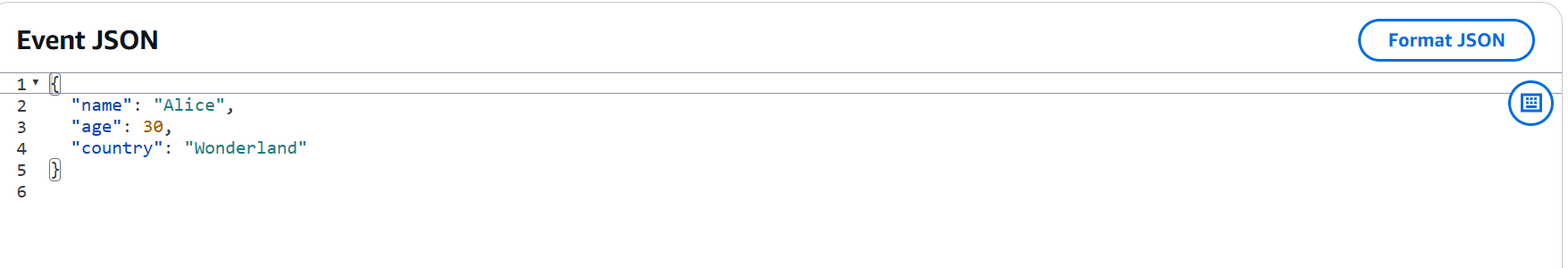
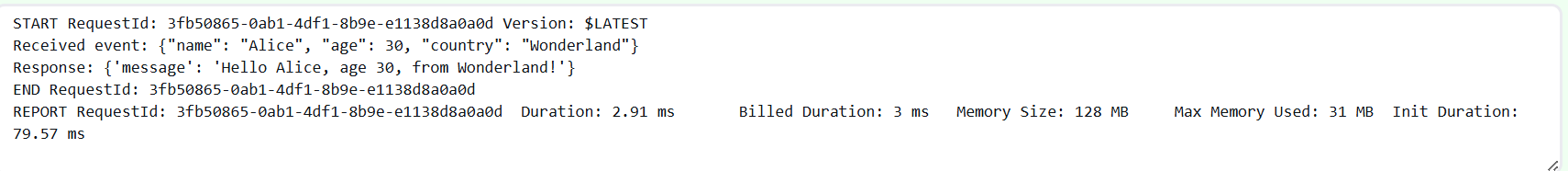
At first I went to lambda functions by simply writing “lambda” in search bar. After that I created function in pycharm and this is the code:



This is the test for my function:  


And after testing I get this output:

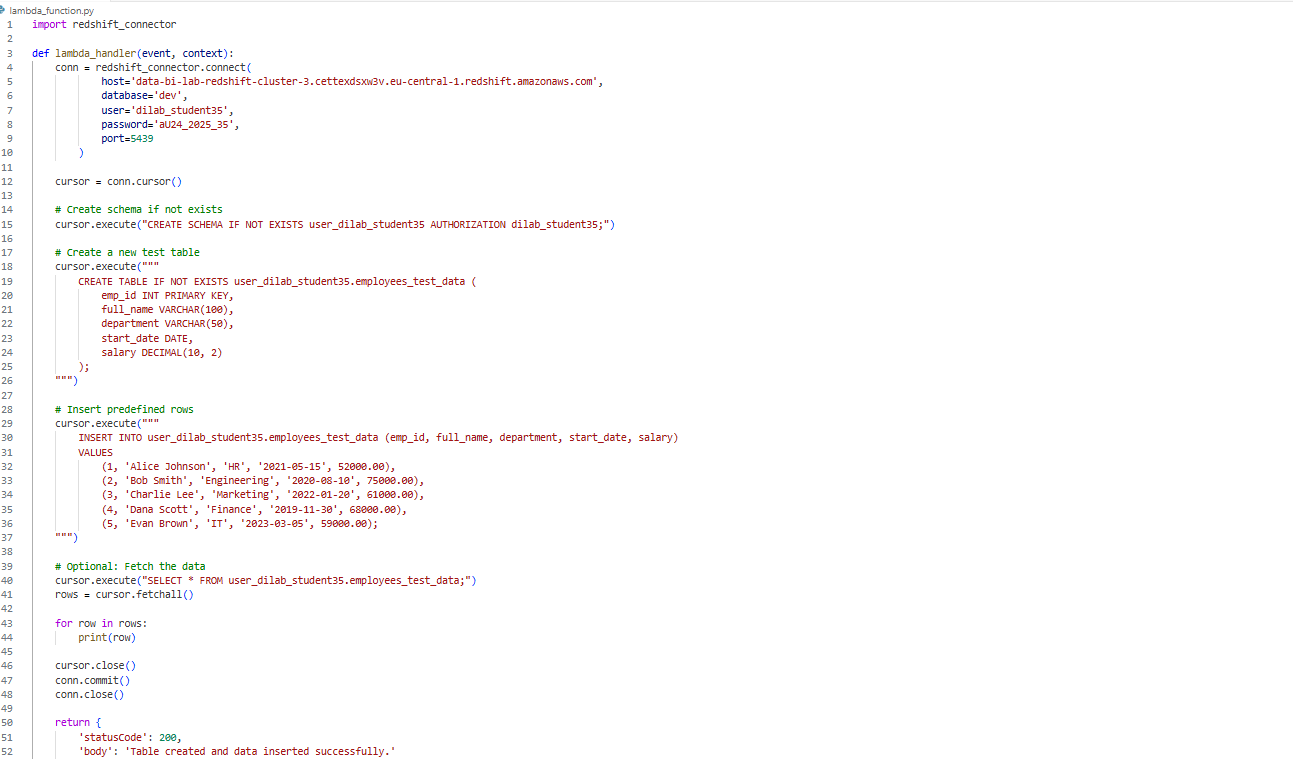




As I understood this task description there was no need to use the exact code that was given in our example in QA session. Also I changed the tests and I hope this is also correct way for test my lambda function.

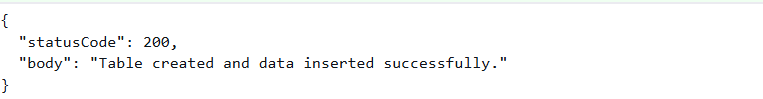
Task 2

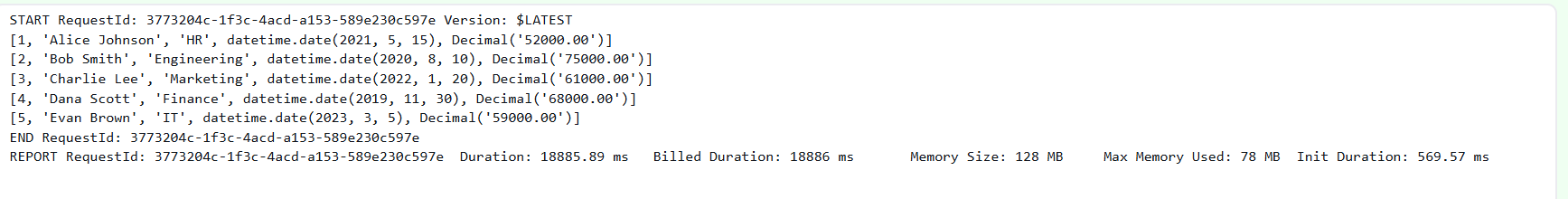
I didn’t had any tables in my cluster so I created lambda function for creation of the schema and the table. this is the code:



[https://eu-central-1.console.aws.amazon.com/lambda/home?region=eu-central-1#/functions/DavitiMatiashvili\_3?fullscreen=true&subtab=general&tab=code](https://eu-central-1.console.aws.amazon.com/lambda/home?region=eu-central-1%23/functions/DavitiMatiashvili_3?fullscreen=true&subtab=general&tab=code) and this is the link

output for this code is this:





After this I created 3 other lambda functions. First one displays some test, second one updates the table and third one again displays some test

First function code looks like this:



And test result looks like this:



Now this is the Update part:



And this is the test result:



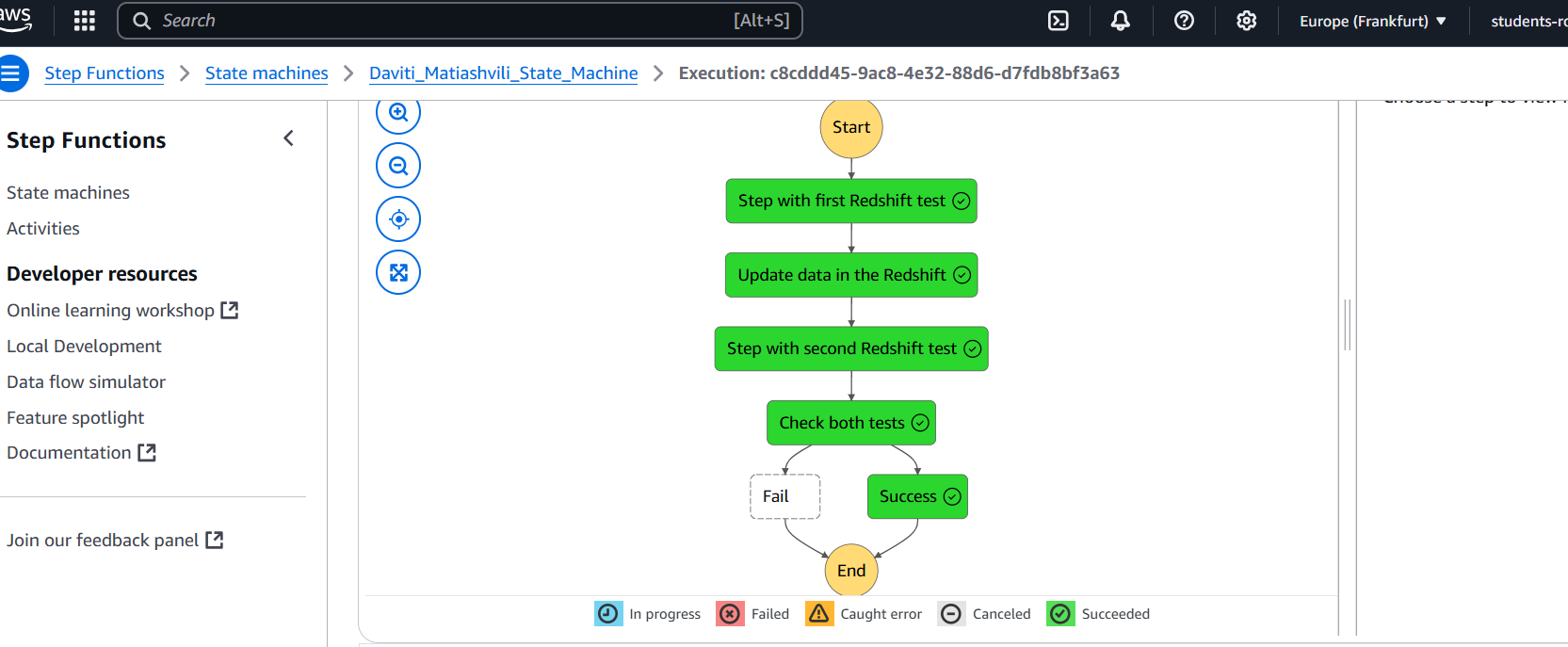
And lastly this is the second Test function:



And this is the test output:



After creating the step function and executing it:



Here is the link

[https://eu-central-1.console.aws.amazon.com/states/home?region=eu-central-1#/statemachines/view/arn%3Aaws%3Astates%3Aeu-central-1%3A260586643565%3AstateMachine%3ADaviti\_Matiashvili\_State\_Machine?type=standard](https://eu-central-1.console.aws.amazon.com/states/home?region=eu-central-1%23/statemachines/view/arn%3Aaws%3Astates%3Aeu-central-1%3A260586643565%3AstateMachine%3ADaviti_Matiashvili_State_Machine?type=standard)

When setting up a Lambda function, I defined the actual logic (in my case, for testing and updating Redshift data), specified the handler, and attached an appropriate IAM execution role. This role was crucial because it controls what services the Lambda is allowed to interact with. For example, my Lambdas needed permission to access Redshift, so I made sure the IAM role included that.

I also used environment variables for things like database connection details, and ensured that any dependencies (like redshift\_connector or other Python libraries) were packaged properly—either through Lambda layers or by uploading a deployment package.

One key point was making sure the Lambda functions could connect to services like Redshift. That meant:

* Placing the Lambda inside the same VPC and subnet if the Redshift cluster was in a VPC.
* Ensuring security groups and network routes (like NAT gateways) allowed outbound traffic.
* Making sure database connections were opened and **properly closed** in the code to avoid timeouts or issues during retries.

To orchestrate the workflow, I used AWS Step Functions. I defined a state machine where each step called a Lambda function using the arn:aws:states:::lambda:invoke resource. I passed inputs and outputs between steps using Payload.$: "$" and made sure each Lambda returned structured data (like {"test\_passed": true}) that Step Functions could understand.

I also added Retry logic in case of temporary Lambda invocation errors, like service exceptions or throttling.

A critical part was the Choice state, where I had to evaluate whether both test steps passed. This required accessing each Lambda’s response properly using JSON path syntax (e.g., $.Payload.test\_passed), and making sure the data structure was correct for Step Functions to process it. I made sure each Lambda returned clear, consistent JSON responses so that the Step Function could parse and evaluate them. For example, after the test steps, I checked both test\_passed and test\_passed\_after\_update values using a condition. If both were true, the flow continued to a success state; otherwise, it went to a fail state.