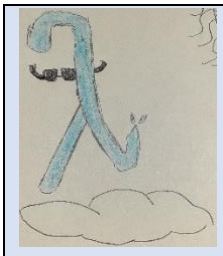
DEI
DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
TÉCNICO LISBOA

Instituto Superior Técnico
Universidade de Lisboa
Integração Empresarial / Enterprise Integration



The goal of this document is to present the details about installing, coding, deploying and testing a lambda service in Amazon AWS Lambda service.
With AWS Lambda, you run functions to process events. You can send events to your function by invoking it with the Lambda API, or by configuring an AWS service or resource to
Function: is a resource that you can invoke to run your code in AWS Lambda. A function has code that processes events, and a runtime that passes requests and responses between Lambda and the function code. You provide the code, and you can use the provided runtimes or create your own.
Runtime: Lambda runtimes allow functions in different languages to run in the same base execution environment. You configure your function to use a runtime that matches your programming language. The runtime sits in between the Lambda service and your function code, relaying invocation events, context information, and responses between the two. You can use runtimes provided by Lambda or build your own.
Event: is a JSON formatted document that contains data for a function to process. The Lambda runtime converts the event to an object and passes it to your function code. When you invoke a function, you determine the structure and contents of the event. When an AWS service invokes your function, the service defines the event.
Concurrency: is the number of requests that your function is serving at any given time. When your function is invoked, Lambda provisions an instance of it to process the event. When the function code finishes running, it can handle another request. If the function is invoked again while a request is still being processed, another instance is provisioned, increasing the function's concurrency.
Trigger: is a resource or configuration that invokes a Lambda function. This includes AWS services that can be configured to invoke a function, applications that you develop, and event source mappings. An event source mapping is a resource in Lambda that reads items from a stream or queue and invokes a function.

The following contents is presented in this document.

Contents

A. Creating a Lambda function using VSCode IDE (Java)

(Partially based on AWS Lambda Developer Guide, 2025 Amazon Web Services, Inc., public available at https://docs.aws.amazon.com/lambda/ *)*

In this step, you start VSCode and create a Maven project. You will add the necessary dependencies and build the project. The build will produce a .jar, which is your deployment package.

A.0. Add a new **Java Project…**
A.1. Select project Type: **Maven**
A.2. Select **No Archetype**
A.3. Write **examples** as group Id
A.4. Write **lambda-java-example** as artifact Id
A.5. **Select your destination folder**
A.6. Add Java class to the project.

a. Open the context (right-click) menu for the src/main/java subdirectory in the project, choose **New**, and then choose **New File…**.add the name **Hello.java**

```
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.RequestStreamHandler;

import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.Map;

import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.JSONValue;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

import java.io.*;

public class Hello implements RequestStreamHandler{
public void handleRequest(
                        InputStream inputStream,
                        OutputStream outputStream,
                        Context context)
  {
        LambdaLogger logger = context.getLogger();
        try
        {
            JSONParser parser = new JSONParser();
            BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));
            JSONObject responseJson = new JSONObject();
            JSONObject event = (JSONObject) parser.parse(reader);
            String name = new String();

            logger.log("input:" + (String) event.toString()+"\n");

            if (event.get("body") != null)
            {
                JSONObject bodyjson = (JSONObject) parser.parse((String) event.get("body"));
                if (bodyjson.get("name") != null)
                        name = (String) bodyjson.get("name");
            }

            JSONObject responseBody = new JSONObject();
            if (name != null)
                responseBody.put("message","Hello " + name + "!");
            else
                responseBody.put("message", "No name defined. Add {\"name\":\"name\"} in the request.");

           JSONObject headerJson = new JSONObject();
           headerJson.put("x-custom-header", "my custom header value");
           responseJson.put("statusCode", 200);
           responseJson.put("headers", headerJson);
           responseJson.put("body", responseBody.toString());
```

```
            OutputStreamWriter writer = new OutputStreamWriter(outputStream, "UTF-8");
            writer.write(responseJson.toString());
            writer.close();
        }
        catch(Exception exc)
        {
                logger.log("Error : " + exc.toString());
        }
  }
}
```

A.7. Verify that your **pom.xml** file contains the following configuration:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>examples</groupId>
    <artifactId>lambda-java-example</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>lambda-java-example</name>

    <properties>
            <maven.compiler.source>11</maven.compiler.source>
            <maven.compiler.target>11</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-lambda-java-core</artifactId>
            <version>1.2.2</version>
        </dependency>
       <dependency>
            <groupId>com.googlecode.json-simple</groupId>
            <artifactId>json-simple</artifactId>
            <version>1.1.1</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>3.4.1</version>
            </plugin>
        </plugins>
    </build>
</project>
```

A.8. Configure **JDK11** to be available on your project root command line with Apache Maven 3.9.9.

A.9. Execute the packaging procedure with

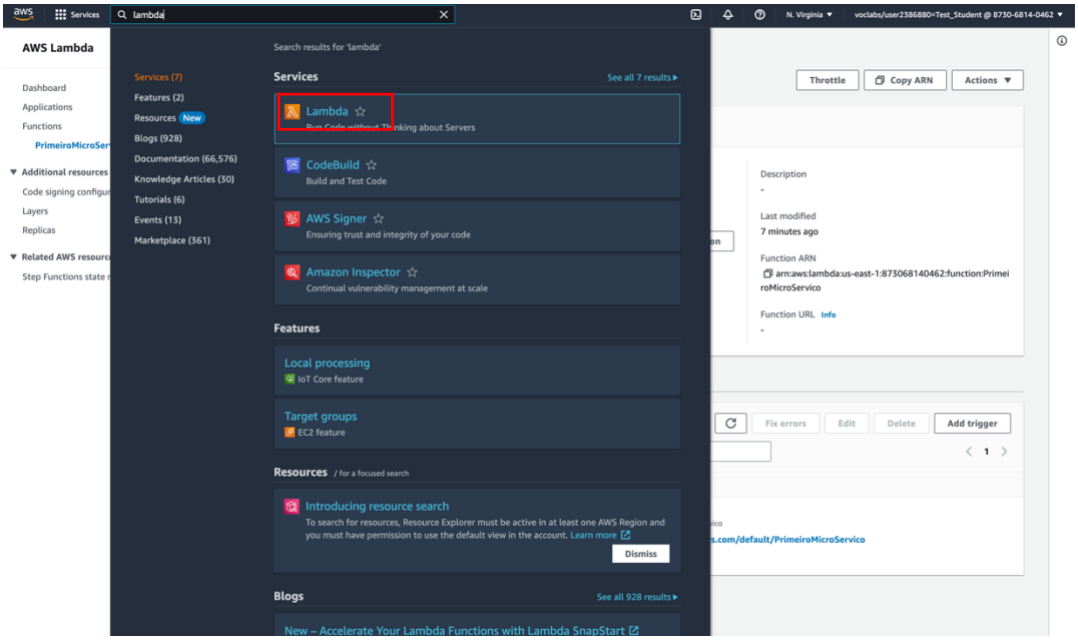
```
mvn package shade:shade
```

A.10. You will the following file in the project target folder:

```
lambda-java-example-0.0.1-SNAPSHOT.jar.
```

A.11. **Login** to AWS Educate, then choose the classroom, and press AWS console. Expand "All services" option.

A.12. Search for the AWS Lambda Service(under Compute option)



A.13. Press Create function

A.14. Choose "Author from scratch"

A.15. Define Function name as: PrimeiroMicroServico

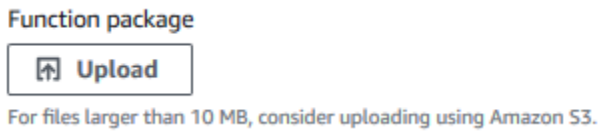
A.16. Choose JAVA 11 for Runtime

A.17. Choose x86_64 Architecture

A.18. Change Default execution role > choose Use an existing role> choose LabRole

A.19. Press Create function

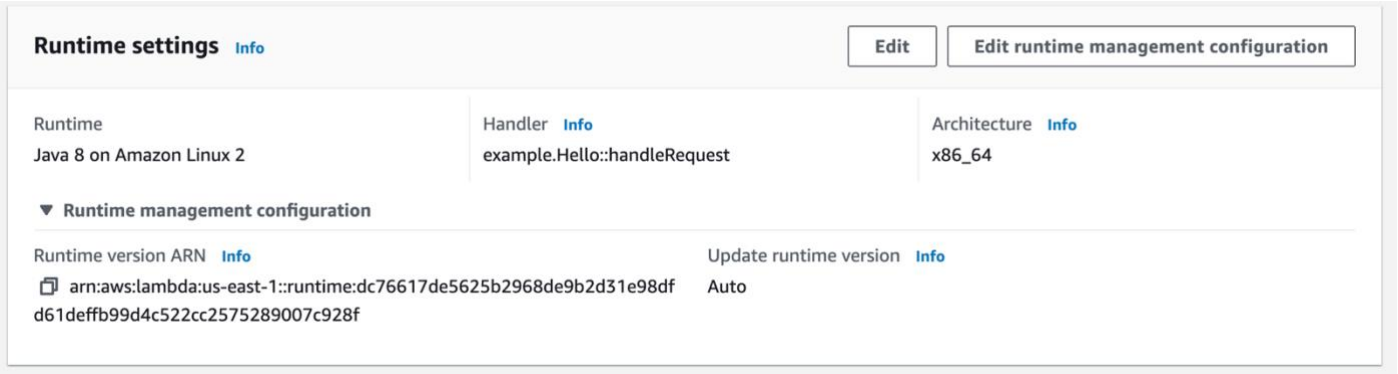
A.20. Then, in the next interface, press the Upload button:



and choose the previously created jar file:

```
lambda-java-example-0.0.1-SNAPSHOT.jar.
```

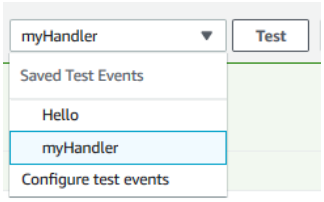
A.21. Check the Handler field with the name of the handler defined within your java class. You will obtain something similar with this:



A.22. Your lambda service is deployed!

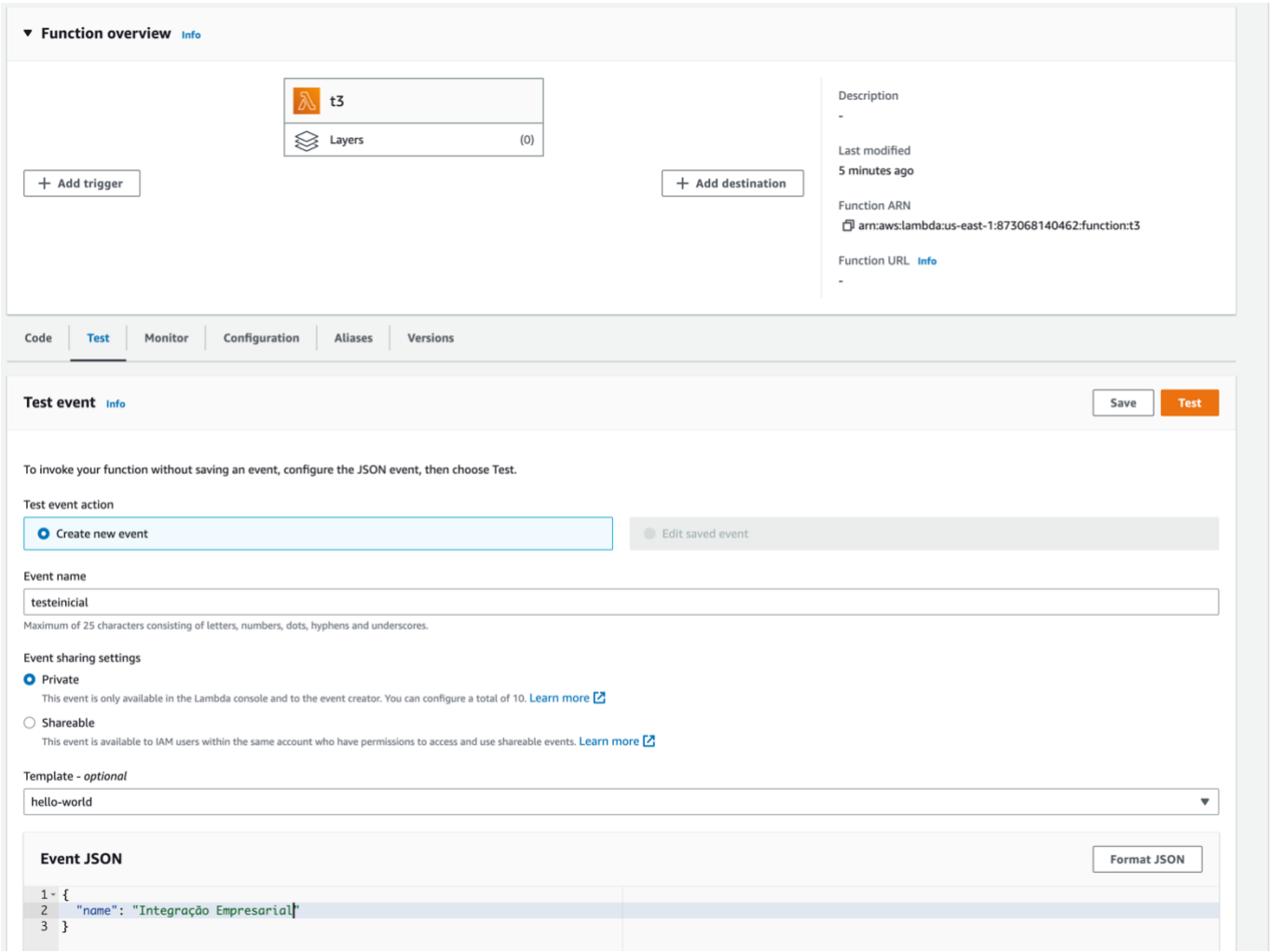
B. Testing a Lambda function internally

C.1. Choose option "Test" and select create new event



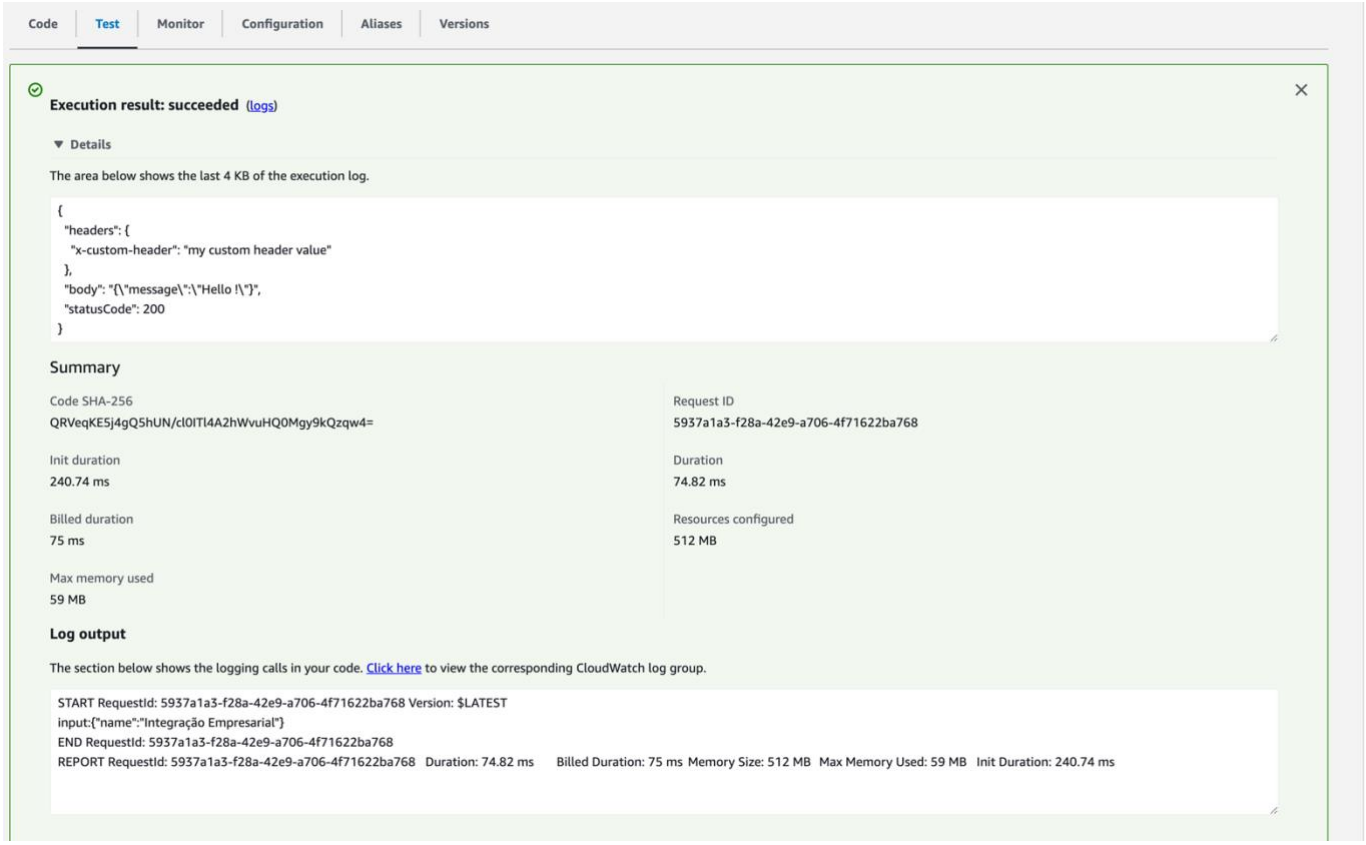
C.2. Define a name for "Event name" field

C.3. Define the JSON message accordingly with the test example, with the example of section A, define the message as follows:



C.4. Press Test button

C.5. You should receive a response similar with:


Code
Test
Monitor
Configuration
Aliases
Versions
Execution result: succeeded (logs)
Details
The area below shows the last 4 KB of the execution log.
{
"headers": {
"x-custom-header": "my custom header value"
},
"body": "{\"message\":\"Hello !\"}",
"statusCode": 200
}
Summary
Code SHA-256
QRVeqKE5j4gQ5hUN/cl0lTl4A2hWvuHQ0Mgy9kQzqw4=
Request ID
5937a1a3-f28a-42e9-a706-4f71622ba768
Init duration
240.74 ms
Duration
74.82 ms
Billed duration
75 ms
Resources configured
512 MB
Max memory used
59 MB
Log output
The section below shows the logging calls in your code. Click here to view the corresponding CloudWatch log group.
START RequestId: 5937a1a3-f28a-42e9-a706-4f71622ba768 Version: $LATEST
input:{"name":"Integração Empresarial"}
END RequestId: 5937a1a3-f28a-42e9-a706-4f71622ba768
REPORT RequestId: 5937a1a3-f28a-42e9-a706-4f71622ba768 Duration: 74.82 ms Billed Duration: 75 ms Memory Size: 512 MB Max Memory Used: 59 MB Init Duration: 240.74 ms

C. Invoking a Lambda function remotely

To allow a Lambda function to be invoked remotely the API Gateway need to be assigned to the trigger of the function

D.1. Access your Lambda function in AWS console as described in the previous sections

D.2. Press the Add trigger button: 

D.3. In the trigger confirmation drop-down menu choose "API Gateway"

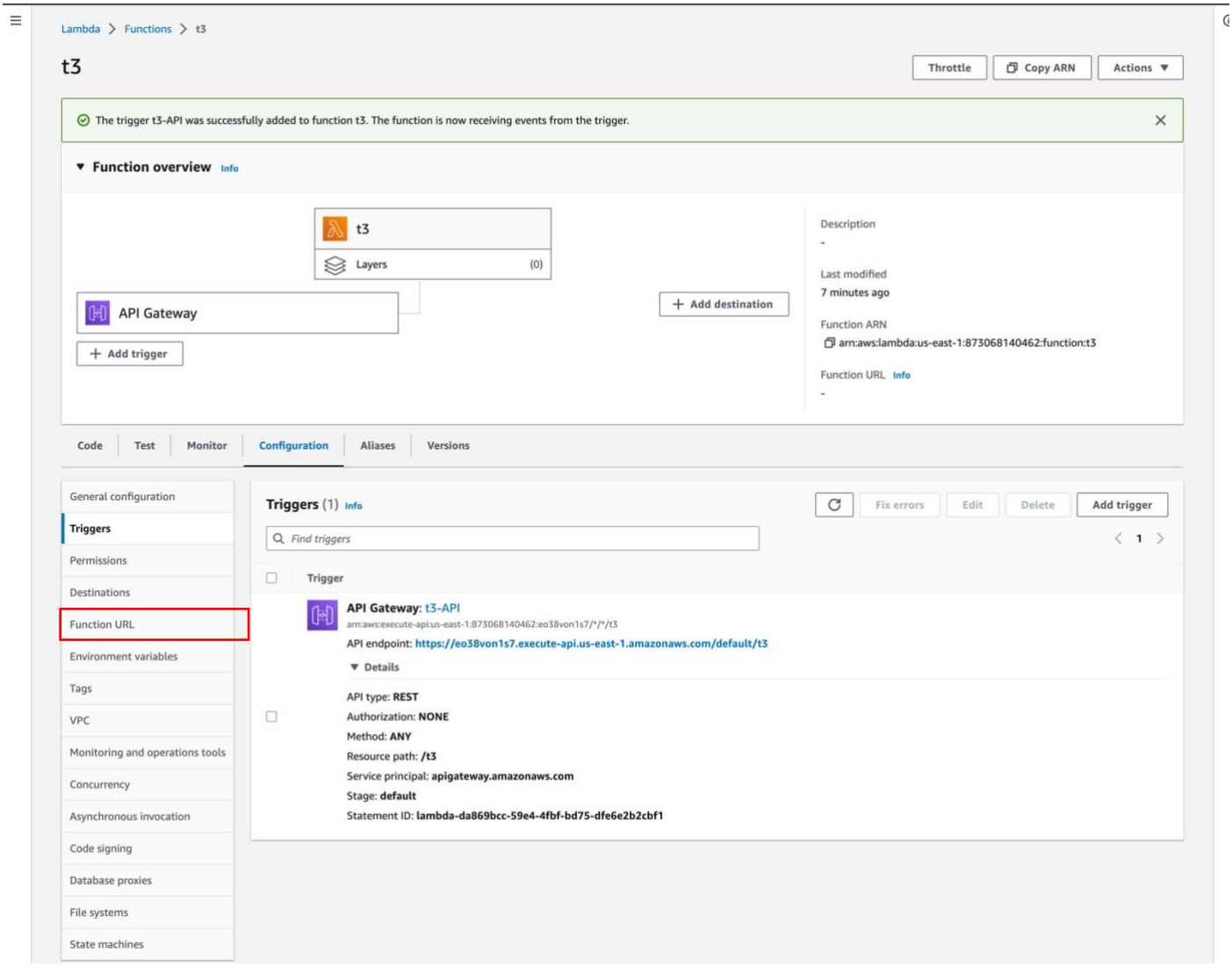
D.4. Then, in the API drop-down menu choose "Create a new API"

D.5. Choose REST API option

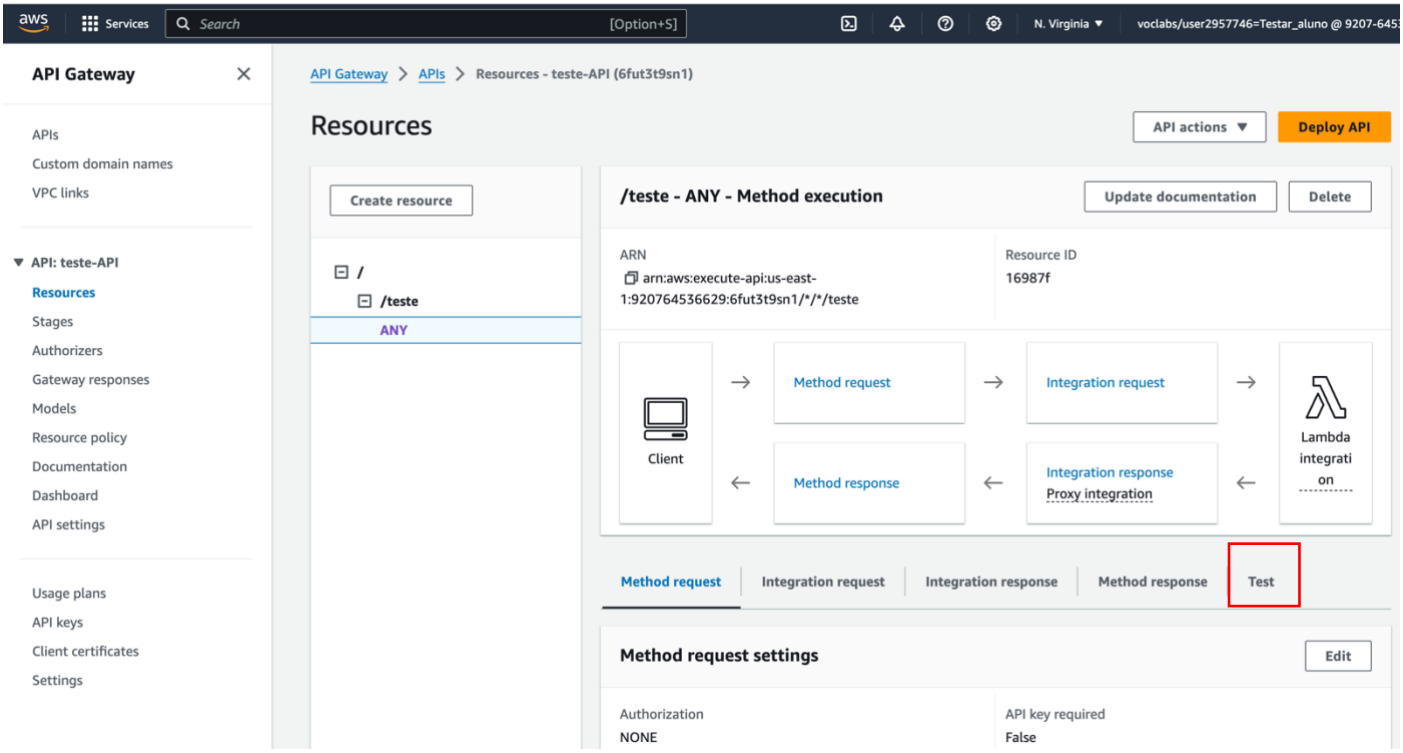
D.6. Choose the following option from Security drop down box: Open

D.7. Press Add button 

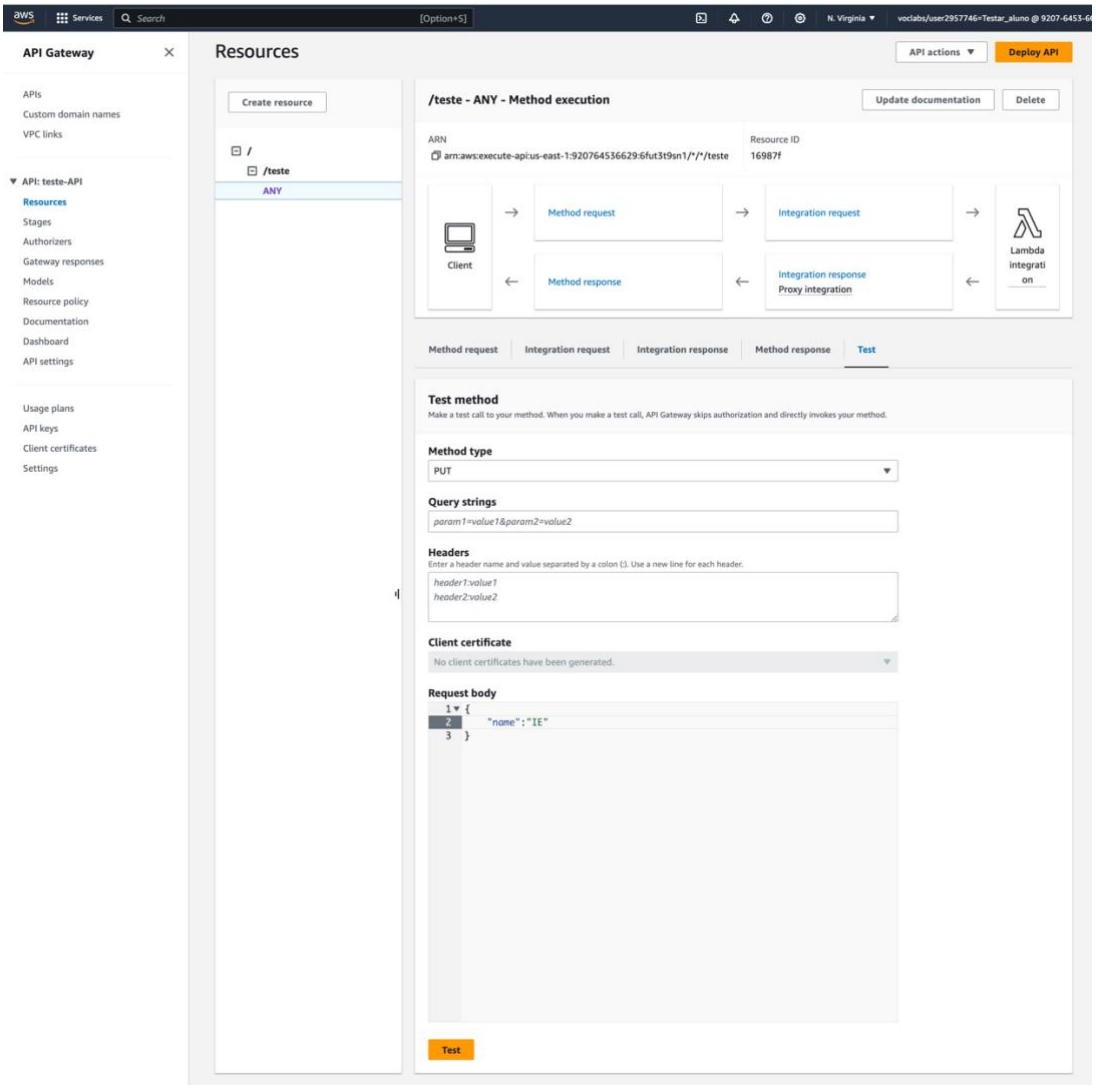
D.8. You will obtain something similar with the following image:



Click in the "t3-API" link, or similar, as identified in the red box of the image, to select the AWS API Gateway service of your lambdaservice. A similar interface is obtained:



D.9. Select the TEST option as identified in the red box of the previous image and obtain the following:



D.10. Alternatively, create the file body.json with the following contents:

```
{
  "name": "Integracao Empresarial"
}
```

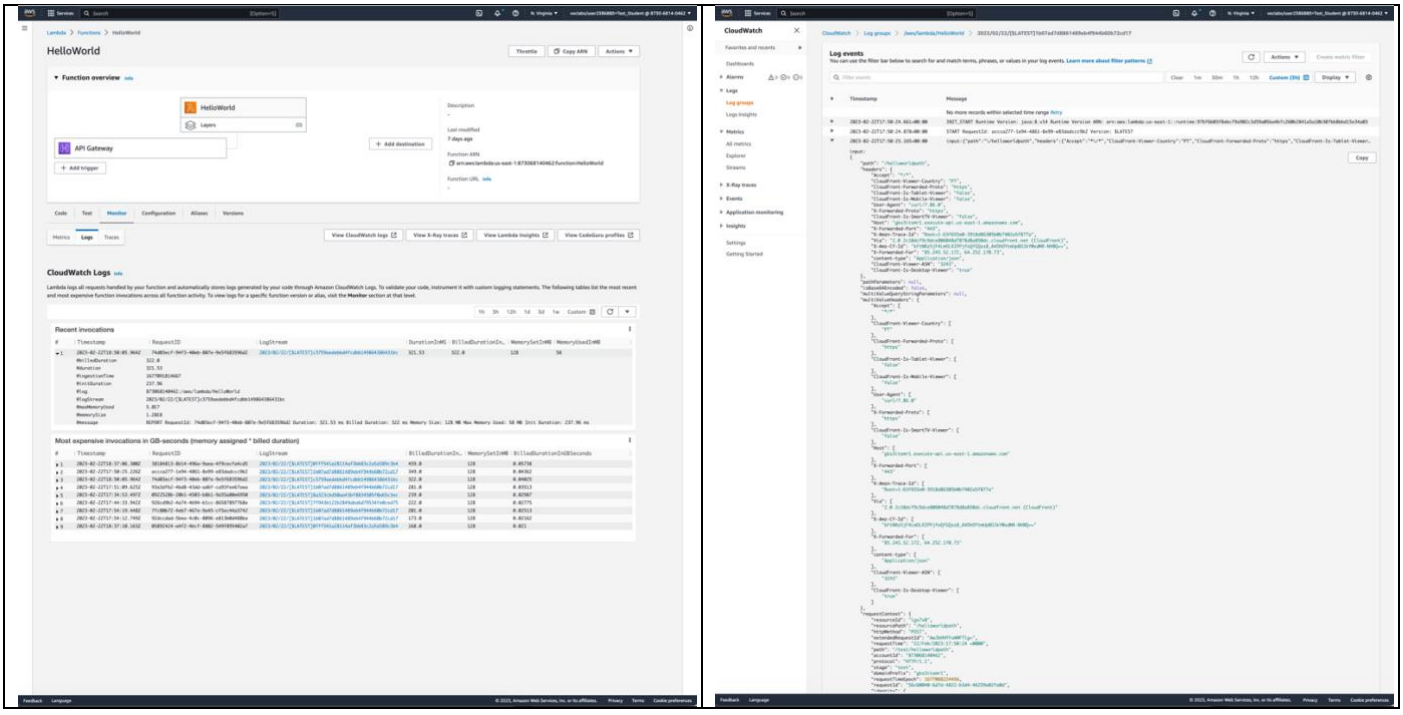
And execute the following command in another ssh or putty session (in linux like environment) where the is available as in step C.8. (e.g.: https://uwajykvb2j.execute-api.us-east-1.amazonaws.com/default/PrimeiroMicroServico):

```
curl -i -H "Content-Type: Application/json" --data "@body.json" -X POST <LAMBDA_URL>
```

It is expected that you receive something similar with:

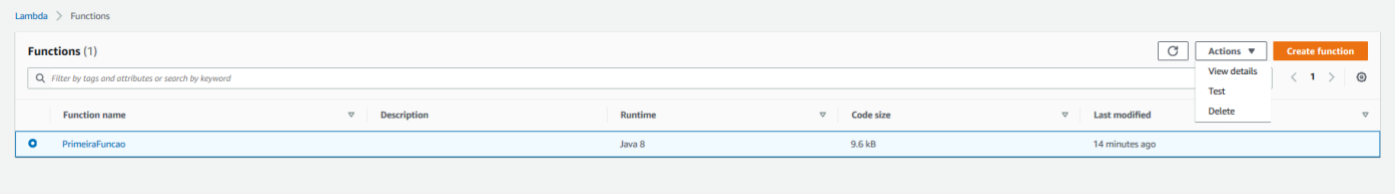
```
$ curl -i -H "Content-Type: Application/json" --data "@body.json" -X POST https://2p3fp5acxj.execute-api.us-
east-1.amazonaws.com/default/PrimeiroMicroServico

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    79  100    43  100    36     43     36  0:00:01 --:--:--  0:00:01   117
HTTP/1.1 200 OK
Date: Tue, 07 Jan 2020 22:26:07 GMT
Content-Type: application/json
Content-Length: 43
Connection: keep-alive
x-amzn-RequestId: a072be98-cefe-4847-8558-2360b6106e30
x-amz-apigw-id: F83L8FjKIAMFzrQ=
x-custom-header: my custom header value
X-Amzn-Trace-Id: Root=1-5e15057f-3a08bd2b9dd68b3d22208a89;Sampled=0

{"message":"Hello Integracao Empresarial!"}
```

The trace of the request can be verified in the "Monitor" tab of the Lambda function. Choosing the desired LogStream (left part of the following figure) will bring you to the right part of the figure with all the details of the invocation (inside AWS CloudWatch).



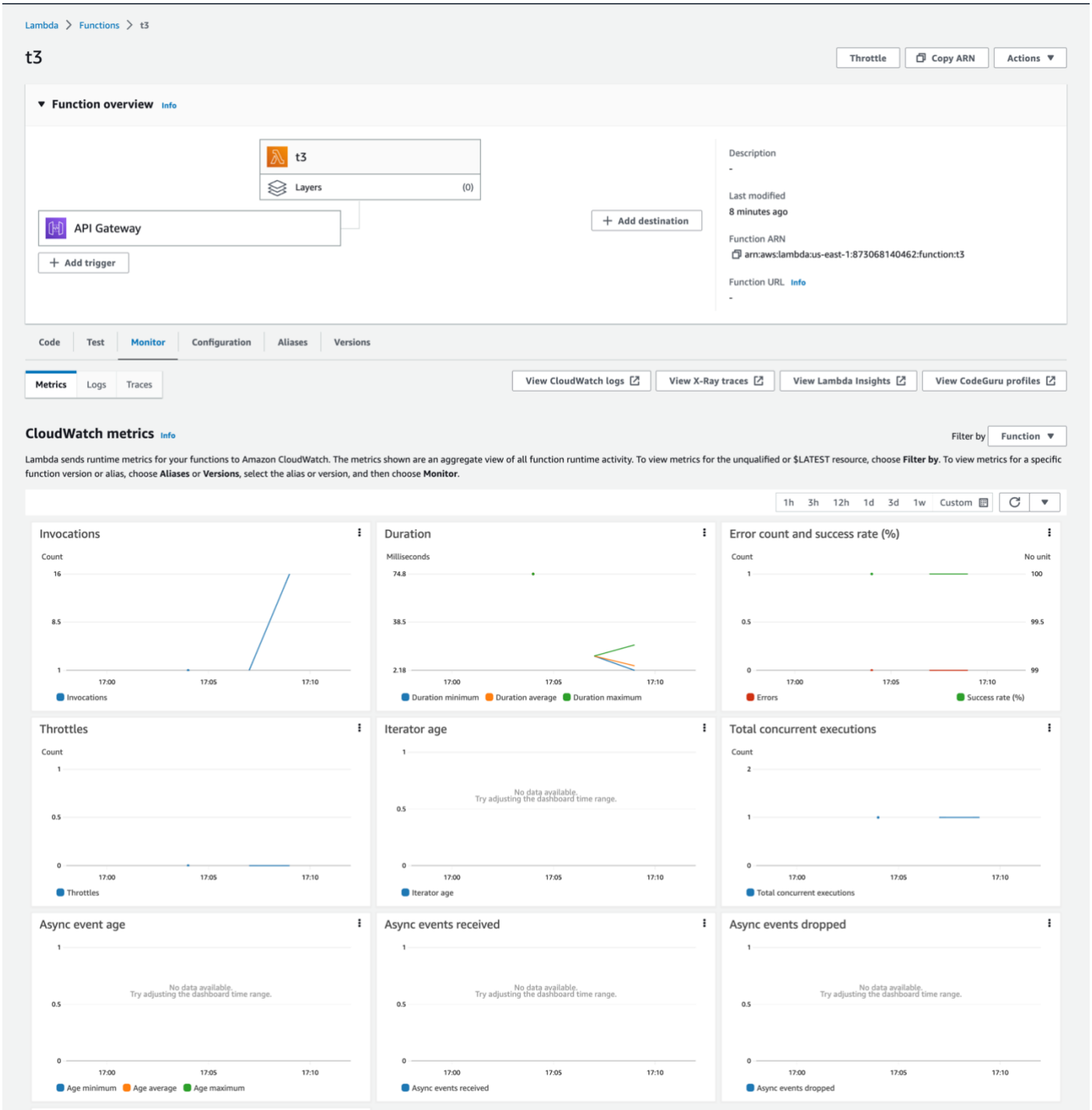
D. Delete a Lambda function

E.1. Choose the Lambda > Functions options, then choose the function to delete and then chose Delete from Action drop-down menu.



E. Checking consumption of a Lambda function

F.1. Choose the Lambda > Dashboard to view all the detail regarding each of the invocations to each of the lambda service. For a specific Function, use the option Monitoring available in all Functions.



Other references

- AWS Lambda Concepts https://docs.aws.amazon.com/lambda/latest/dg/welcome.html, available in PDF.
- Amazon API Gateway Developer Guide https://docs.aws.amazon.com/pt_br/apigateway/latest/developerguide/welcome.html, available in PDF.
- Set up an Integration Response in API Gateway, https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-integration-settings-integration-response.html
- https://www.mkyong.com/java/json-simple-example-read-and-write-json/
- https://www.tutorialspoint.com/json_simple/json_simple_quick_guide.htm
- https://www.baeldung.com/aws-lambda-api-gateway
- https://github.com/eugenp/tutorials/blob/master/aws-lambda/src/main/java/com/baeldung/lambda/apigateway/APIDemoHandler.java
- Amazon Relational Database Service User Guide - https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/rds-ug.pdf#CHAP_GettingStarted