

AWS Serverless, API gateway(SAM) 4th assignment

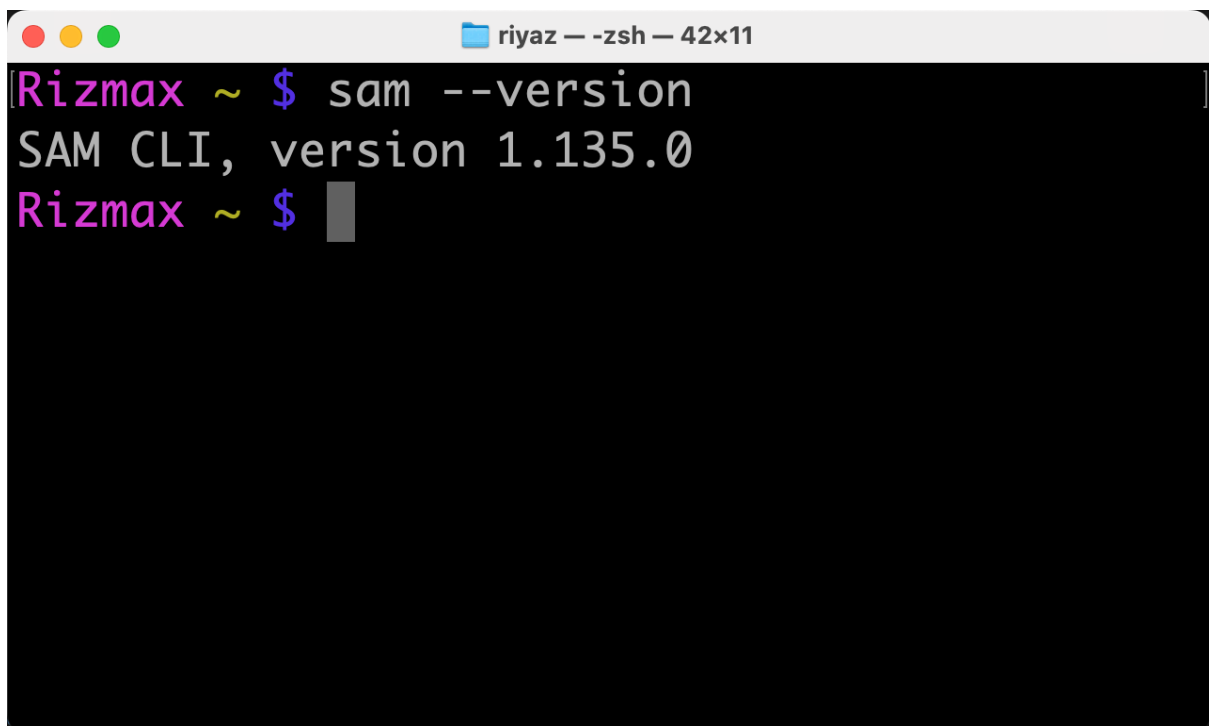
Task 1 - Install and Set Up AWS SAM

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`

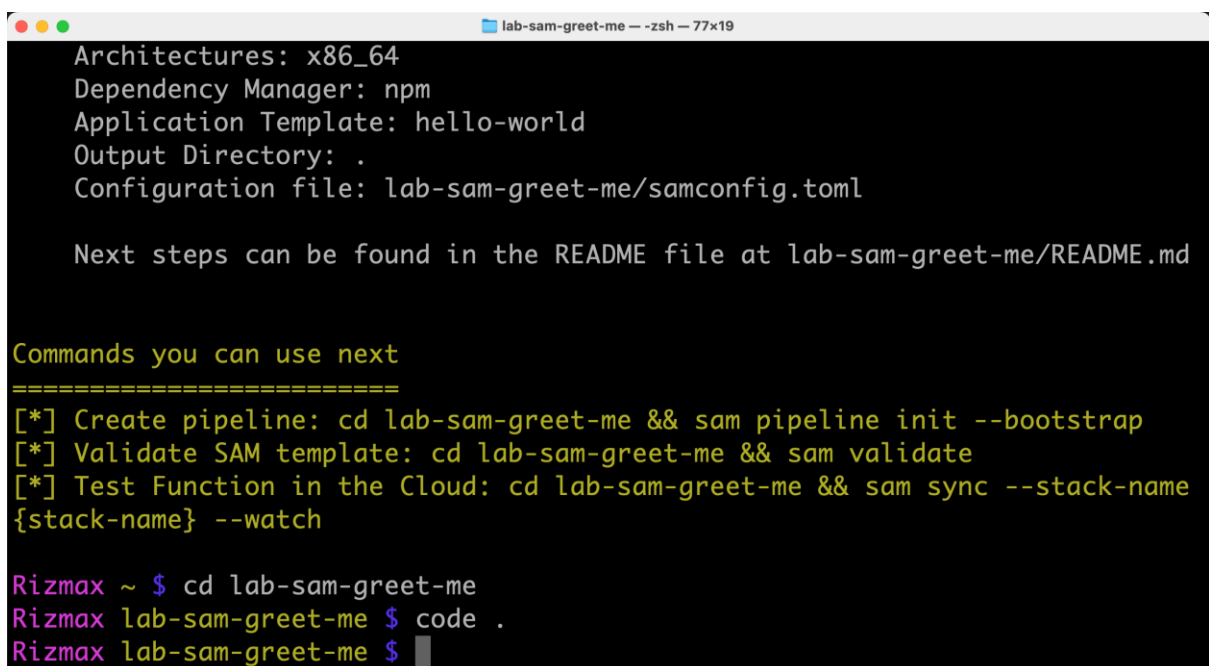
A terminal window titled 'riyaz — -zsh — 42x11' shows a user named 'Rizmax' at a prompt '~ \$' running the command 'sam --version'. The output is 'SAM CLI, version 1.135.0'. The prompt then returns to '~ \$' with a cursor.

```
Rizmax ~ $ sam --version
SAM CLI, version 1.135.0
Rizmax ~ $
```

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 — 77x19
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`

The screenshot shows a VS Code editor with a project named 'lab-sam-greet-me'. The Explorer sidebar on the left shows the file structure, including 'app.mjs'. The main editor area displays the content of 'app.mjs', which is a JavaScript Lambda function handler. The terminal window at the bottom shows the following commands and output:

```
greet-me > JS app.mjs > ...
1 export const lambdaHandler = async (event, context) => {
2   let response;
3
4   try {
5     if (event.httpMethod === "GET") {
6       const name = event.pathParameters?.name || "Guest";
7       const lang = event.queryStringParameters?.lang || "en";
8
9       const greetings = {
10         en: `Hello ${name}!`,
11         es: `¡Hola ${name}!`,
```

```
Rizmax lab-sam-greet-me $ mv hello-world greet-me
Rizmax lab-sam-greet-me $ cd greet-me
Rizmax greet-me $ npm install

added 1 package, and audited 104 packages in 523ms

26 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
Rizmax greet-me $
```

- 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`

```

greet-me > JS app.mjs > ...
1 export const lambdaHandler = async (event, context) => {
Rizmax greet-me $ cd ..
Rizmax lab-sam-greet-me $ curl -o events/get-event.json https://nfclabs-members.s3.us-east-1.amazonaws.com/lab-10-4/get-event.json
curl -o events/post-event.json https://nfclabs-members.s3.us-east-1.amazonaws.com/lab-10-4/post-event.json
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 253 100 253 0 0 1400 0 --:--:-- --:--:-- --:--:-- 1405
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 300 100 300 0 0 1799 0 --:--:-- --:--:-- --:--:-- 1807
Rizmax lab-sam-greet-me $ sam local invoke GreetMeFunction --event events/get-event.json
No current session found, using default AWS::AccountId
Invoking app.lambdaHandler (nodejs22.x)
Local image is up-to-date
Using local image: public.ecr.aws/lambda/nodejs:22-rapid-x86_64.

Mounting /Users/riyaz/lab-sam-greet-me/greet-me as /var/task:ro,delegated, inside runtime container
START RequestId: e67028e4-ec6b-46e4-92fc-68e037099a02 Version: $LATEST
END RequestId: 78067029-dbae-46db-b3cf-bd971f9643b4
REPORT RequestId: 78067029-dbae-46db-b3cf-bd971f9643b4 Init Duration: 0.39 ms Duration: 1099.68 ms Billed Duration: 1100 ms Memory Size: 128 MB Max Memory Used: 128 MB
{"statusCode": 200, "headers": {"Content-Type": "application/json"}, "body": "{\"message\": \"Bonjour Riyaz!\"}"}
Rizmax lab-sam-greet-me $

```

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

```

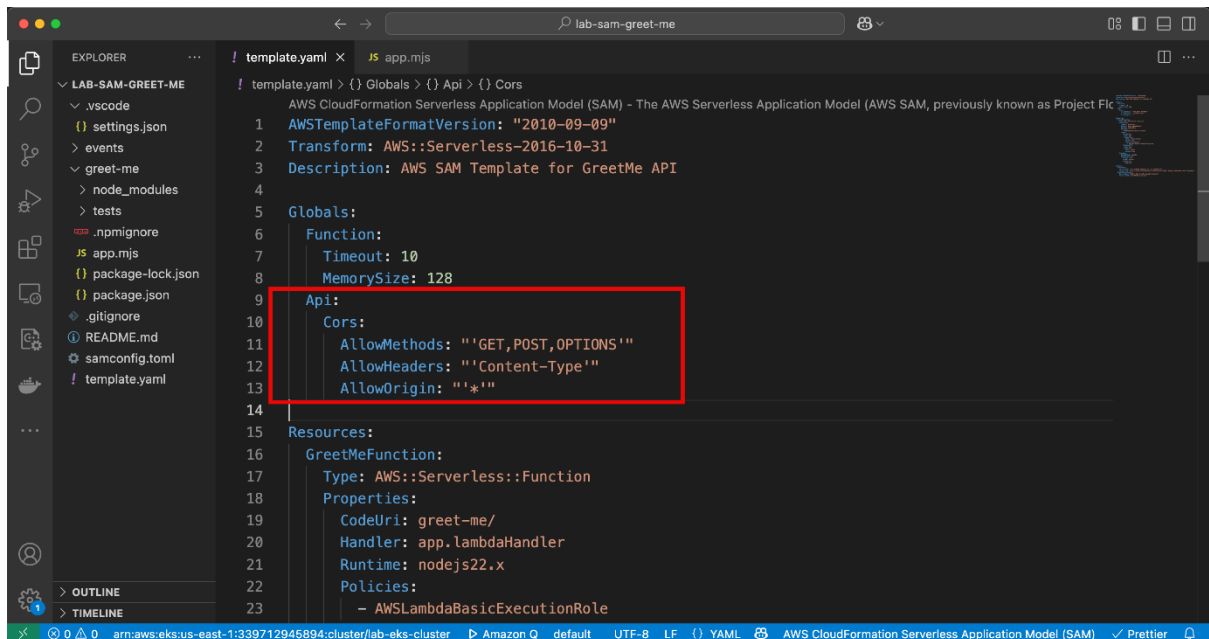
greet-me > JS app.mjs > ...
1 export const lambdaHandler = async (event, context) => {
Rizmax lab-sam-greet-me $ sam local invoke GreetMeFunction --event events/post-event.json
No current session found, using default AWS::AccountId
Invoking app.lambdaHandler (nodejs22.x)
Local image is up-to-date
Using local image: public.ecr.aws/lambda/nodejs:22-rapid-x86_64.

Mounting /Users/riyaz/lab-sam-greet-me/greet-me as /var/task:ro,delegated, inside runtime container
START RequestId: bc1fa75f-abe9-4cd8-8605-af92570ff755 Version: $LATEST
END RequestId: e5e05da0-fc58-4b4a-81a6-cbea009a3ebc
REPORT RequestId: e5e05da0-fc58-4b4a-81a6-cbea009a3ebc Init Duration: 0.40 ms Duration: 906.34 ms Billed Duration: 907 ms Memory Size: 128 MB Max Memory Used: 128 MB
{"statusCode": 200, "headers": {"Content-Type": "application/json"}, "body": "{\"message\": \"Welcome, riyaz. Your preferred language is en.\"}"}
Rizmax lab-sam-greet-me $

```

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: "'*'"



```

1  template.yaml > {} Globals > {} Api > {} Cors
2  AWS CloudFormation Serverless Application Model (SAM) - The AWS Serverless Application Model (AWS SAM, previously known as Project Fl
3  AWSTemplateFormatVersion: "2010-09-09"
4  Transform: AWS::Serverless-2016-10-31
5  Description: AWS SAM Template for GreetMe API
6
7  Globals:
8    Function:
9      Timeout: 10
10     MemorySize: 128
11   Api:
12     Cors:
13       AllowMethods: "'GET,POST,OPTIONS'"
14       AllowHeaders: "'Content-Type'"
15       AllowOrigin: "'*'"
16
17   Resources:
18     GreetMeFunction:
19       Type: AWS::Serverless::Function
20       Properties:
21         CodeUri: greet-me/
22         Handler: app.lambdaHandler
23         Runtime: nodejs22.x
24         Policies:
25           - AWSLambdaBasicExecutionRole

```

Task 6 - Deploy and Test the API

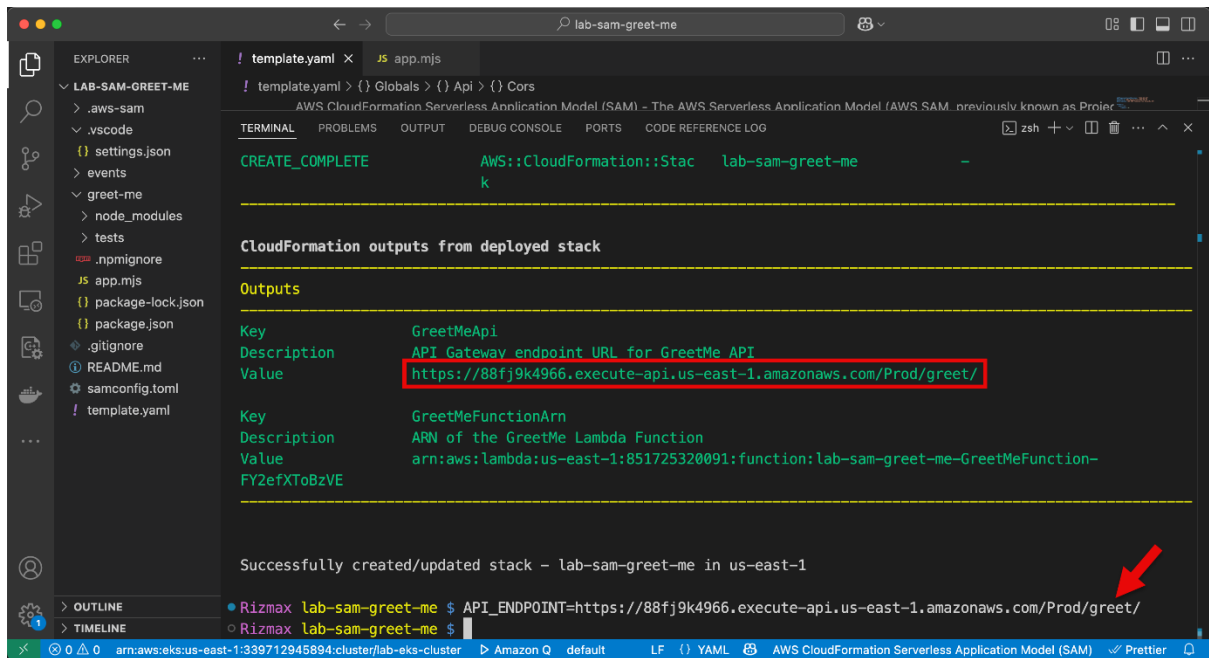
Note: if you do not have esbuild installed on your local computer, install it using the command `npm install -g esbuild`

- 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.

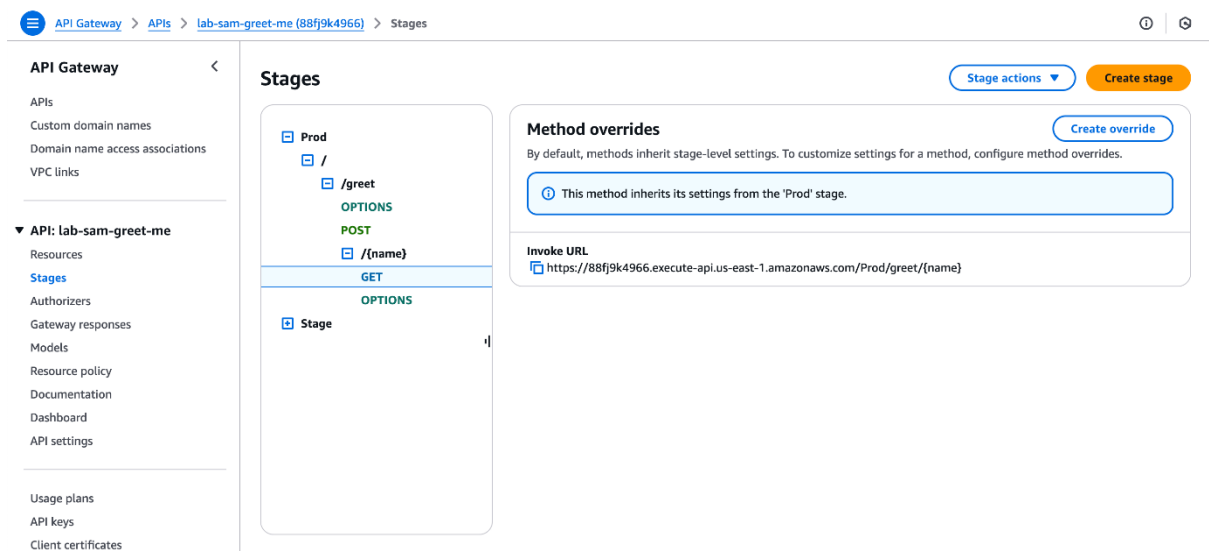
Once deployed, copy the GreetMeApi Endpoint URL from the **Outputs** section. Set the GreetMeApi Endpoint URL into an environment variable `API_ENDPOINT`

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



```
! template.yaml x JS app.mjs
! template.yaml | {} Globals > {} Api > {} Cors
AWS CloudFormation Serverless Application Model (SAM) - The AWS Serverless Application Model (AWS SAM, previously known as Project
TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE PORTS CODE REFERENCE LOG
CREATE_COMPLETE AWS::CloudFormation::Stack lab-sam-greet-me
k
CloudFormation outputs from deployed stack
Outputs
Key GreetMeApi
Description API Gateway endpoint URL for GreetMe API
Value https://88fj9k4966.execute-api.us-east-1.amazonaws.com/Prod/greet/
Key GreetMeFunctionArn
Description ARN of the GreetMe Lambda Function
Value arn:aws:lambda:us-east-1:851725320091:function:lab-sam-greet-me-GreetMeFunction-
FY2efXT0BzVE
Successfully created/updated stack - lab-sam-greet-me in us-east-1
Rizmax lab-sam-greet-me $ API_ENDPOINT=https://88fj9k4966.execute-api.us-east-1.amazonaws.com/Prod/greet/
Rizmax lab-sam-greet-me $
```

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"`

```

greet-me > JS app.mjs > lambdaHandler
1  export const lambdaHandler = async (event, context) => {
8
9      const greetings = {
10         en: 'Hello ${name}!',
11         es: '¡Hola ${name}!',
12         fr: 'Bonjour ${name}!',
13         hi: 'Namaste ${name}!',
14     };
15
16     // ... (rest of the function)

```

```

Rizmax lab-sam-greet-me $ curl -X GET ${API_ENDPOINT}"John?lang=fr"
{"message":"Bonjour John!"}
Rizmax lab-sam-greet-me $ curl -X GET ${API_ENDPOINT}"Jiya?lang=hi"
{"message":"Namaste Jiya!"}
Rizmax lab-sam-greet-me $

```

- 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.

```

stId: 329f6ca9-60f4-405f-a022-5b599f0a4084 Version: $LATEST
2025-03-16T18:54:55.762000+00:00 2025/03/16/[$LATEST]c6f2e591d45b4563b52bb662ed2686ef END Request
Id: 329f6ca9-60f4-405f-a022-5b599f0a4084
2025-03-16T18:54:55.762000+00:00 2025/03/16/[$LATEST]c6f2e591d45b4563b52bb662ed2686ef REPORT Reque
stId: 329f6ca9-60f4-405f-a022-5b599f0a4084 Duration: 1.65 ms Billed Duration: 2 ms M
emory Size: 128 MB Max Memory Used: 69 MB
2025-03-16T18:54:57.085000+00:00 2025/03/16/[$LATEST]c6f2e591d45b4563b52bb662ed2686ef START Reque
stId: 2a47fbcc-740d-4377-98c3-f20aaf3dffd2 Version: $LATEST
2025-03-16T18:54:57.087000+00:00 2025/03/16/[$LATEST]c6f2e591d45b4563b52bb662ed2686ef END Request
Id: 2a47fbcc-740d-4377-98c3-f20aaf3dffd2
2025-03-16T18:54:57.087000+00:00 2025/03/16/[$LATEST]c6f2e591d45b4563b52bb662ed2686ef REPORT Reque
stId: 2a47fbcc-740d-4377-98c3-f20aaf3dffd2 Duration: 1.64 ms Billed Duration: 2 ms M
emory Size: 128 MB Max Memory Used: 69 MB

```

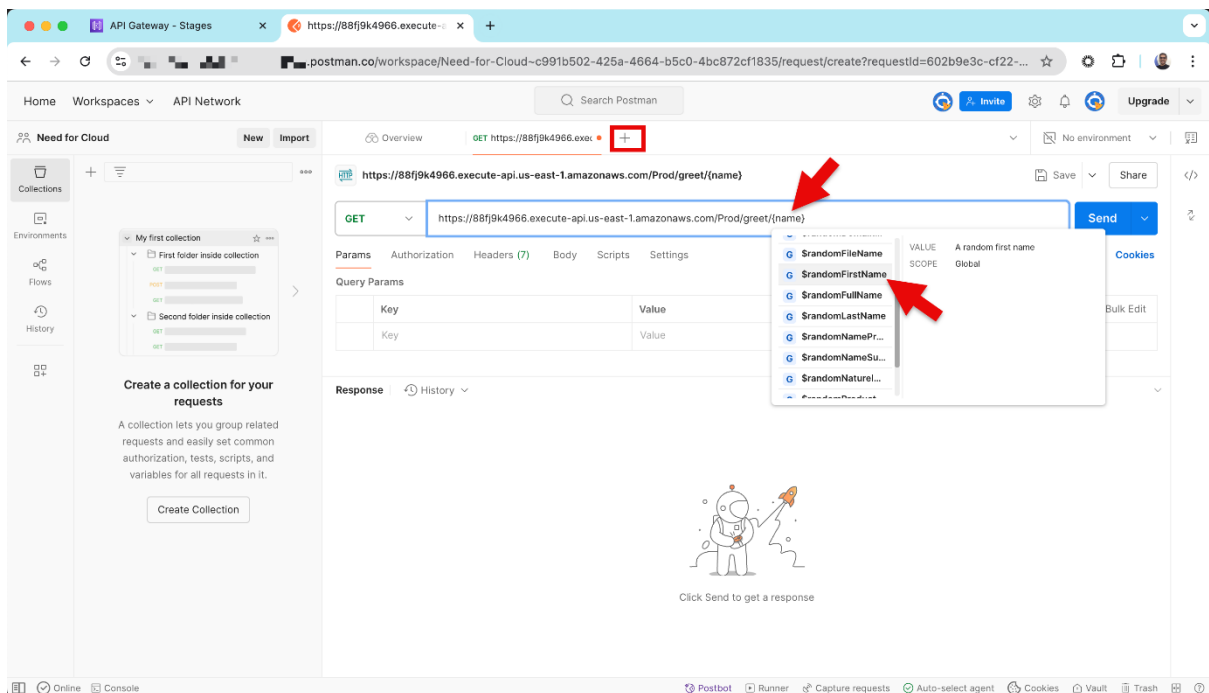
- Use **Ctrl + C** to exit.

Task 7 - Test API Using Postman

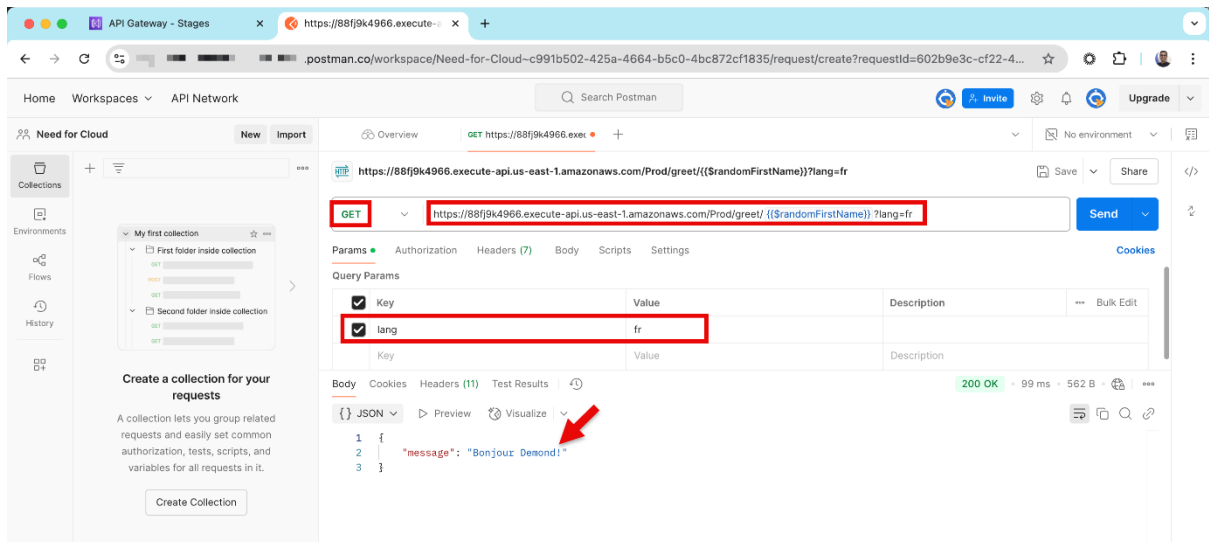
Now that the API is deployed, it's time to test its functionality using **Postman**. You will perform various tests to validate that API Gateway and Lambda process requests correctly.

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 }}`

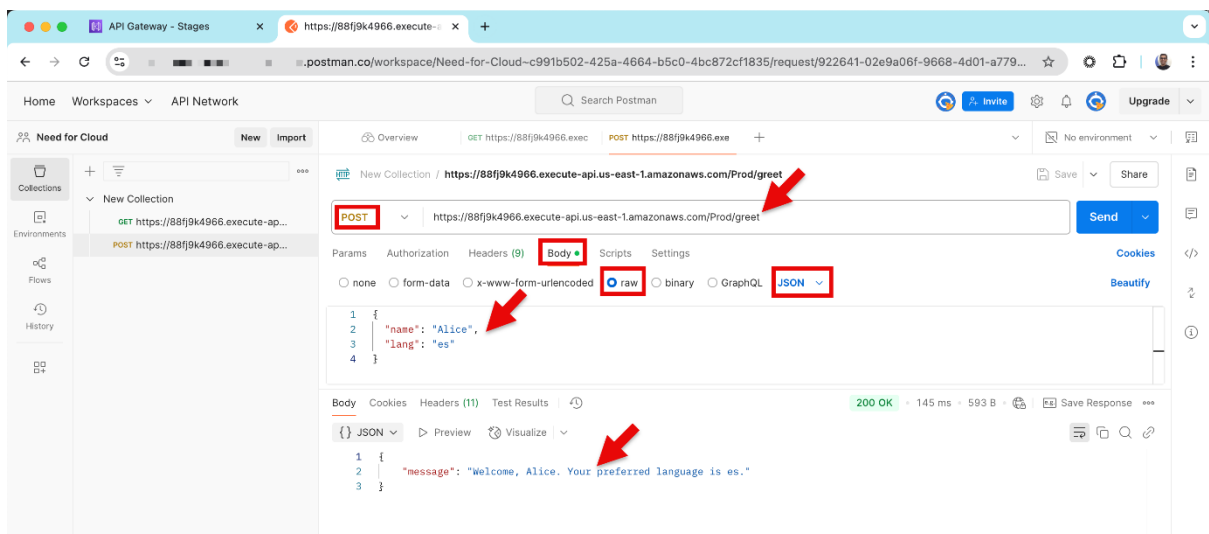


- Add `lang=fr` as Query Params.
 - Click **Send** and verify the response
1. `{ "message": "Bonjour John!" }`



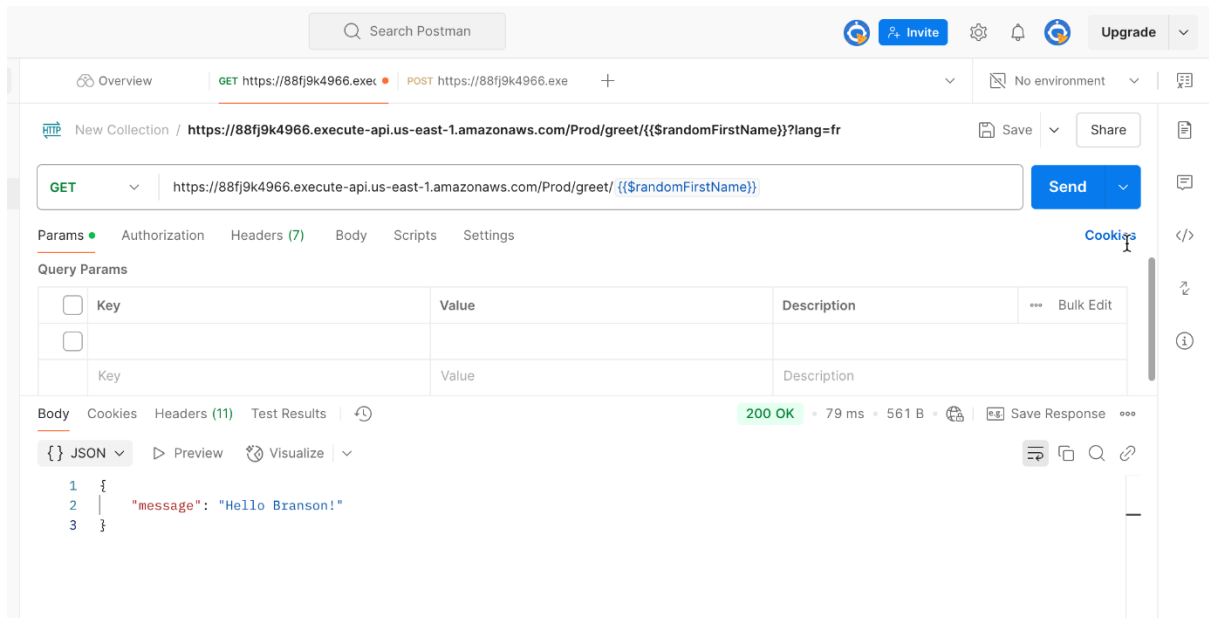
Test a valid POST request

- 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." }`



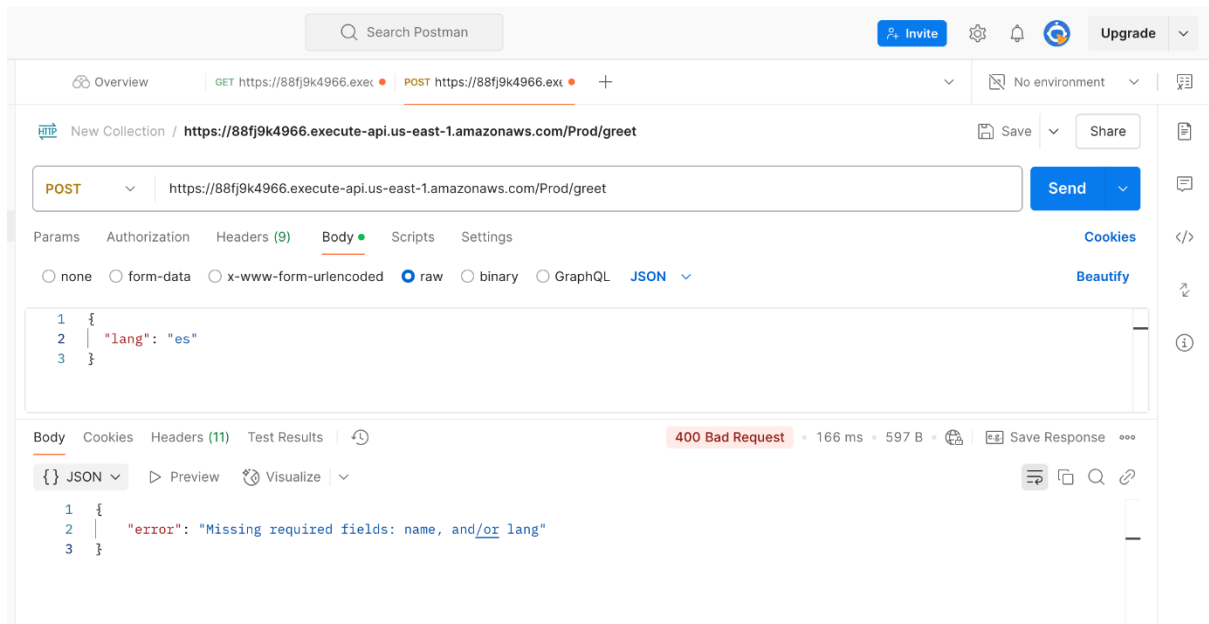
Test missing query parameters in GET request

- 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. `aws sam delete --stack-name lab-sam-greet-me`
- Enter `y` for any confirmation prompts.

