Building Operational Decision Services

with

OpenRules AWS Lambda AWS API Gateway

OpenRules, Inc.

www.openrules.com

September-2019

TABLE OF CONTENTS

INTRODUCTION

Nowadays many enterprises move their decision services to clouds utilizing new serverless architecture. They simply upload their code and serverless takes care of everything required to run and scale this code with high availability. The most popular today's serverless environment is Amazon Web Services (AWS) Lambda – it allows you to deploy and run your decision services without even thinking about servers and to pay only for the execution time your services consume.

AWS Lambda and AWS API Gateway have made creating serverless APIs extremely easy. You can simply upload your decision service to AWS Lambda, configure an API Gateway, and start responding to RESTful endpoint calls. However, the way how you build and deploy your Lambda Decision Service is usually not as straightforward as it seems on the surface. OpenRules Decision Manager simplifies this process as much as possible avoiding any coding.

This tutorial provides a sampling with all details of how to build AWS Lambda Decision Services without assuming any preliminary knowledge of the AWS environment.

WHAT SHOULD BE PRE-INSTALLED

We assume that you've already installed:

- Java 1.8 or later
- MS Excel
- OpenRules Decision Manager evaluation or production version 8.0.1 or later by downloading the workspace "openrules.dm.models".

Instead of MS Excel you may use or Google Sheets. We also assume that you already created an AWS account.

What You'll Do

We will explain what you need to do to create, test, and deploy an OpenRules business decision model as AWS Lambda Function, and then execute it from a Java-based decision-making application.

Following step-by-step instructions below, you do the following:

1. Select the standard OpenRules business decision model from the workspace "openrules.dm.models", build and run it locally
2. Upload it to AWS Lambda
3. Create a new AWS API that will provide you with an invocation URL
4. Use this URL to test the Lambda service from POSTMAN or a Java client.

In the end, you will be ready to create, deploy, and execute your own AWS Lambda decision services.

Defining Decision Model

The standard OpenRules Decision Manager installation workspace "openrules.dm.models" comes with a set of sample decision models such as "VacationDays" or "PatientTherapy" – click on these links to see their detailed descriptions. Similarly, you may create and test your own decision model inside this workspace using only MS Excel or Google Sheets.

We will start our explanations with the decision model "VacationDays" that calculates the number of vacation days given to an employee based on the age and years of service in accordance with the following business rules:

The number of vacation days depends on age and years of service.

Every employee receives at least 22 days.
Additional days are provided according to the following criteria:

1) Only employees younger than 18 or at least 60 years, or employees with at least 30 years of service will receive 5 extra days.

2) Employees with at least 30 years of service and also employees of age 60 or more, receive 3 extra days, on top of possible additional days already given.

3) If an employee has at least 15 but less than 30 years of service, 2 extra days are given. These 2 days are also provided for employees of age 45 or more. These 2 extra days can not be combined with the 5 extra days.

This decision model has been implemented a business analyst using a set of xls-files and built locally by a click on the standard file "build.bat". Then it was tested by a click on the standard file "run.bat" using the test-cases also defined in the Excel file "Test.xls" by the same business person. All parameters of this decision model were provided in the file "settings.bat":

```
set MODEL_NAME=DecisionModelVacationDays
set GOAL_NAME="Vacation Days"
set MODEL_FILE=rules/DecisionModel.xls
set TEST_FILE=rules/Test.xls
set DEBUG=On
set PACKAGE_NAME=vacation.days
```

This model deals only with one business concept Employee defined in the table "Glossary":

Glossary glossary		
Variable Name	**Business Concept**	**Attribute**
Vacation Days	Employee	vacationDays
Eligible for Extra 5 Days		eligibleForExtra5Days
Eligible for Extra 3 Days		eligibleForExtra3Days
Eligible for Extra 2 Days		eligibleForExtra2Days
Age in Years		age
Years of Service		service

Internally this business decision model was transformed into Java classes placed in the package "generated/vacation.days" and zipped in one small file "dist/DecisionModelVacationDays.jar".

To prepare our model for AWS Lambda deployment, we need to add only one more setting to the file "settings.bat":

```
set LAMBDA=On
```

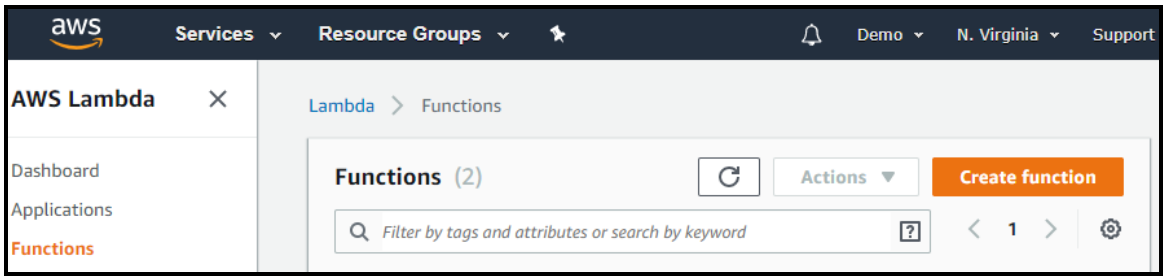
It will prompt “build.bat” to generate several Java classes used during the deployment and Lambda function invocation. For this decision model, the build will generate the following classes in the package “generated/vacation.days”:

- DecisionModelVacationDaysHandler.java – an interface for AWS Lambda
- DecisionModelVacationDaysRequest.java – an input interface for AWS Lambda
- DecisionModelVacationDaysResponse.java – an output interface for AWS Lambda
- DecisionModelVacationDaysClient.java – a client for Lambda function invocation.

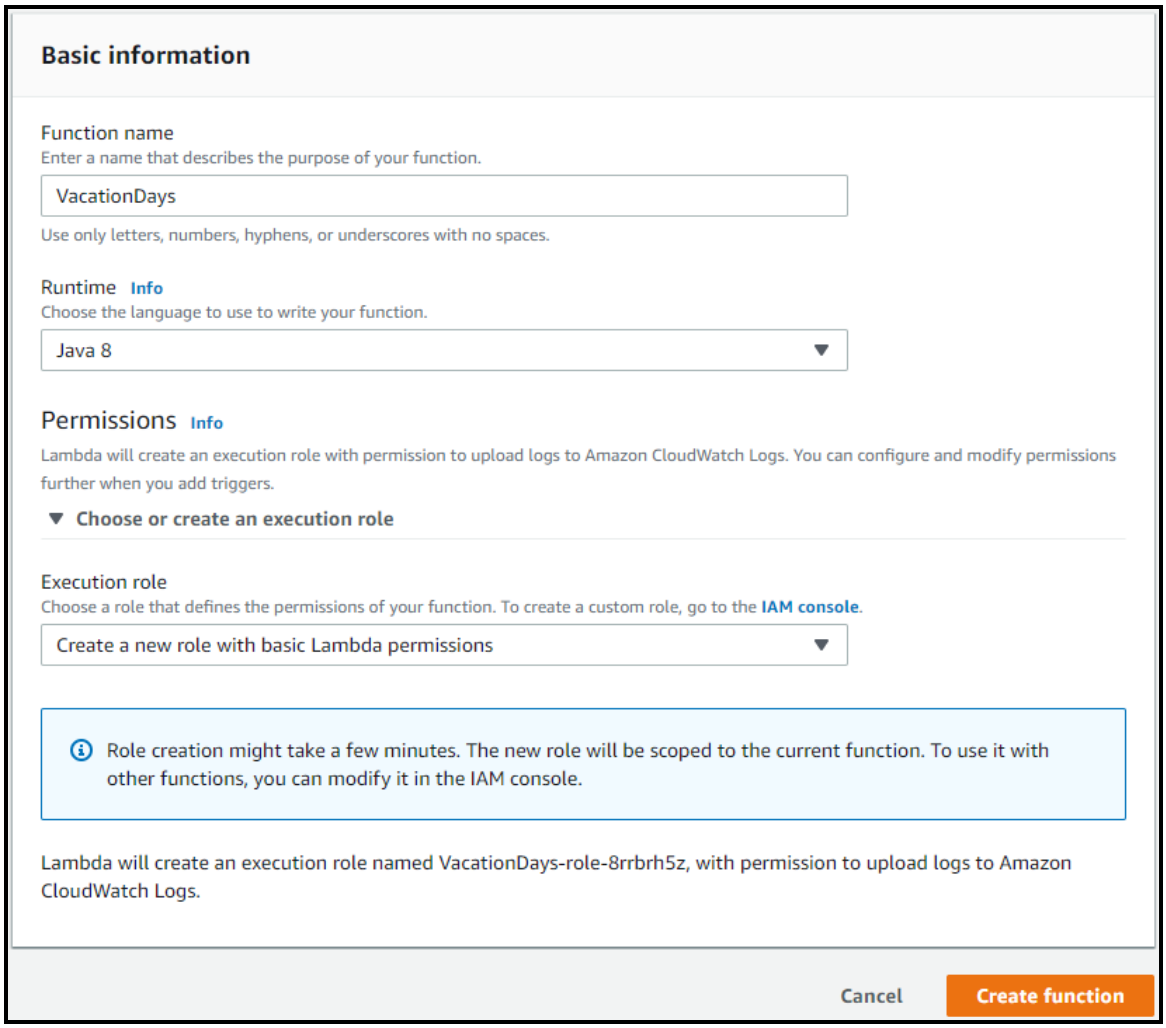
It will also generate file “dist/DecisionModelVacationDays.lambda.jar”. It’s important to note that all these actions are hidden from a business user (the author of the decision model) who simply clicks on “build.bat”.

Uploading Decision Model to AWS Lambda

Let’s assume that you signed in to the AWS Lambda console:



Click on “**Create function**”. The next screen will look as follows:



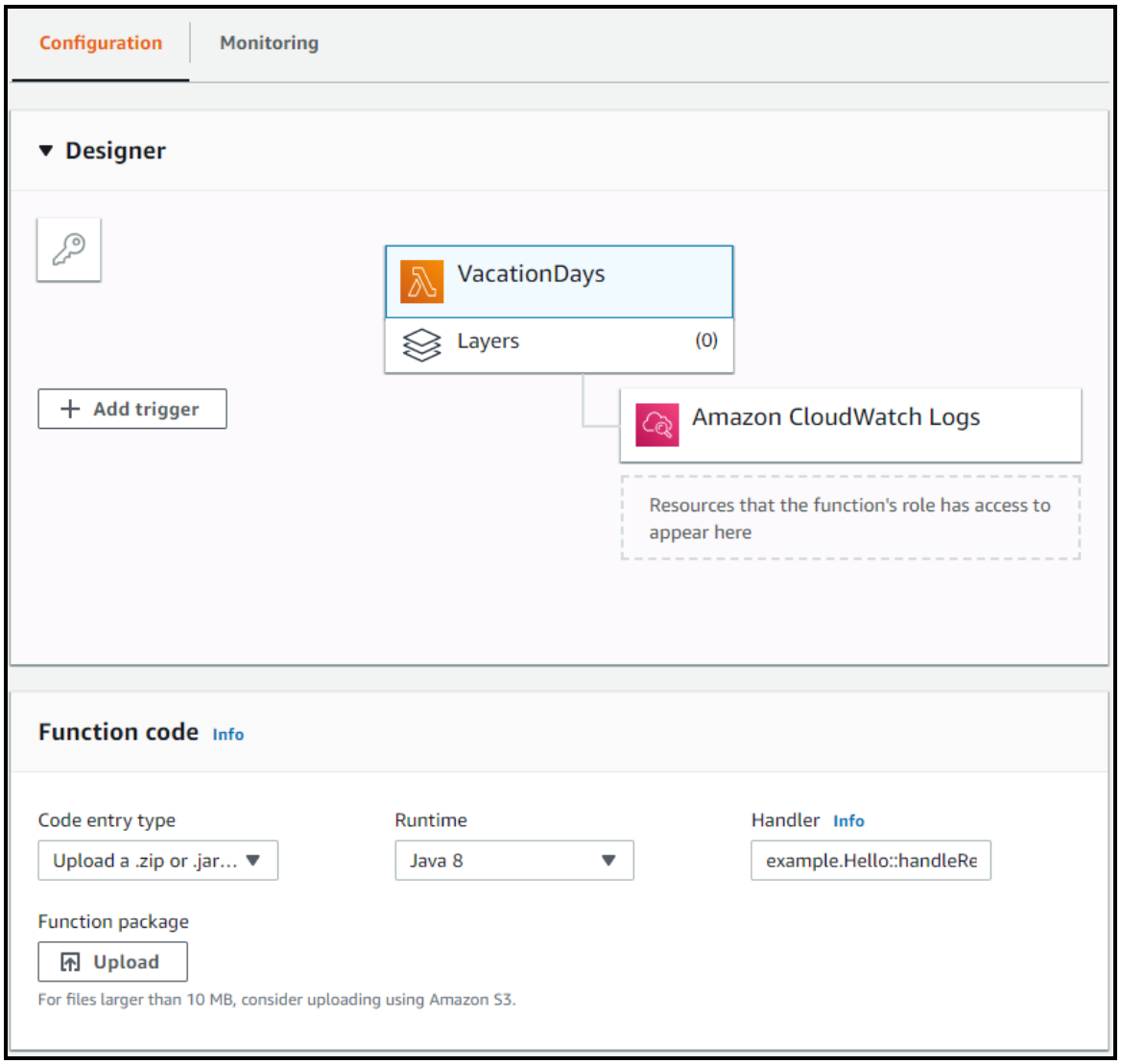
Enter Function name "VacationDays".

Select "Java 8" from the combo-box "**Runtime**".

Click on a little triangle on the left of "**Choose or create an execution role**" and select "Create a new role with basic Lambda permissions".

Click on "**Create function**".

It will display the following screen:



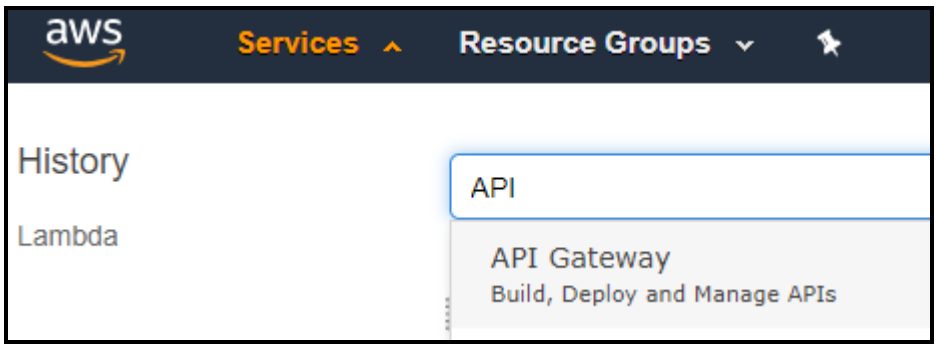
Click on the button "**Upload**". It will open File Explorer allowing you to choose the already generated file "**DecisionModelVacationDays.lambda.zip**" from the folder "openrules.dm.models/VacationDays/dist/".

In the box "**Handler**" replace "example.Hello::handleRequest" with "**vacation.days.DecisionModelVacationDaysHandler::handleRequest**". Make sure that you use two colons.

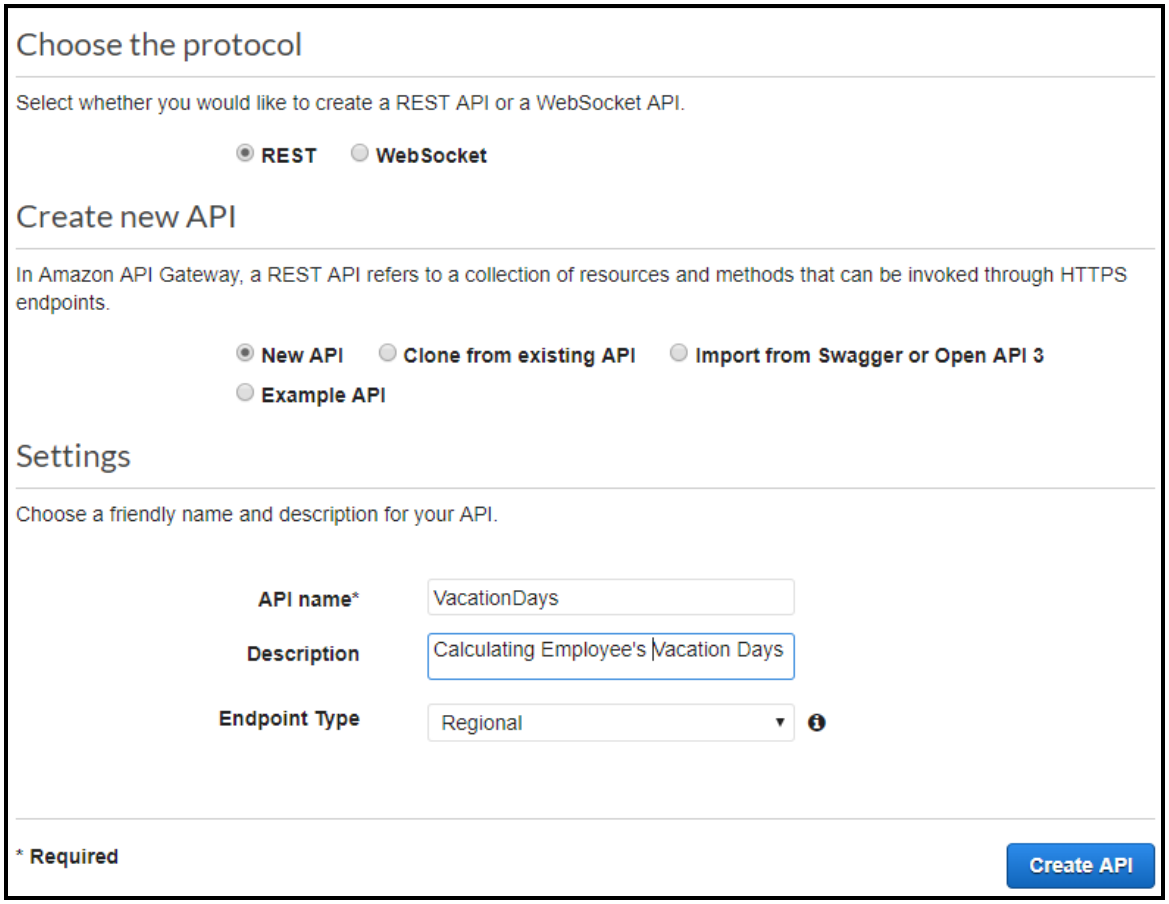
Click on the button "**Save**" in the top right corner. Our AWS Lambda Function "VacationDays" is ready. Now we need to create an API for its invocation.

CREATING API FOR LAMBDA INVOCATION

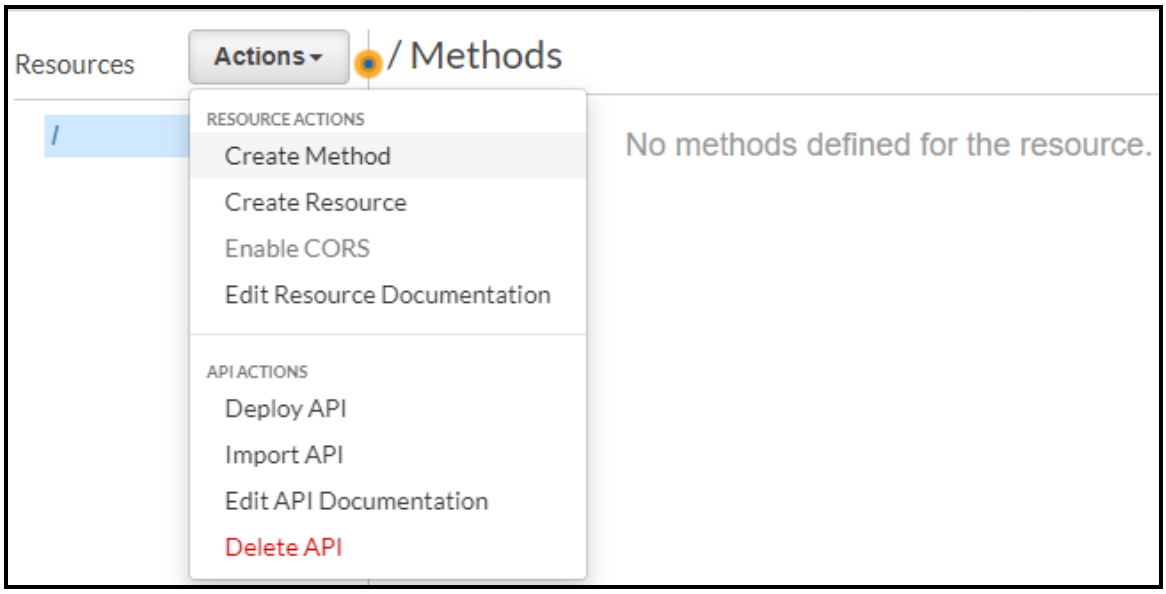
Let's switch to AWS API Gateway. Click on the menu item "Services" and find "API Gateway" by starting to type "API" in the search box:



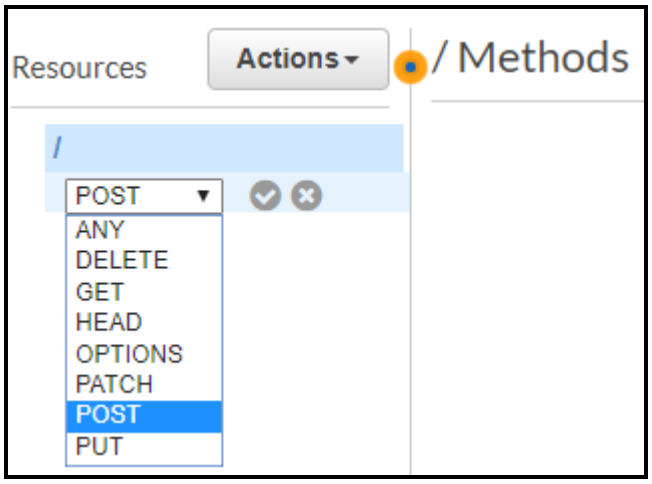
On the next screen click on  and it will display:



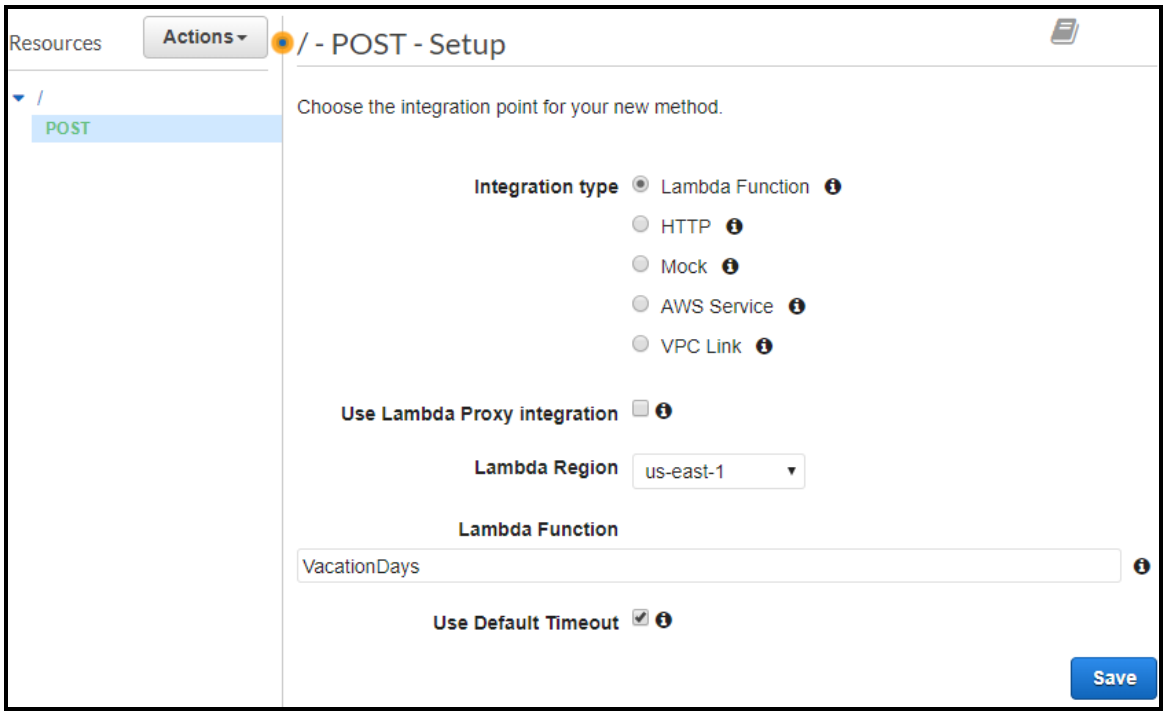
Make sure that “REST” and “New API” are selected (the defaults) and enter the API name and Description as on the above screen. Click on the button “**Create API**”. On the next screen



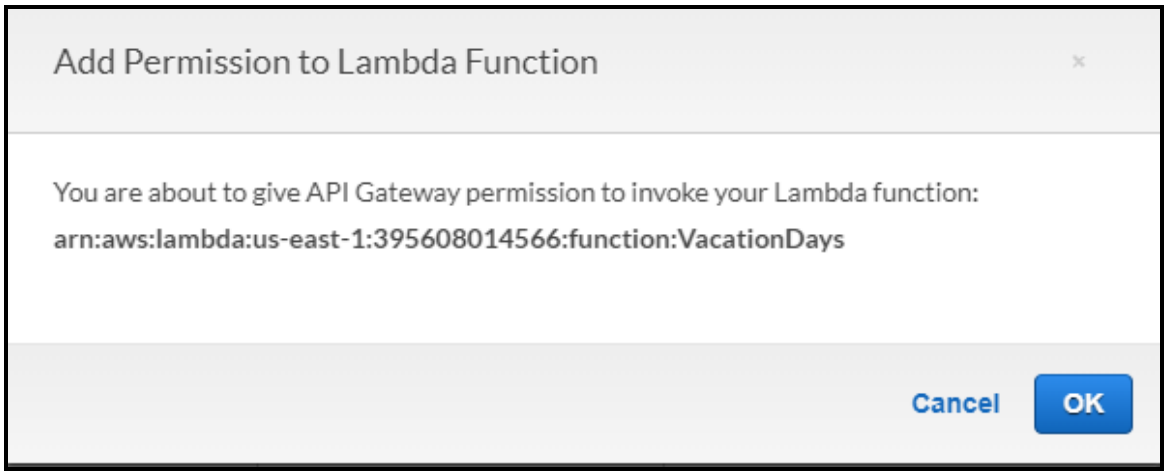
select “**Create Method**” from the combo-box “**Actions**”. Then choose the method “POST” and click on  from this view:



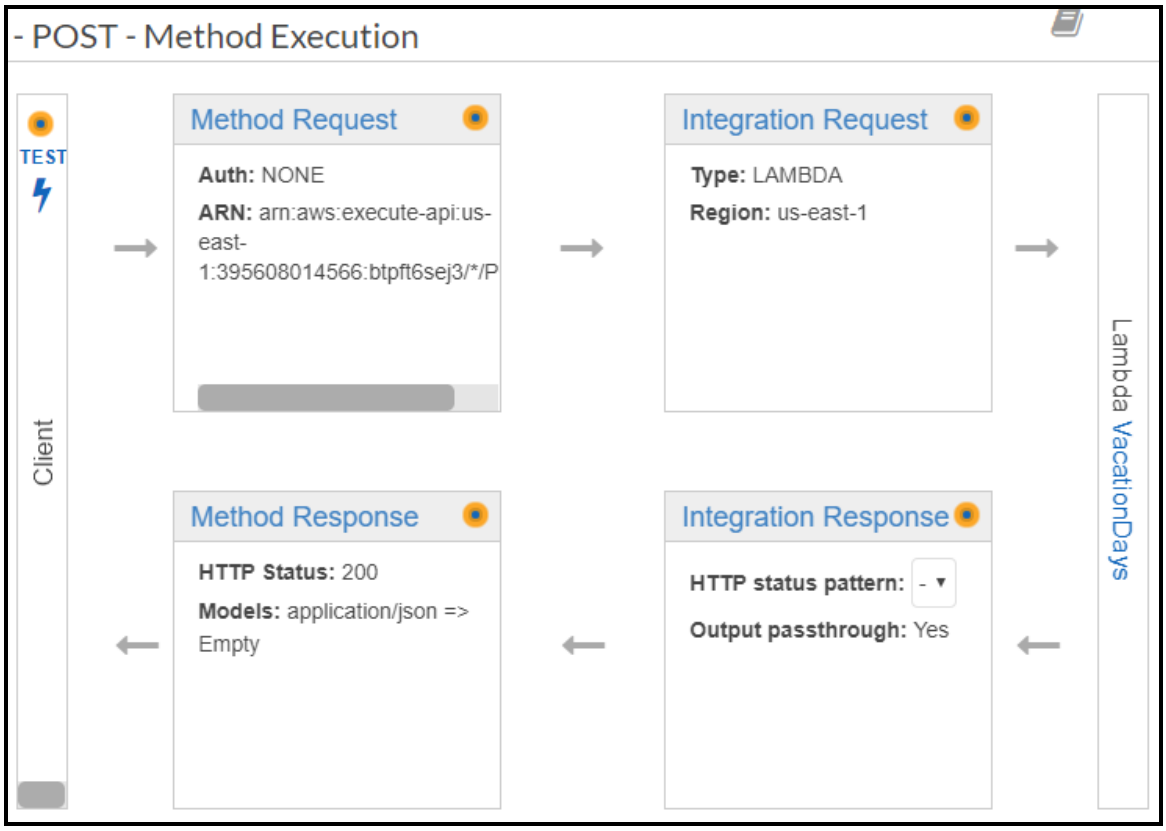
It will display the following “POST – Setup”:



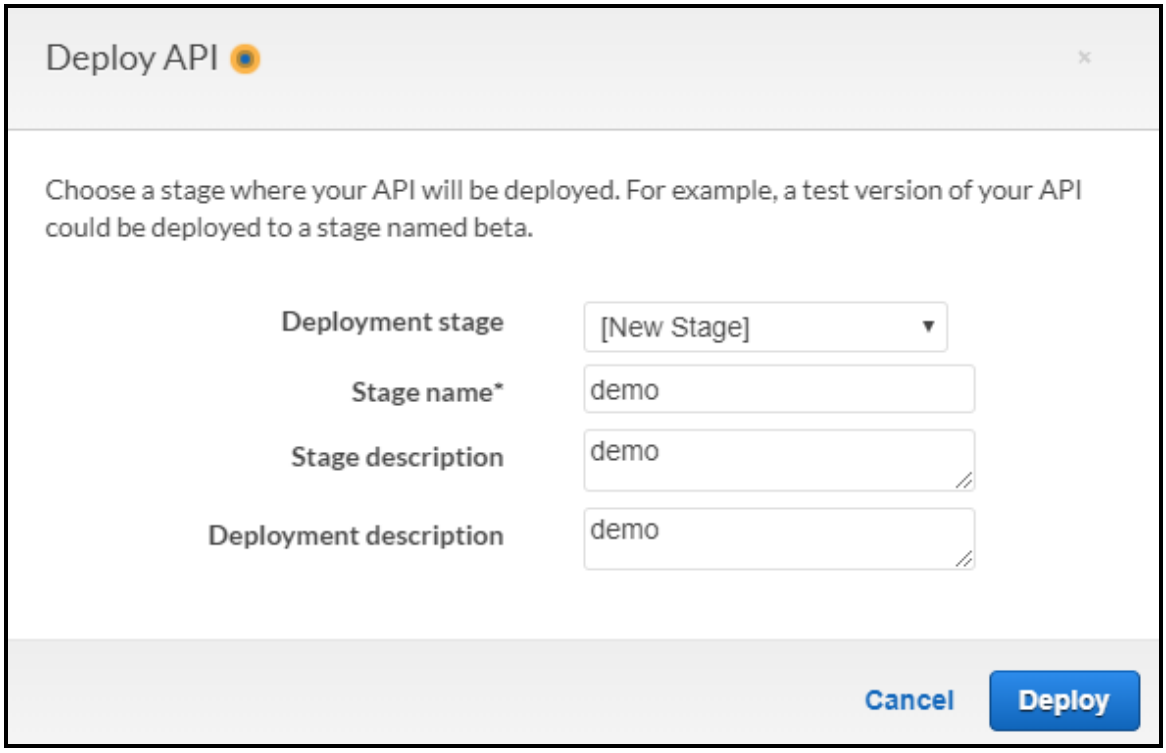
Click on the input box below “Lambda Function” and choose VacationDays. Click on the button “Save”. It will display a box similar to this one:



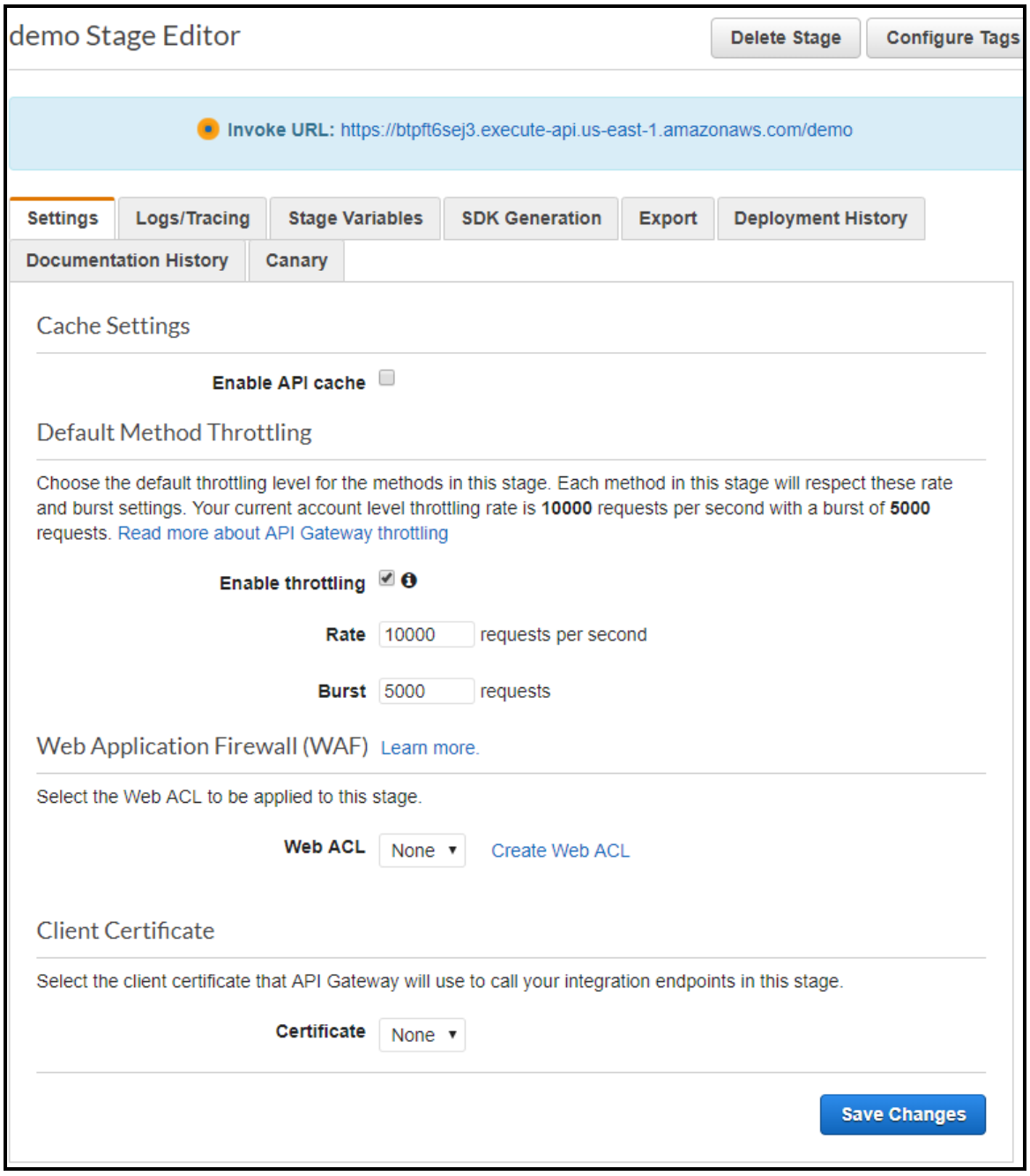
Click “OK”. It will display this view:



Now go back to the combo-box "**Actions**" and select "**Deploy API**". It will display this dialog:



Select [New Stage] as a **Deployment stage** and enter a description similar to the above ones. Click on the button "**Deploy**". You will see the following screen:



You may modify different deployment options later on, but by now we already have our **Invoke URL** specified in the blue box on the top. Right-click on this URL and select "**Copy link address**". Click on the button "Save Changes". Out Lambda Function is successfully deployed and we know its invocation URL. We may test this lambda function.

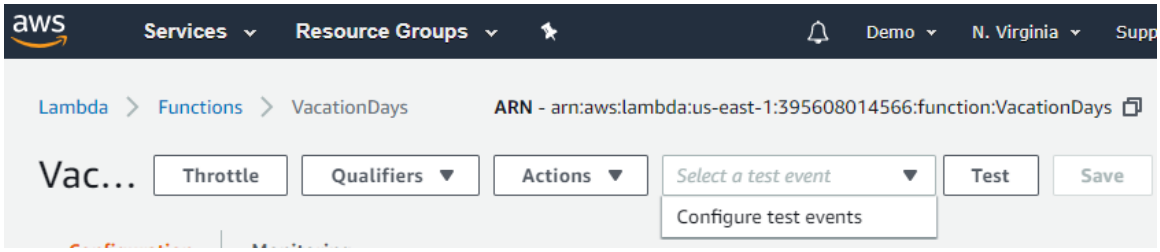
TESTING LAMBDA DECISION SERVICE

You may test our Lambda Decision service using different ways, but here we will describe 3 testing facilities:

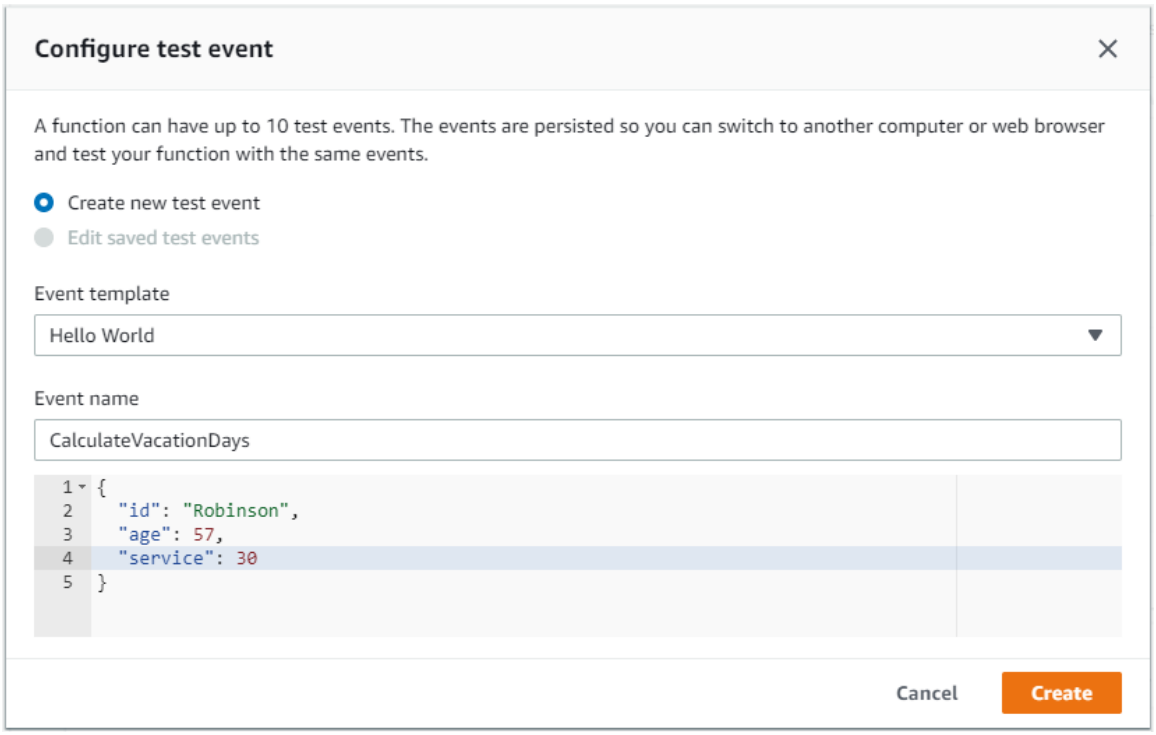
- Using Lambda Test
- Using POSTMAN
- Using Java Client.

Testing using Lambda Test

First, we will test out AWS Lambda function "VacationDays" using Lambda itself. Select Lambda function "VacationDays" and from the screen



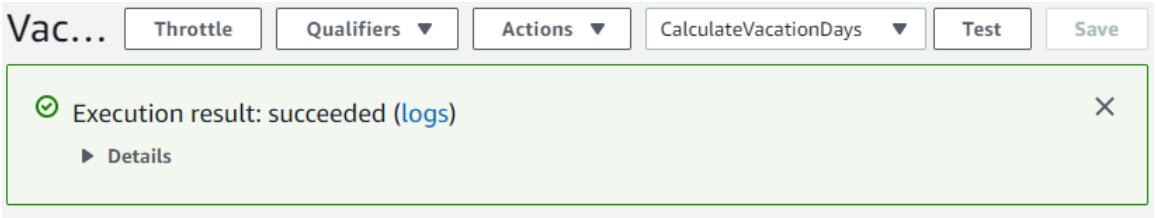
and click on the combo-box "Select a test event" choose "Configure test events". It will open the following dialog:



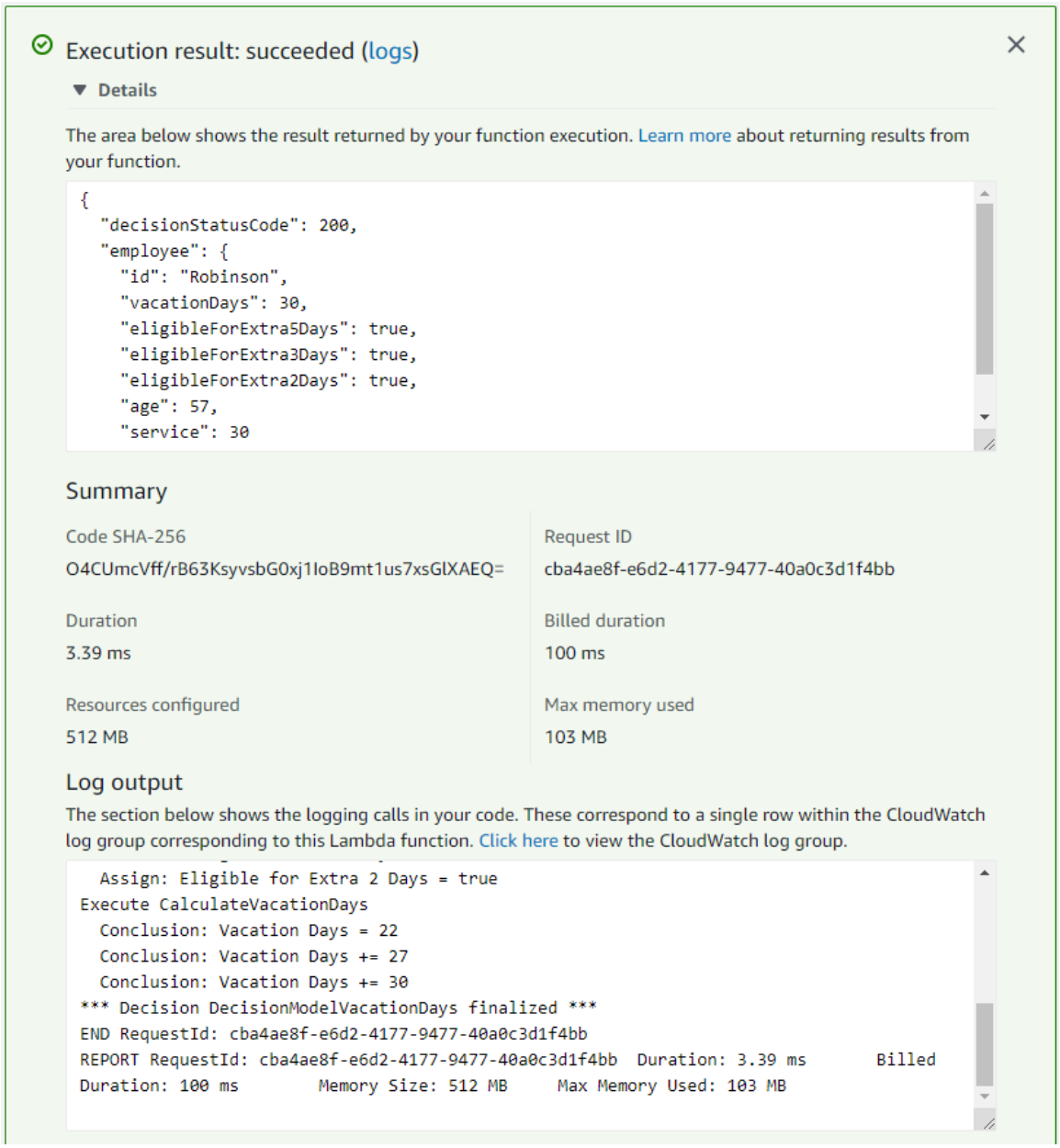
Select “Create new test event”, enter Event Name such as “CalculateVacationDays”, and type test data

```
{
    “id”: “Robinson”,
    “age”: 57,
    “service”: 30
}
```

Click on the button “**Create**”. After the dialog is closed, click on the button “**Test**”. It will display

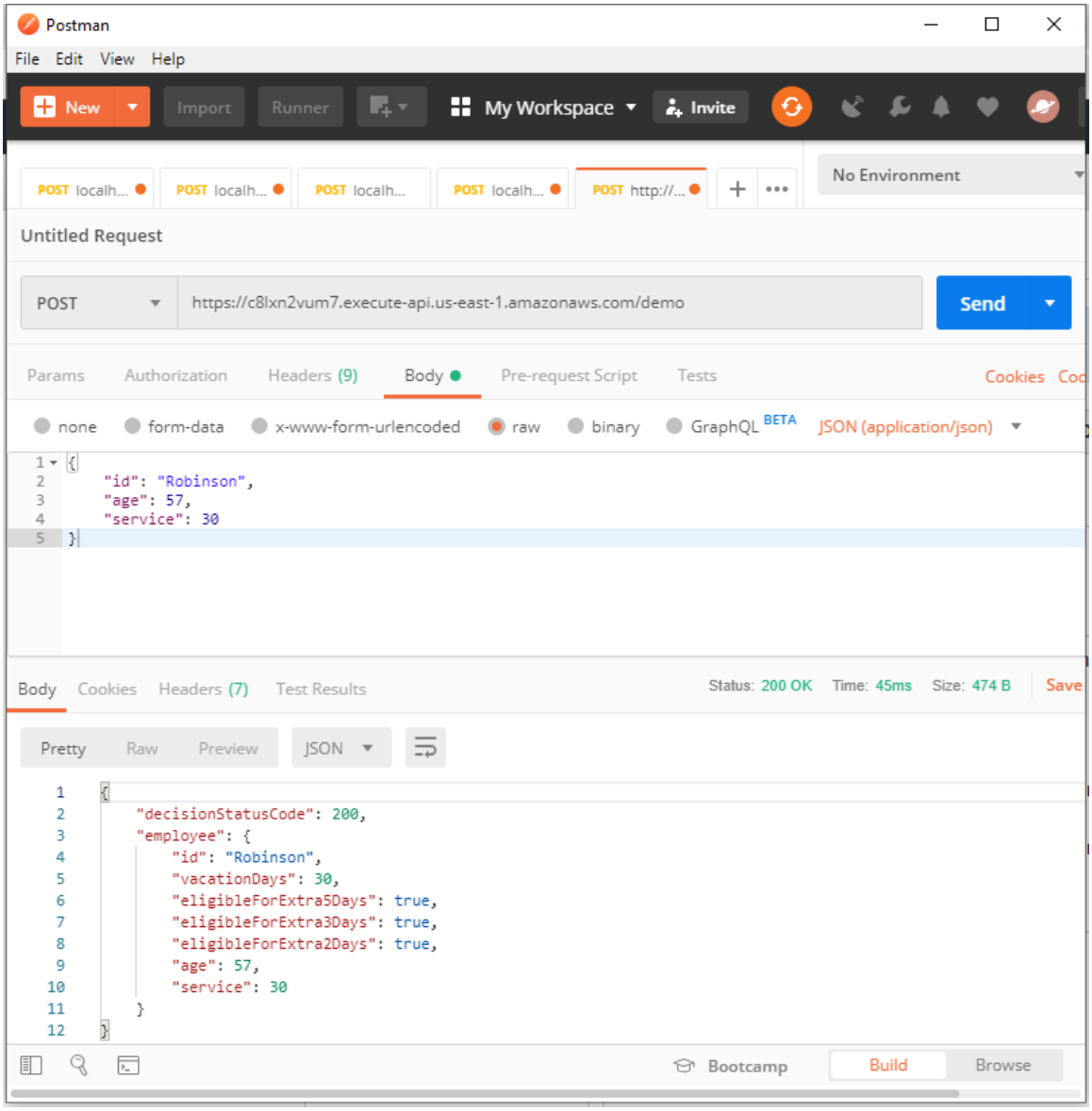


Click on “**Details**” and you will see the execution results for this test:



Testing using POSTMAN

Now, we will show how to test our AWS Lambda function “VacationDays” using POSTMAN, a popular tool that can be downloaded for free from https://www.getpostman.com/. After installation and start, you may fill out this POSTMAN’s form:



In this form, select the method “POST” (from the drop-down list), and paste our copied Invoke URL to the box next to the “POST”. Select “Body” and enter a simple JSON structure for a test-employee:

```
{
    “id”: “Robinson”,
    “age”: 57,
    “service”: 30
}
```

After a click on “Send”, the POSTMAN will send the proper HTTP request to our Lambda function, that will execute our decision service “VacationDays”, and will return back the Employee that includes the calculated number of vacation days “30” and other calculated goals.

You may enter different combinations of “age” and “service” to make sure that our OpenRules-based Lambda function works as expected.

Testing using a Java Client

Now we may test our running service from any Java program similar to what we did with POSTMAN. OpenRules Decision Manager has already generated for a Java class DecisionModelVacationDaysClient.java that can be used to invoke our lambda function “VacationDays” from any Java program. Here is an example of the main class, called LaunchClient, that can be used to test our Lambda function “VacationDays”:

```
public class LaunchClient {

    public static void main(String[] args) throws Exception {

        Employee[] employees = employeesArray.get();

        String endpoint = "https://c8lxn2vum7.execute-api.us-east-1.amazonaws.com/demo";
        DecisionModelVacationDaysClient client = new DecisionModelVacationDaysClient(endpoint);

        for (Employee employee : employees) {
            System.out.println("\nCalculate Vacation Days for Employee " + employee.getId());
            DecisionModelVacationDaysRequest request = new DecisionModelVacationDaysRequest();
            request.setEmployee(employee);
            Employee resultEmployee = client.request(request);
            System.out.println("Result: " + resultEmployee);
        }
    }
}
```

First, it gets an array of employees that will be used for testing – they are the same employees which were created by the author of our decision model in the file “Test.xls”:

```
Employee[] employees = employeesArray.get();
```

Then it creates an instance of the DecisionModelVacationDaysClient using the endpoint URL which we copied from the AWS API Gateway and already used in POSTMAN. Then it will loop over the array of employees and will call the client’s method request(employee). For each “round-trip” from our local Java program to AWS Lambda and back the launcher will display the resulting employee along with the execution time. Here are the execution results for our 6 test-employees:

```
Calculate Vacation Days for Employee A
Result:
Employee(id=A,vacationDays=27,eligibleForExtra5Days=true,eligibleForExtra3Days=false
eligibleForExtra2Days=false,age=17,service=1)
Elapsed time: 1042 mills

Calculate Vacation Days for Employee B
Result:
Employee(id=B,vacationDays=22,eligibleForExtra5Days=false,eligibleForExtra3Days=fals
,eligibleForExtra2Days=false,age=25,service=5)
Elapsed time: 46 mills

Calculate Vacation Days for Employee C
Result:
Employee(id=C,vacationDays=30,eligibleForExtra5Days=true,eligibleForExtra3Days=true,
eligibleForExtra2Days=true,age=49,service=30)
Elapsed time: 41 mills

Calculate Vacation Days for Employee D
Result:
Employee(id=D,vacationDays=24,eligibleForExtra5Days=false,eligibleForExtra3Days=fals
,eligibleForExtra2Days=true,age=49,service=29)
Elapsed time: 39 mills

Calculate Vacation Days for Employee E
Result:
Employee(id=E,vacationDays=30,eligibleForExtra5Days=true,eligibleForExtra3Days=true,
eligibleForExtra2Days=true,age=57,service=32)
Elapsed time: 45 mills

Calculate Vacation Days for Employee F
Result:
Employee(id=F,vacationDays=30,eligibleForExtra5Days=true,eligibleForExtra3Days=true,
eligibleForExtra2Days=true,age=64,service=42)
Elapsed time: 47 mills
```

It's interesting to analyze the execution time for each “round-trip”. The very first one took a little bit more than 1 second. This relatively “long” time was taken for our Lambda function to “wake up” – by AWS design, when not being used lambda-functions fall asleep not to use AWS resource (and for you not to pay for them!). However, as you can see, all consecutive lambda executions were done within 39 to 47 milliseconds! Note that this time includes networking: if you look at the pure AWS Lambda Test above, you will notice that the actual execution time of the Lambda service was under 4 milliseconds.

Making Changes in Lambda Decision Service

When you make changes in the business decision model "VacationDays", e.g. modify some rules in Excel, you need to rebuild your model by executing "build.bat". OpenRules Decision Manager will re-generate all necessary files including those that were used by your already deployed Lambda function. To make sure that your Lambda function is up-to-date, you need to Open it in AWS Lambda, repeat "Upload" and "Save". The update Lambda function is ready to be executed.

Conclusion

This tutorial provides step-by-step instructions for deployment of OpenRules decision models as AWS Lambda functions. It demonstrates that OpenRules Decision Manager generates all necessary components for business decision models deployment on-premises and on-cloud using the highly popular serverless approach. No programming knowledge was required to create, deploy, and test OpenRules operational decision services. At the same time, OpenRules generates all necessary Java classes to simplify the creation of Java clients for the deployed AWS Lambda services.

Technical Support

Direct all your technical questions to support@openrules.com or to this Discussion Group.