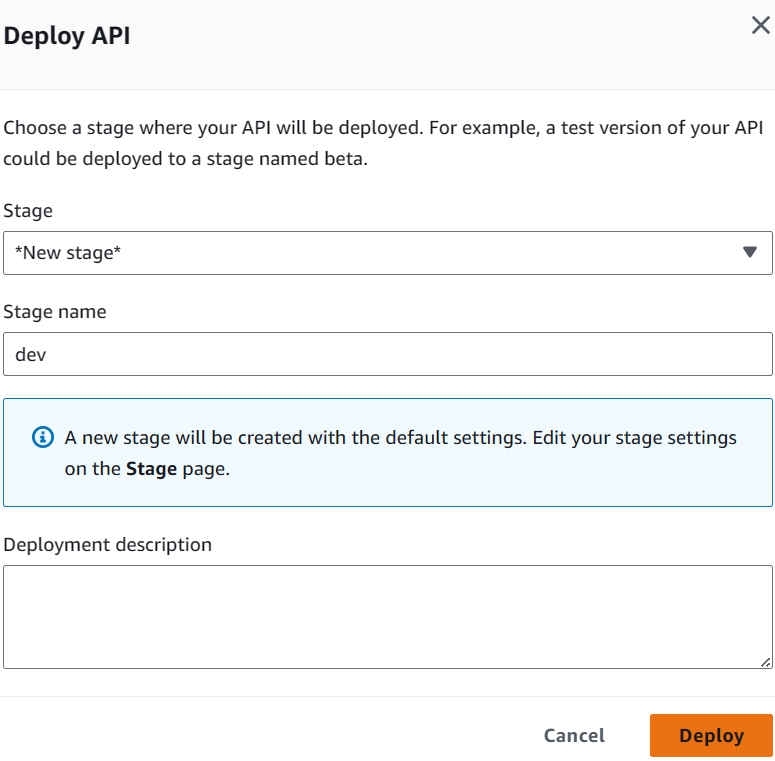


**Testing and Deploying the API:**

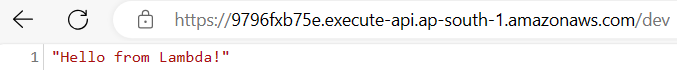
1. **Test the "/houses" method using the API Gateway "Test" feature.**
   * Observe the response:
     + Status code: 200.
     + Body: "Hello from my pretty house".



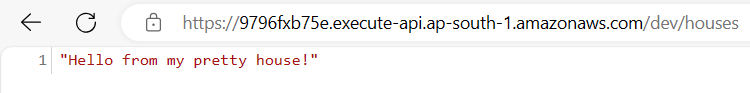
1. **Deploy the API to a new stage named "dev".**



1. **Use the provided Invoke URL to access the API in a web browser:**
   * Access "/dev" to see "Hello from Lambda".



* + Access "/houses" to see "Hello from my pretty house".



* + Access an incorrect path (e.g., "/wrong") to see an error message.

