AWS Serverless, API gateway(SAM) 4th assignment

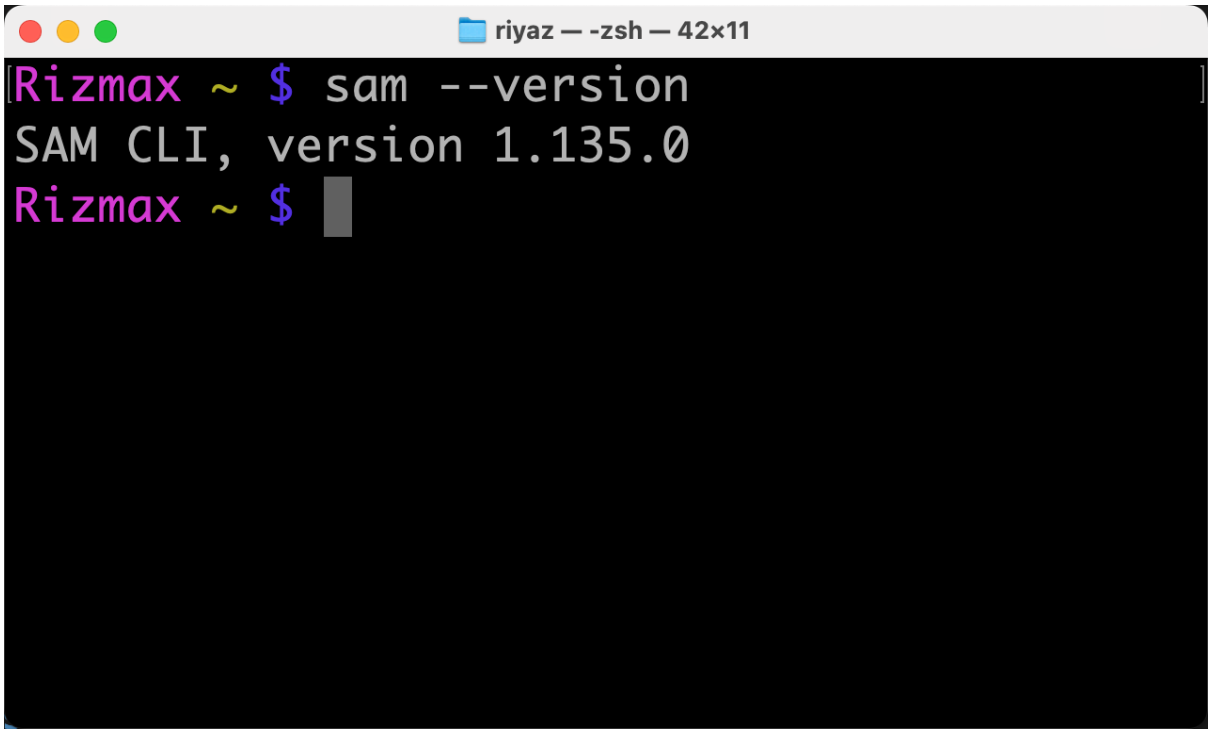
Task 1 - Install and Set Up AWS SAM

AWS Serverless Application Model (SAM) simplifies deploying and managing serverless applications. In this lab, you will use SAM to:

- Define an API Gateway with **GET and POST methods**
- Implement **path parameters and query parameters** in API Gateway
- Create an AWS Lambda function in **Node.js** to process incoming API requests
- Configure **CORS** to allow cross-origin requests
- Deploy and **test API behavior using Postman**, including negative tests

Install AWS SAM CLI

- Follow the [AWS SAM CLI installation guide](#).
- Verify installation by running:
    1. sam --version



Task 2 - Initialize an AWS SAM Project

- Run the command sam init to initialize a new SAM project:
- Choose:
    - (1) AWS Quick Start Templates

- (1) Hello World Example
- (N) Use the most popular runtime and package type?
- (11) Node.js 22.x as the runtime
- (1) ZIP as the package type
- (1) Hello World Example as your starter template
- (y) Enable X-Ray tracing
- (N) CloudWatch Application Insights
- (y) Structured logging in JSON format
- Enter lab-sam-greet-me as the project name

- Navigate into the project folder and open it **VS Code**.

1. cd lab-sam-greet-me
2. code .

```
lab-sam-greet-me — -zsh — 77×19
    Architectures: x86_64
    Dependency Manager: npm
    Application Template: hello-world
    Output Directory: .
    Configuration file: lab-sam-greet-me/samconfig.toml

    Next steps can be found in the README file at lab-sam-greet-me/README.md


Commands you can use next
=========================
[*] Create pipeline: cd lab-sam-greet-me && sam pipeline init --bootstrap
[*] Validate SAM template: cd lab-sam-greet-me && sam validate
[*] Test Function in the Cloud: cd lab-sam-greet-me && sam sync --stack-name
{stack-name} --watch

Rizmax ~ $ cd lab-sam-greet-me
Rizmax lab-sam-greet-me $ code .
Rizmax lab-sam-greet-me $
```

- Make sure you have AWS CLI configured or have set AWS temporary credentials in your terminal session.

Task 3 - Define the API Gateway with GET and POST Methods

Update the SAM template

- Open template.yaml in a code editor.
- Replace its contents with the following code: https://github.com/rizmaxed/sls-labs-code/blob/main/lab-10-4/lab-sam-greet-me/template.yaml
- Save the file.

Understanding the SAM Template

This **AWS SAM template** is the blueprint for deploying your serverless API and its supporting Lambda function. Let's break it down so you can understand what each section does and how it all comes together.

At the top, you'll see AWSTemplateFormatVersion and Transform. These tell AWS that this is a **CloudFormation template** using **SAM** to simplify serverless deployments. Essentially, SAM extends CloudFormation, making it easier to define APIs and Lambda functions with minimal configuration.

Next, under **Globals**, we set some defaults for all Lambda functions in this project. Here, we specify a **timeout of 10 seconds** and allocate **128MB of memory** to each function. These settings ensure the function has enough resources while keeping costs optimized.

Moving on to **Resources**, this is where we define the actual infrastructure. The **GreetMeFunction** is our Lambda function, responsible for handling API requests. It points to app.lambdaHandler, meaning when the API is triggered, AWS will execute the lambdaHandler function in the app.mjs file. The function runs on nodejs22.x, which is the latest stable Node.js runtime at the time of writing. We also attach the **AWSLambdaBasicExecutionRole** to grant the necessary permissions for logging and execution.

Inside the function, we define two API Gateway events. The first one, **GetGreetAPI**, sets up a GET request at /greet/{name}. The {name} is a **path parameter**, meaning users can pass a name directly in the URL. It also accepts an optional **query parameter** called lang, which allows users to specify their preferred language. The second event, **PostGreetAPI**, is a POST request at /greet. This method expects a JSON request body, allowing users to send dynamic input instead of using URL parameters.

Now, let's talk about **Metadata**. This section configures esbuild, a super-fast JavaScript bundler that optimizes the Lambda function for deployment. We enable **minification** and set the target to **es2020**, making sure the function runs efficiently in modern Node.js environments.

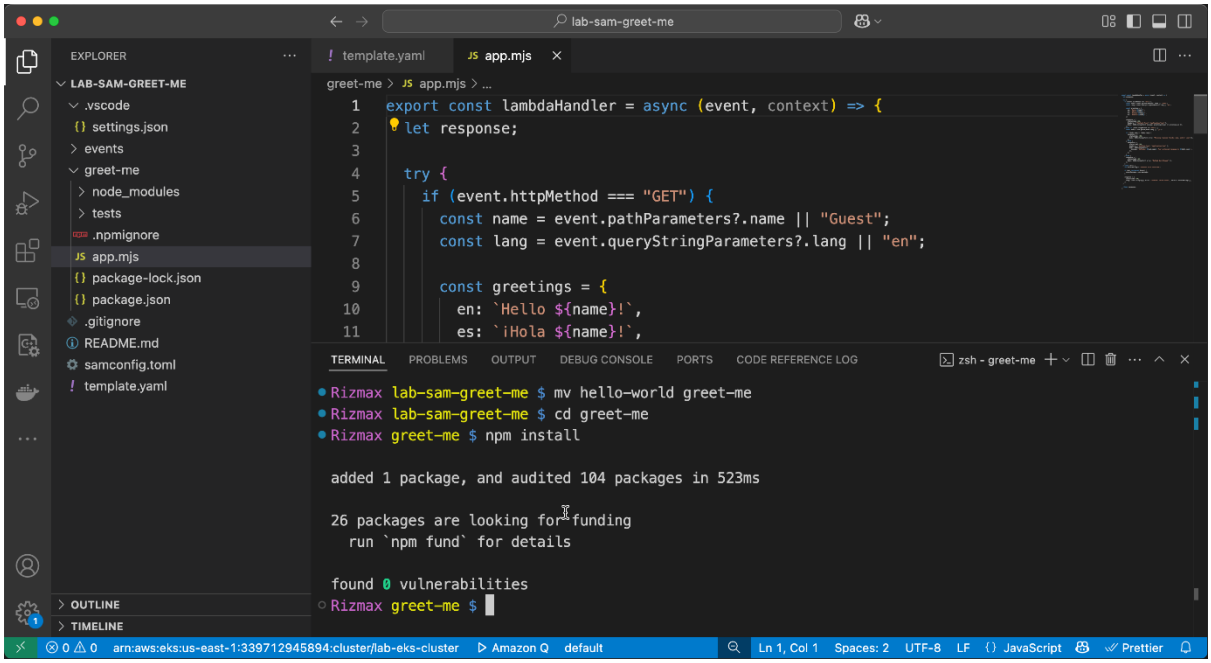
Finally, we have **Outputs**, which provides useful information after deployment.
The GreetMeApi output returns the API Gateway URL, so you can easily test the API once it's deployed. The GreetMeFunctionArn output gives you the function's unique identifier (ARN), which can be useful if you need to reference it elsewhere.

With this understanding, you now have a solid grasp of what this SAM template does. Once deployed, it sets up an API Gateway and Lambda function, allowing users to interact with a simple API. Now, let's move forward and implement the function logic! ?

Task 4 - Implement the Lambda Function

In this task, we will implement the Lambda backend for our API.

- Let's rename the hello-world directory to greet-me, and navigate to it.
    1. mv hello-world greet-me
    2. cd greet-me
- Install dependencies
    1. npm install



- Open app.mjs in a code editor.
- Replace its contents with the following code: https://github.com/rizmaxed/sls-labs-code/blob/main/lab-10-4/lab-sam-greet-me/greet-me/app.mjs
- Save the file.

Understanding the Lambda Function

This **Lambda function** is the heart of your serverless API, handling both GET and POST requests. Let's break it down so you know exactly what's happening.

When a GET request is received at /greet/{name}, the function extracts the **name** from the path parameter and the **lang** parameter from the query string. It then returns a greeting message in the specified language, defaulting to English if no language is provided.

For POST requests to /greet, the function expects a JSON body containing **name** and **lang** fields. If either of these fields is missing, it returns a 400 Bad Request error. Otherwise, it constructs a welcome message and returns it in the response.

Finally, the function includes basic **error handling**. If an unsupported HTTP method is used, it returns 405 Method Not Allowed. If something unexpected happens, it returns a 500 Internal Server Error with a helpful error message.

This function ensures clean and structured request handling while keeping the logic simple and effective. Now, let's move on to testing it locally! ?

Test the Lambda function locally

Note that you must have Docker installed and running. If not already done, you can install Docker Desktop on your local computer.

- Make sure that you are in the project root directory i.e. ~/lab-sam-greet-me/. If you're in the ./greet-me/ folder, use cd .. to go a level up.

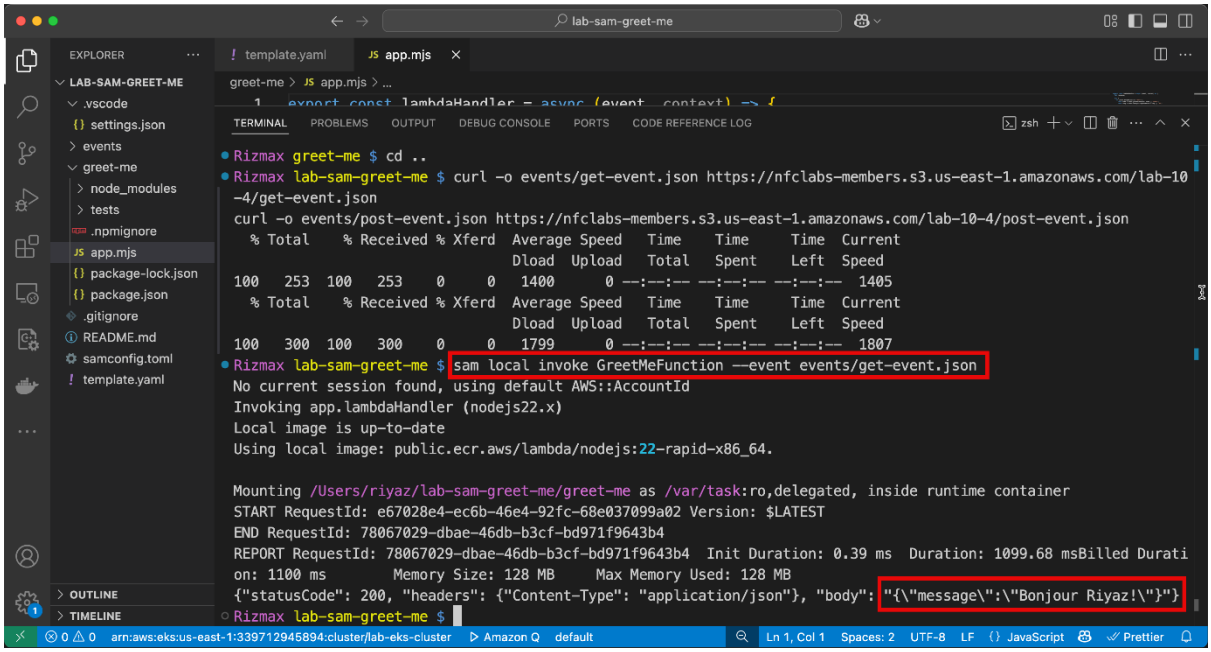
- Download the sample event data files and review them. The event JSON uses standard event structure for an API Gateway request.

1. curl -o events/get-event.json https://nfclabs-members.s3.us-east-1.amazonaws.com/lab-10-4/get-event.json

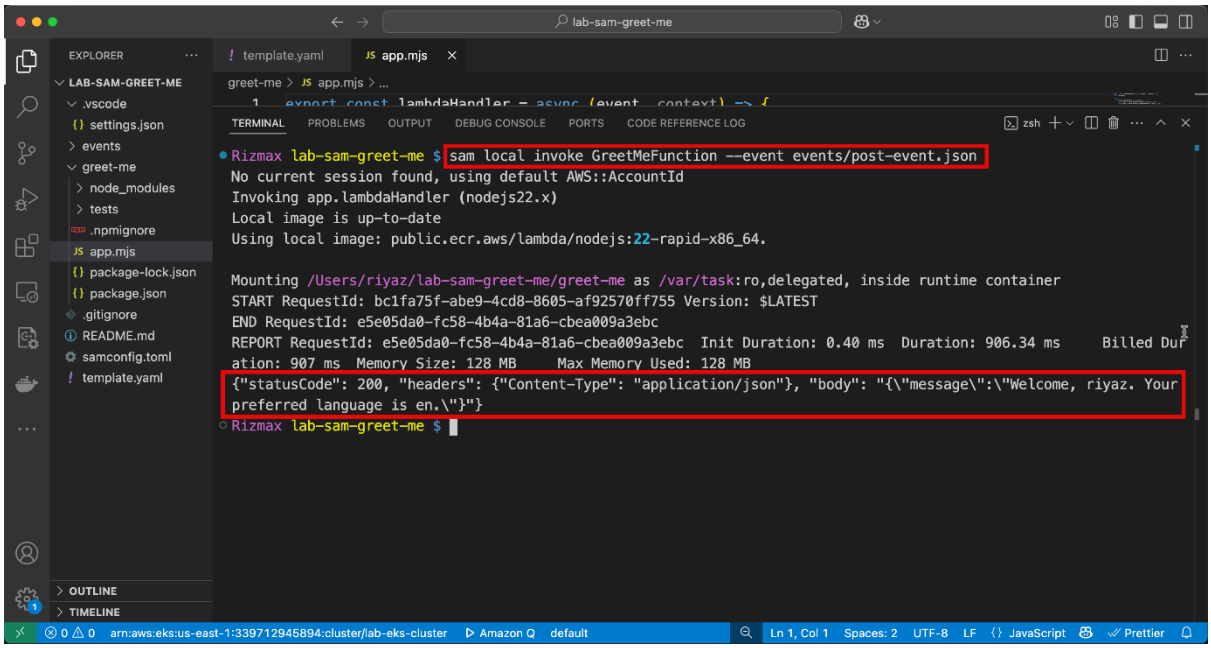
2. curl -o events/post-event.json https://nfclabs-members.s3.us-east-1.amazonaws.com/lab-10-4/post-event.json

- Run the following command.

1. sam local invoke GreetMeFunction --event events/get-event.json



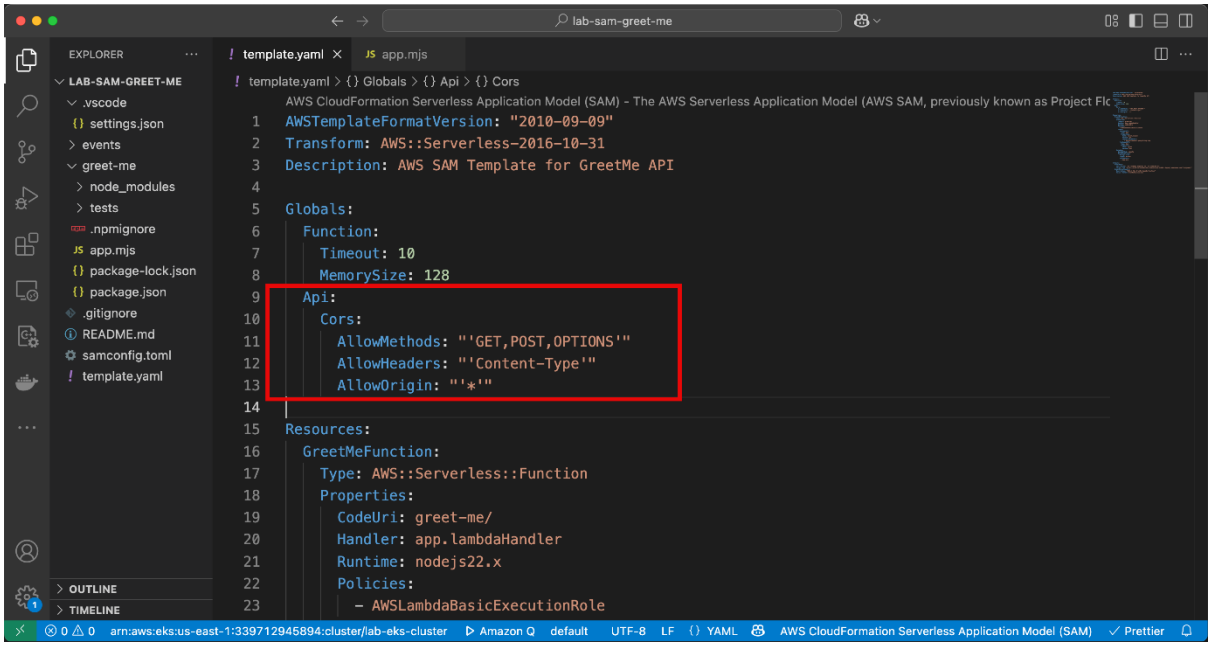
1. sam local invoke GreetMeFunction --event events/post-event.json



Task 5 - Enable CORS

- Modify template.yaml and add the following under Globals section:

1. Api:
2. Cors:
3. AllowMethods: "'GET,POST,OPTIONS'"
4. AllowHeaders: "'Content-Type'"
5. AllowOrigin: "'*'"



Task 6 - Deploy and Test the API

Note: if you do not have esbuild installed on your local computer, install it using the command npm install -g esbuild

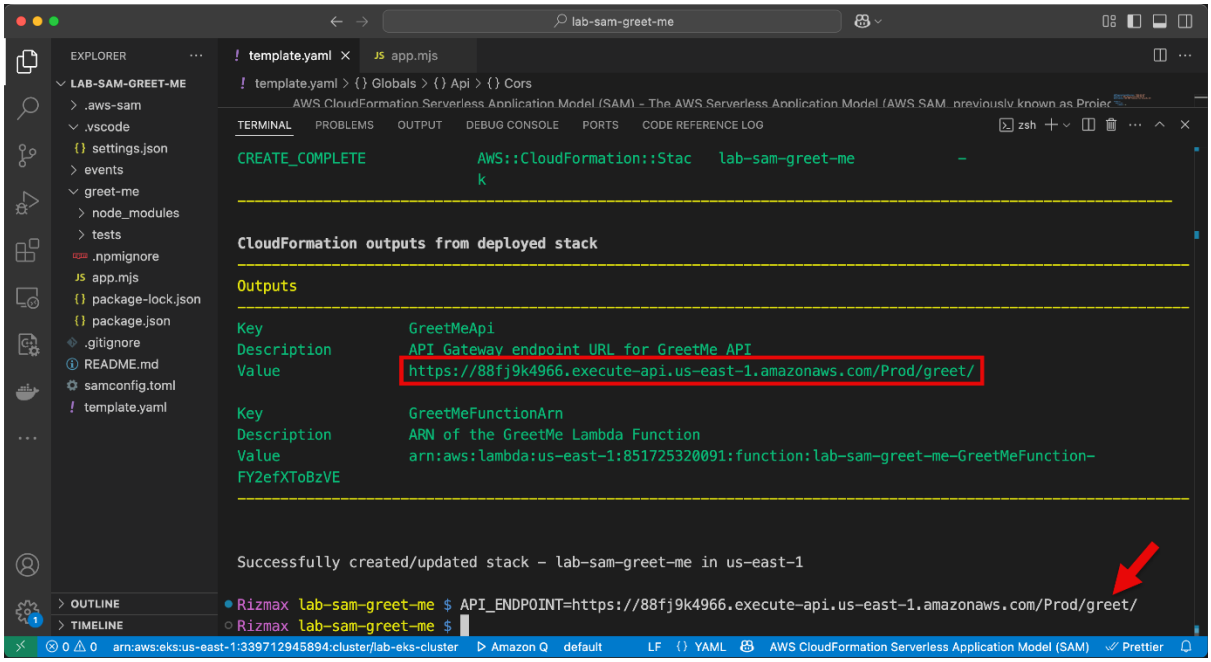
- Make sure you have AWS CLI configured or have set AWS temporary credentials in your terminal session.
- Build the application:

1. sam build
1. sam deploy

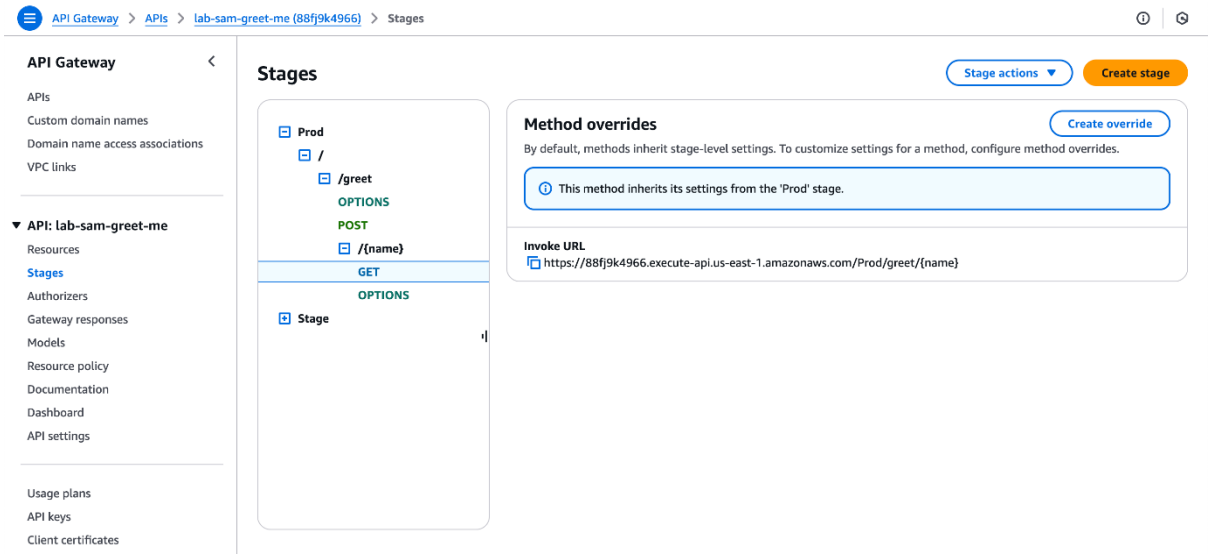
Enter y for any confirmation prompts.

Once deployed, copy the GreetMeApi Endpoint URL from the **Outputs** section. Set the GreetMeApi Endpoint URL into an environment variable API_ENDPOINT

1. API_ENDPOINT=https://your-api-id.execute-api.region.amazonaws.com/Prod/greet/



You may also view the API and the Lambda function in the AWS Console.

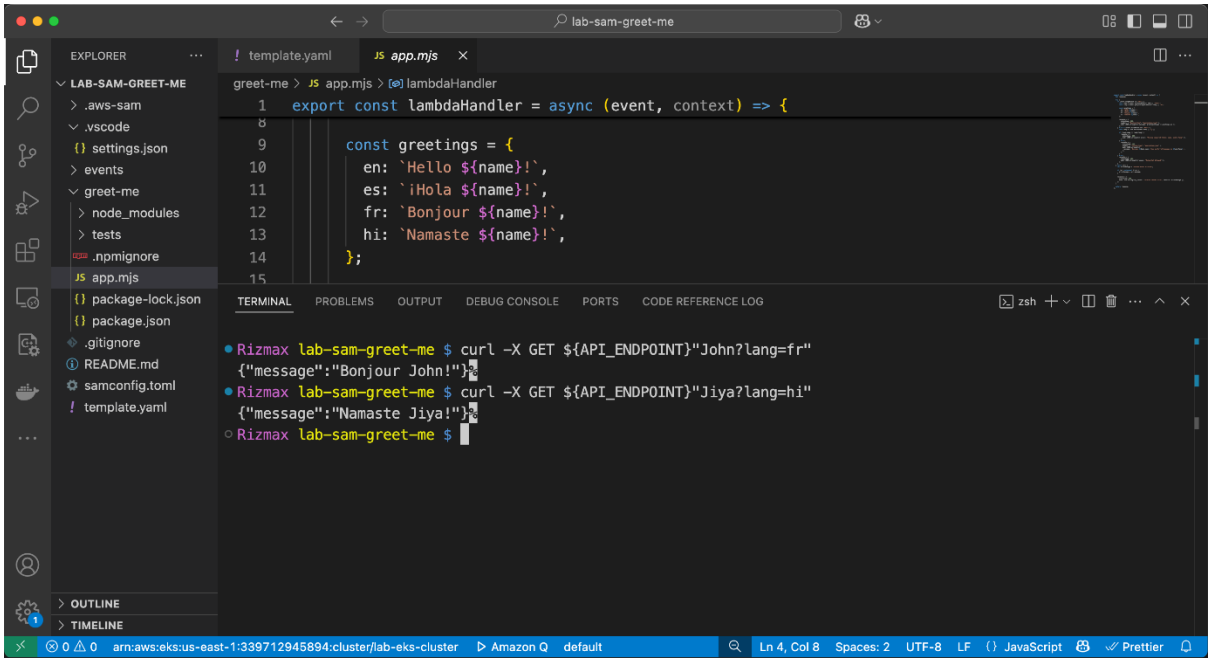


Test Logging in CloudWatch

- Invoke the API using **cURL**:

1. curl -X GET ${API_ENDPOINT}"John?lang=fr"

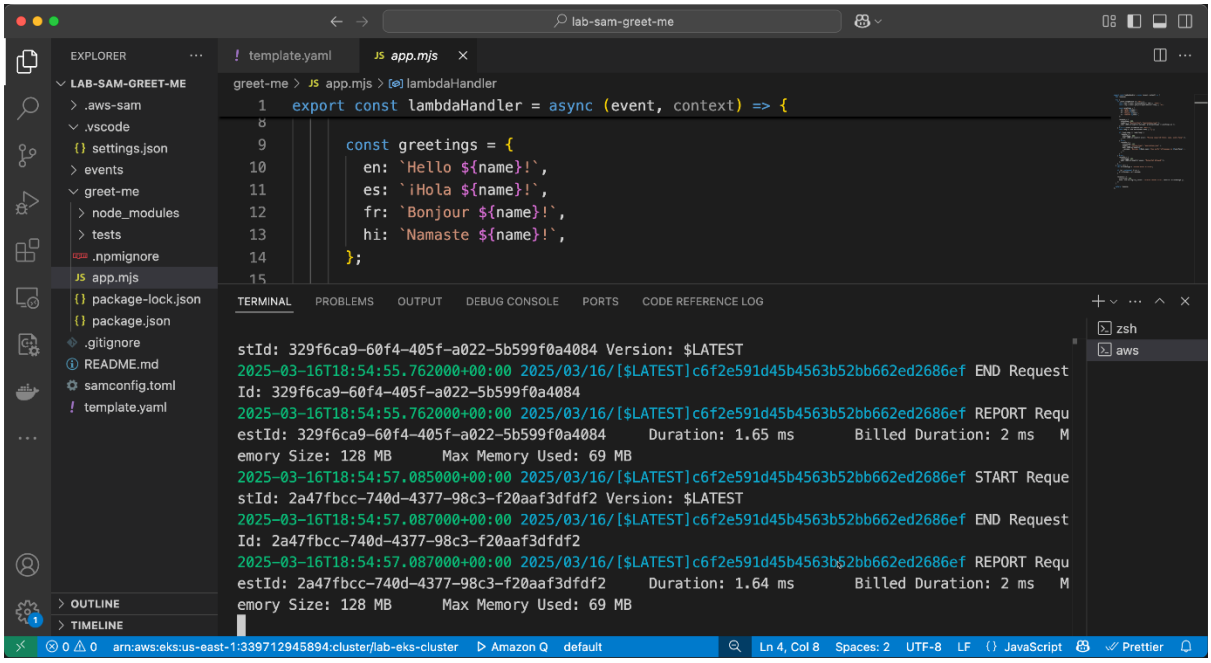
1. curl -X GET ${API_ENDPOINT}"Jiya?lang=hi"



- Open another terminal tab and check the logs tail using **AWS CloudWatch** (Make sure you have credentials available in the terminal if using temporary AWS credentials)

1. aws logs tail $(aws logs describe-log-groups --query "logGroups[?contains(logGroupName, 'GreetMeFunction')].logGroupName" --output text) --follow

- Invoke the API a few times from the other terminal tab to see log tail in action.



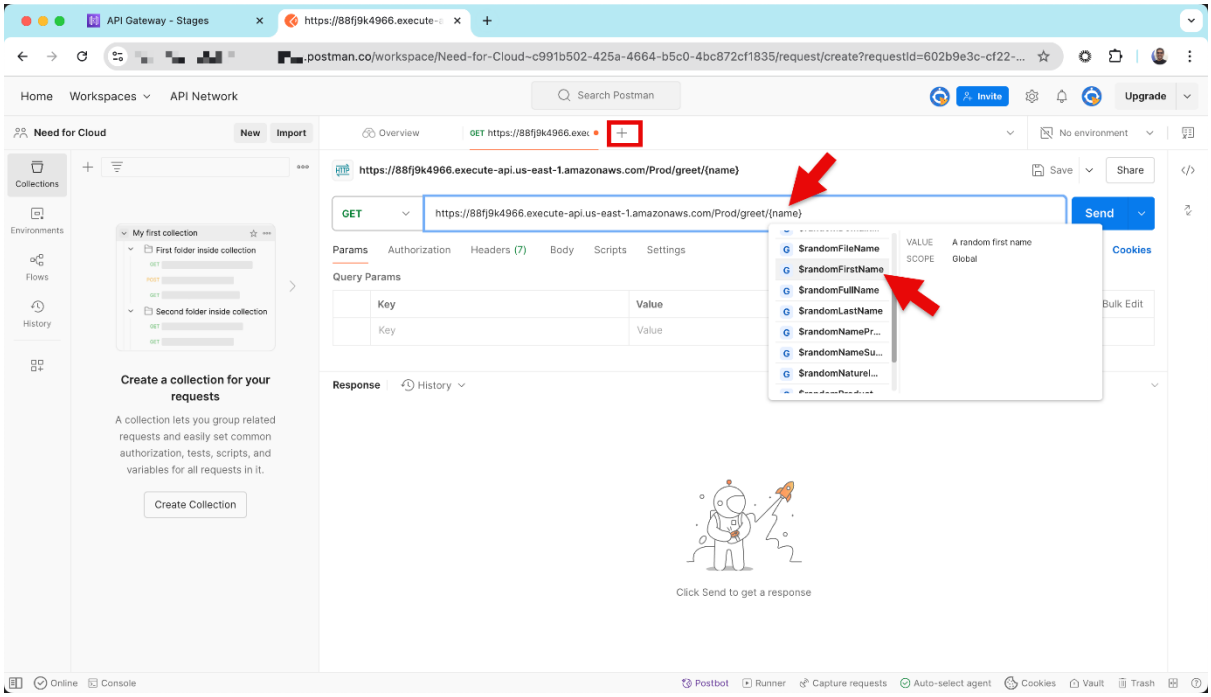
- Use Ctrl + C to exit.

Task 7 - Test API Using Postman

Now that the API is deployed, it's time to test its functionality using **Postman**. You will perform various tests to validate that API Gateway and Lambda process requests correctly.

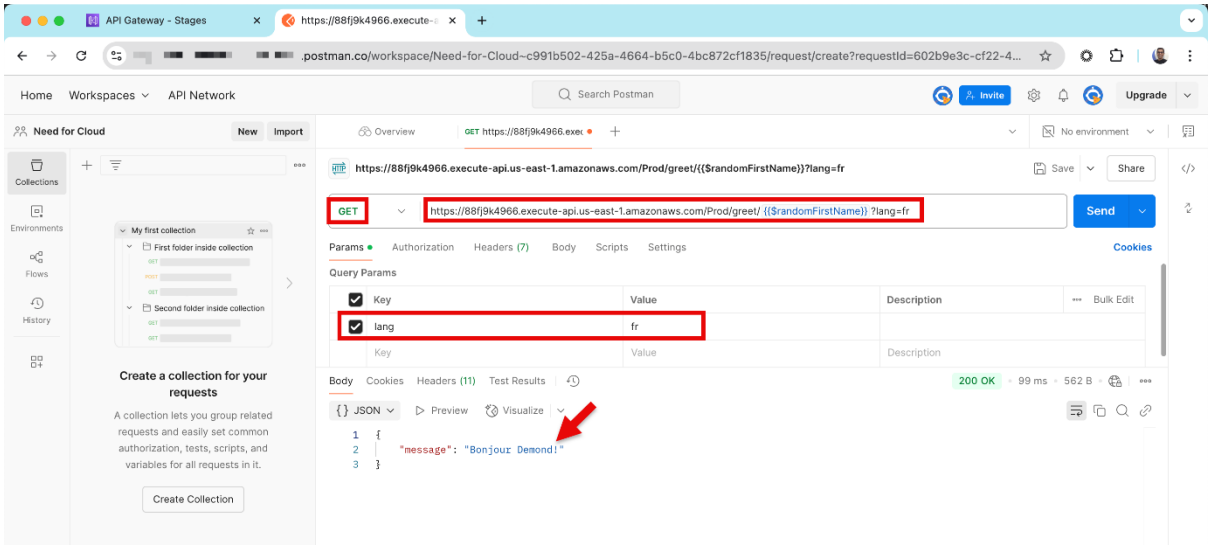
Test a valid GET request

- Open **Postman** app, either installed locally or via web browser at https://www.postman.com/ and login to it.
- Create a workspace and inside the workspace, use the + option to create a new request.
- Create a **GET request** to your API endpoint. e.g. https://your-api-id.execute-api.us-east-1.amazonaws.com/Prod/greet/{name}.
- Within Postman, replace {name} with {{$randomFirstName}}



- Add lang=fr as Query Params.
- Click **Send** and verify the response

1. { "message": "Bonjour John!" }



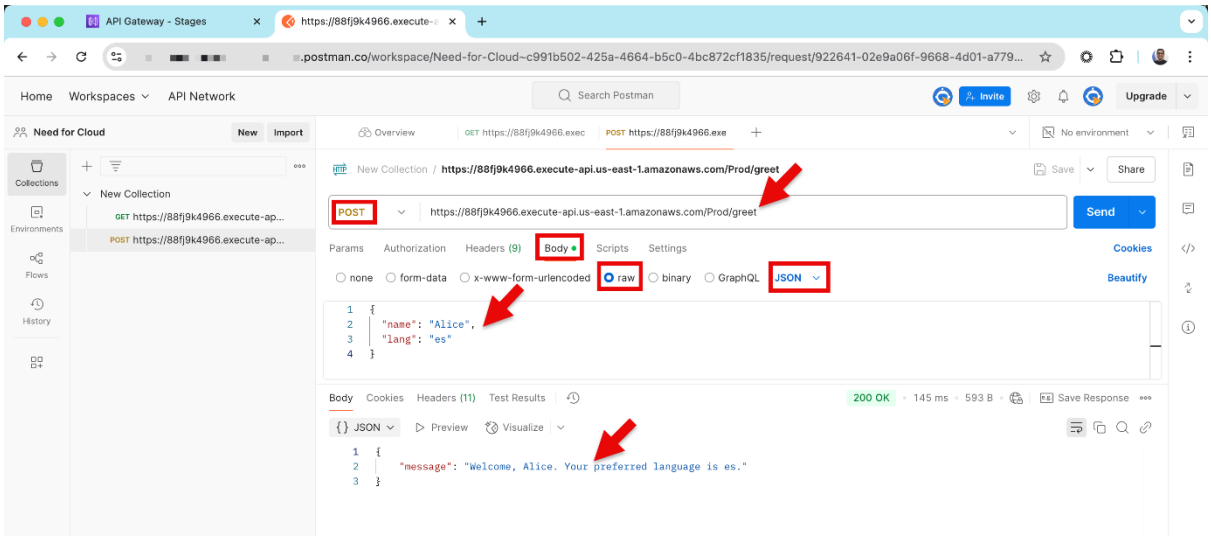
Test a valid POST request

- In **Postman**, create a **POST request** to https://your-api-id.execute-api.region.amazonaws.com/Prod/greet
- Set the **Content-Type** to application/json
- In the **Body** tab, select **raw** and enter the following JSON

1. {
2. "name": "Alice",
3. "lang": "es"
4. }

- Click **Send** and verify the response

1. { "message": "Welcome, Alice. Your preferred language is es." }



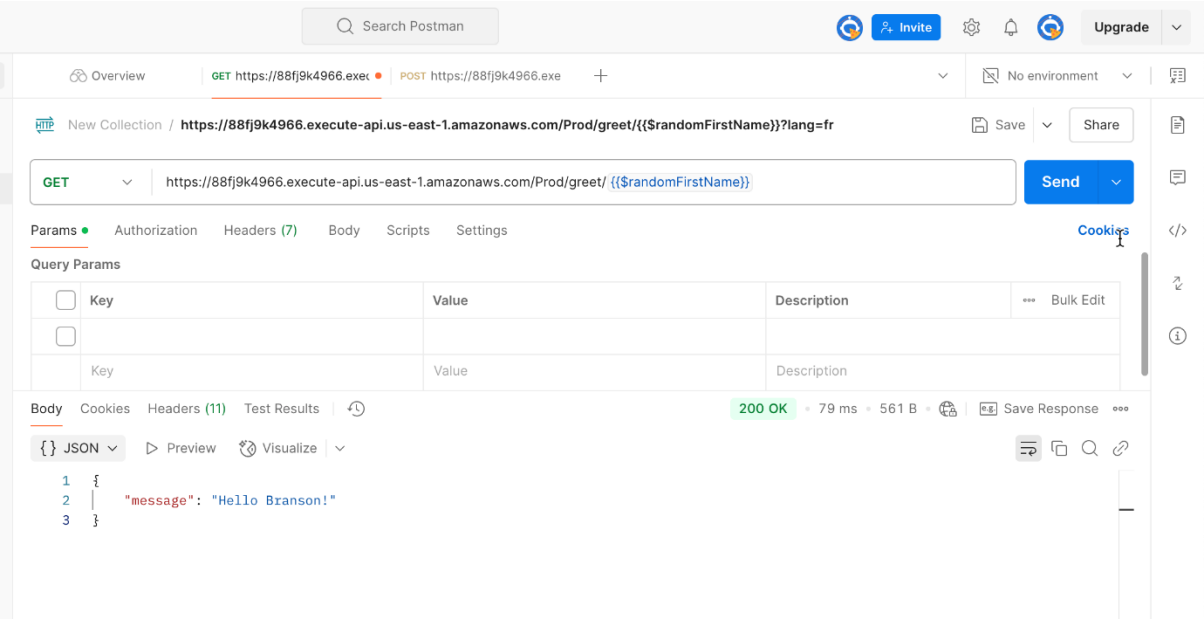
Test missing query parameters in GET request

- Remove the lang query parameter from the **GET request**

1. https://your-api-id.execute-api.region.amazonaws.com/Prod/greet/John

- Click **Send** and verify the response

1. { "message": "Hello John!" }



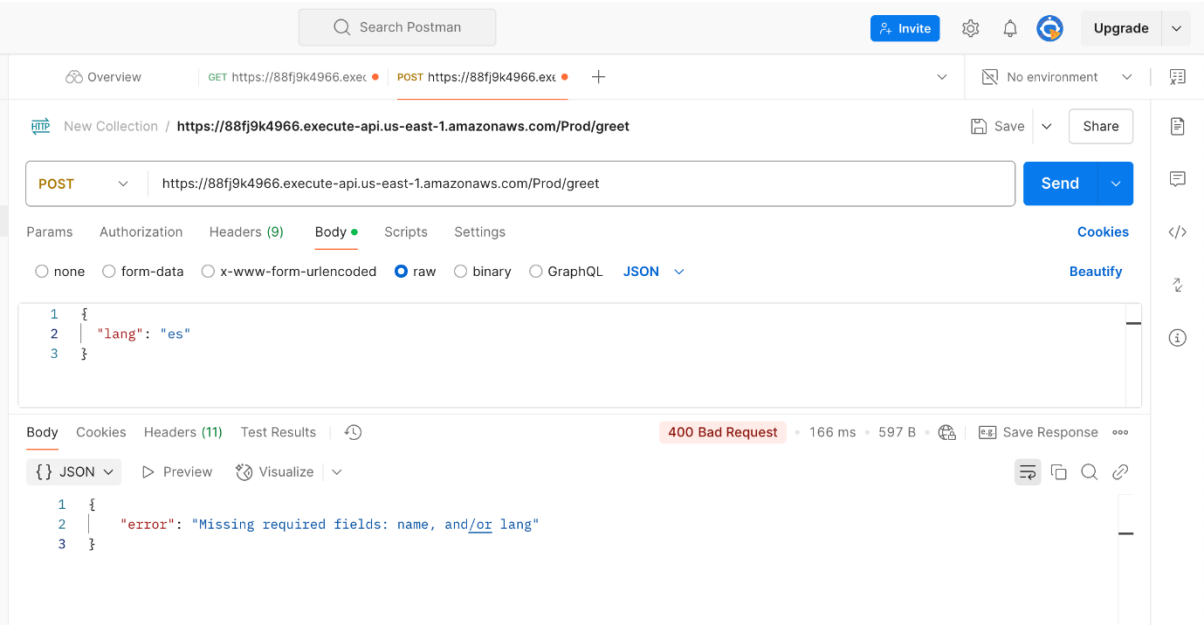
Test missing required fields in POST request

- In **Postman**, send a **POST request** but remove the name field

1. {
2. "lang": "es"
3. }

- Click **Send** and verify the response. You should receive HTTP 400 Bad Request error.

1. { "error": "Missing required fields: name, lang" }



Verify logs in CloudWatch

- Run the following command in the terminal to check the logs

1. aws logs tail $(aws logs describe-log-groups --query "logGroups[?contains(logGroupName, 'GreetMeFunction')].logGroupName" --output text) --follow

By completing these tests, you have verified that API Gateway and Lambda process different types of requests correctly while handling errors appropriately.

Task 8 - Clean Up Resources

- Optionally, delete all deployed resources using the following command:

1. sam delete --stack-name lab-sam-greet-me

- Enter y for any confirmation prompts.