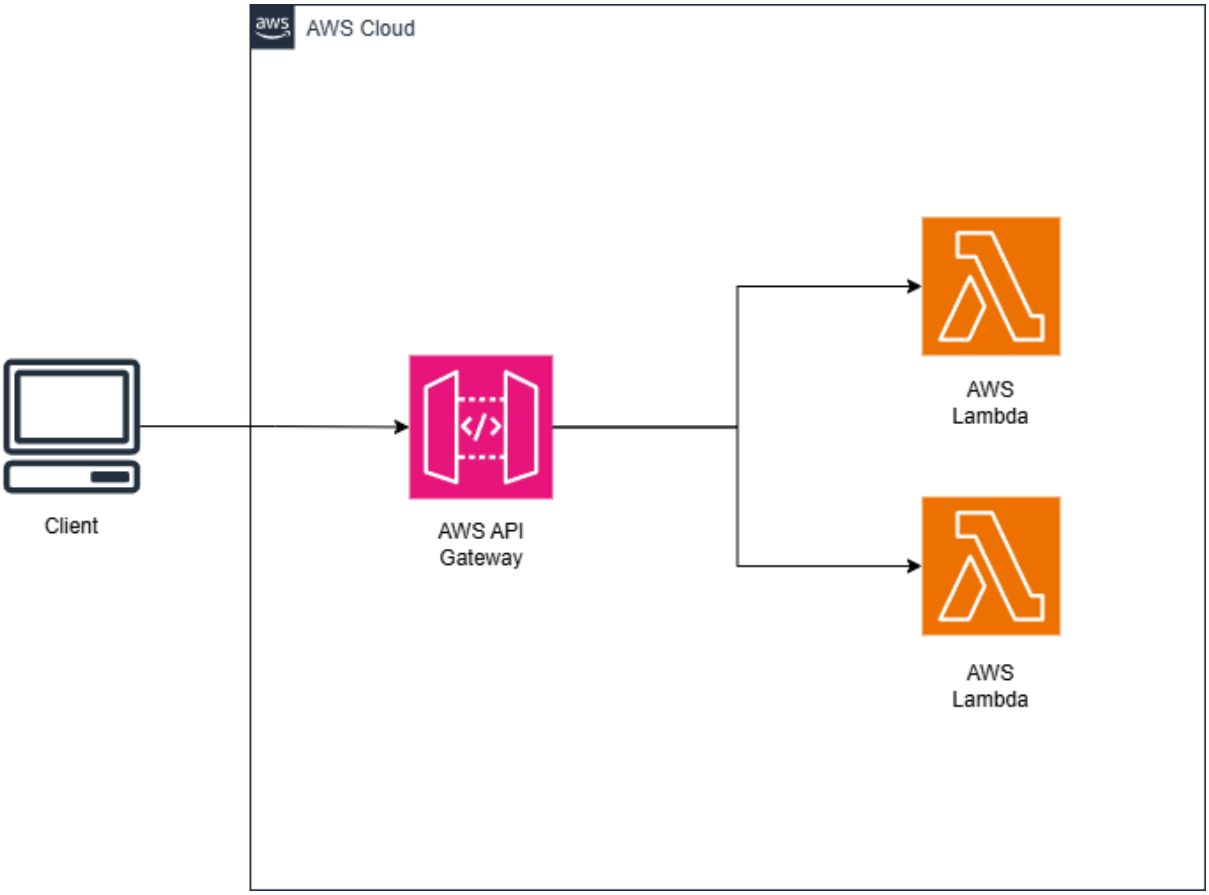
Integrate AWS with Lambda and API Gateway and Monitoring

Architecture :



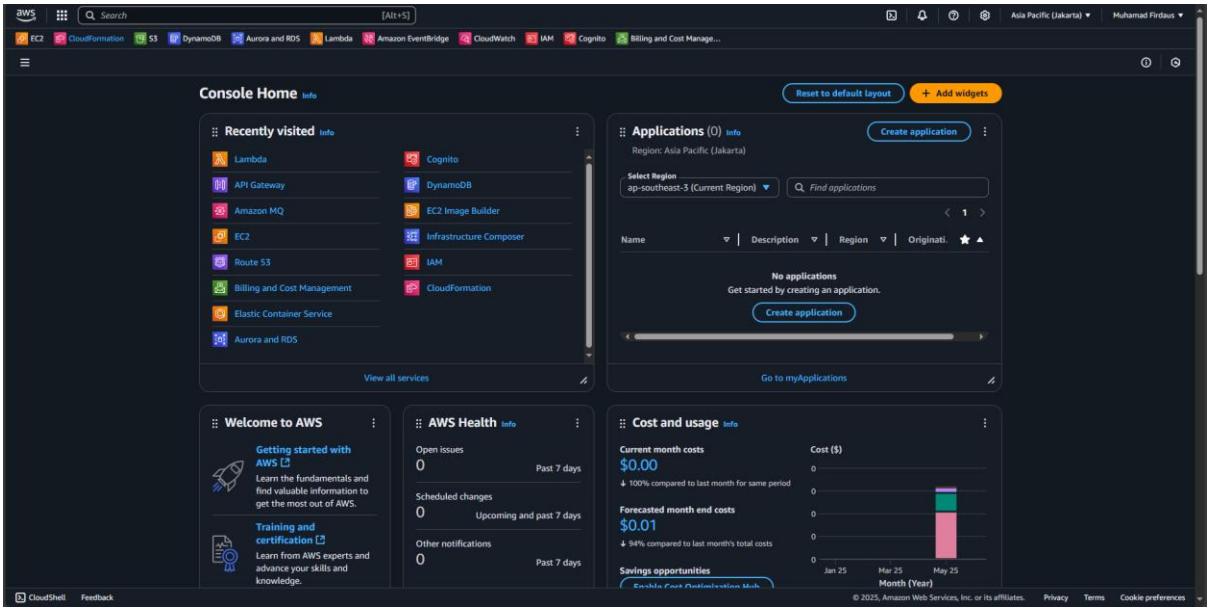
Service used :

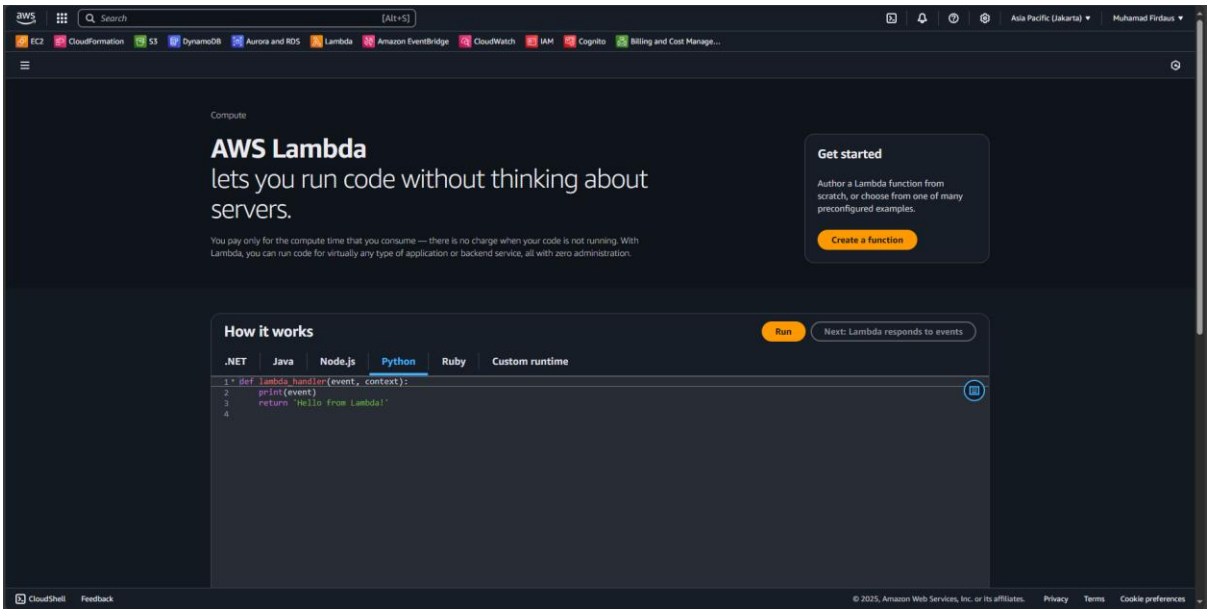
- AWS Lambda
- AWS API Gateway

Walkthrough:

PART 1 : Creating the Lambda and monitor the function logs

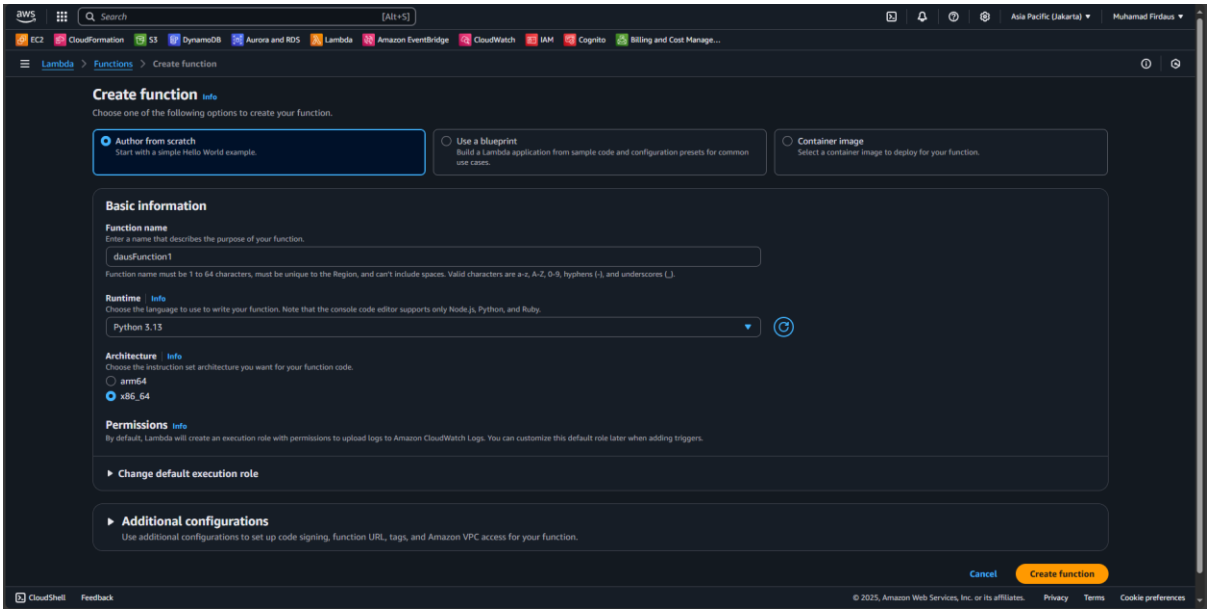
1. Open up the AWS Console and open the Lambda services, choose to create a new Function.



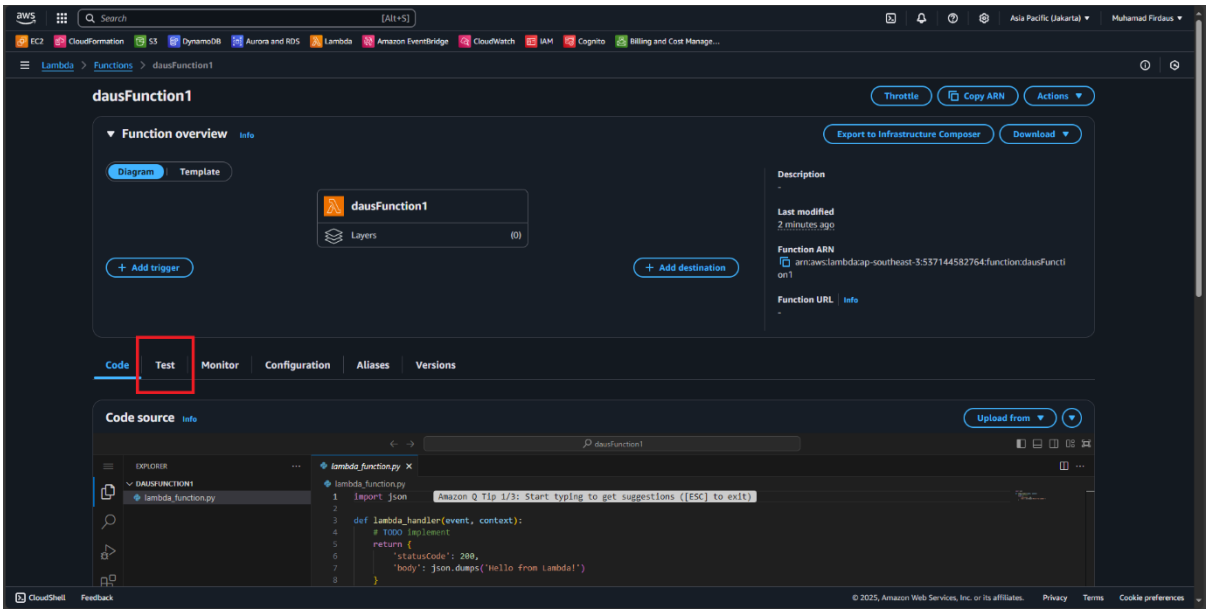


2. Fill the function details with information that you want, in this case I am using below options to fill the details and after that choose to create the function :

Option : Author from scratch Function name : dausFunction1 Runtime : Python Architecture : x64

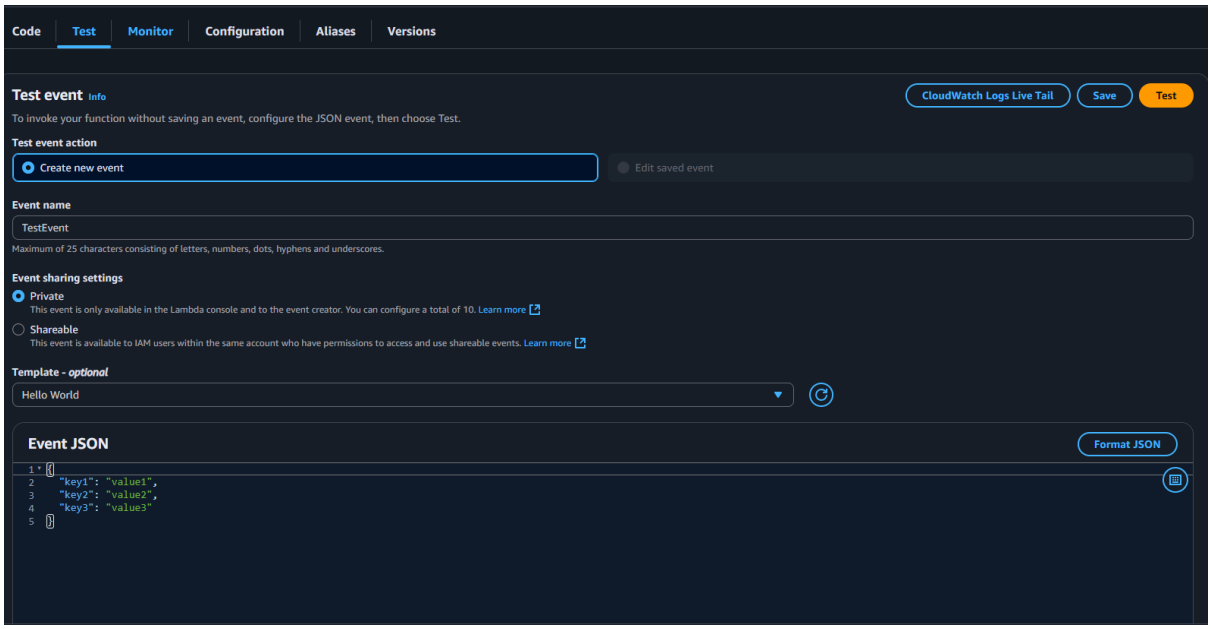


3. A pop up will show after the service created the new function, after the pop up choose the “Test” button on the bottom screen to open the section.

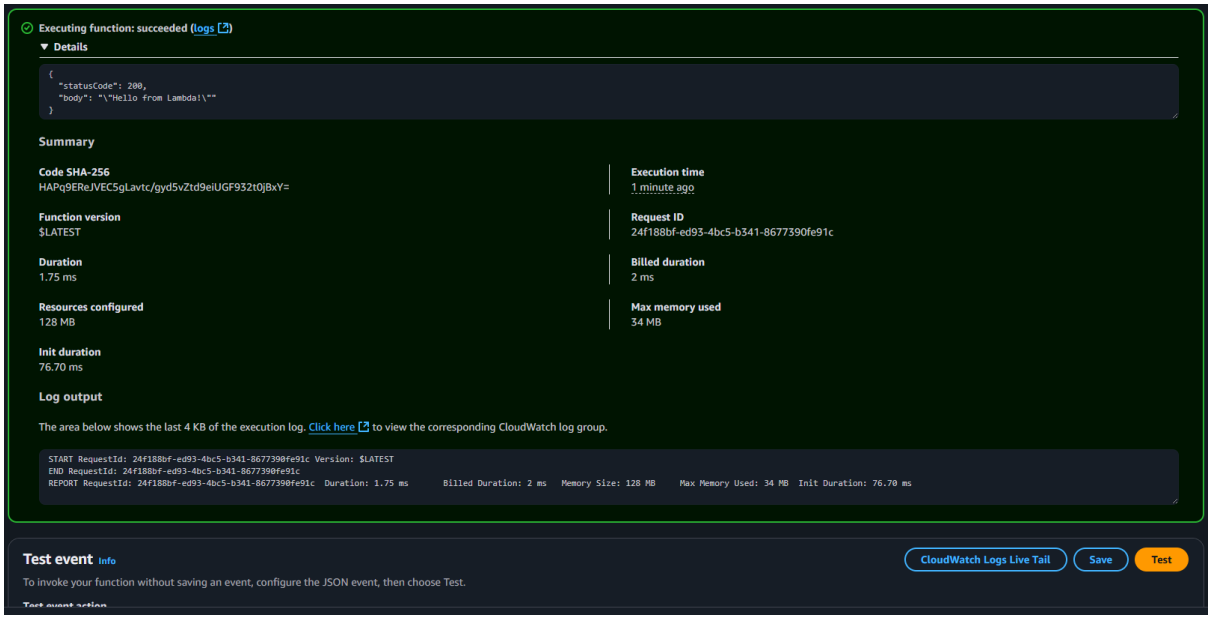


4. On the “Test” section, edit the details with the following details so we can use it to test the function and after that click on the orange “Test” button on the right above screen.

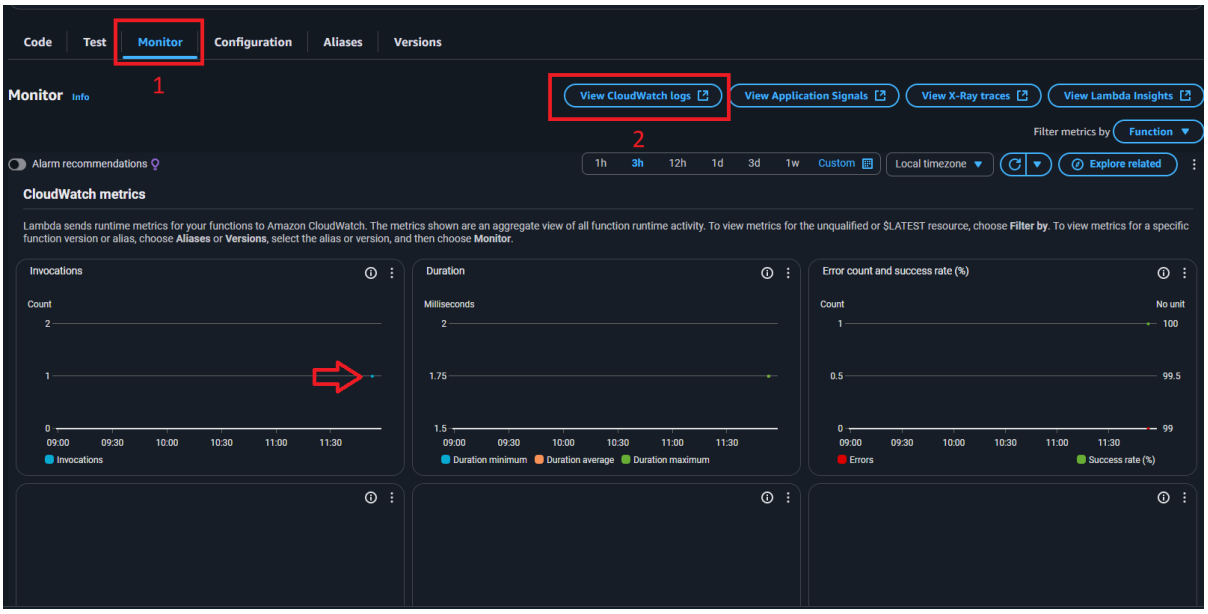
Test event action : Create new event Event name : TestEvent Event sharing settings : Private Template : Hello World



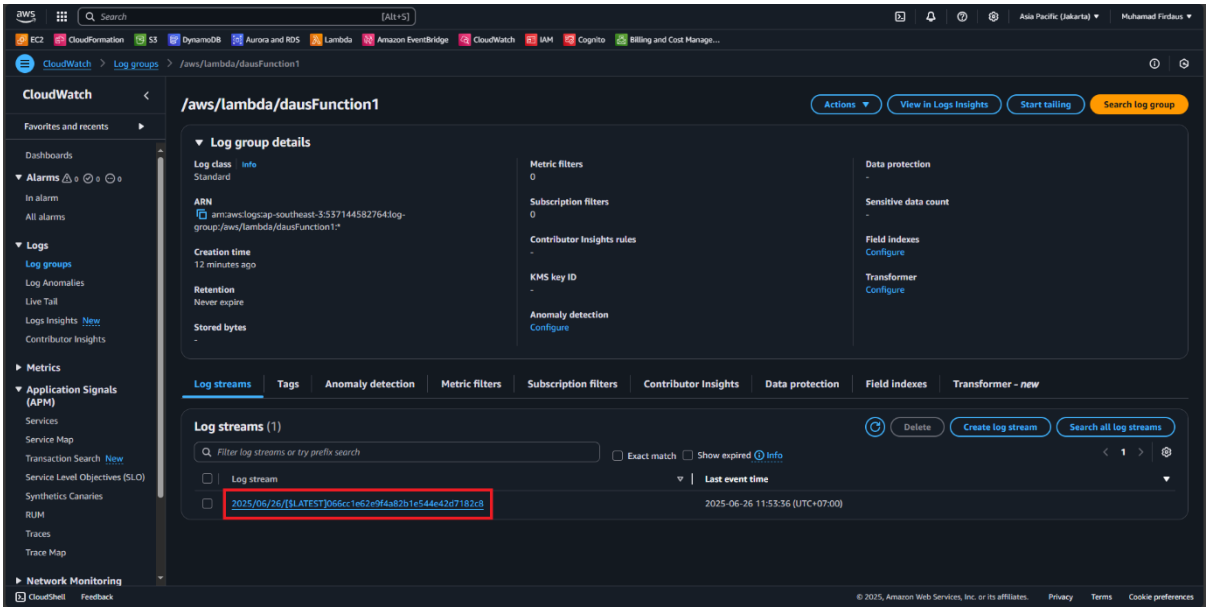
5. The result of the function test will then pop up to give us the details regarding the function testing.



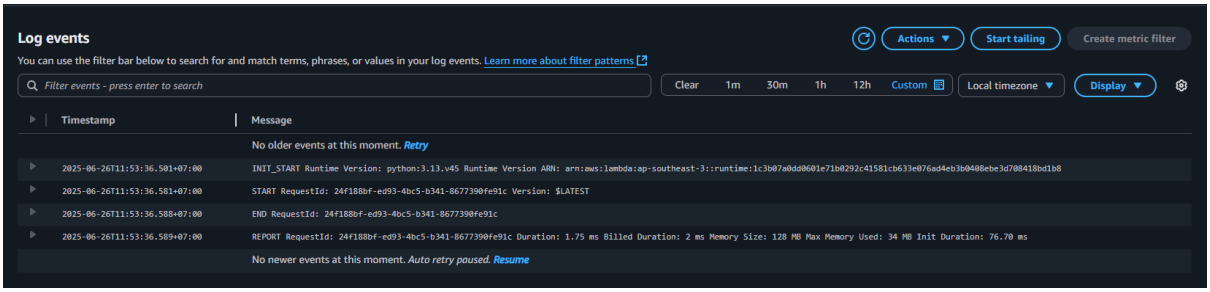
6. Now we can see the CloudWatch logs regarding the function testing on the CloudWatch section, click on “Monitoring” and choose “View CloudWatch Logs”. On the monitoring tab you can also see a dot that list our testing history activity.



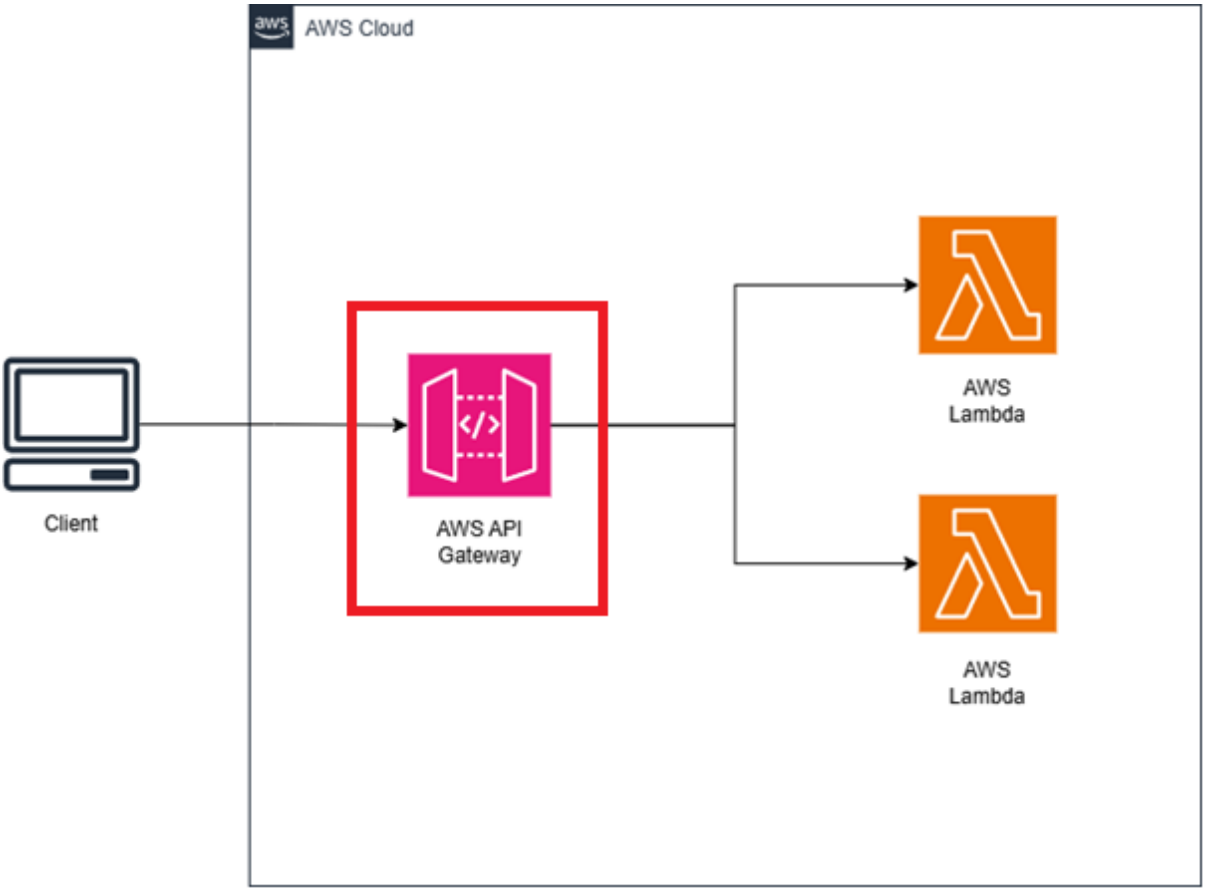
7. On the CloudWatch pop up, choose the newest log name on the log stream (the blue one) and click it to get redirected to the logs menu.



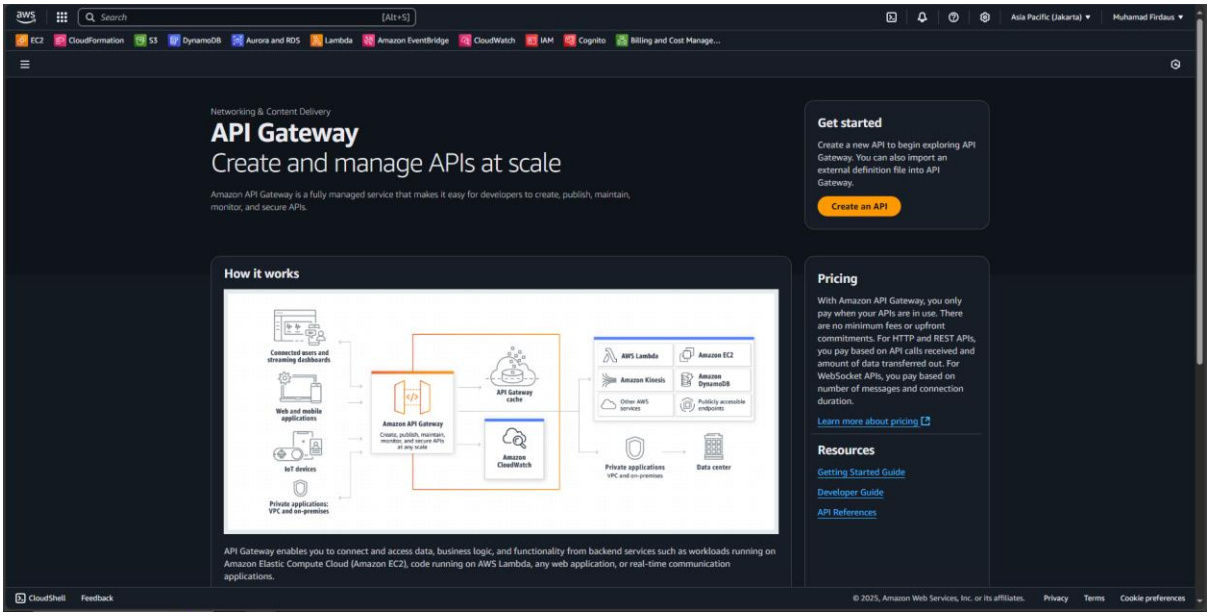
8. The log event then will show the detail regarding the running function process that we can see.



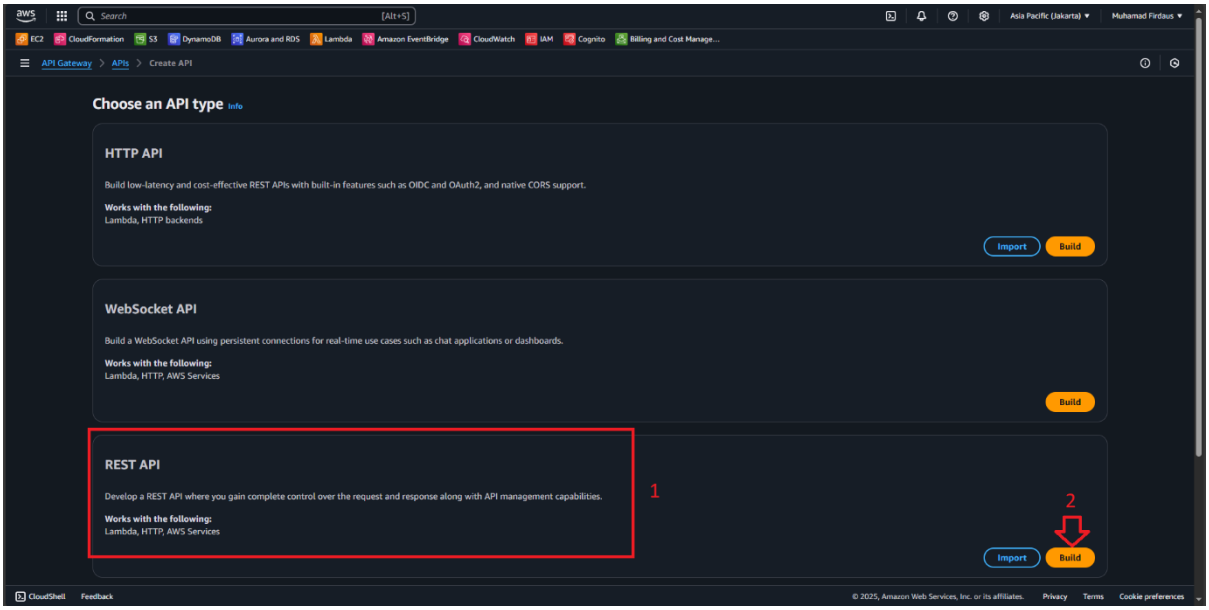
PART 2 : Connecting Lambda function with the API Gateway



1. On the AWS Console, search for API Gateway service and enter the service. We will create the API Gateway (I will mention it further as APIGW) to connect to the lambda function. Choose to create a new API.

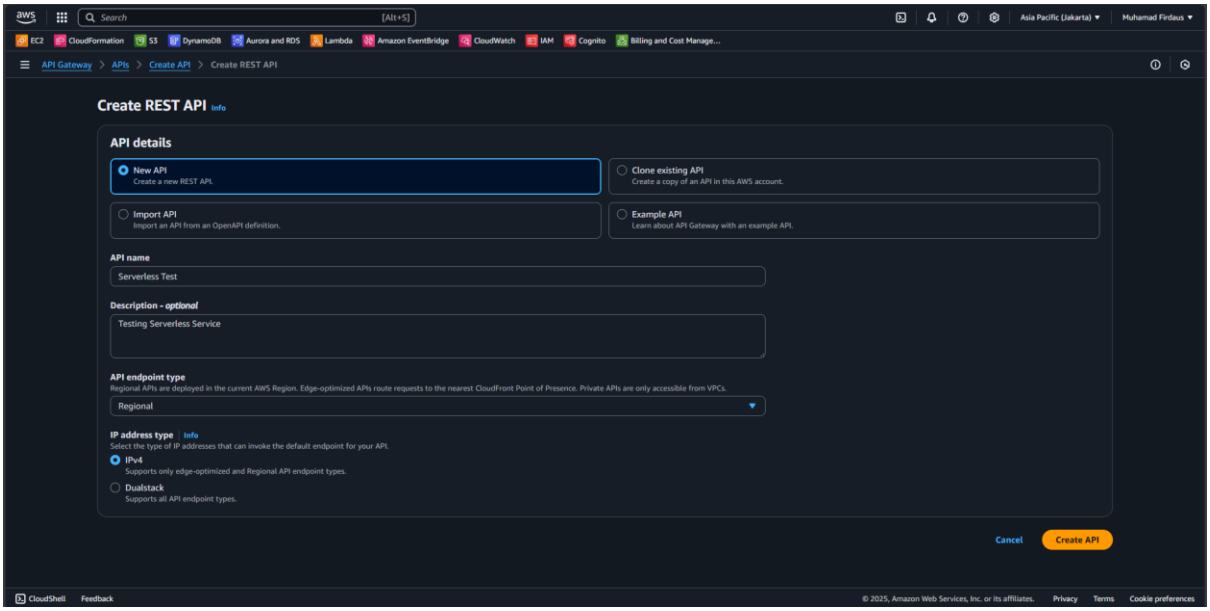


2. On the creation page, you will be asked with what kind of API you want to create. We will continue by using REST API (not the private one).

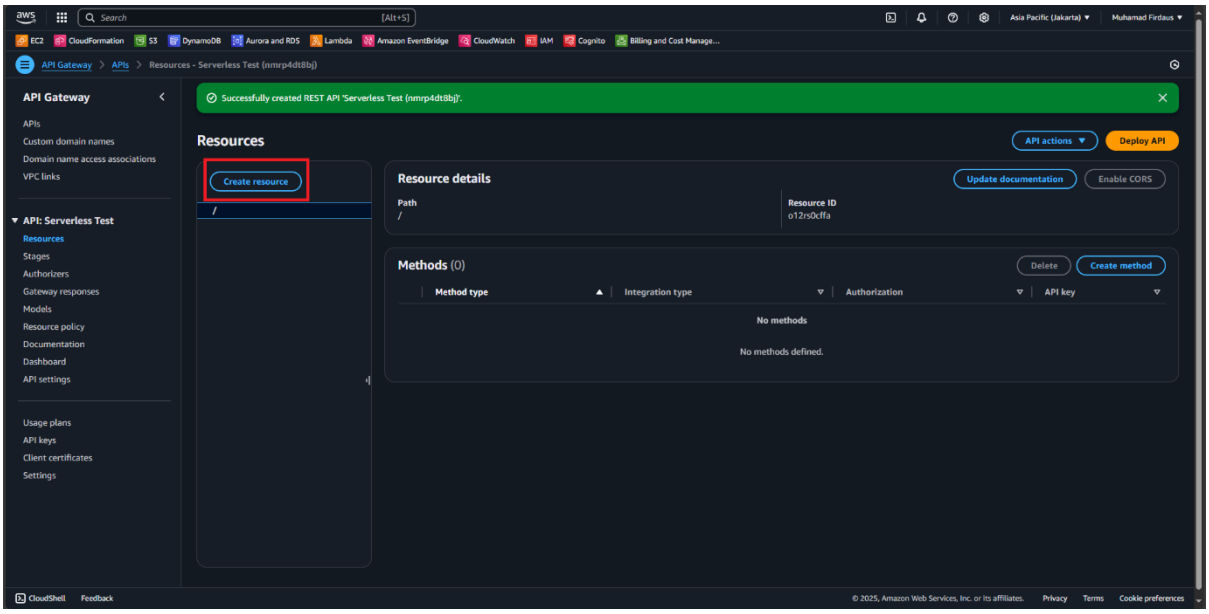


3. On the API creation page, choose to fill the details as you see fit. In this case I will be using below details to continue and choose to create the API :

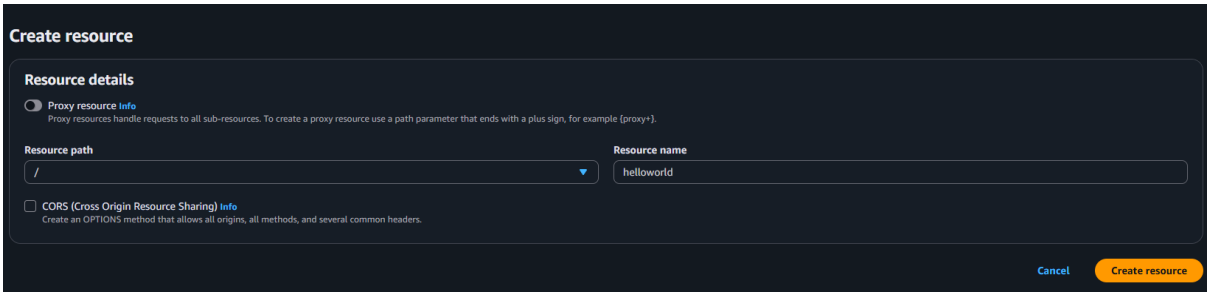
API Details : New API
API name : Serverless Test
Description : Testing Serverless Service
API endpoint type : Regional
IP address type : IPv4



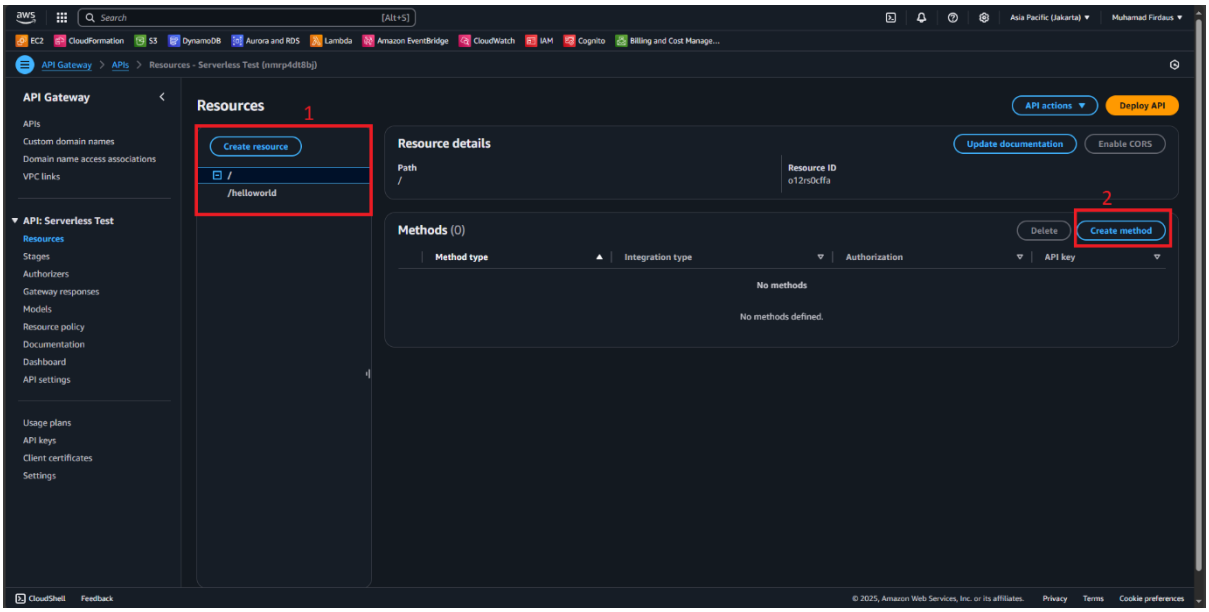
4. A page will then popup showing you the finished process of creating the new API, now on that popup page we will choose the “Create Resource” option to create a new resources.



5. On the create resource section, fill in the details with default setting while adding the name of the resource as “helloworld”.

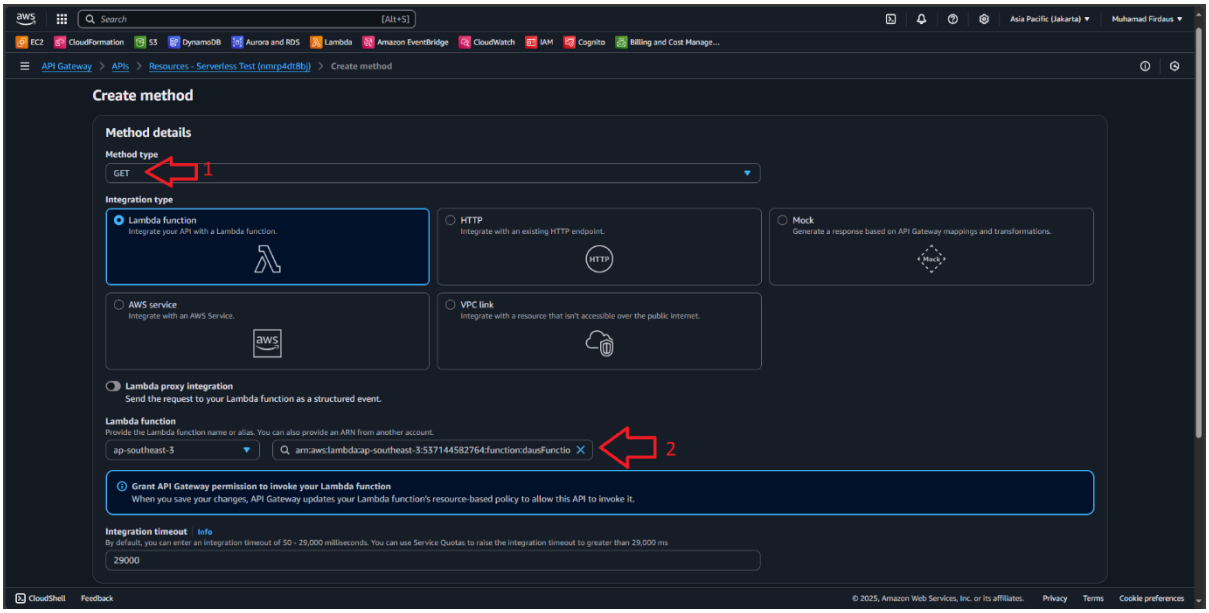


6. After creating the resource, a new pop up will show us that the resource is successfully created. And then we choose the “create method” option AFTER we click on the /helloworld to create a new method for our resource under the /helloworld resource.

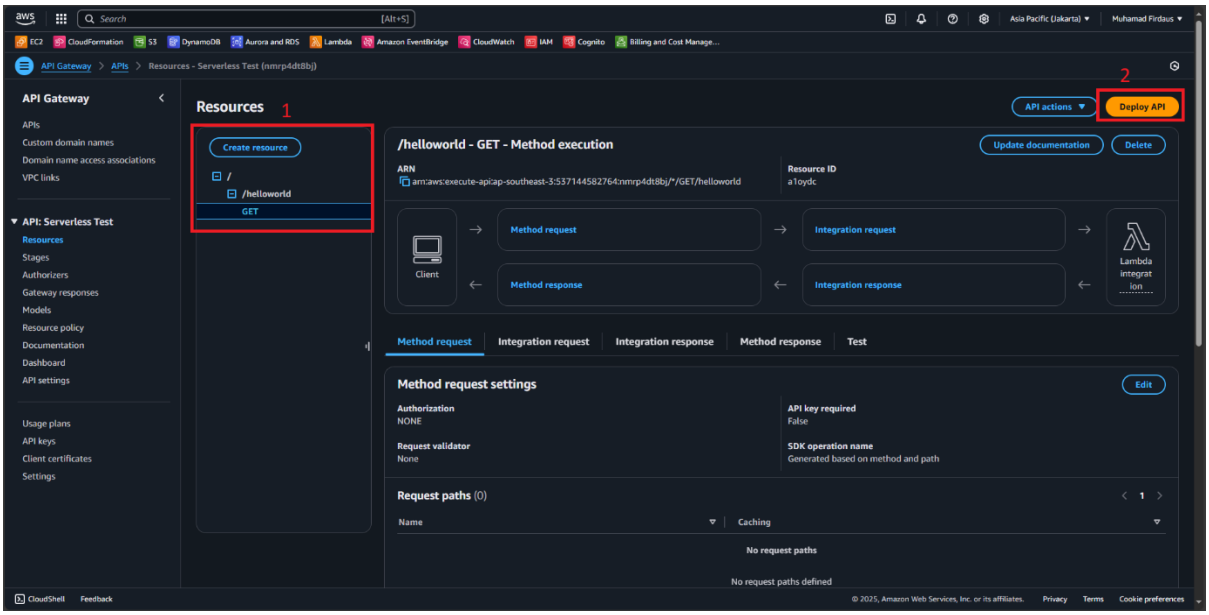


7. On the create method page, we fill the details with the below information to proceed with our method creation and then choose create method.

Method type : GET Integration type : Lambda function Lambda proxy integration : off Lambda fucntion : our created function earlier in my case it is dausFunction1 Integration timeout : left it default

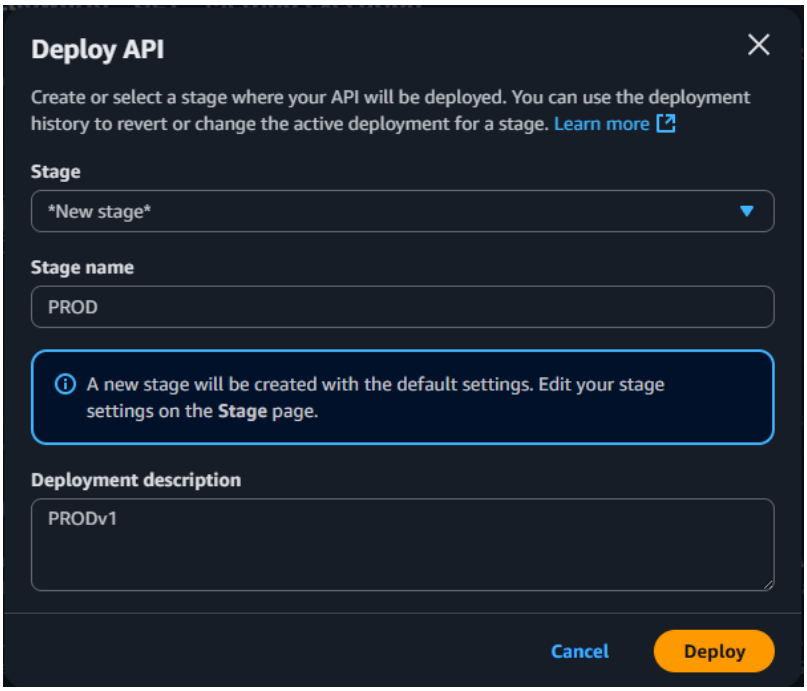


8. After the method was created, the GET method will appear under the /helloworld on the resource tab, confirm this to proceed. Then click on the “Deploy API” so we can begin to prepare to deploy our API

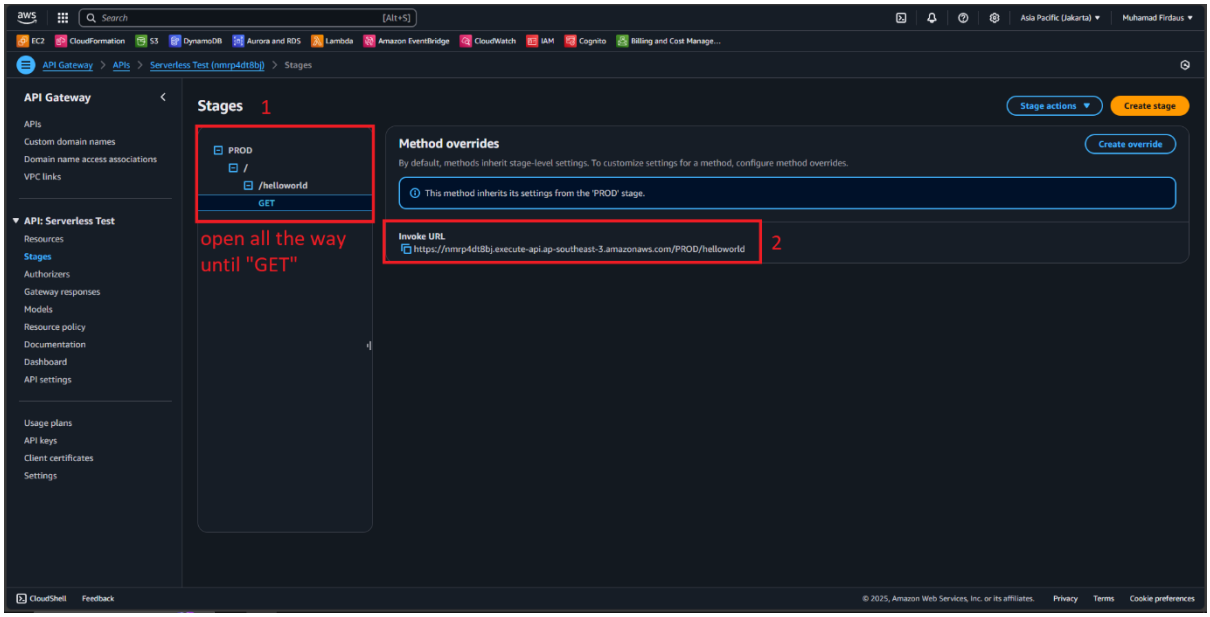


9. A popup will appear for us to configure the API deployment, use below option to fill the required field and then “Deploy”.

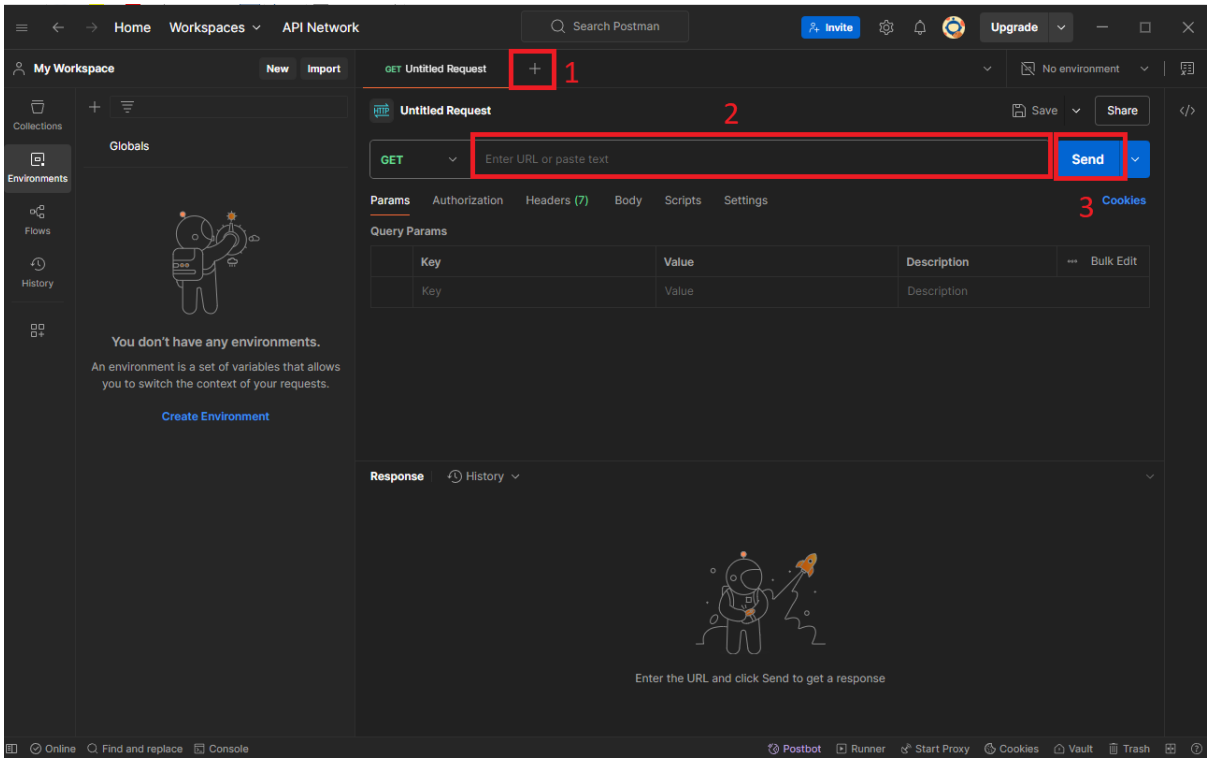
10. Stage : *New stage* (Its a dropdown menu)
11. Stage name : PROD
12. Description : PRODv1



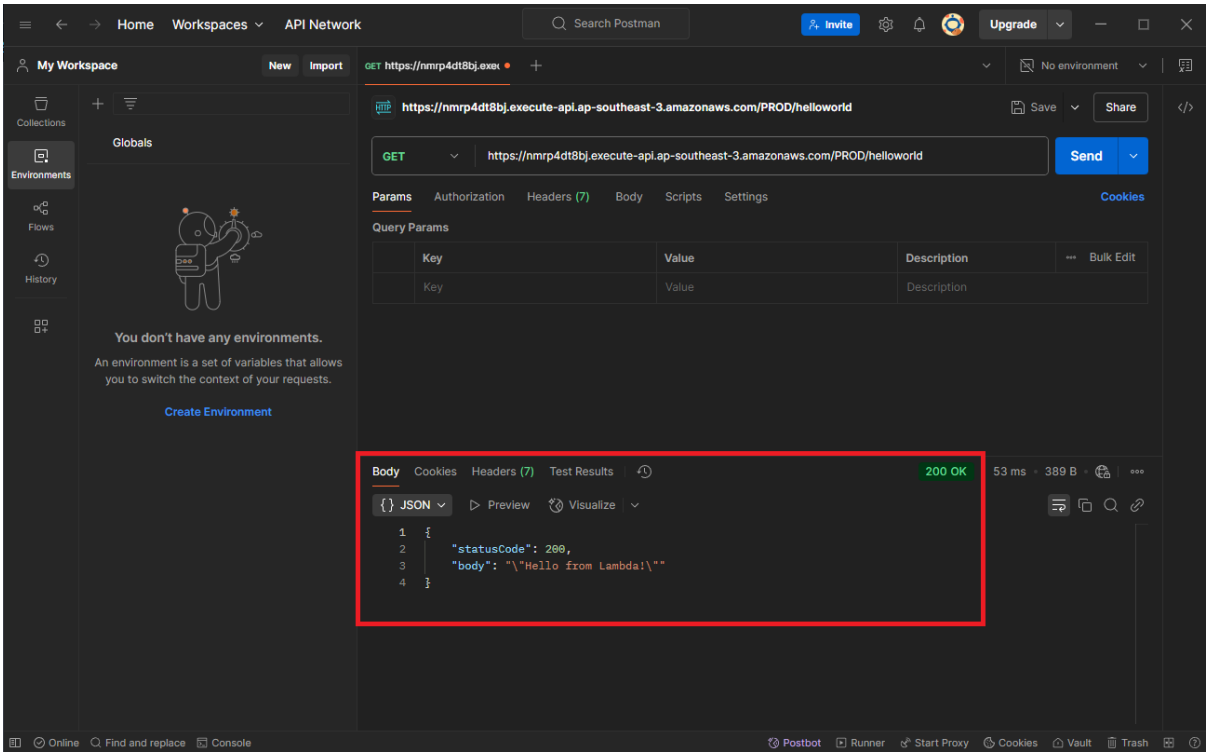
13. After we clicked the deploy button, a new page pop up showing us the stages of the method, open all the stages until you get to the GET stages, click on it to reveal the method content and then copy the “Invoke URL” for us to be able to use it on API testing tools.



14. Open up your API testing tools such as Postman API or Insomnia, then click on a new testing environment, use the GET method and then paste the Invoke URL we previously copied on the AWS API Gateway and then click on send request.

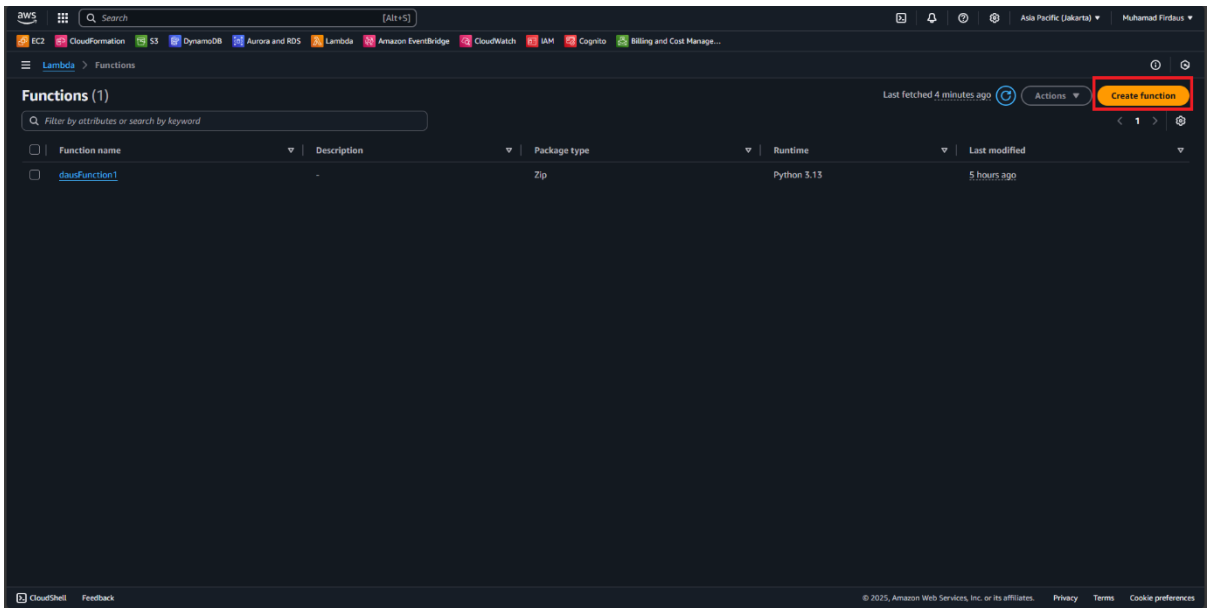


15. If the configuration is correct, we will get the response from the AWS API Gateway with status 200 OK and the Hello from lambda message.



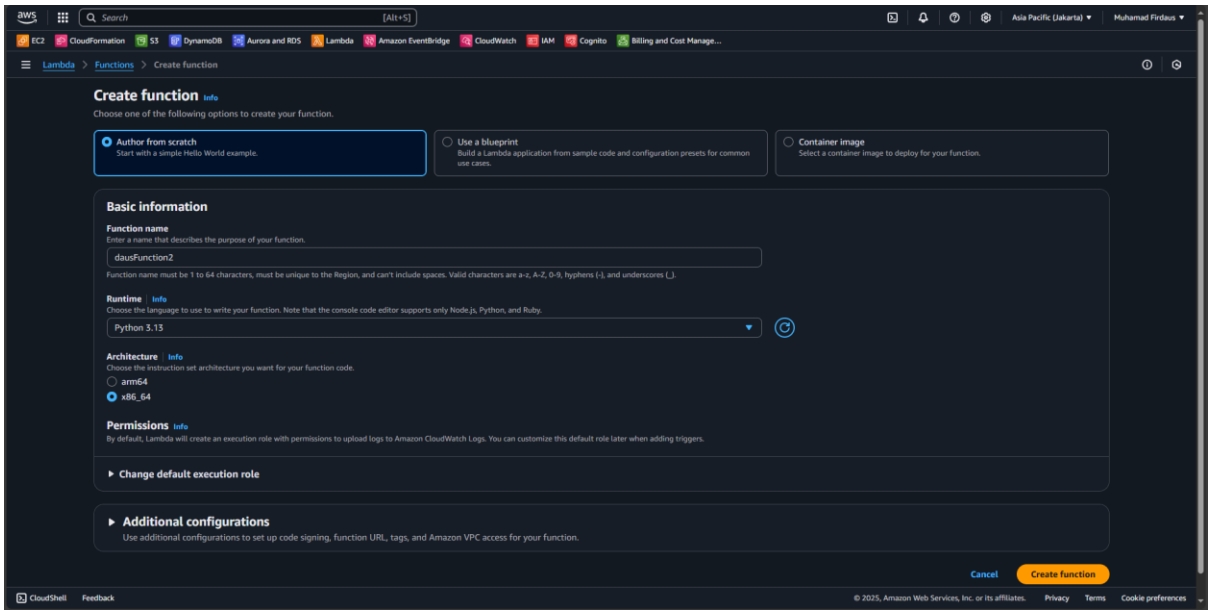
PART 3 : Implement API Canary on the API Gateway

1. Open up the lambda service menu, we will create a new function this time.



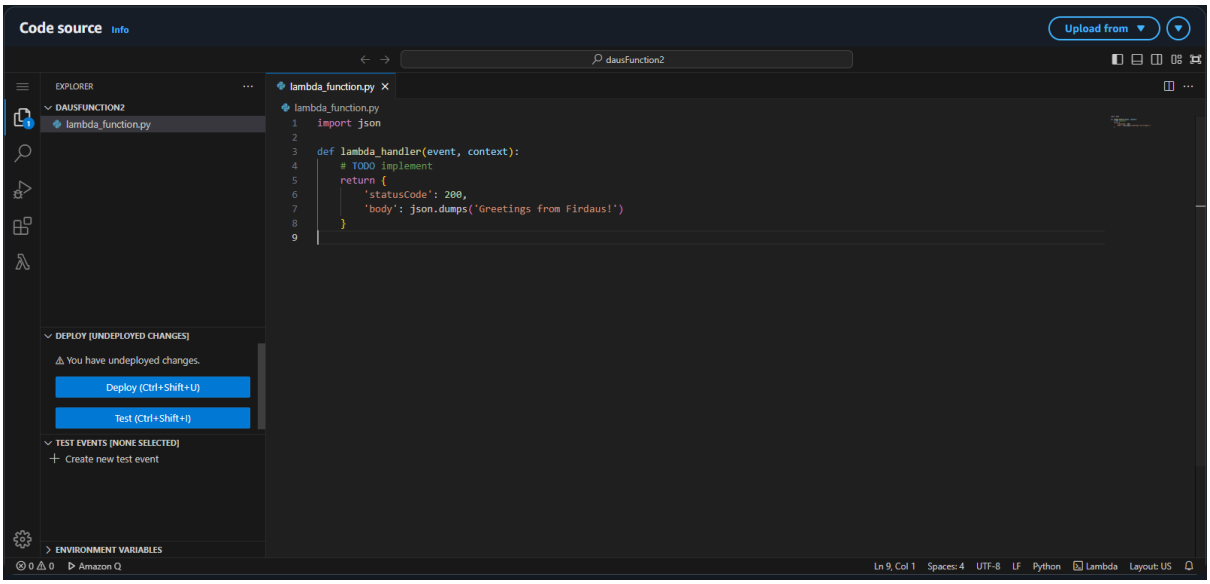
2. Fill out the required information for our second function, I use the specification below for my second function.

Create function : Author from scratch Function name : dausFunction2 Runtime : Python Architecture : x64

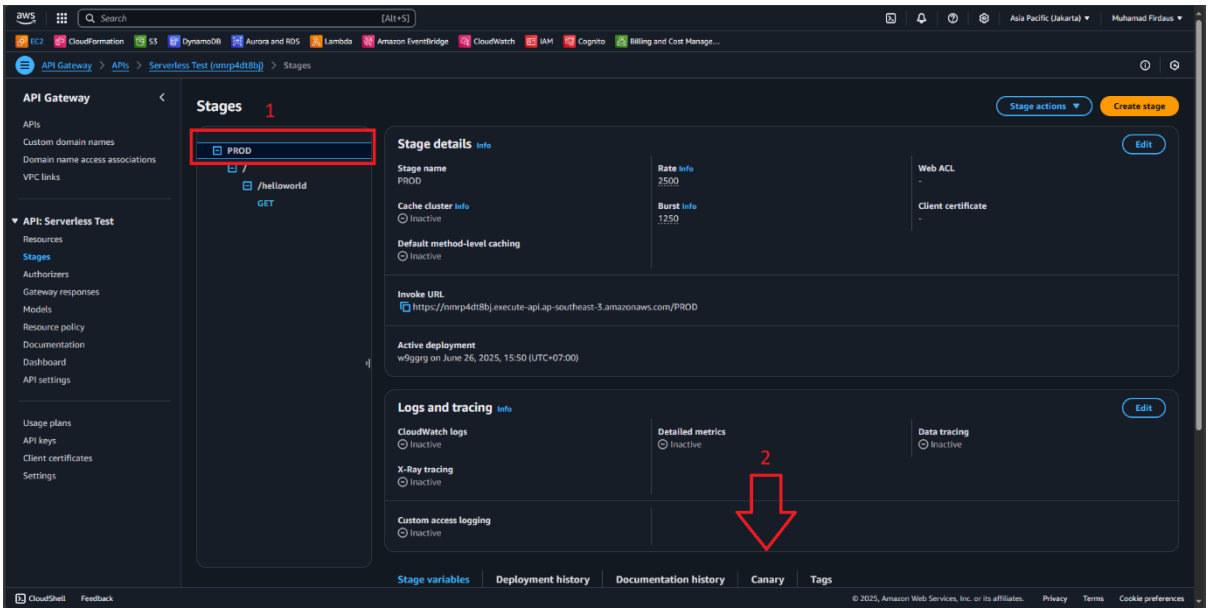


3. On the code source, we change the successful message a little bit. You can edit it however you want and DON'T FORGET TO DEPLOY THE CHANGES FIRST before proceeding.

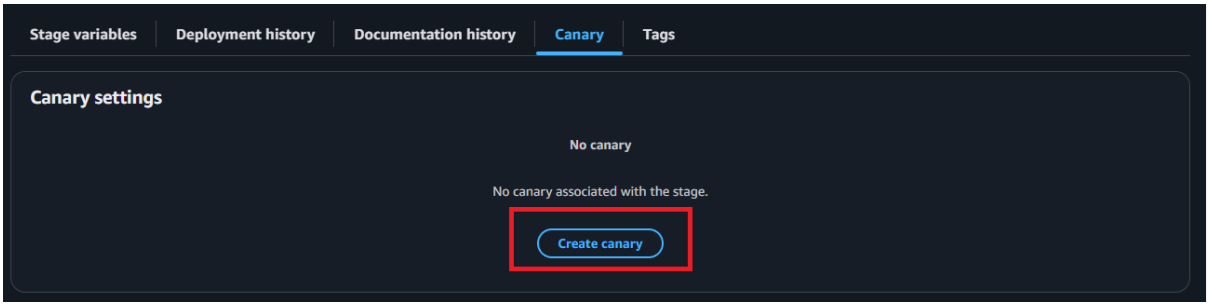
```
import json

def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Greetings from Firdaus!')
    }
```



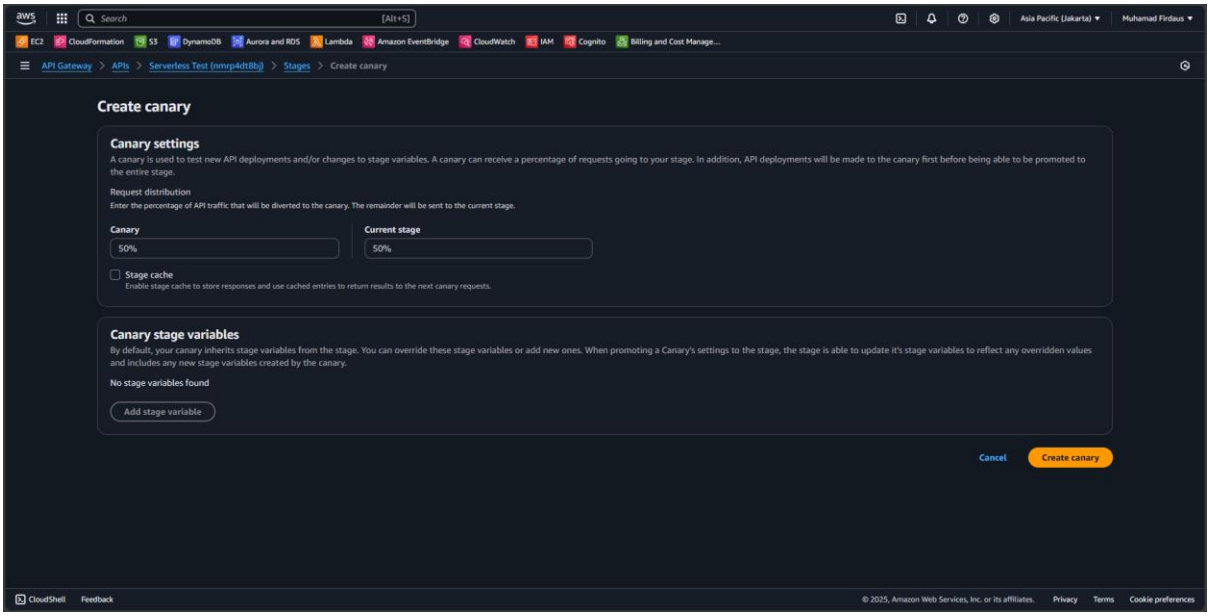
4. On the stages section, click on the “PROD” option, and choose “canary” menu on the right bottom of the page.



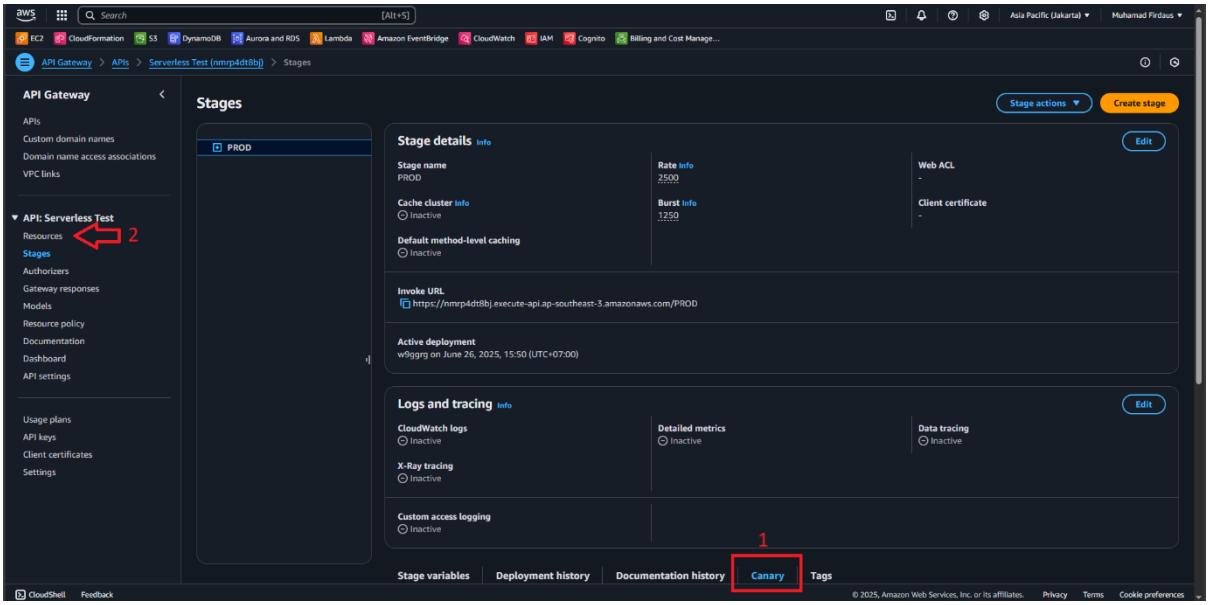
5. Choose to create a new canary by clicking on “create canary” button.



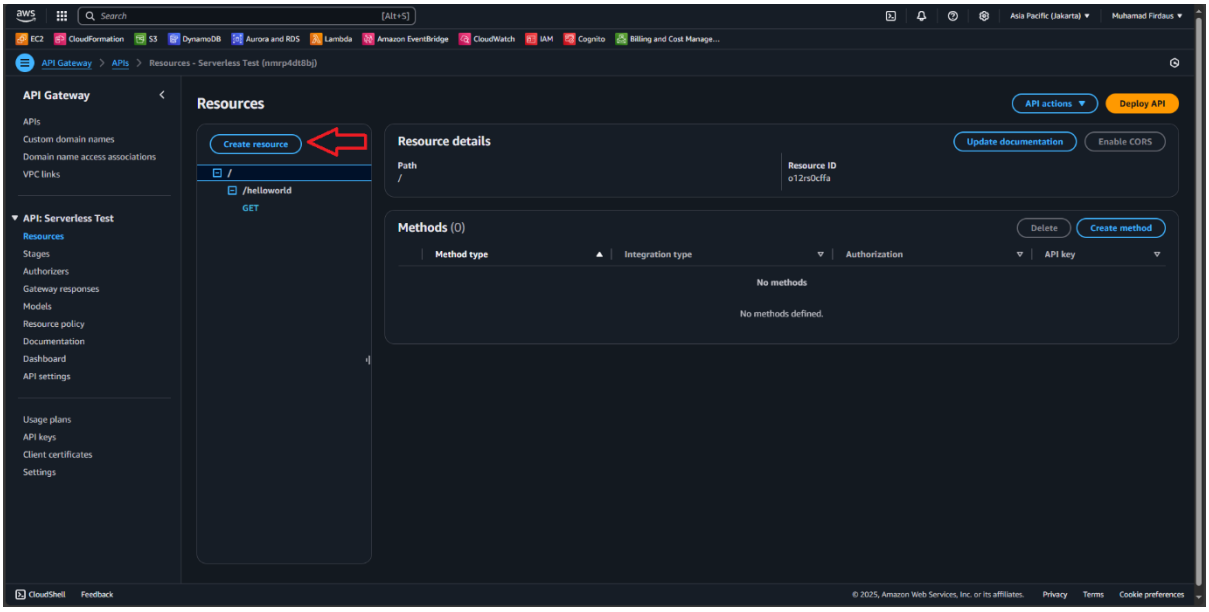
6. On the create canary menu, I will be splitting the canary and the current stages with equal percentages of 50%, this will split the traffic in equal way towards both stages.



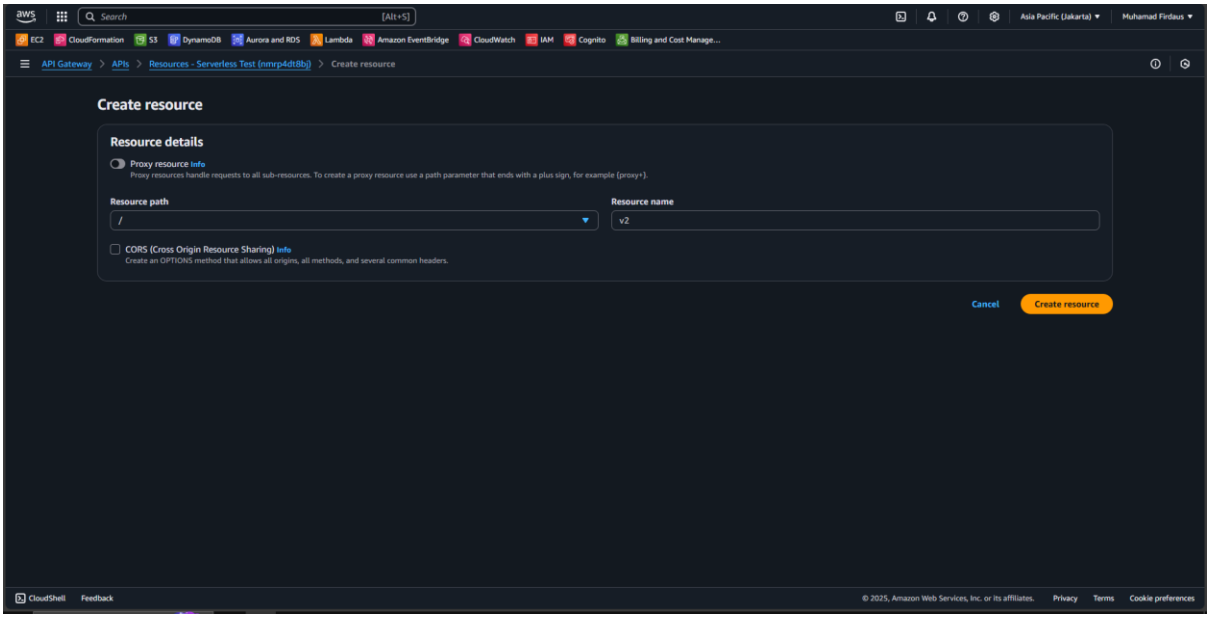
7. Back to the stages menu, check if your new canary is already created, and then click on the resource tab as we will create a new resource for the canary stages.



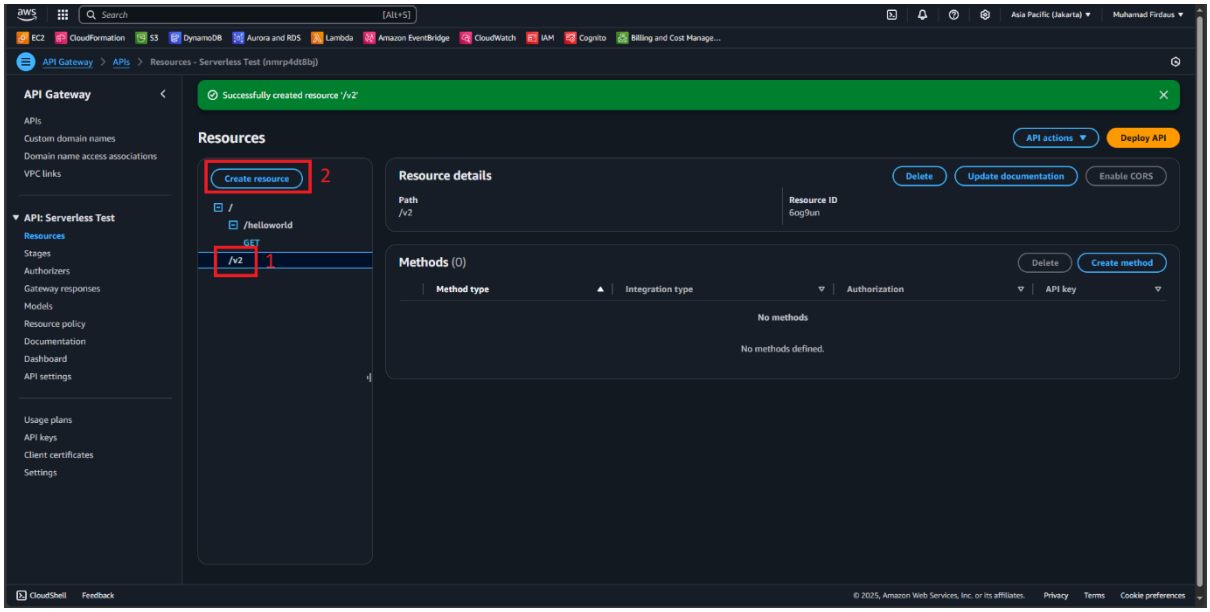
8. Click on the “Create resource” button to open up the creation menu.



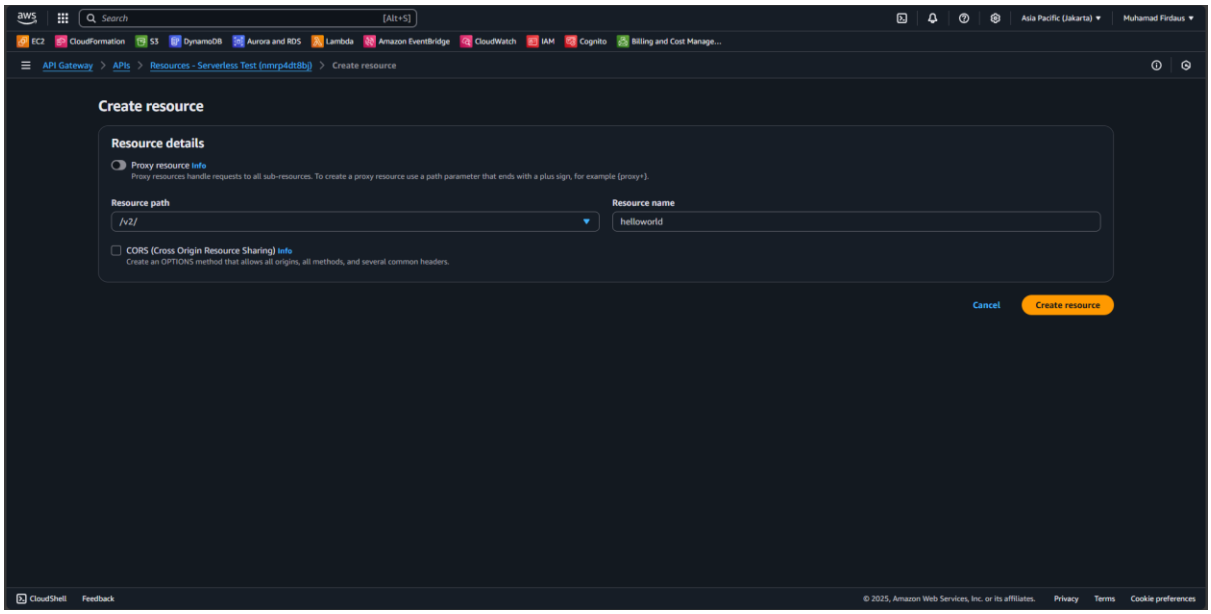
9. On the resource detail page, enter “v2” or any other resource name you want to name it, still within the “/” path, then choose to create this resource.



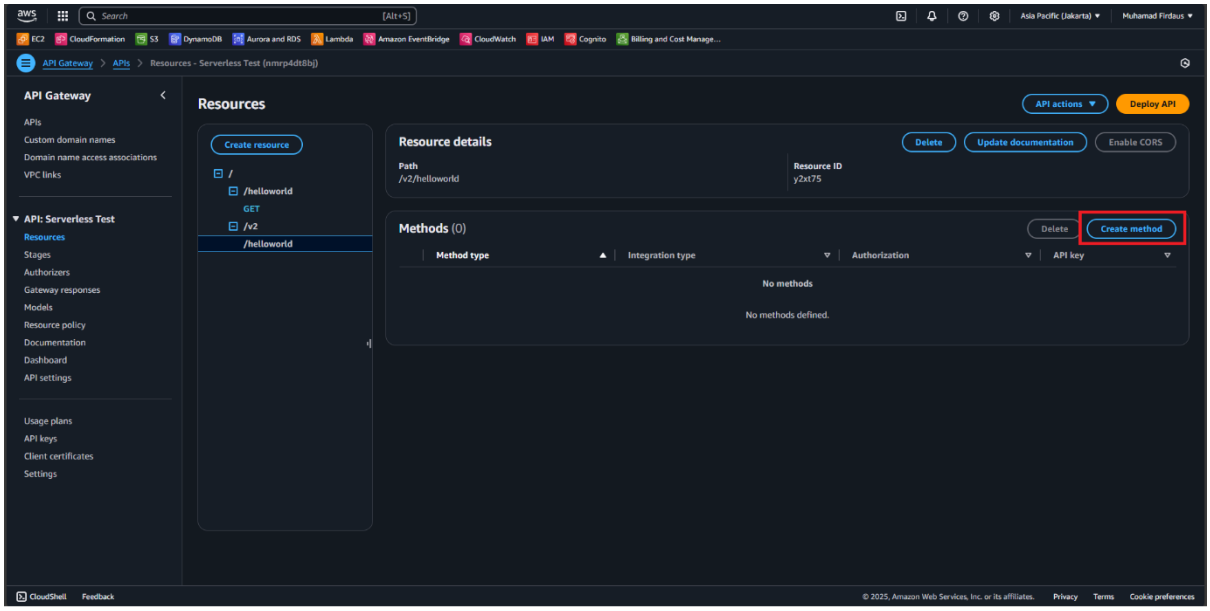
10. Click on the “/v2” to choose it as the current path and then click again on the create resource menu to create a new resource under the “v2” path.



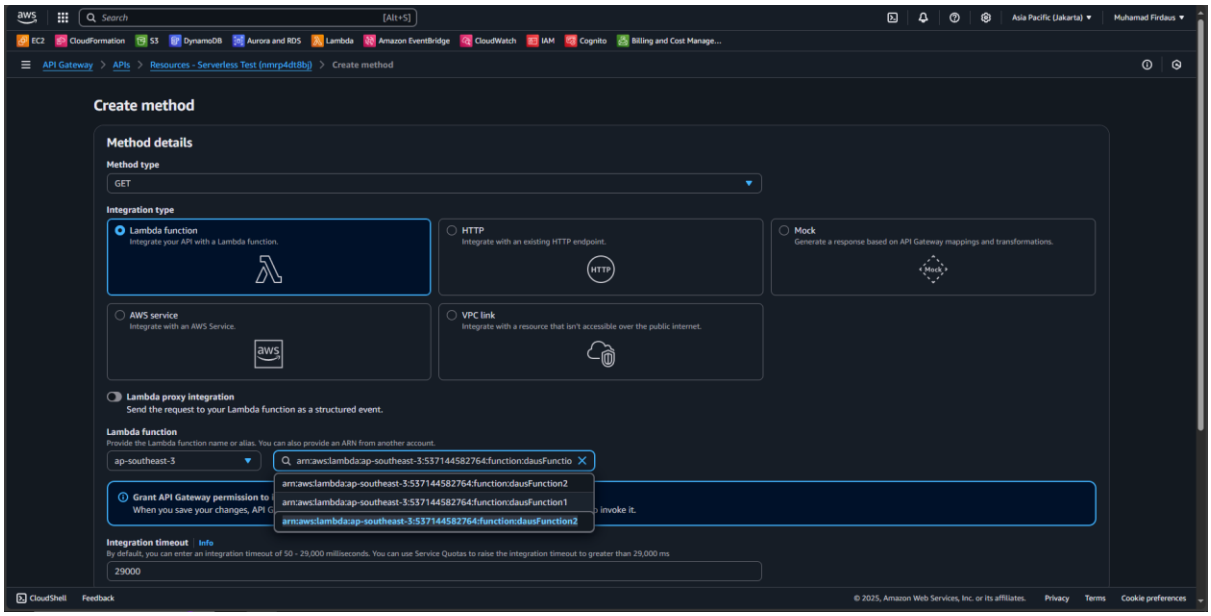
11. Make sure to check the resource path is on the “v2” and then input the name you want to be using as the resource name, I will be choosing another “helloworld”.



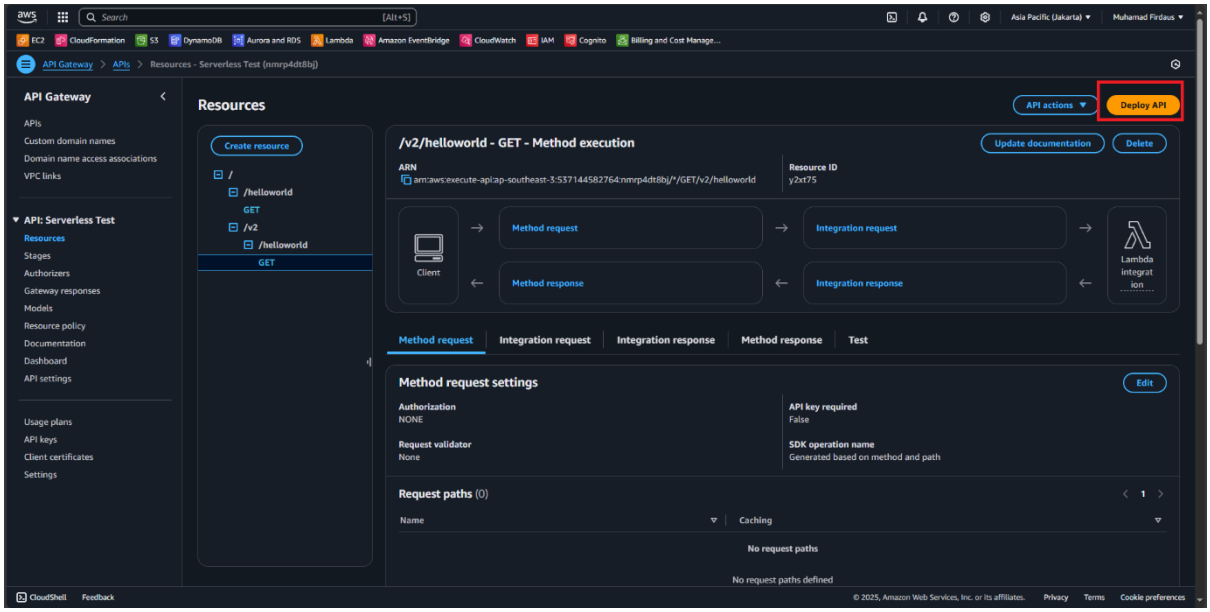
12. When the “helloworld” resource is created inside the “v2”, click on it and then choose to create a new method inside the helloworld resource.



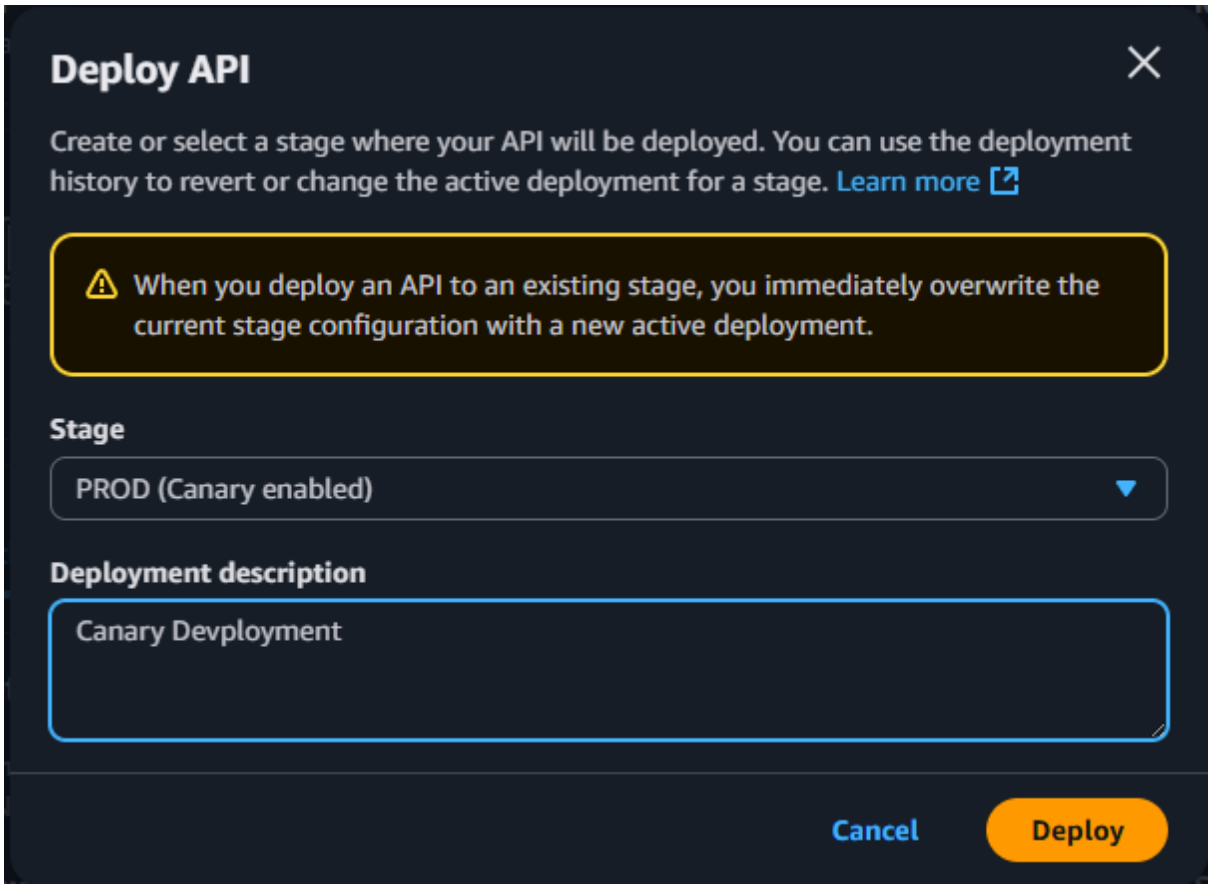
13. We’ll be using the GET method again, and this time we choose the “dausFunction2” as the lambda function. Create the method after leaving all the settings in default.



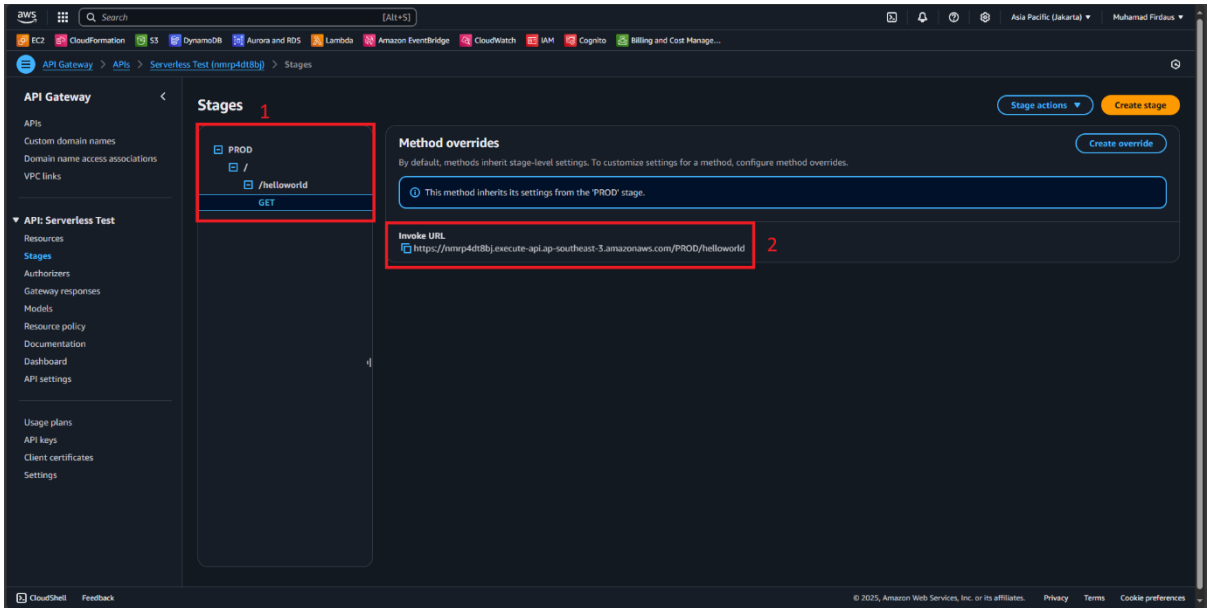
14. Click on the GET method inside the v2 helloworld, then choose to deploy a new API inside of it.



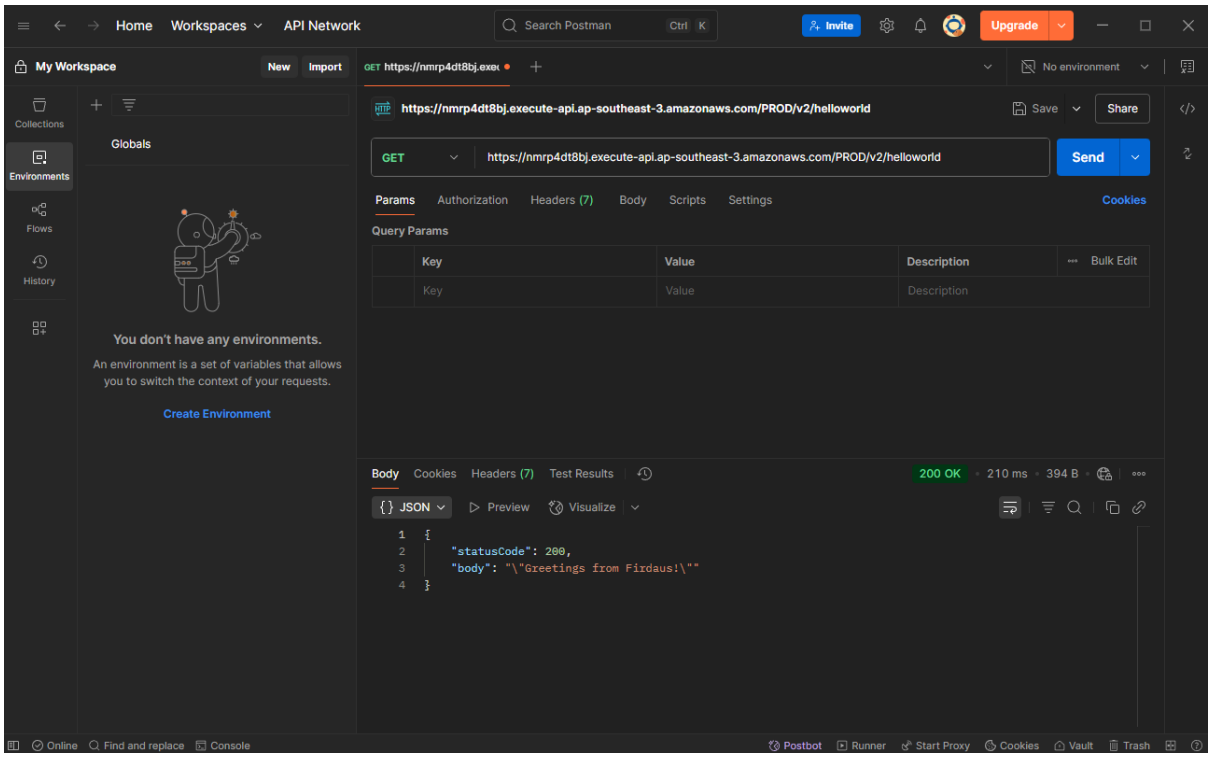
15. On the stage, we'll be choosing "PROD" it will mention that canary is now enabled if we choose it, fill the description however you like and create the API.



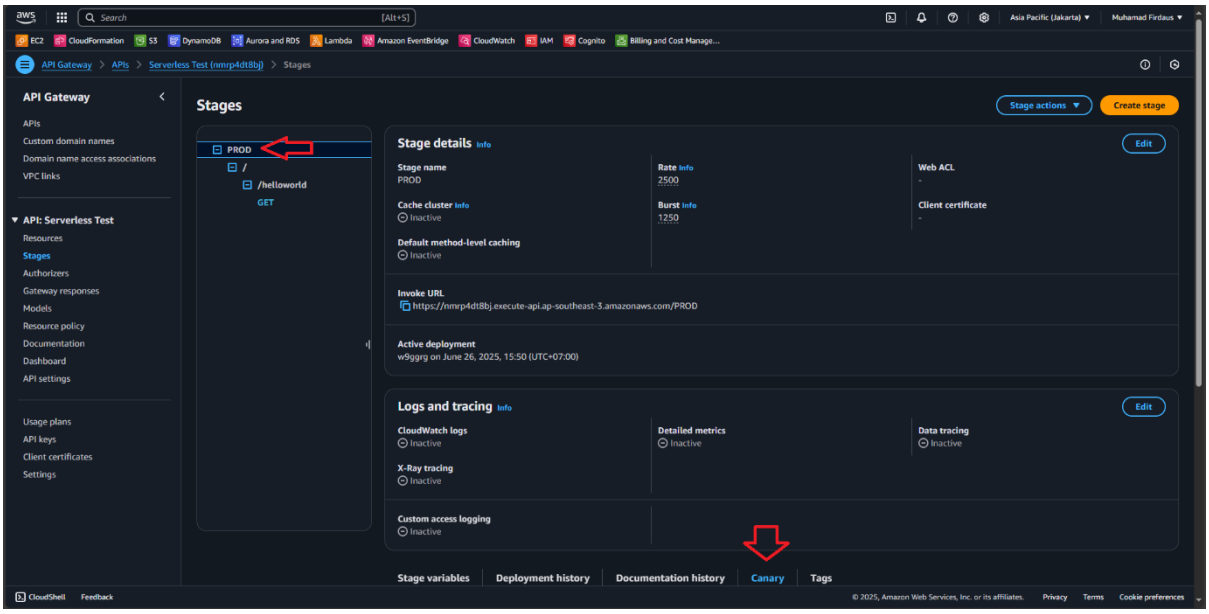
16. On the stages menu, open all the stages until you click the GET section so you can see the invoke url, copy it so we can use it to send a request to the canary gateway.



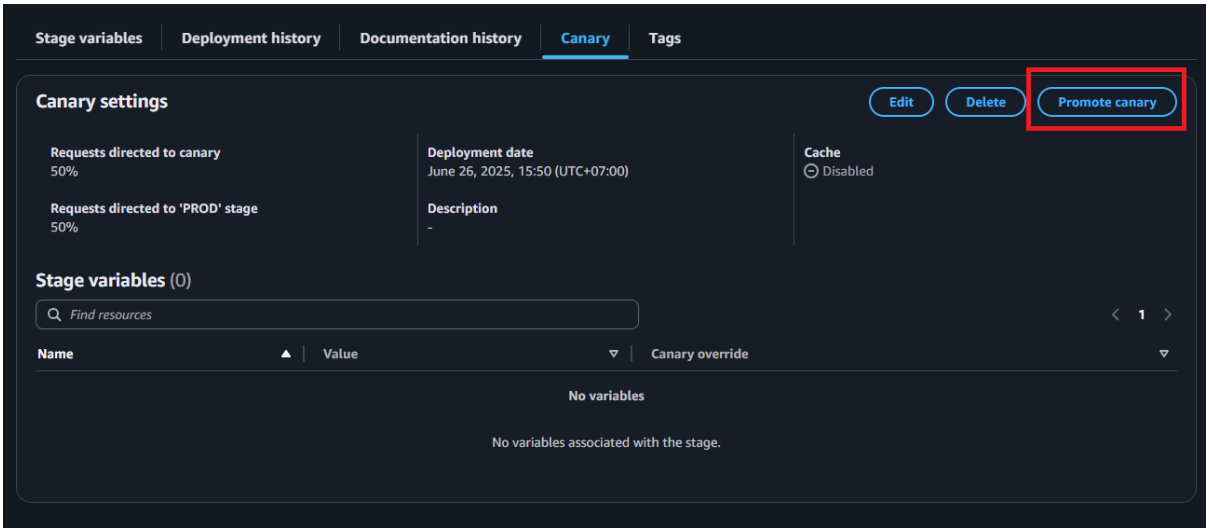
17. Open up your API testing tools, again I will be using Postman, enter a new task, choose HTTP and use the GET method, then paste the Invoke URL we copied before (don't forget to add the /v2/ path before the helloworld since we havent configured it to be automatically invoked on the URL) then choose to sent your request. If your configuration is correct then you will see your message in the different setup than the first lambda function as seen here.



18. Return to the AWS console and click on the PROD menu, and choose on canary option.

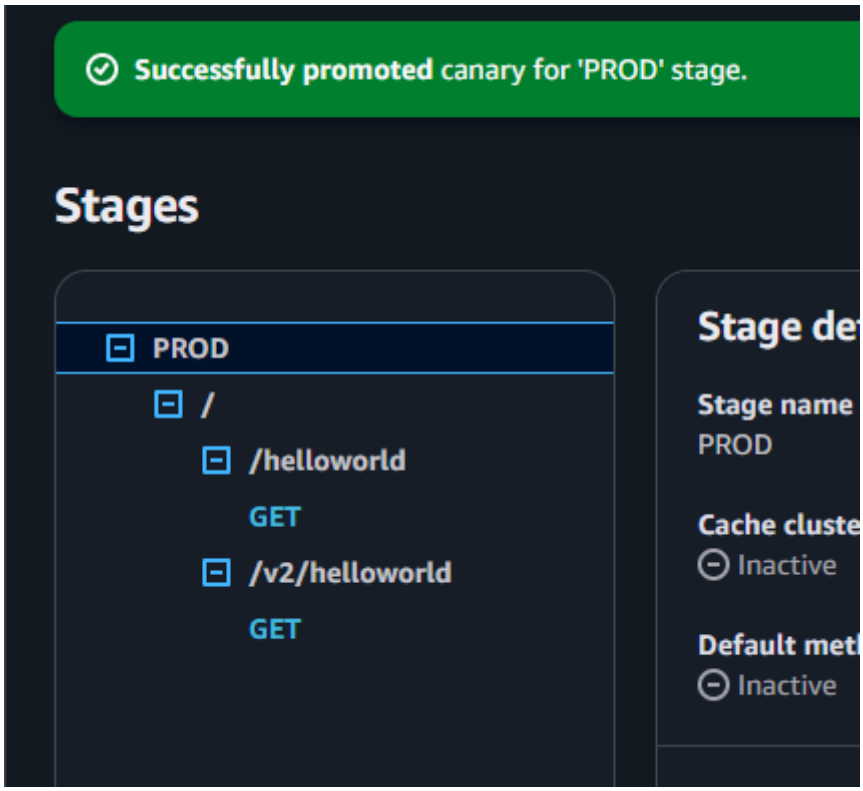


19. Now after confirming that the canary is reachable, we can promote the canary by clicking on the “Promote Canary” button inside the canary option menu.



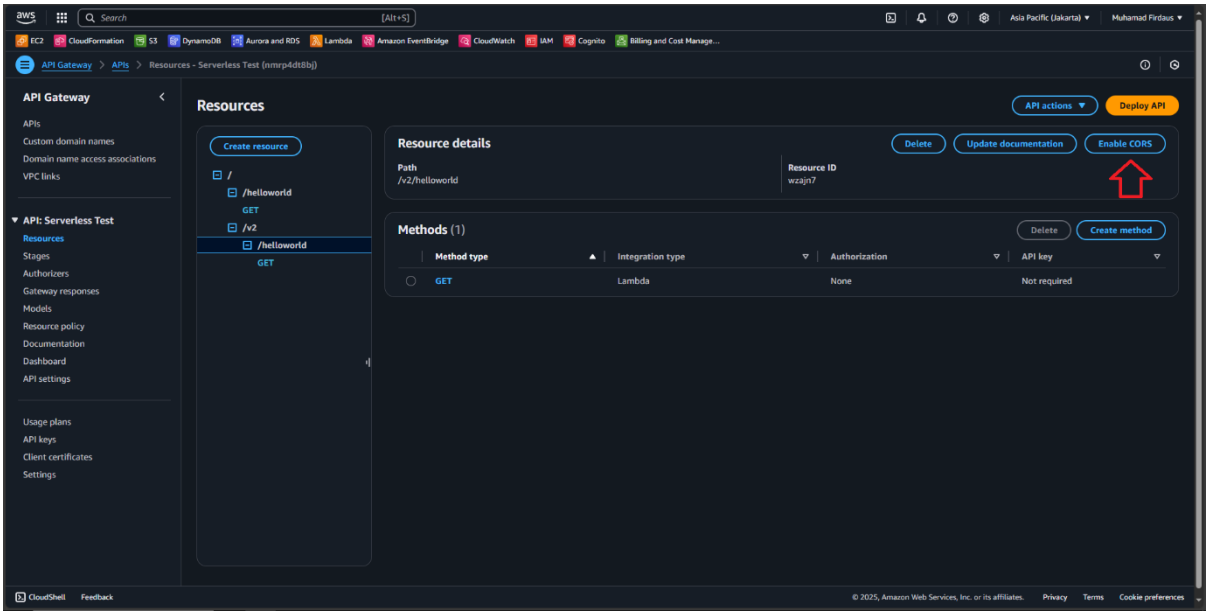
20. A new pop up will show and ask you regarding the configuration for the canary promotion, checks all the box to update the settings of the gateway and then click on “Promote canary” to promote the new developed canary.

21. After a successful promotion we can see the GET method from the v2/helloworld is now live and deployed to the PROD. This means the helloworld will be changed with the v2 version of it.

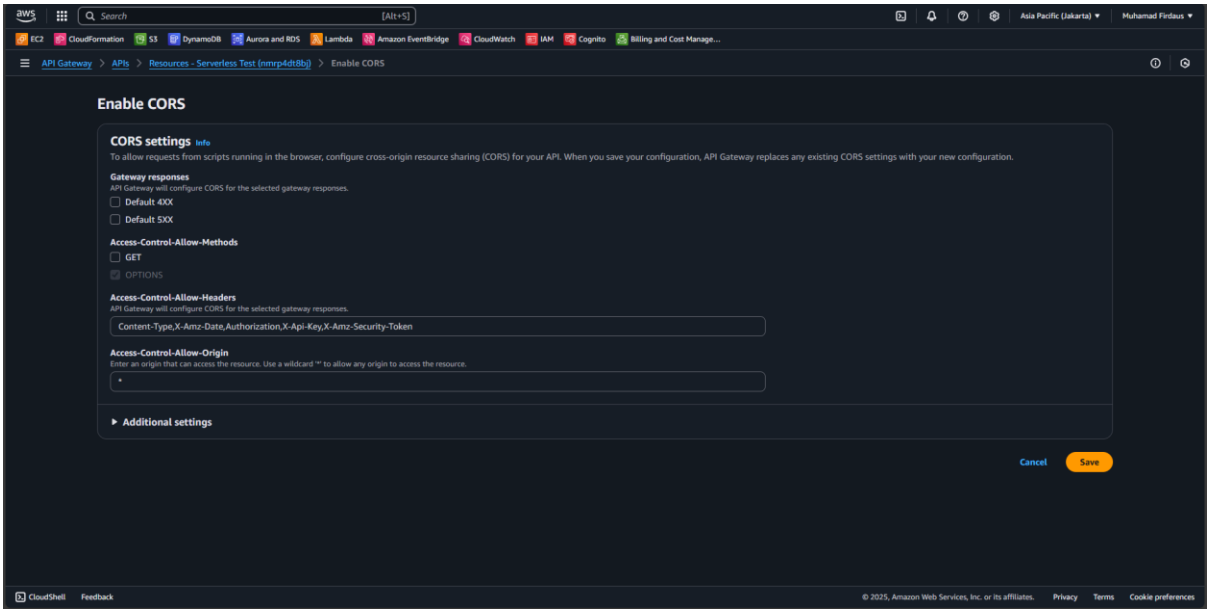


PART 4 : Enabling CORS for the API Gateway

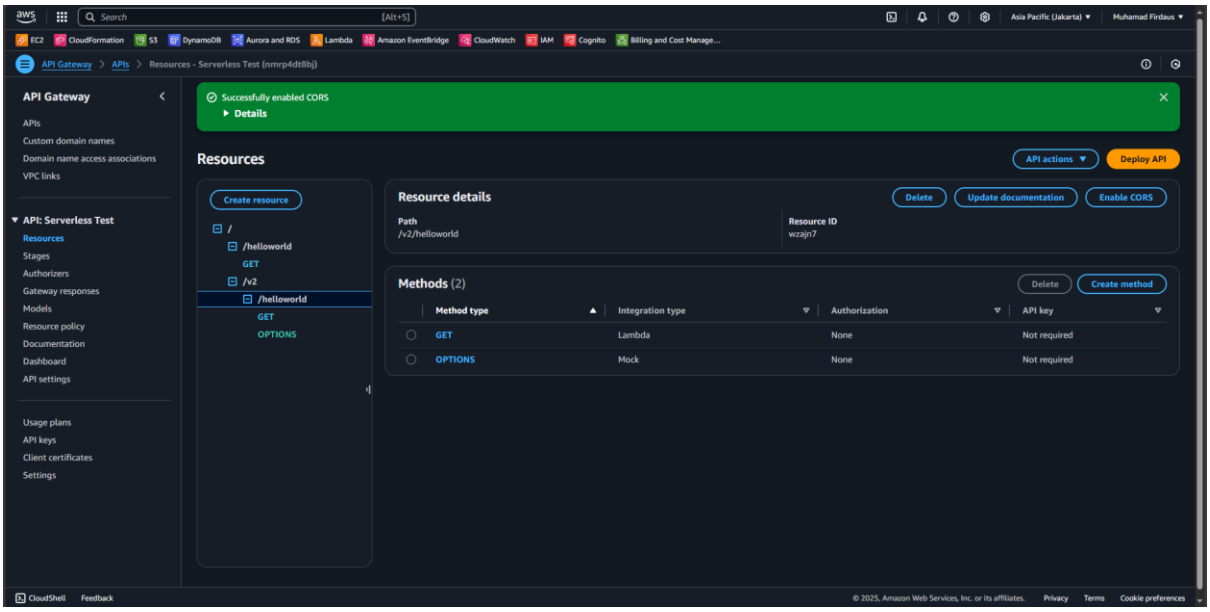
1. On the “Resources” menu, click on the resource you want to use to enable the CORS method. Click on the “Enable CORS” button on the top right section.



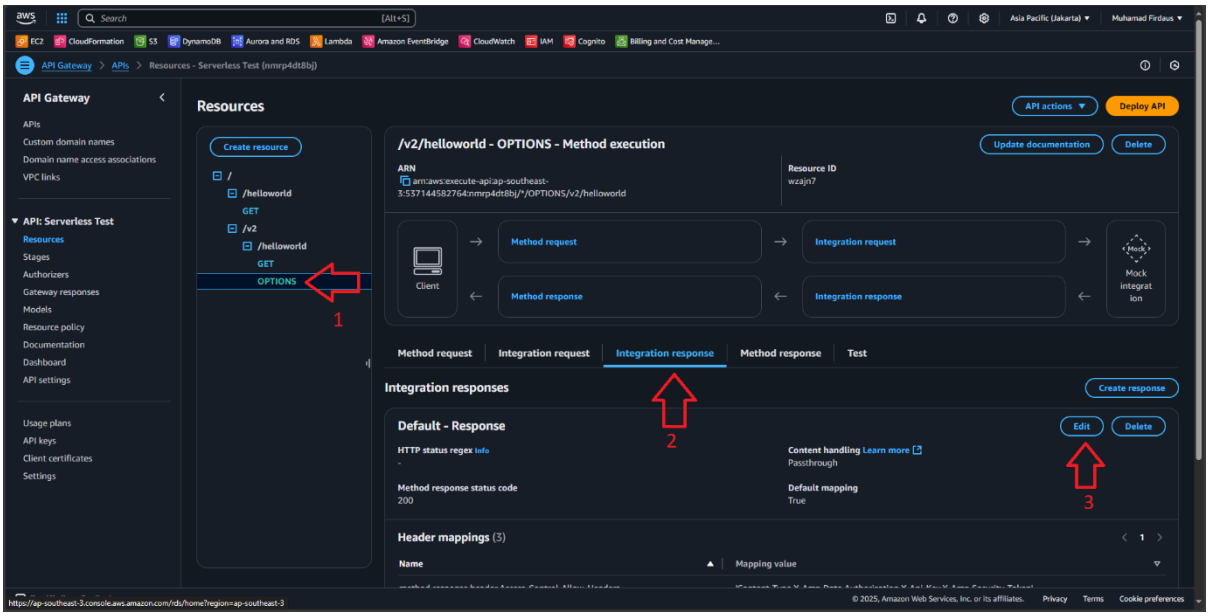
2. A pop up will show the CORS enabling menu, just choose the default and save this configuration to proceed (We will be using the OPTIONS method).



3. An OPTION resource will appear under the V2 and GET method we created earlier, this confirms that the CORS is already activated but not yet deployed on the PROD since we enable canary.

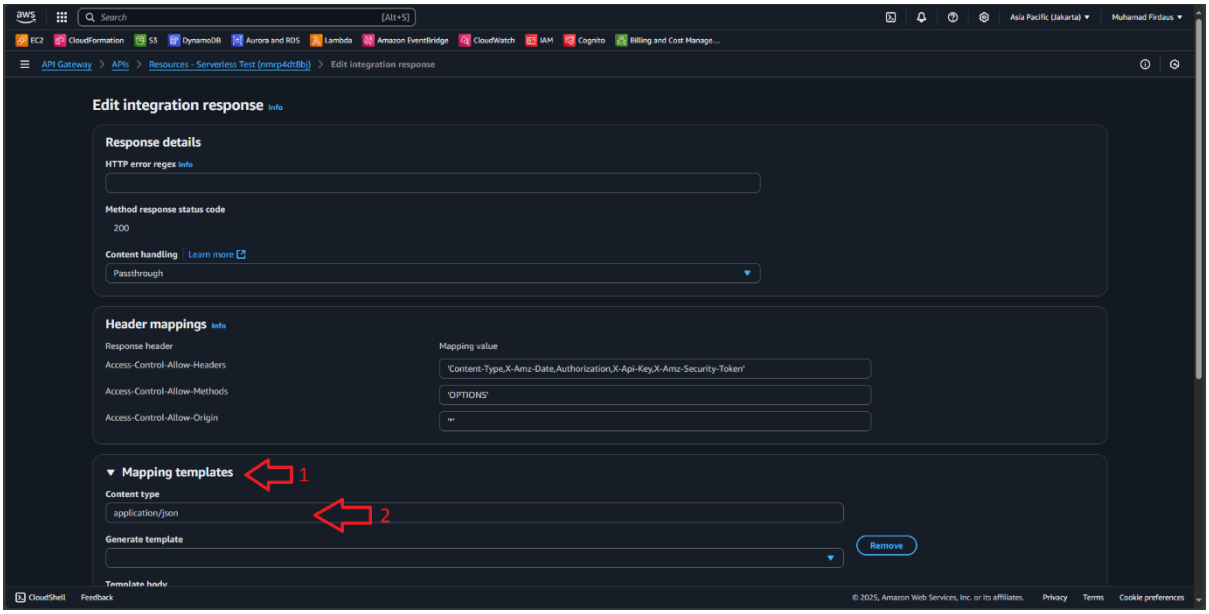


4. Before we deploy to the PROD env, click on the OPTION resource and choose the "intergration response" then "edit" to edit the response as we will use this after this one.



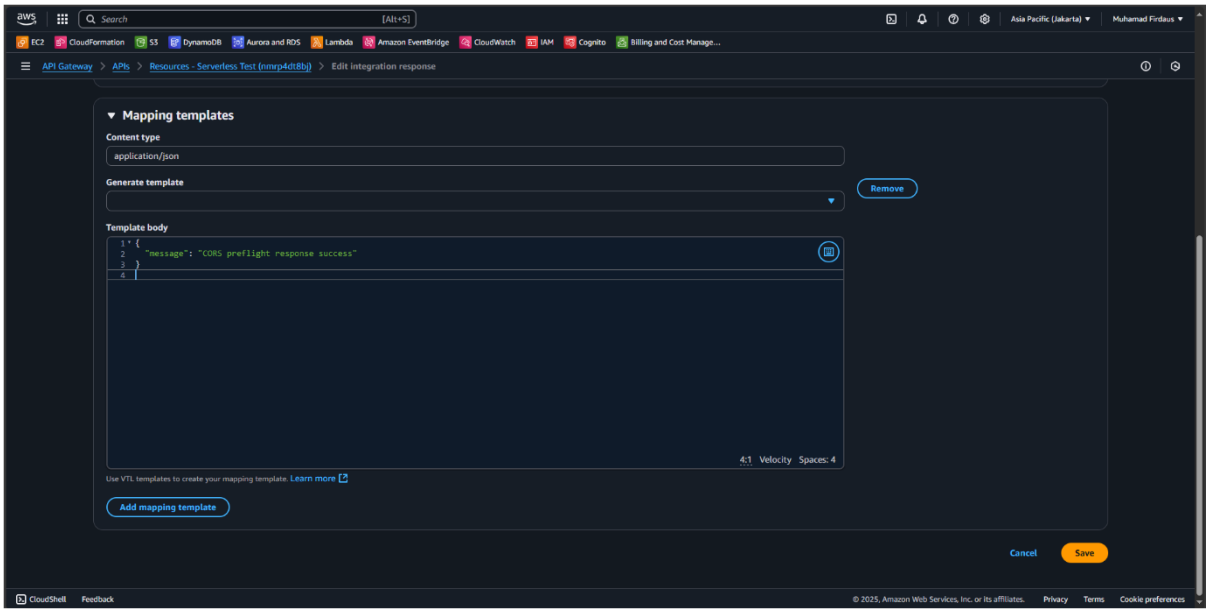
5. Immediately scroll to the bottom section and click the “Mapping Template” to open the advanced menu, then fill the content type name as below or however you like.

```
application/json
```

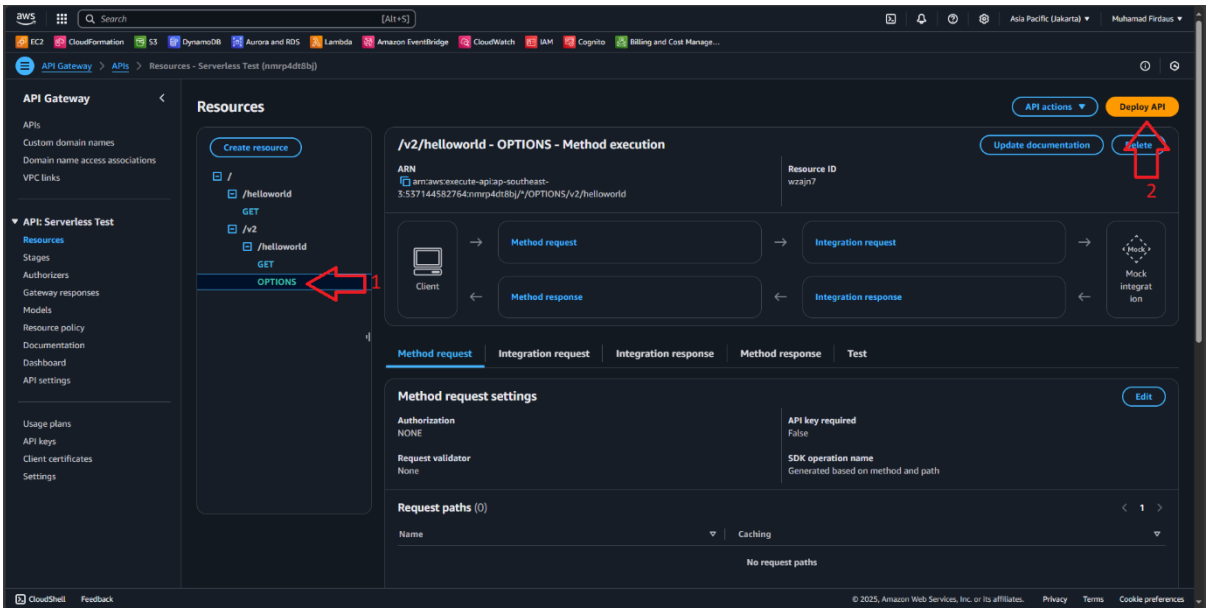


6. Scroll to the bottom again until you see the “Template body” section, fill it with the message you want to get during another API testing we will be having after this edit.

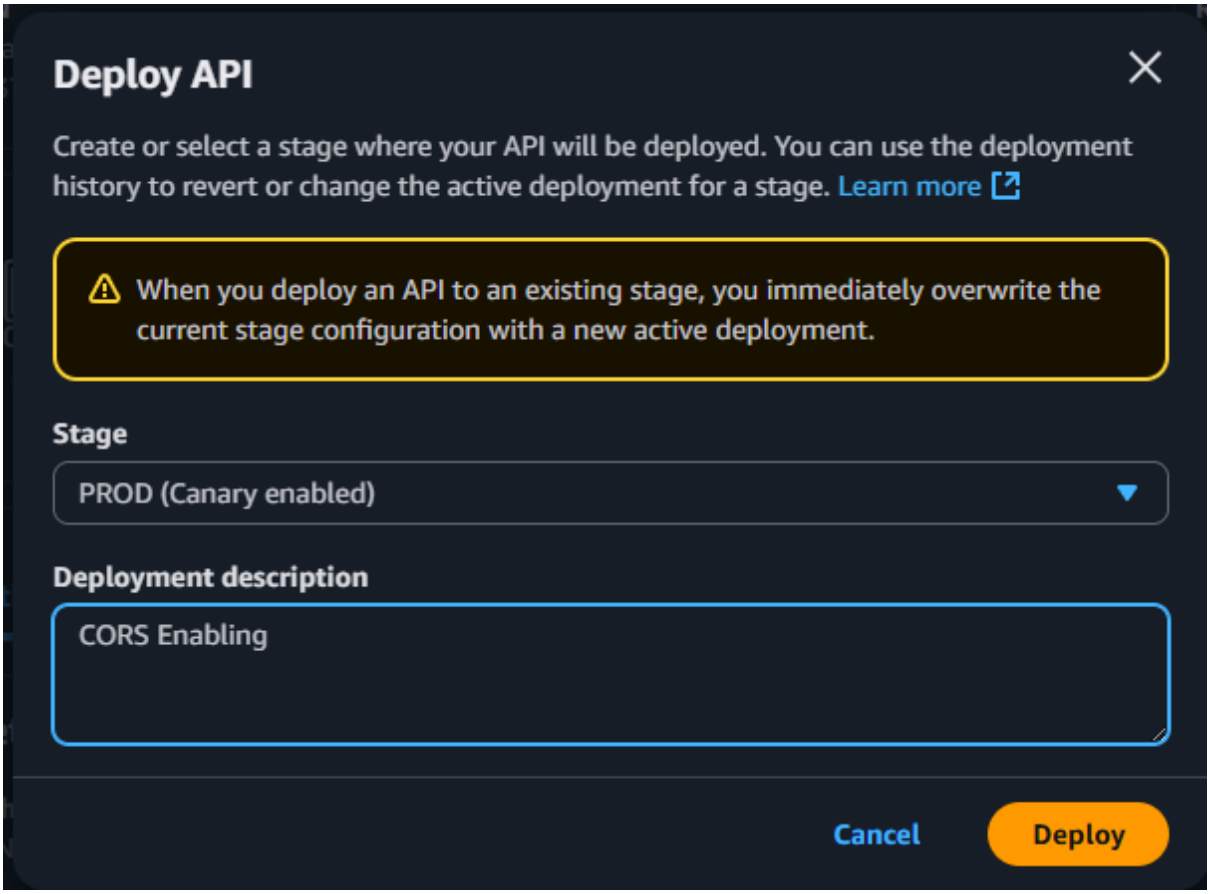
```
{
        "message": "CORS preflight response success"
}
```



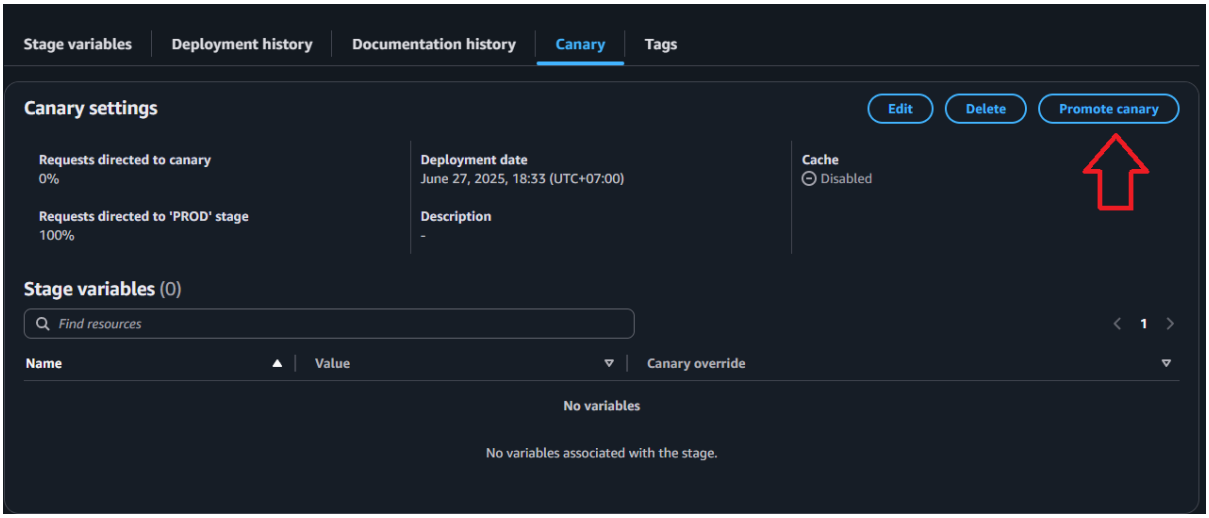
7. After successfully editing the response, finally its time to deploy this version to the current PROD env. Choose the “Deploy API” to deploy this version of the API to the gateway.



8. Confirm the changes and updates.

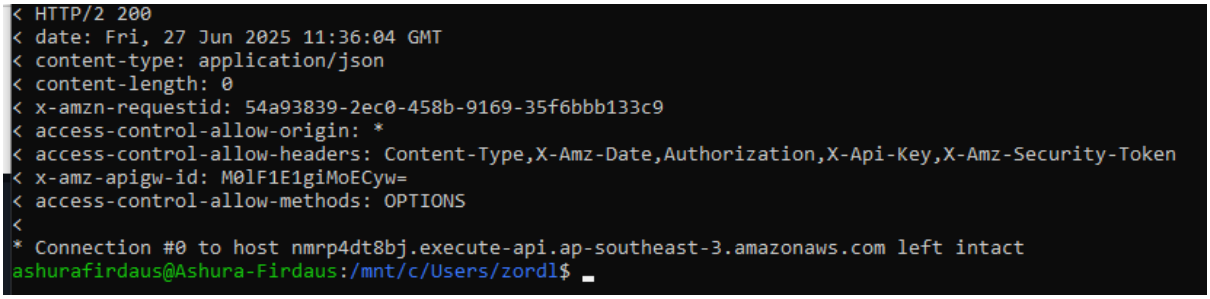


9. Finally, we go to the canary section and use that Promote canary button to promote this API version to the PROD env.

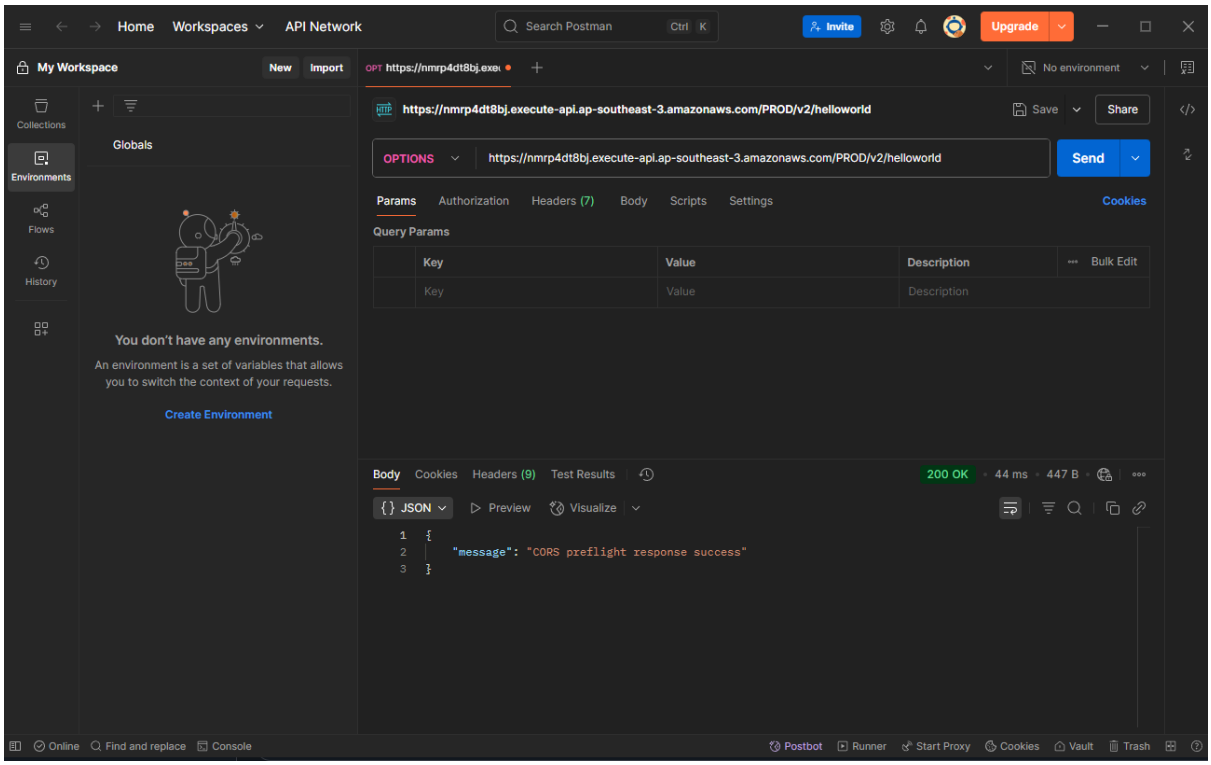


10. Promote the canary using the configurations you want.

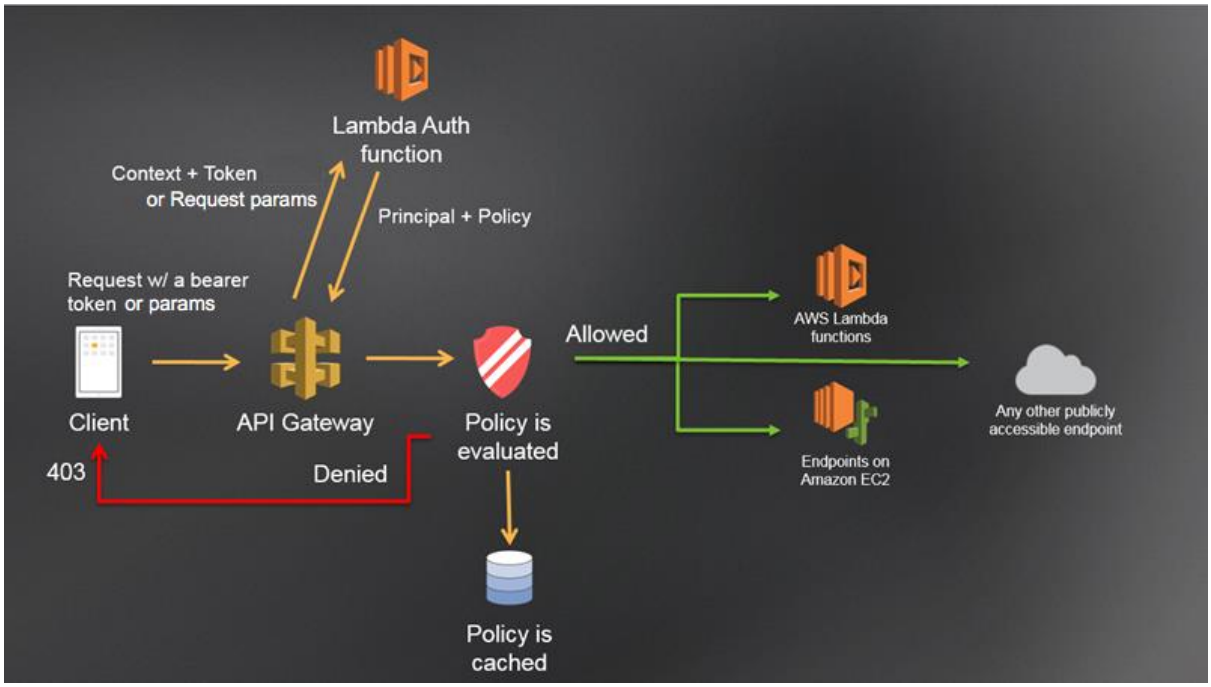
11. Check it via curl to see if the API Gateway is responsing with the 200 OK message.



12. Or use Postman API or any API testing tools you want to use to check the changes and see the response now confirming that the CORS is now already applied to the PROD env using the OPTIONS method.



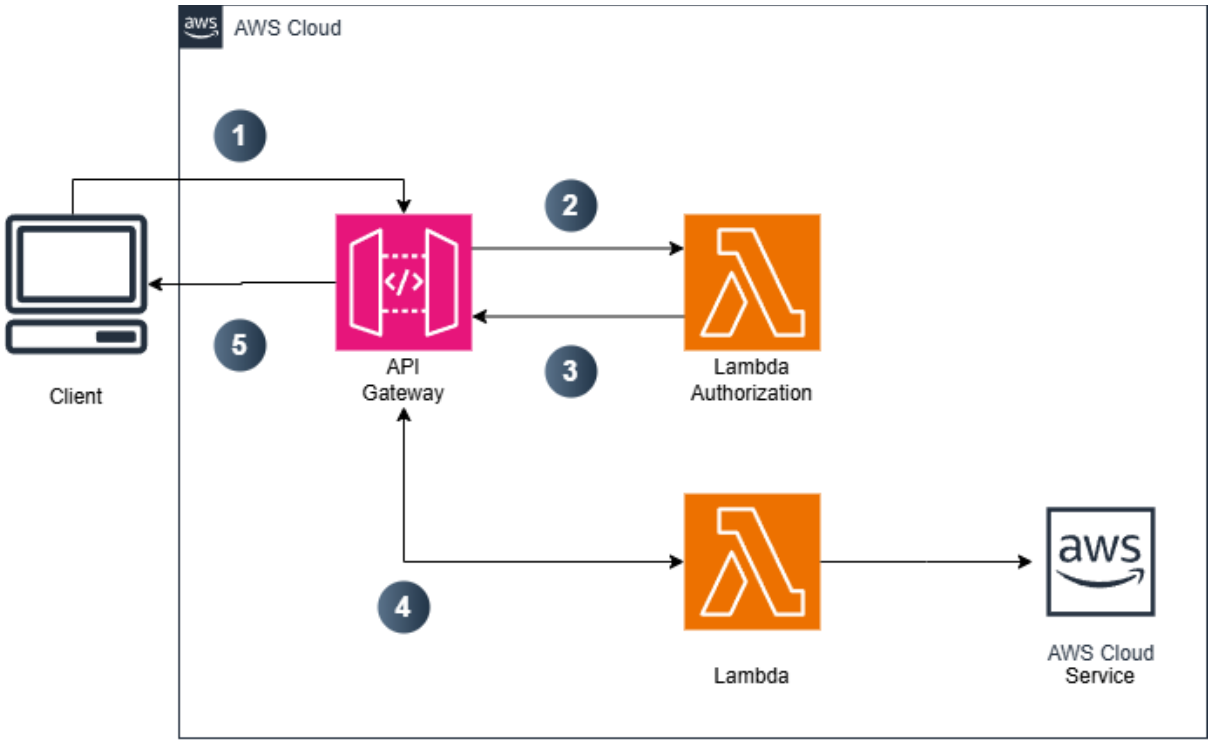
PART 5 : Control your access using Lambda Authorizer



The diagram shows the authorization workflow for a Lambda authorizer.

(Source : https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-use-lambda-authorizer.html)

Architecture :



1. Open up command line to use the openssl rand command, in this case I'll be using my WSL environment to create the secret key for the AWS lambda authorizer.

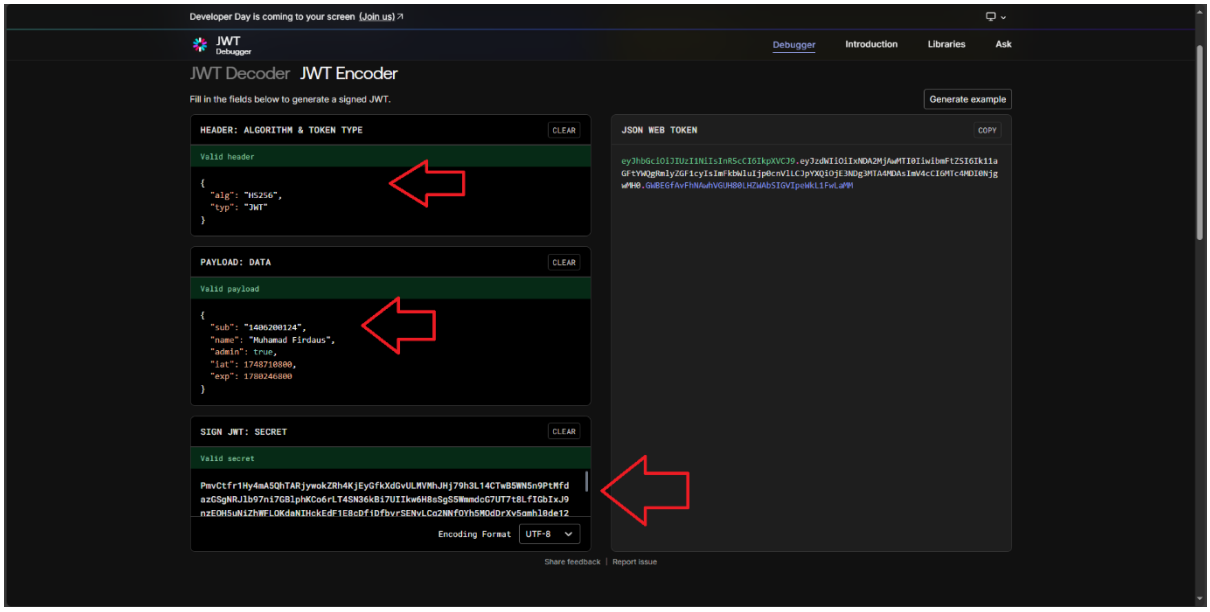
```
openssl rand 256 | base64
```

2. Copy the secret and go to the jwt.io website to and choose the JWT Encoder and fill the value such as below.

```
Header: Algorithm & Token Type

{
  "alg": "HS256",
  "typ": "JWT"
}
```

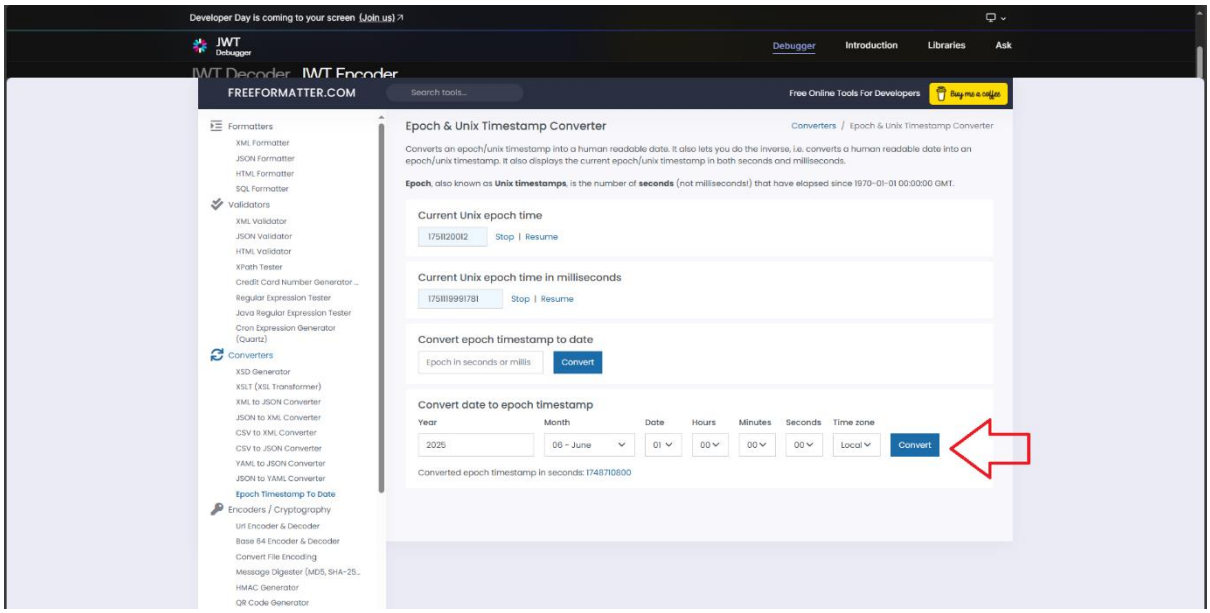
```
Payload: Data

{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true,
  "iat": 1516239022
}
```

```
Sign JWT: Secret

fill with your secret
```

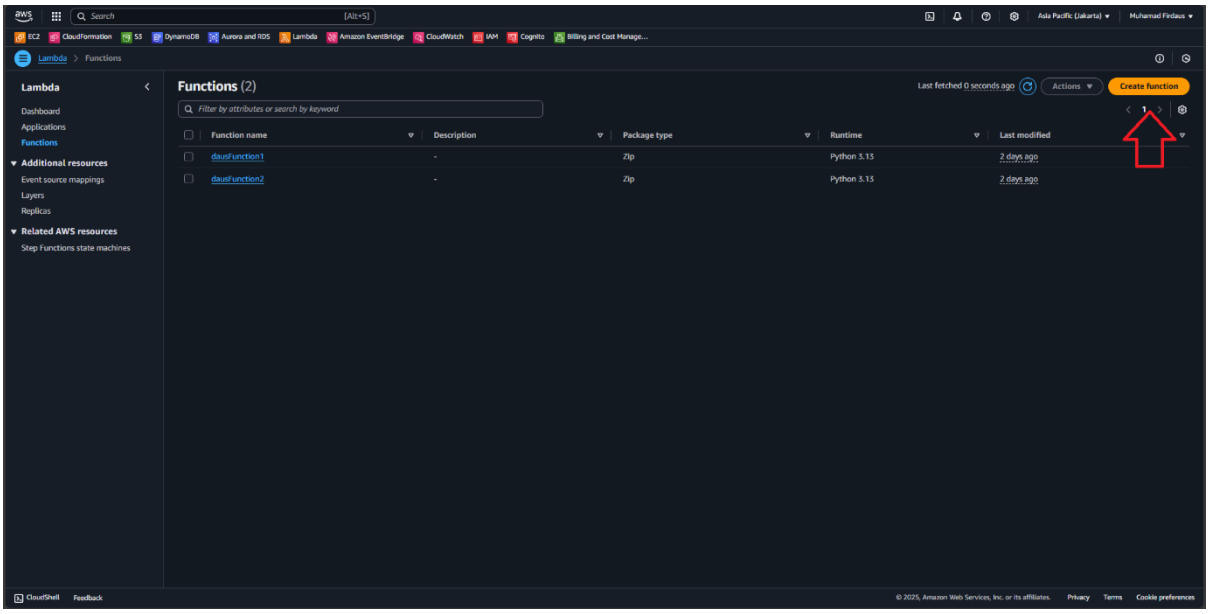


3. For the iat, we can use the freeformatter tools below and fill with the date format you'll be using.

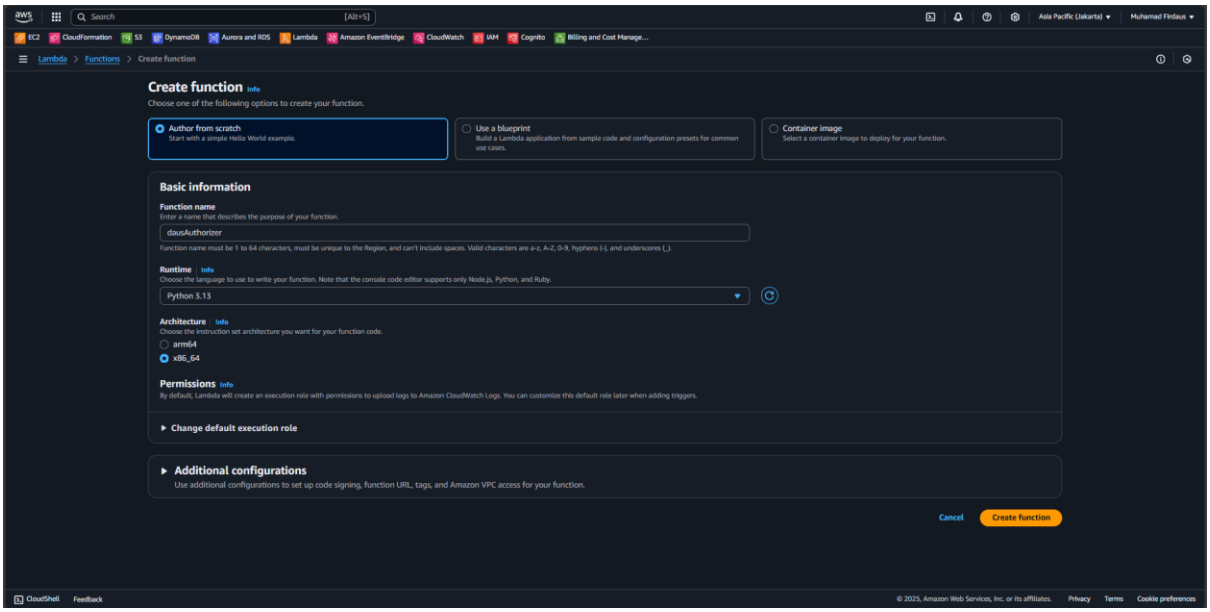
https://www.freeformatter.com/epoch-timestamp-to-date-converter.html



4. Open up AWS console and proceed to the lambda section to create a new function.

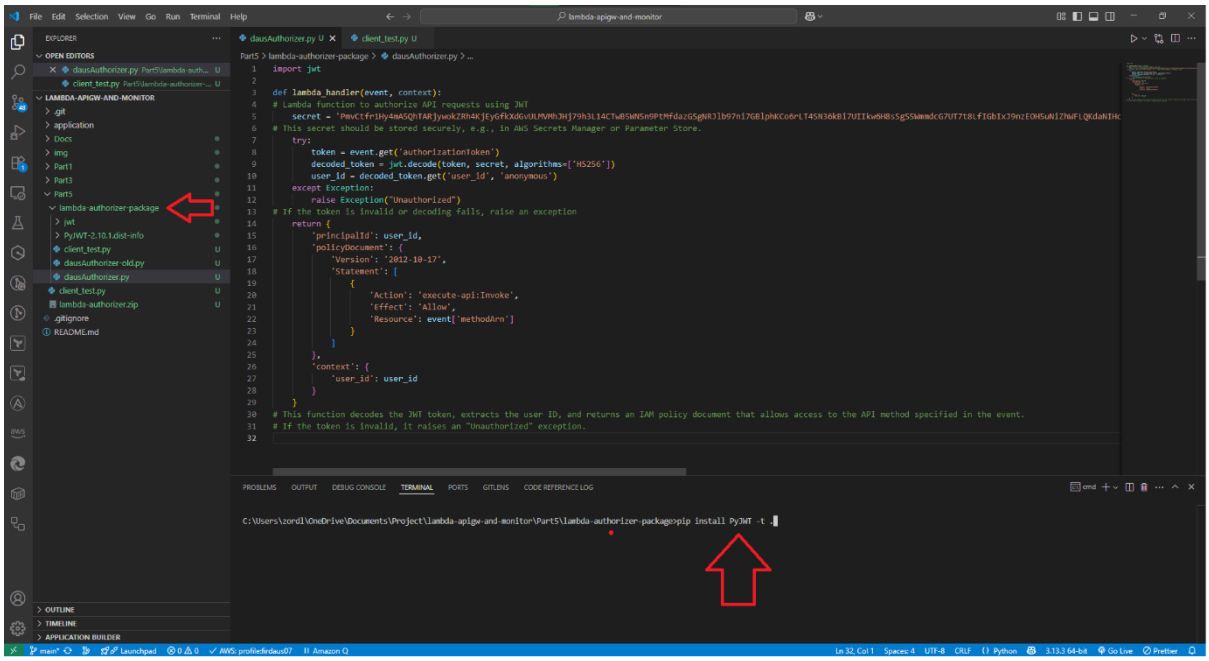


5. Fill out the new function to set it up as our lambda authorizer.



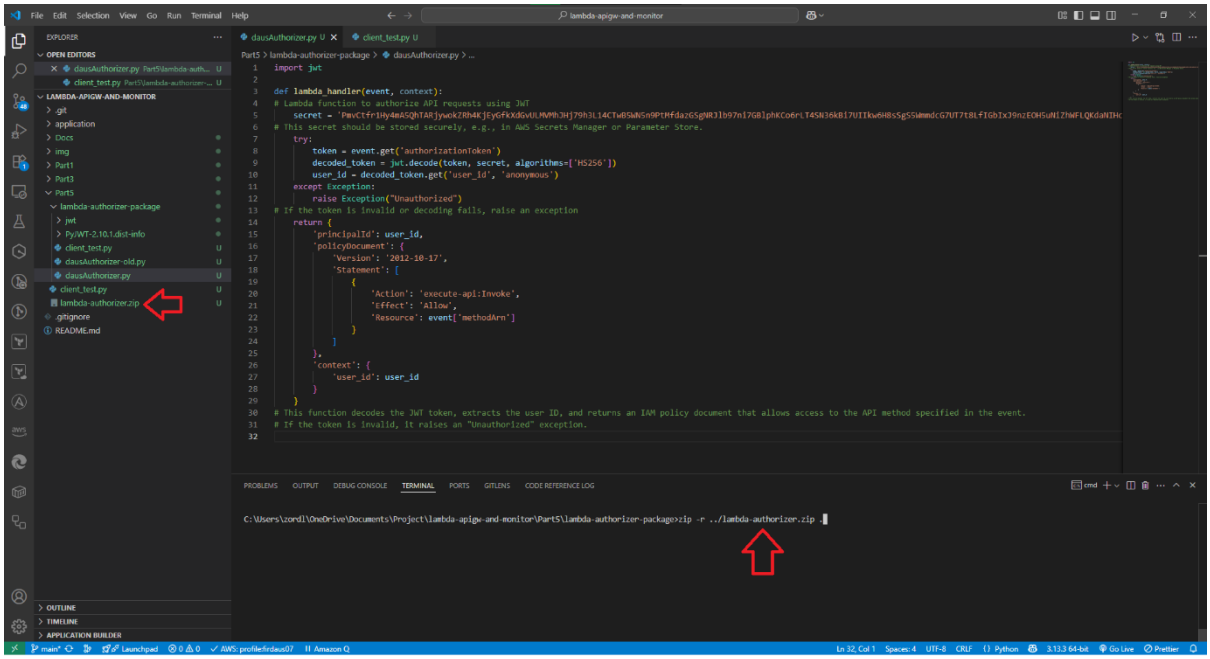
6. Create a new directory to set up as a zip file that we will be using later and install the PyJWT library inside of it (Your directory will look empty at first).

```
pip install PyJWT -t .
```

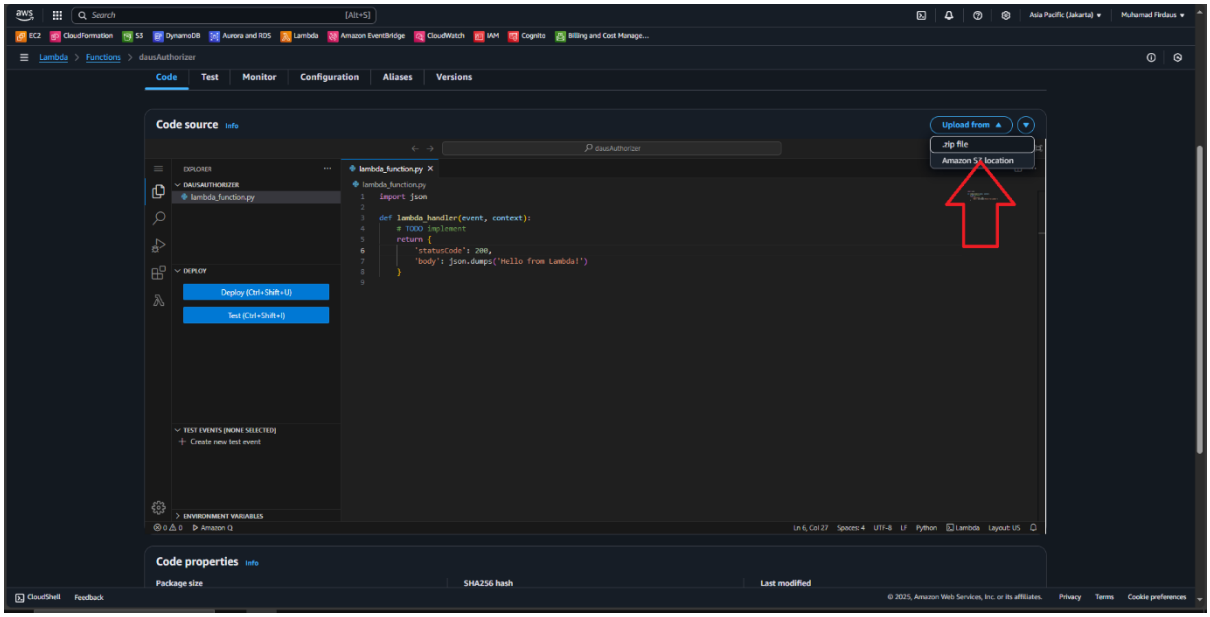


7. After the library has show up inside the directory, we proceed to zip all the directories using below command.

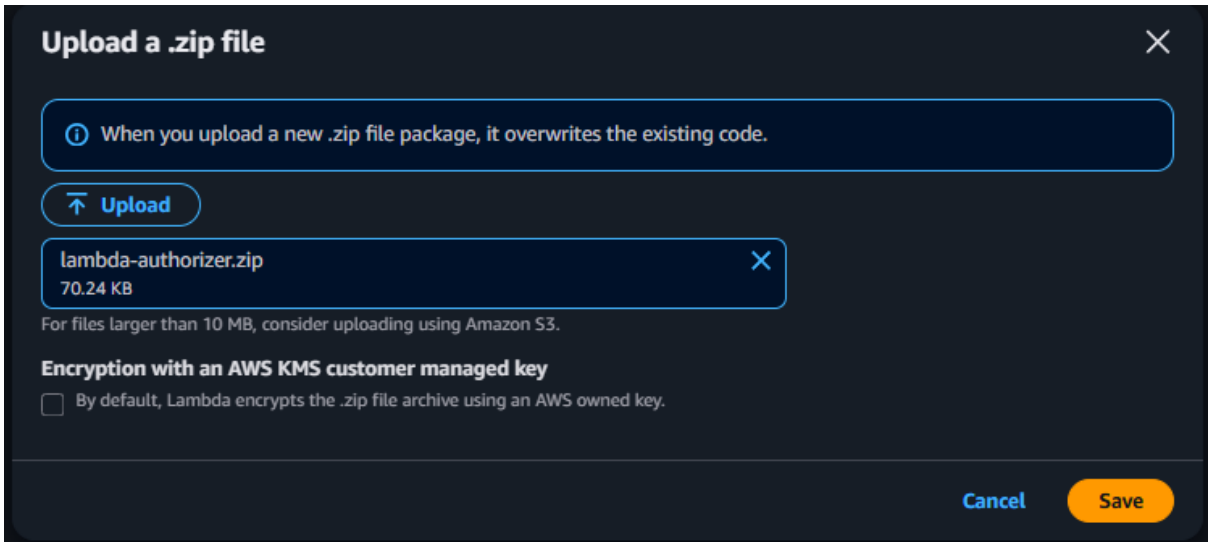
```
zip -r ../lambda-authorizer.zip .
```



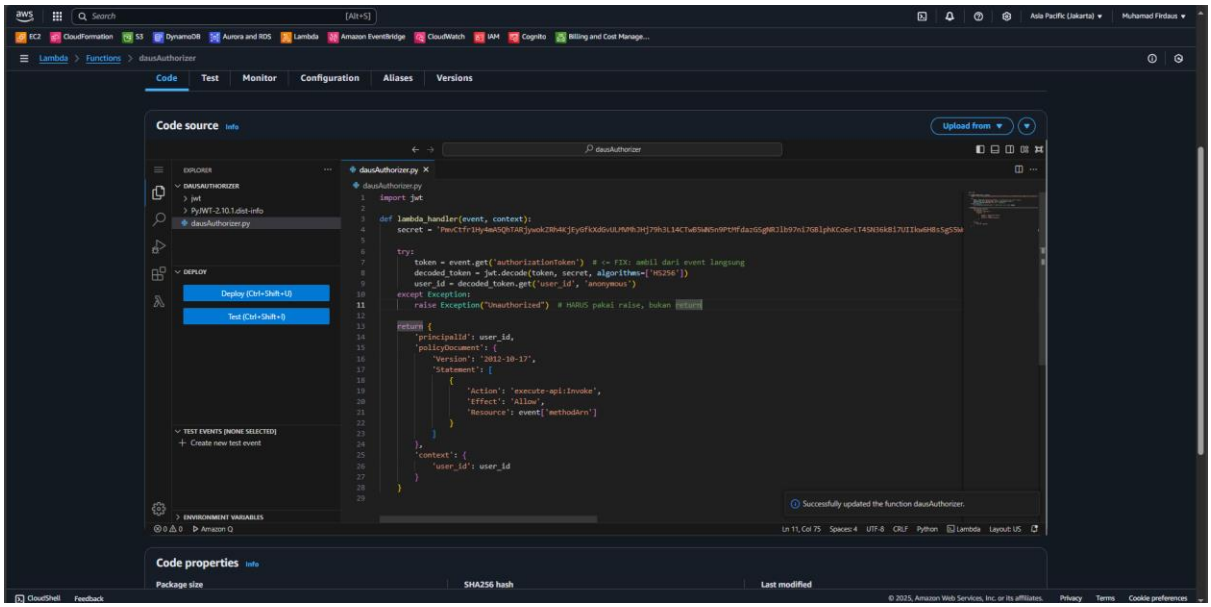
8. Back at the newly launched lambda authorizer function, choose to upload from the zip file.



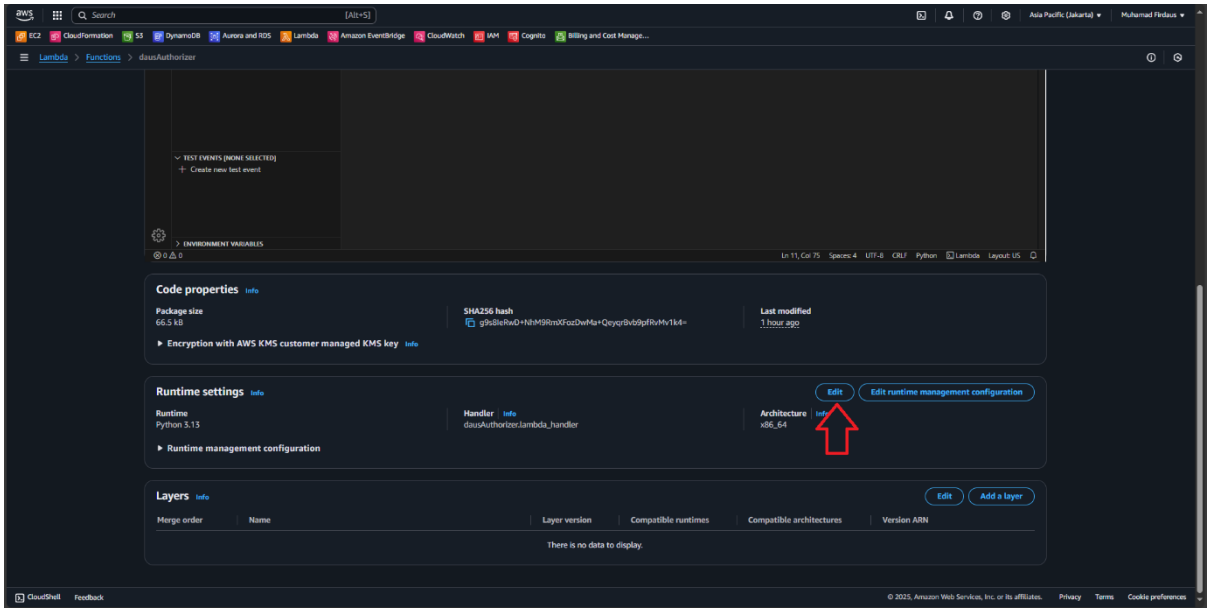
9. Confirm that the uploaded file is the zip we just created earlier.



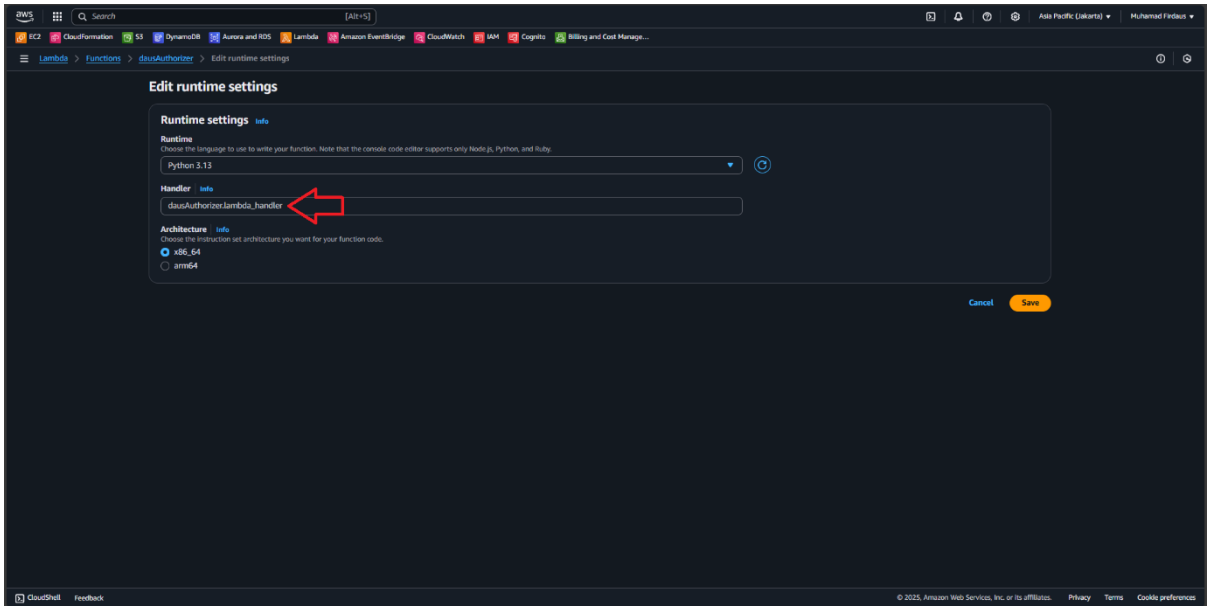
10. The python code then will show up and check if its correct.



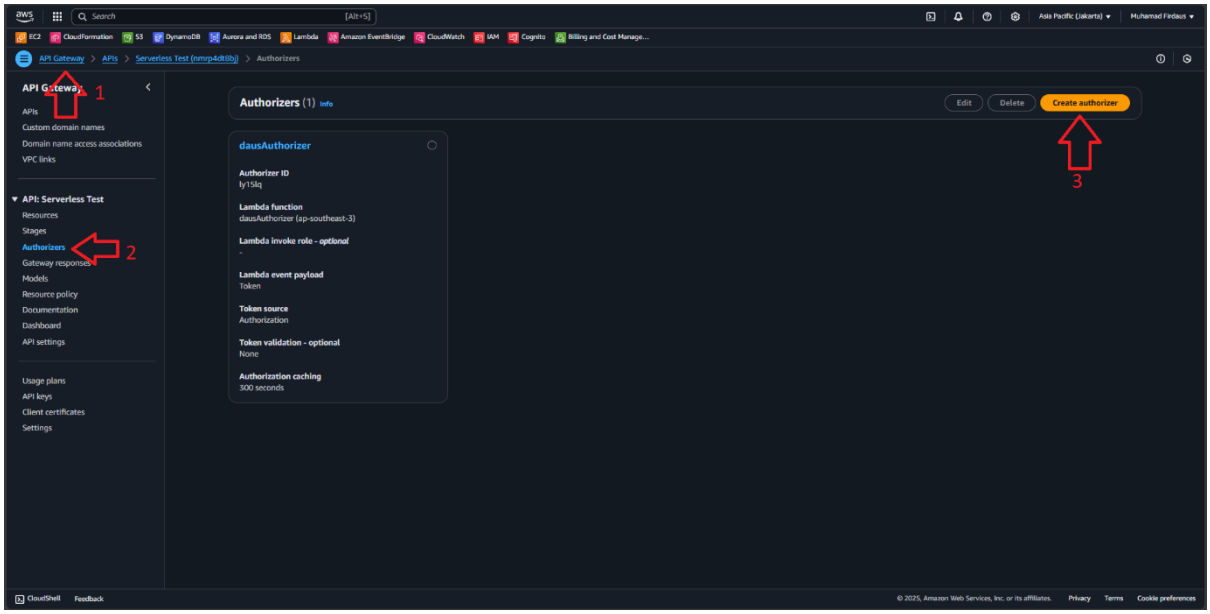
11. Scroll down and choose edit button on the runtime settings.



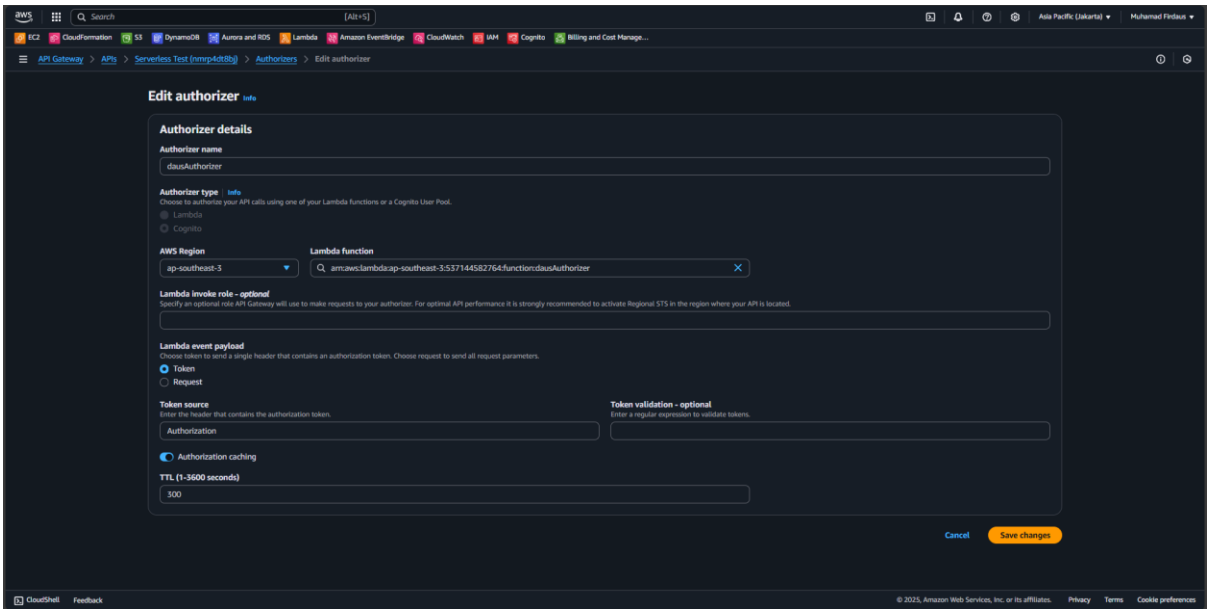
12. Choose the header as your authorizer file name, make sure it is the same to avoid connection error (in this case it is dausAuthorizer).



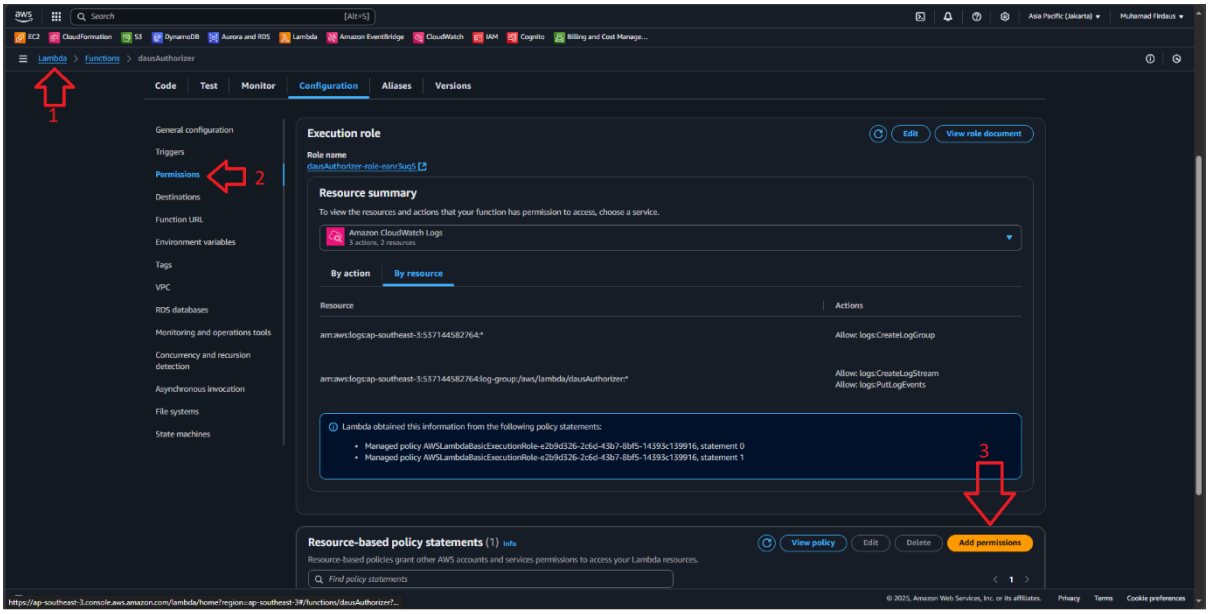
13. Go to the API Gateway and choose the authorize menu, click on the "Create authorizer" button to create a new authorizer.



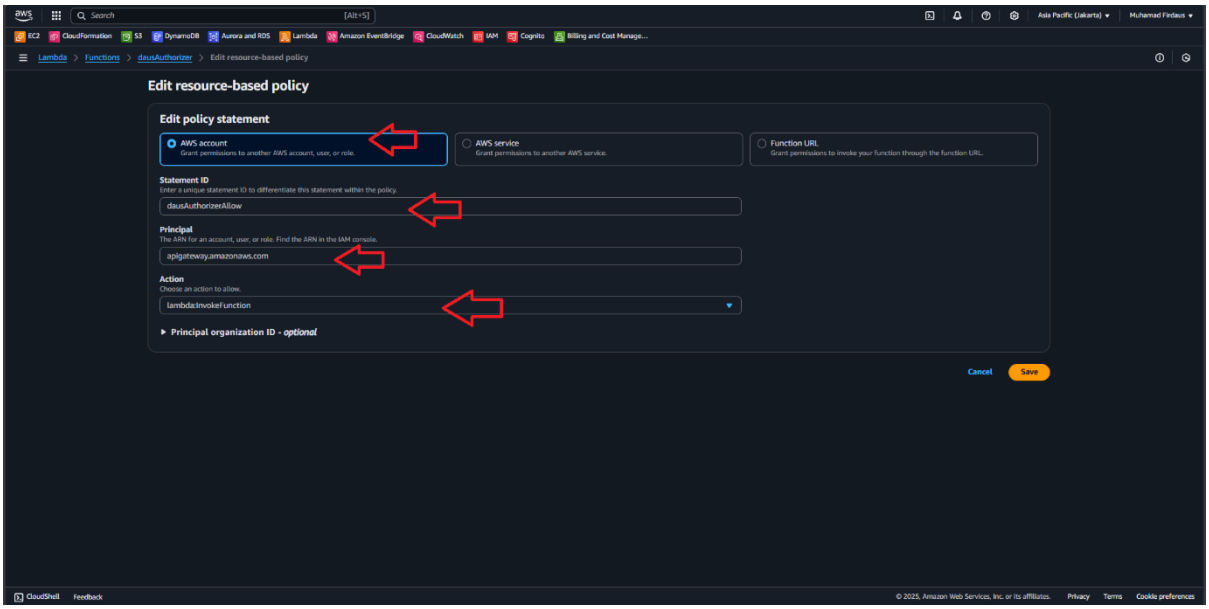
14. At the Edit authorizer menu, fill the details with the same file you created earlier, in my case the name is “dausAuthorizer”, choose the lambda function as the same “dausAuthorizer” and then fill the token source as “Authorization”.



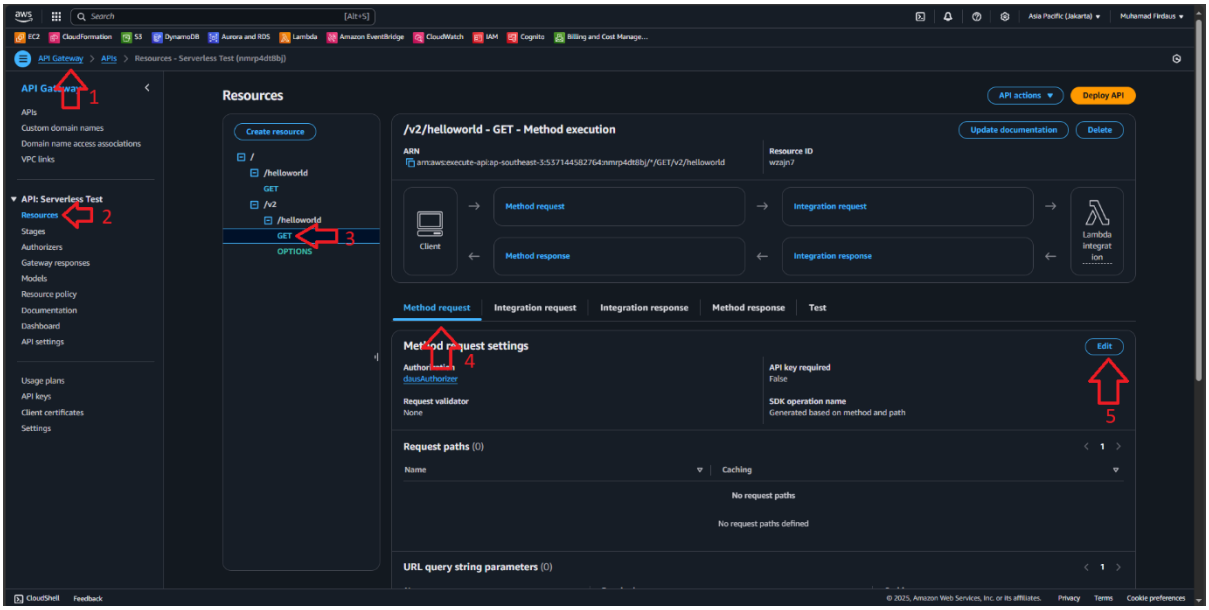
15. Return back to lambda and proceed to the permission section, then click on the “Add Permission” button so we can add a permission for the lambda authorizer.



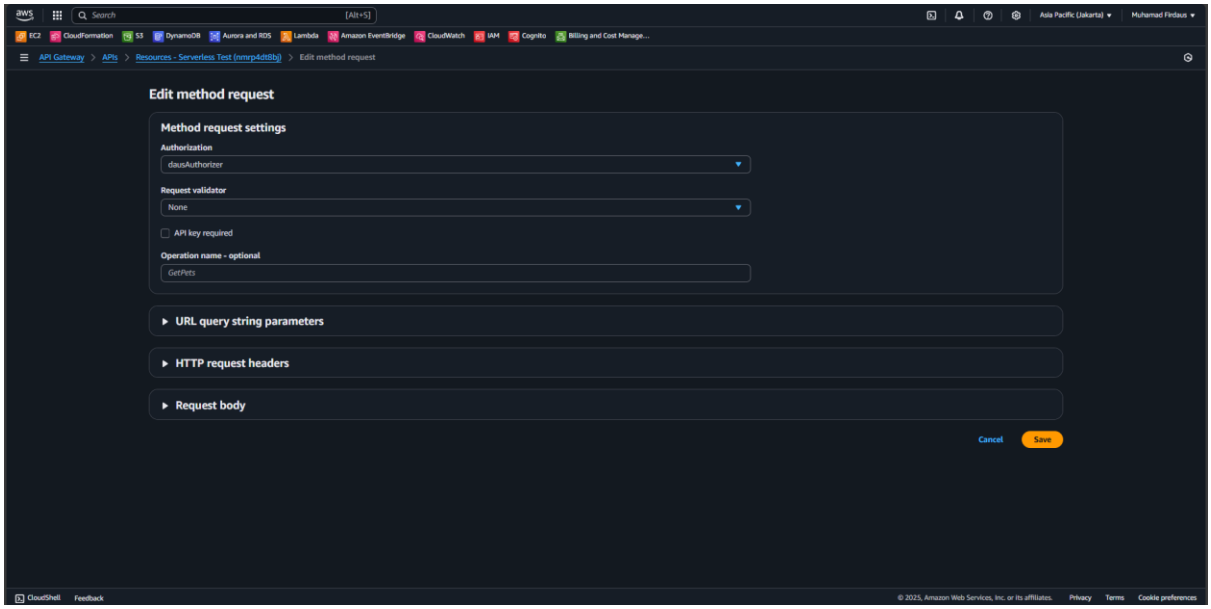
16. On the policy statement configuration, choose 1. AWS account, 2. Statement ID = dausAuthorizerAllow or anything you wish to name it, 3. Set the Principal as “apigateway.amazonaws.com” (Because API Gateway needs permission to invoke the Lambda function on our behalf. We grant API Gateway the permission to invoke the Lambda Authorizer function, since it needs to validate tokens before forwarding requests to the backend.) and the action is lambdaInvokeFunction and click save.



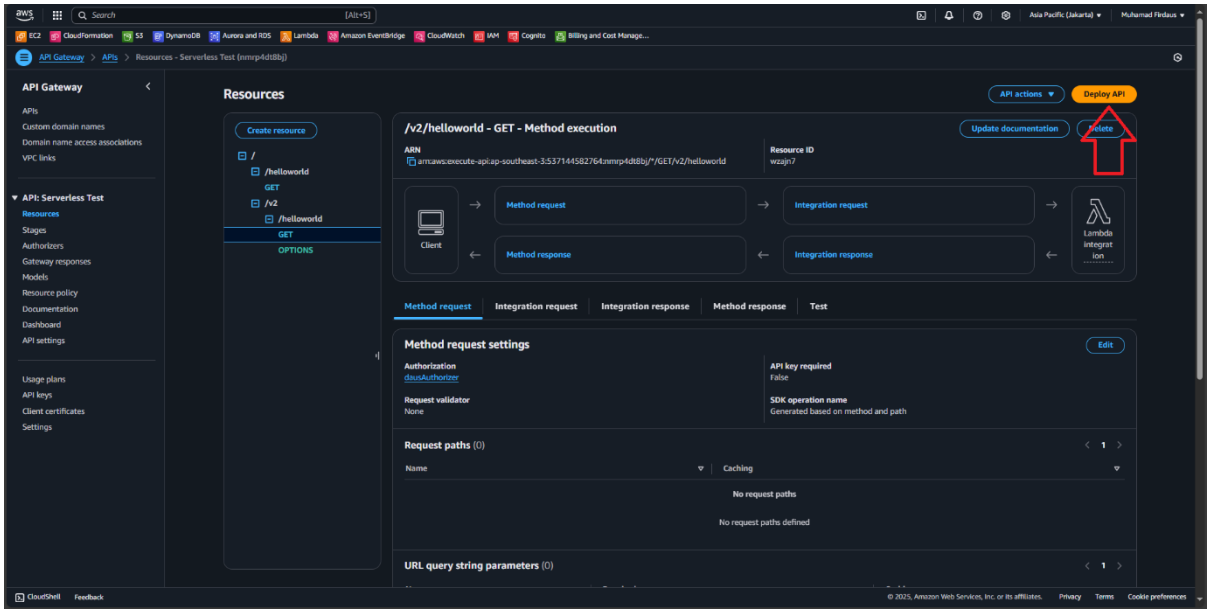
17. Then return to the API Gateway, click on resource and choose the v2 GET menu, choose method request and edit this section.



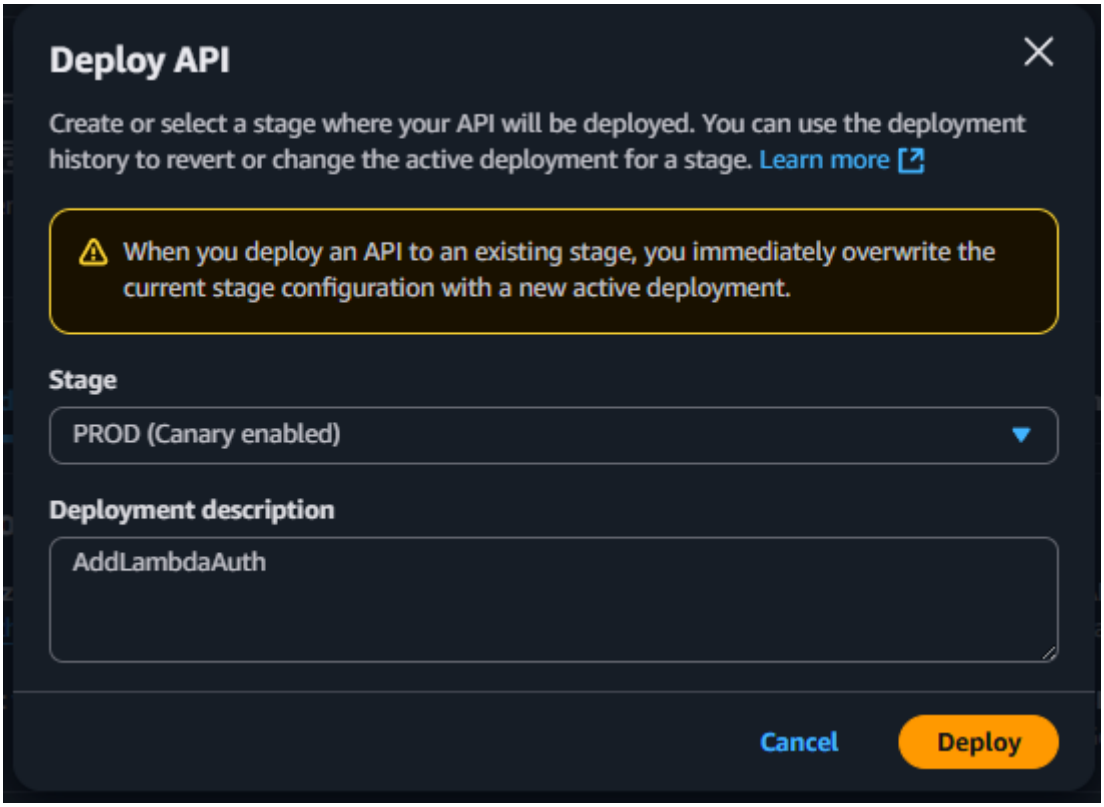
18. On the edit method request, fill out the details with the set up we created earlier making sure its matching. In my case it is Authorization = dausAuthorizer and click save.



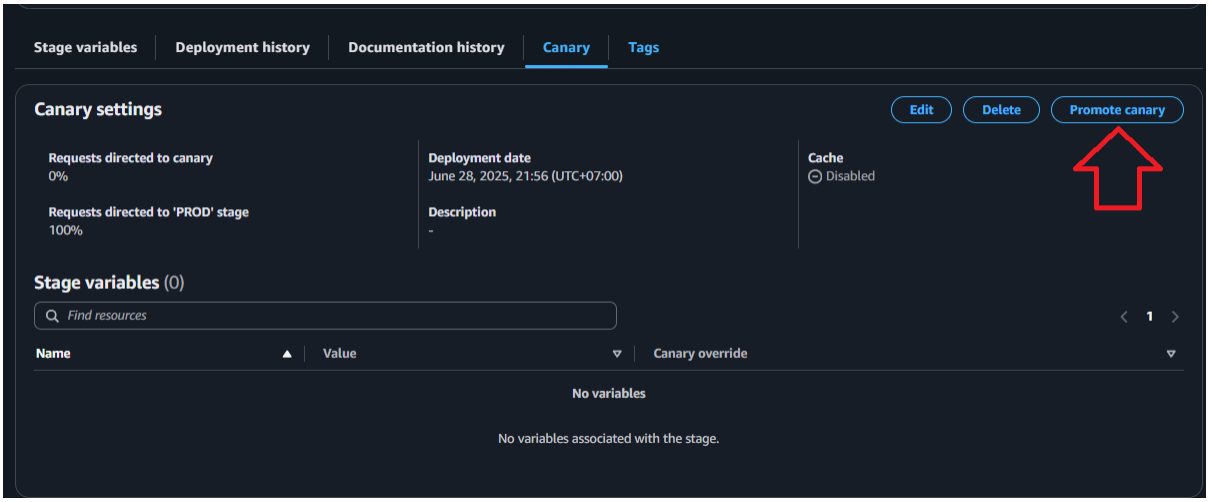
19. Click on Deploy API to deploy our changes.



20. Choose the stage, in my case it is still on the PROD with canary enabled, and add the description to make sure you know which version you just deployed.

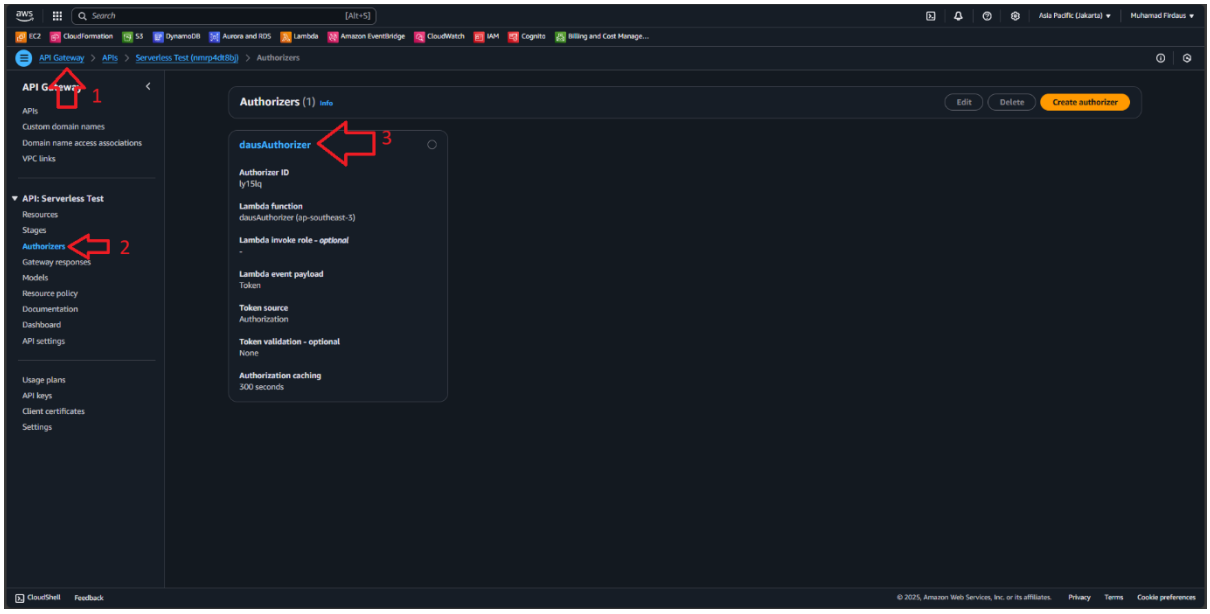


21. After successful deployment, as usual since we use canary we promote this canary to the PROD before it can be invoked.

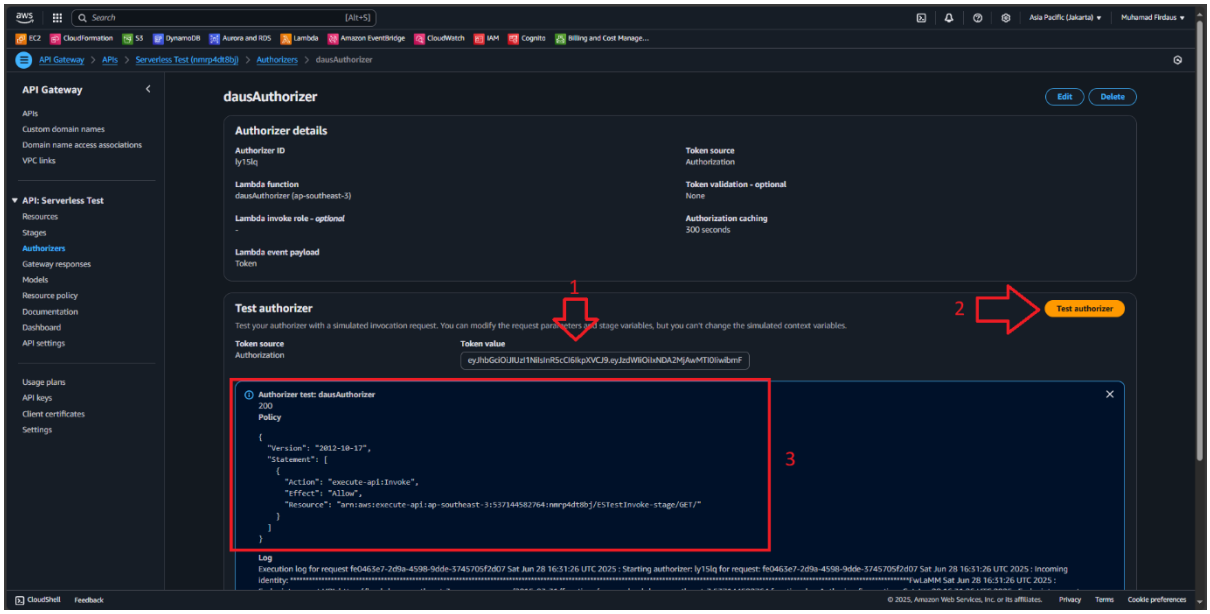


22. Promote the canary to proceed.

23. Then return to the API Gateway to test our authorization by going to the authorizer and click on the name of the authorizer we created earlier.



24. Put the JWT token you get from the secret you encoded on the JWT.io website and test.



25. Or we can use a python code that used the JWT token to request to the API Gateway, if successful you will see the 200 message with the greetings.

python client_test.py