



**OpenRules
Decision Manager**

+



**AWS Lambda
Function**

+



**AWS API
Gateway**

BUILDING OPERATIONAL DECISION SERVICES

WITH

OPENRULES AWS LAMBDA AWS API GATEWAY

OpenRules, Inc.

www.openrules.com

September-2019

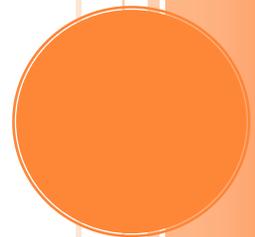


TABLE OF CONTENTS

<i>Table of Contents</i>	1
<i>Introduction</i>	3
<i>What Should Be Pre-Installed</i>	3
<i>What You'll Do</i>	4
<i>Defining Decision Model</i>	4
<i>Uploading Decision Model to AWS Lambda</i>	6
<i>Creating API for Lambda Invocation</i>	9
<i>Testing Lambda Decision Service</i>	14
Testing using Lambda Test	14
Testing using POSTMAN.....	16
Testing using a Java Client.....	18
<i>Making Changes in Lambda Decision Service</i>	20
<i>Conclusion</i>	20
<i>Technical Support</i>	20

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
```

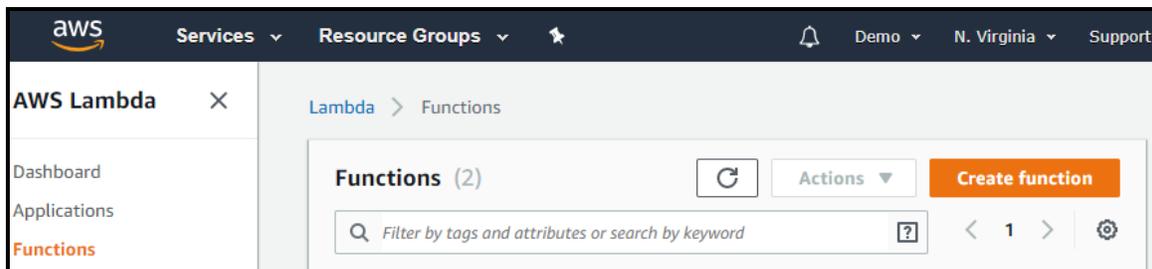
It will prompt “build.bat” to generate several Java classes used during the deployment and Lambda function invocation. For this decision model, the build will generate the following classes in the package “generated/vacation.days”:

- 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:

Basic information

Function name
Enter a name that describes the purpose of your function.

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime [Info](#)
Choose the language to use to write your function.

Permissions [Info](#)
Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and modify permissions further when you add triggers.

▼ **Choose or create an execution role**

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

i Role creation might take a few minutes. The new role will be scoped to the current function. To use it with other functions, you can modify it in the IAM console.

Lambda will create an execution role named VacationDays-role-8rrbrh5z, with permission to upload logs to Amazon CloudWatch Logs.

Cancel **Create function**

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:

The screenshot shows the AWS Lambda console Configuration tab for a function named "VacationDays". The interface is divided into two main sections: "Designer" and "Function code".

Designer Section:

- At the top, there are two tabs: "Configuration" (selected) and "Monitoring".
- Below the tabs is a "Designer" section with a dropdown arrow.
- On the left, there is a key icon and a button labeled "+ Add trigger".
- In the center, there is a box for the function "VacationDays" with a Lambda icon. Below it is a "Layers" section with a stack icon and "(0)" next to it.
- On the right, there is a box for "Amazon CloudWatch Logs" with a CloudWatch icon.
- Below the logs box is a dashed box containing the text: "Resources that the function's role has access to appear here".

Function code Section:

- The section is titled "Function code" with a link to "Info".
- There are three dropdown menus: "Code entry type" (set to "Upload a .zip or .jar..."), "Runtime" (set to "Java 8"), and "Handler" (set to "example.Hello::handleRe").
- Below these is a "Function package" section with an "Upload" button.
- At the bottom, there is a note: "For files larger than 10 MB, consider uploading using Amazon S3."

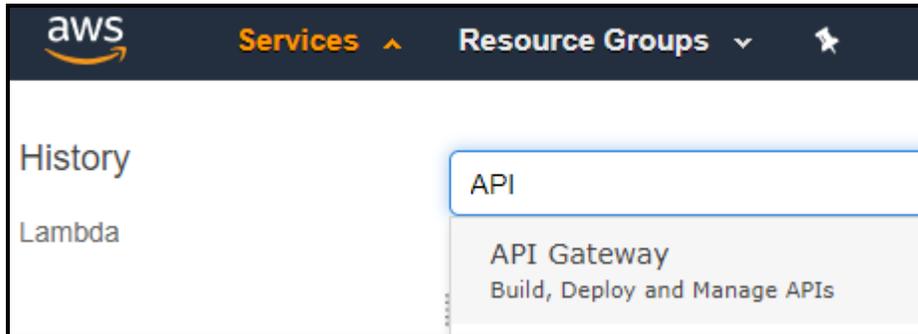
Click on the button **"Upload"**. It will open File Explorer allowing you to choose the already generated file **"DecisionModelVacationDays.lambda.zip"** from the folder **"openrules.dm.models/VacationDays/dist/"**.

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

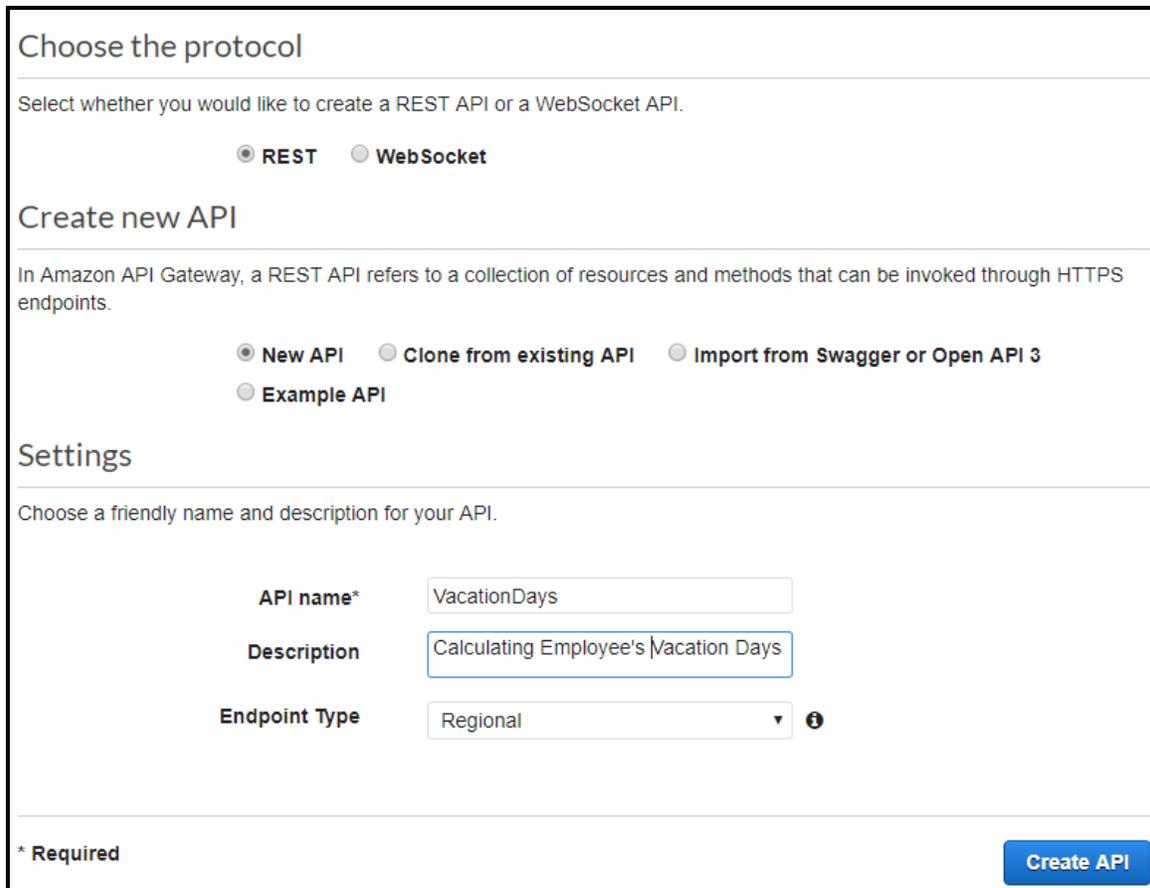
Click on the button **"Save"** in the top right corner. Our AWS Lambda Function **"VacationDays"** is ready. Now we need to create an API for its invocation.

CREATING API FOR LAMBDA INVOCATION

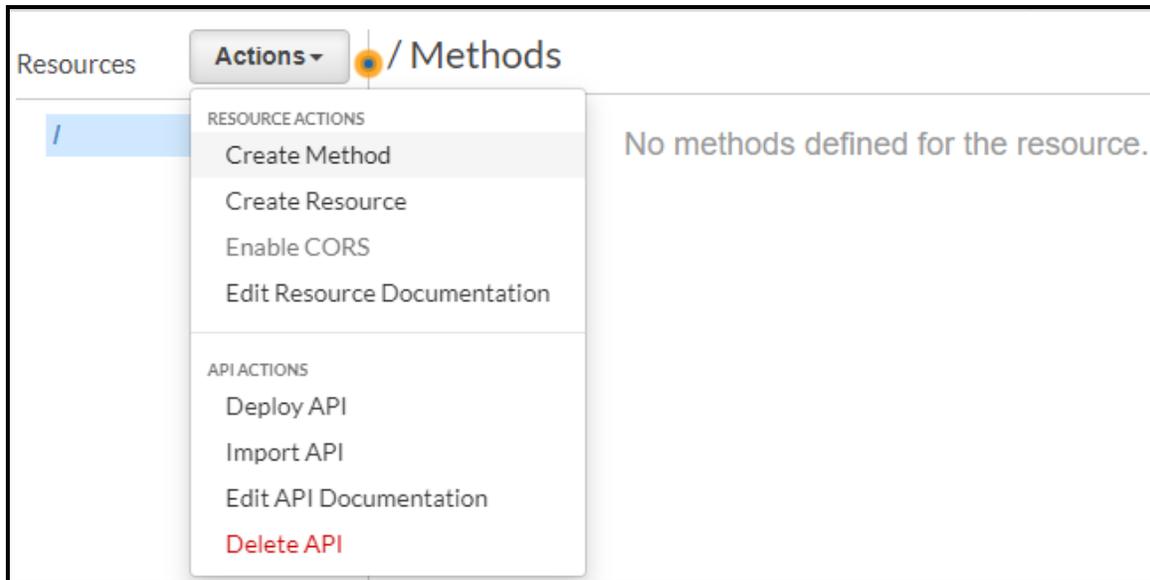
Let's switch to AWS API Gateway. Click on the menu item "Services" and find "API Gateway" by starting to type "API" in the search box:



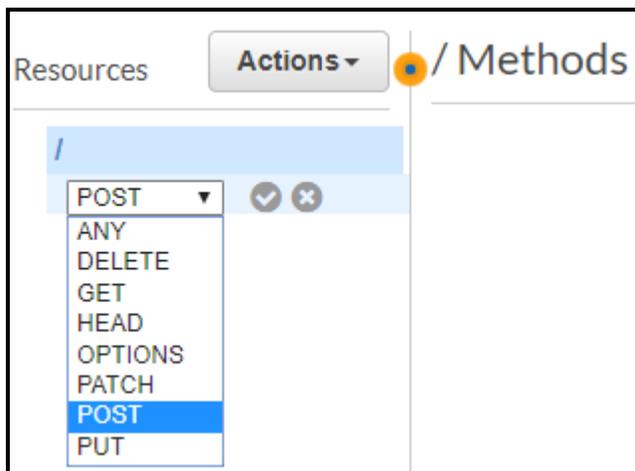
On the next screen click on  and it will display:

A screenshot of the "Choose the protocol" screen in the AWS console. The screen is divided into three sections: "Choose the protocol", "Create new API", and "Settings".
1. "Choose the protocol": A heading followed by the instruction "Select whether you would like to create a REST API or a WebSocket API." Below this are two radio buttons: "REST" (selected) and "WebSocket".
2. "Create new API": A heading followed by the instruction "In Amazon API Gateway, a REST API refers to a collection of resources and methods that can be invoked through HTTPS endpoints." Below this are four radio buttons: "New API" (selected), "Clone from existing API", "Import from Swagger or Open API 3", and "Example API".
3. "Settings": A heading followed by the instruction "Choose a friendly name and description for your API." Below this are three form fields:
- "API name*": A text input field containing "VacationDays".
- "Description": A text input field containing "Calculating Employee's Vacation Days".
- "Endpoint Type": A dropdown menu with "Regional" selected and an information icon to its right.
At the bottom left, there is a note "* Required". At the bottom right, there is a blue "Create API" button.

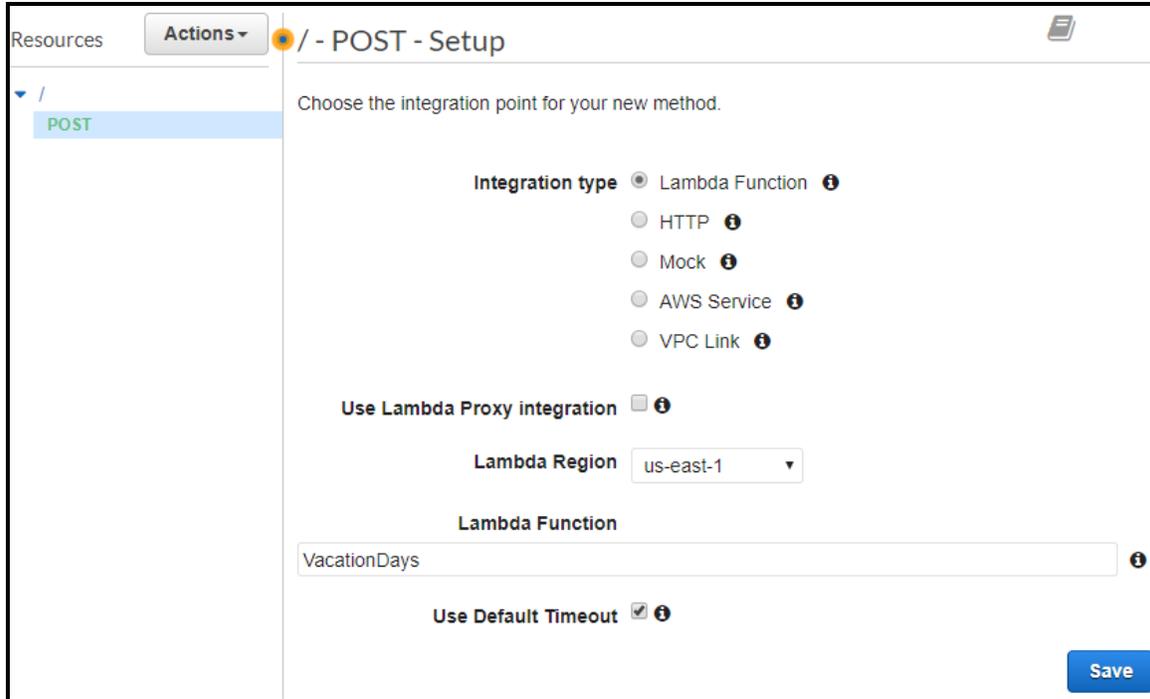
Make sure that “REST” and “New API” are selected (the defaults) and enter the API name and Description as on the above screen. Click on the button “**Create API**”. On the next screen



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”:

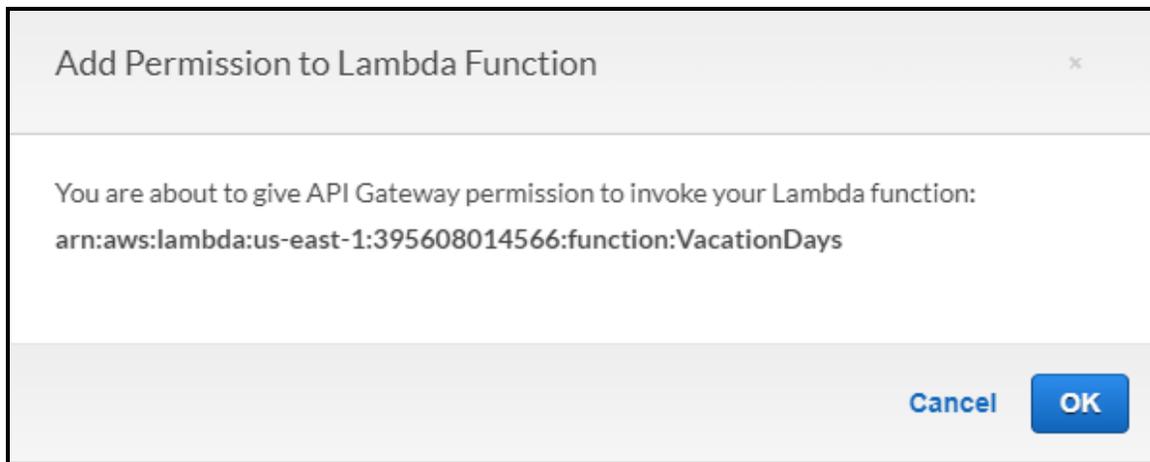


The screenshot shows the AWS API Gateway console interface for configuring a new POST method. The left sidebar shows the resource path `/` with the `POST` method selected. The main area is titled `/ - POST - Setup` and contains the following configuration options:

- Integration type:** Radio buttons for `Lambda Function` (selected), `HTTP`, `Mock`, `AWS Service`, and `VPC Link`.
- Use Lambda Proxy integration:** A checkbox that is currently unchecked.
- Lambda Region:** A dropdown menu set to `us-east-1`.
- Lambda Function:** A text input field containing `VacationDays`.
- Use Default Timeout:** A checkbox that is checked.

A `Save` button is located at the bottom right of the configuration area.

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:

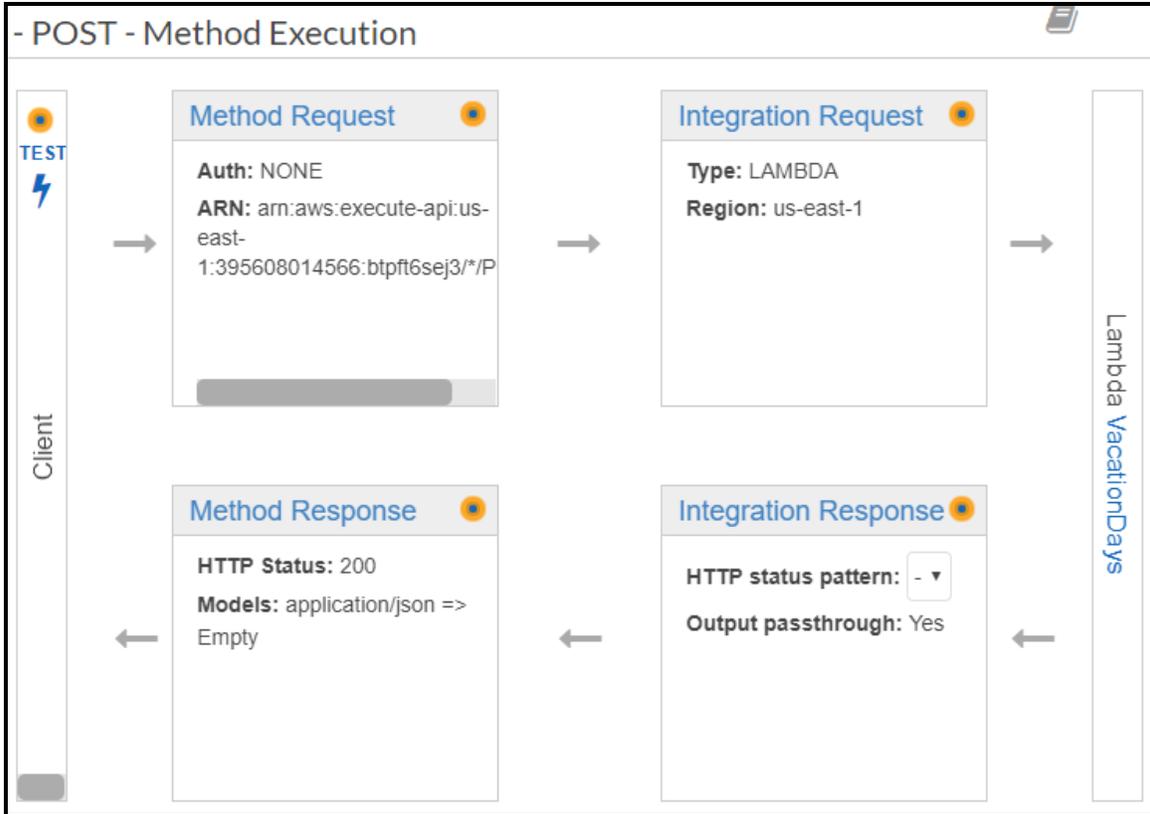


The screenshot shows a dialog box titled `Add Permission to Lambda Function`. The dialog contains the following text:

You are about to give API Gateway permission to invoke your Lambda function:
`arn:aws:lambda:us-east-1:395608014566:function:VacationDays`

At the bottom right of the dialog, there are two buttons: `Cancel` and `OK`.

Click “OK”. It will display this view:



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

Deploy API

Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.

Deployment stage: [New Stage]

Stage name*: demo

Stage description: demo

Deployment description: demo

Buttons: Cancel, Deploy

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:

The screenshot shows the 'demo Stage Editor' interface. At the top right, there are buttons for 'Delete Stage' and 'Configure Tags'. Below this is a blue box containing the 'Invoke URL: https://btpft6sej3.execute-api.us-east-1.amazonaws.com/demo'. A navigation bar includes tabs for 'Settings', 'Logs/Tracing', 'Stage Variables', 'SDK Generation', 'Export', and 'Deployment History'. Below the navigation bar are tabs for 'Documentation History' and 'Canary'. The main content area is divided into sections: 'Cache Settings' with an 'Enable API cache' checkbox; 'Default Method Throttling' with a description, an 'Enable throttling' checkbox, and input fields for 'Rate' (10000 requests per second) and 'Burst' (5000 requests); 'Web Application Firewall (WAF)' with a 'Web ACL' dropdown set to 'None' and a 'Create Web ACL' link; and 'Client Certificate' with a 'Certificate' dropdown set to 'None'. A 'Save Changes' button is located at the bottom right.

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**”. Our Lambda Function is successfully deployed and we know its invocation URL. We may test this lambda function.

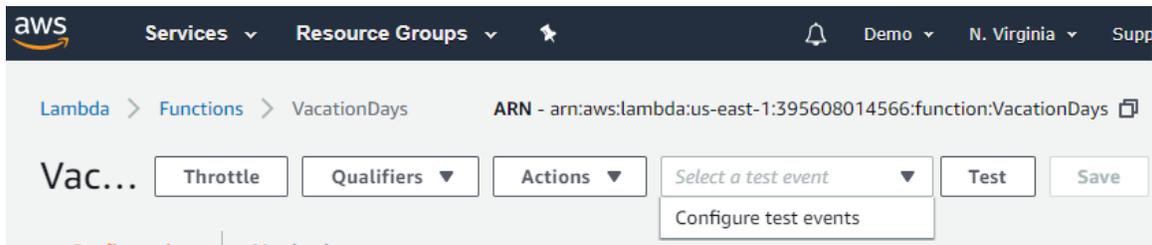
TESTING LAMBDA DECISION SERVICE

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:

Configure test event ✕

A function can have up to 10 test events. The events are persisted so you can switch to another computer or web browser and test your function with the same events.

Create new test event
 Edit saved test events

Event template
Hello World ▼

Event name
CalculateVacationDays

```
1 {  
2   "id": "Robinson",  
3   "age": 57,  
4   "service": 30  
5 }
```

Cancel Create

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

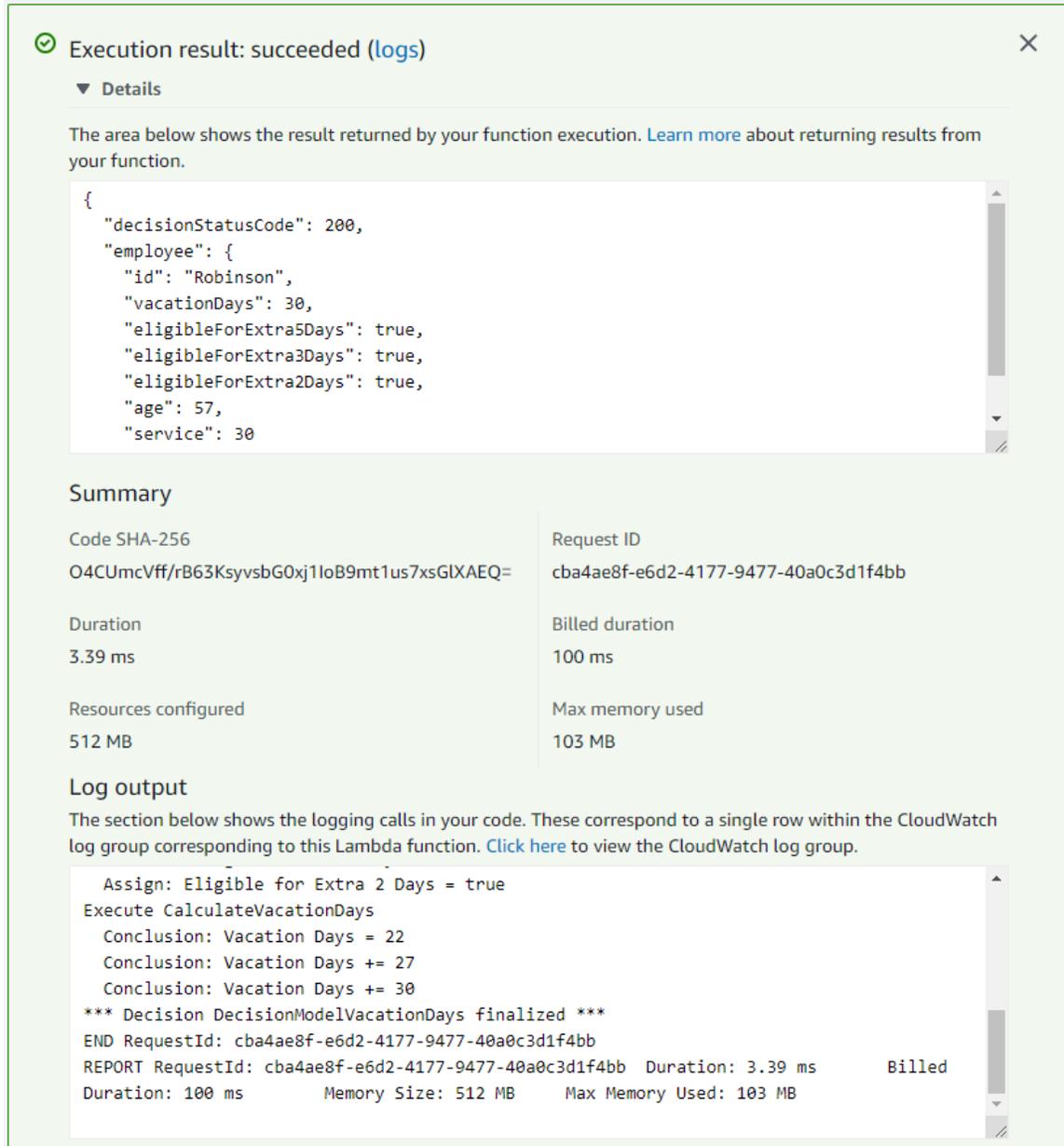
```
{  
  "id": "Robinson",  
  "age": 57,  
  "service": 30  
}
```

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

Vac... Throttle Qualifiers ▼ Actions ▼ CalculateVacationDays ▼ Test Save

✔ Execution result: succeeded (logs) ✕
▶ Details

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



Execution result: succeeded (logs) ✕

▼ Details

The area below shows the result returned by your function execution. [Learn more](#) about returning results from your function.

```
{
  "decisionStatusCode": 200,
  "employee": {
    "id": "Robinson",
    "vacationDays": 30,
    "eligibleForExtra5Days": true,
    "eligibleForExtra3Days": true,
    "eligibleForExtra2Days": true,
    "age": 57,
    "service": 30
  }
}
```

Summary

Code SHA-256	Request ID
O4CUmcVff/rB63KsyvsbG0xj1loB9mt1us7xsGlXAEQ=	cba4ae8f-e6d2-4177-9477-40a0c3d1f4bb
Duration	Billed duration
3.39 ms	100 ms
Resources configured	Max memory used
512 MB	103 MB

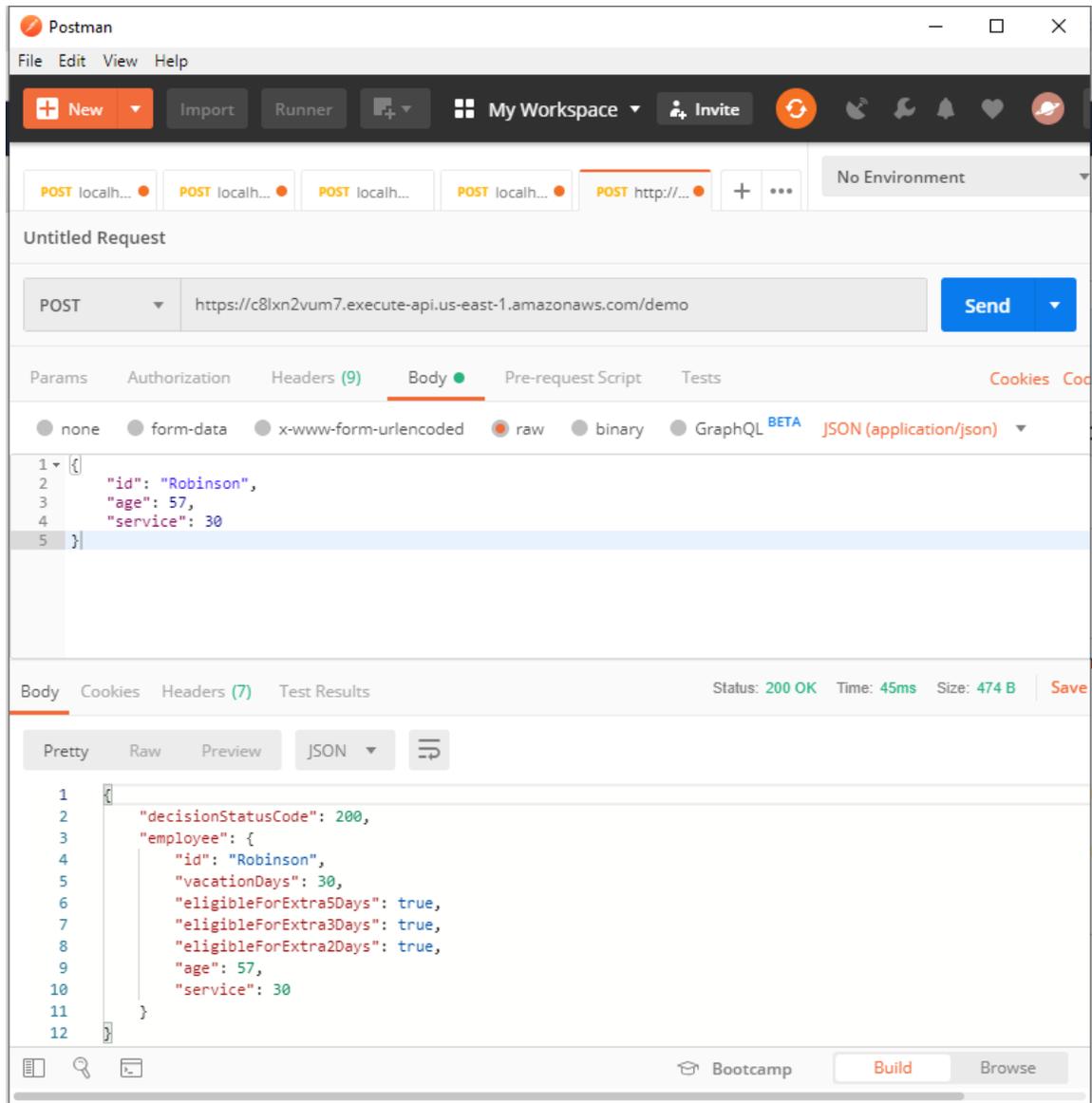
Log output

The section below shows the logging calls in your code. These correspond to a single row within the CloudWatch log group corresponding to this Lambda function. [Click here](#) to view the CloudWatch log group.

```
Assign: Eligible for Extra 2 Days = true
Execute CalculateVacationDays
Conclusion: Vacation Days = 22
Conclusion: Vacation Days += 27
Conclusion: Vacation Days += 30
*** Decision DecisionModelVacationDays finalized ***
END RequestId: cba4ae8f-e6d2-4177-9477-40a0c3d1f4bb
REPORT RequestId: cba4ae8f-e6d2-4177-9477-40a0c3d1f4bb Duration: 3.39 ms Billed
Duration: 100 ms Memory Size: 512 MB Max Memory Used: 103 MB
```

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://c81xn2vum7.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=false,
,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=false,
,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](#).