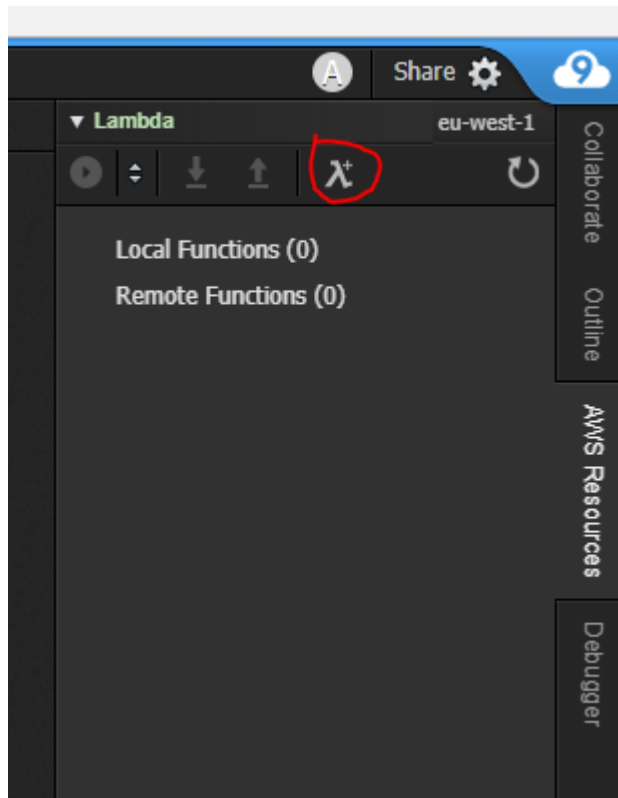




## Building HelloWorld

Let's start off create a new application for the mandatory "Hello World" example:

**Create a new lambda function by using the AWS Resources menu and selecting the lambda+ icon:**



**Provide a function and an application name:**

The screenshot shows the 'Create serverless application' dialog box. It has a title bar with a close button. The main content area contains explanatory text and two input fields. The first input field is labeled 'Function name:' and contains the text 'HelloWorld'. Below it, a message states 'Function name may only contain alphanumeric characters'. The second input field is labeled 'Application name:' and also contains 'HelloWorld', with a similar message below it. At the bottom left, the 'Region' is set to 'eu-west-1'. At the bottom right, there is a green 'Next' button. The bottom of the dialog shows a dark bar with a tab labeled 'ip-172-31-16', a status 'Immediate', and a '+', along with some icons on the right.

**Create serverless application**

A serverless application can have one or more AWS Lambda functions, along with triggers and integrations for each of those functions. All of its configuration is stored in an AWS CloudFormation template file

Application name is used for the folder name for both your application's files and the AWS CloudFormation stack name when you deploy this application.

Function name:

Function name may only contain alphanumeric characters

Application name:

Application name may only contain alphanumeric characters

Region: eu-west-1 **Next**

The function needs to be developed. Let's **select an empty Python 3.6+ template** to write the Hello World example as **Python code**:



**Create serverless application**

Select runtime

All runtimes

Select blueprint

<b>empty-nodejs</b> An empty NodeJS function nodejs · nodejs6.10	<b>empty-python</b> An empty Python function python · python3.6
<b>alexa-skill-kit-sdk-factskill</b> Demonstrate a basic fact skill built with the ASK NodeJS SDK nodejs6.10 · alexa	<b>alexa-skills-kit-color-expert</b> Demonstrates a basic skill built with the Amazon Alexa Skills Kit. nodejs6.10 · alexa
<b>alexa-skills-kit-color-expert</b> Demonstrates a basic skill built with the Amazon	<b>alexa-smart-home-skill-adapter</b> Provides the basic framework for a skill adapter for a

Region: eu-west-1

Previous Next

**For now, we don't require a function trigger** (we will test the function manually first):

**Create serverless application**

Function trigger

none

Region: eu-west-1

Previous Next

Also, **default memory settings and a default generated role should be enough** - we're not doing any heavy lifting, nor are we using any asynchronous functionality:



### Create serverless application

Memory (MB): 128 MB

Your function is allocated CPU proportional to the memory configured.

Role: Automatically generate role

Region: eu-west-1

PreviousNext

### Review the settings and finish:

### Create serverless application

Review your serverless application details below. You can go back to make changes for each section. When you are ready, click Finish to complete the setup process.

After AWS Cloud9 creates your new application, it will be automatically deployed using AWS CloudFormation.

#### Lambda function

Region	eu-west-1
Name	HelloWorld
ApplicationName	HelloWorld
Runtime	python3.6
Blueprint	empty-python
Handler	lambda_function.lambda_handler
Memory-size	128
Role	Automatically Generated

Region: eu-west-1

PreviousFinish

From the setting above, you should note that the Handler is "`lambda_function.lambda_handler`". What does this mean? Well, if the function is invoked, it will look for a Python **source file** named "`lambda_function`" (.py, actually) and try and execute the **function** "`lambda_handler`" inside this file. To isolate this project from other projects in the same IDE, Cloud9 will create a so-called "Virtual Environment" for Python, where all its dependencies will be stored and not pollute the global namespace - reducing the chances of conflicts.



As you can see, the AWS Lambda framework will provide the function with an event and a context: the event is the event that triggered the invocation of the function, the context consists of AWS specific environment information, the so-called execution context.

**Let's insert code to examine both the event and the context passed in and return a greeting using "Hello " plus the name if present in the input event, otherwise name should be defaulting to "World" .**

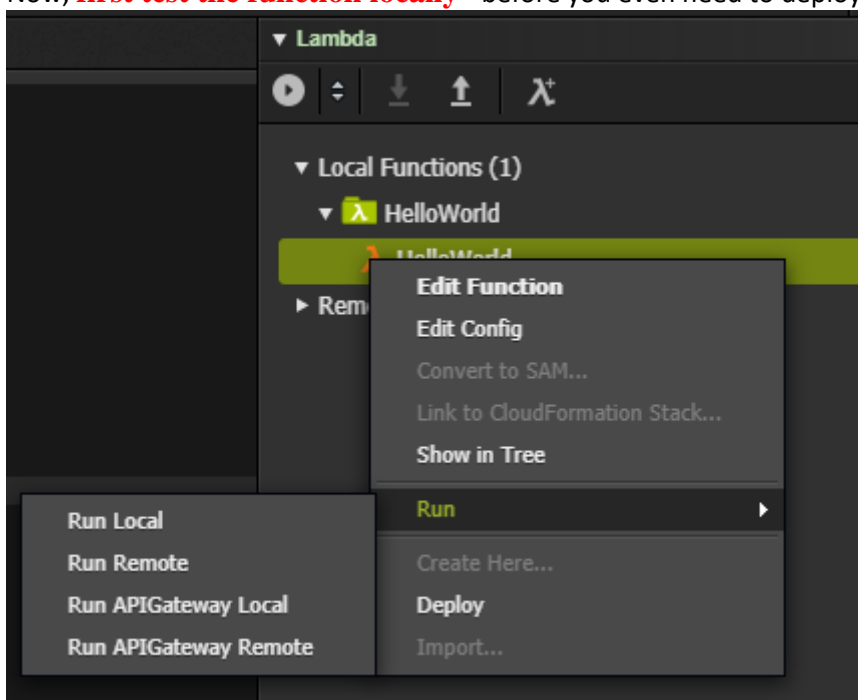
```

1 import json
2 print('Loading...')
3
4
5 def lambda_handler(event, context):
6     # event is a dict object
7     print('Got an event: ' + json.dumps(event, indent=2))
8
9     # context is an object - https://docs.aws.amazon.com/lambda/latest/dg/python-context-object.html
10    print("Execution context - {} ms remaining".format(context.get_remaining_time_in_millis()))
11    print('Execution context: function {} version {}, size {} mb'.format(context.function_name, context.function_version, context.memory_limit_in_mb))
12    print('Logging to log group {} and log stream {}'.format(context.log_group_name, context.log_stream_name))
13
14    naam = 'World' if 'name' not in event else event['name']
15
16    groet = 'Hello ' + naam + "!"
17    print('We gaan groeten ... ' + groet)
18
19    return groet
20
21 print('Done loading!')
```



Relevant code snippets (code for the handler function, or the payload for the test event) can be found in the GitHub repository <https://github.com/mnuman/syntouch-aws-lambda-workshop/tree/master/2-Hello%20World>.

Now, **first test the function locally** - before you even need to deploy it to the real cloud:



This opens a window that lets you specify an **event payload**. **As we are expecting a JSON string element called name at a minimum set this element in the event payload:**



```

bash - "ip-172-31-16" x [A] HelloWorld x +
Run /HelloWorld/.debug/HelloWorld/lambda_function.py

▼ Test payload
Function: HelloWorld
Payload:
1 {
2   "name" : "SynTouch Colleagues",
3   "array" : [ "This", "is", "a", "structured", "object"],
4   "error" : false
5 }

▼ Execution results
Response
"Hello SynTouch Colleagues!"

Function Logs
Loading...
Done loading!
Got an event: {
  "name": "SynTouch Colleagues",
  "array": [
    "This",
    "is",
    "a",
    "structured",
    "object"
  ],
  "error": false
}
Execution context - 15000 ms remaining
Execution context: function test version $LATEST, size 128 mb
Logging to log group /aws/lambda/test and log stream 2018/7/4/[$LATEST]ffda7bf63c5909fe

```

When supplying a name attribute in the JSON event, the contents are used for the greeting.

You can see the function loading (initializing), finish the initialization and then executing on the supplied event. As we're running locally, the actual names ("test" for the function name) may be different from the real run-time environment values.

What Cloud9 is doing behind the scenes, is creating a SAM (Serverless Application Model) template to provide all resources required (template.yaml). Actually, this is an extension to the Amazon CloudFormation service, which creates or updates an "application stack" based on an input template ('infrastructure' as code!)

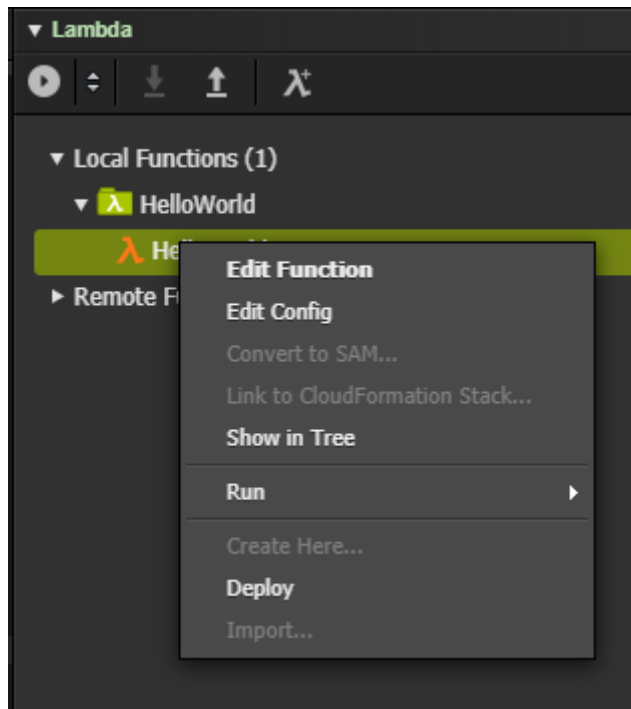
**Before deployment, we need to make a small change to the template that is used to provision the stack (if we don't we cannot control the actual name assigned to the function completely): add a FunctionName attribute to the HelloWorld function and set this to HelloWorld:**

```

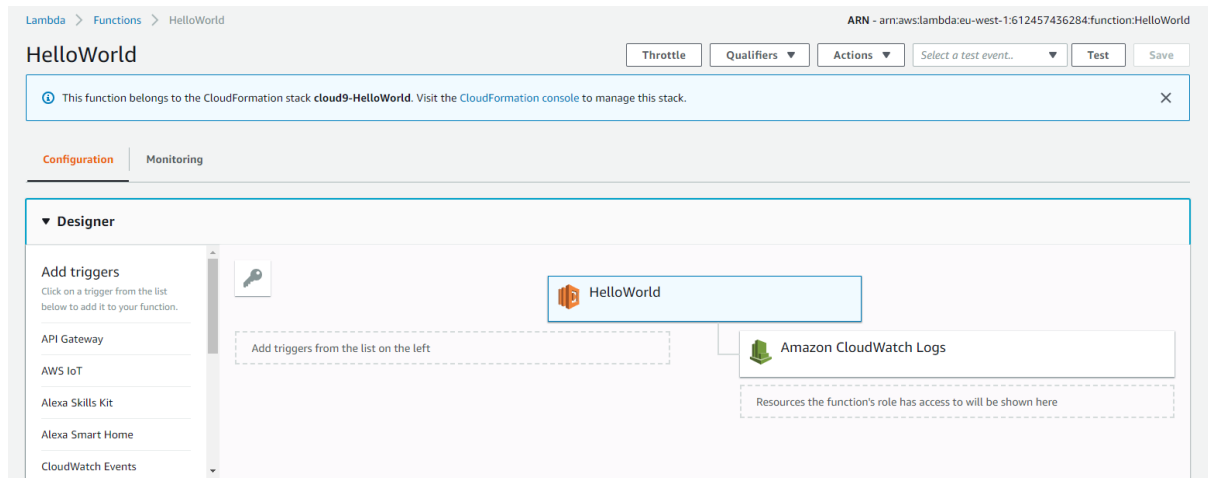
lambda_function.py x template.yaml x +
1 AWSTemplateFormatVersion: '2010-09-09'
2 Transform: 'AWS::Serverless-2016-10-31'
3 Description: An AWS Serverless Specification template describing your function.
4 Resources:
5   HelloWorld:
6     Type: 'AWS::Serverless::Function'
7     Properties:
8       Handler: HelloWorld/lambda_function.lambda_handler
9       FunctionName: HelloWorld
10      Runtime: python3.6
11      Description: 'This is my super duper HelloWorld function!'
12      MemorySize: 128
13      Timeout: 15
14      CodeUri: .debug/
15

```

Now the code is ready to rock & roll! **Deploy your code**



After some seconds, a stack has been provisioned using CloudFormation. **Examine the actual HelloWorld Lambda function from the AWS Lambda console:**



**Examine the code:**



HelloWorld Throttle Qualifiers Actions Select a test event... Test Save

Function code [Info](#)

Code entry type: Edit code inline Runtime: Python 3.6 Handler: HelloWorld/lambda\_function.lambda\_handler

```

1 import json
2 print('Loading...')
3
4 def lambda_handler(event, context):
5     # event is a dict object
6     print('Got an event: ' + json.dumps(event, indent=2))
7
8     # context is an object - https://docs.aws.amazon.com/lambda/latest/dg/python-context-object.html
9     print('Execution context: ' + json.dumps(context.get_remaining_time_in_millis()))
10    print('Execution context: function {} version {}, size {} mb'.format(context.function_name, context.function_version, context.memory_limit_in_mb))
11    print('Logging to log group {} and log stream {}'.format(context.log_group_name, context.log_stream_name))
12
13
14    name = 'World' if 'name' not in event else event['name']
15
16    greet = 'Hello ' + name + '!'
17    print('We gaan groeten ... ' + greet)
18
19    return greet
20
21 print('Done loading!')
```

1:1 Python Spaces: 4

And **inspect other relevant settings:**

**Execution role**

Defines the permissions of your function. Note that new roles may not be available for a few minutes after creation. [Learn more](#) about Lambda execution roles.

Choose an existing role

Existing role

You may use an existing role with this function. Note that the role must be assumable by Lambda and must have Cloudwatch Logs permissions.

cloud9-HelloWorld-HelloWorldRole-11HH2LFWGK063

**Basic settings**

Description

This is my super duper HelloWorld function!

Memory (MB) [Info](#)

Your function is allocated CPU proportional to the memory configured.

128 MB

Timeout [Info](#)

0 min 15 sec

**Examine the role generated and verify this only allows writing logging to CloudWatch:**

**Summary**

Role ARN: arn:aws:iam::612457436284:role:cloud9-HelloWorld-HelloWorldRole-11HH2LFWGK063

Role description: Edit

Instance Profile ARNs:

Path: /

Creation time: 2018-07-04 20:59 UTC+0200

Maximum CLI/API session duration: 1 hour Edit

Permissions: **Attach policies** Attached policies: 1 [Add inline policy](#)

Policy name: AWSLambdaBasicExecutionRole Policy type: AWS managed policy

Policy summary [JSON](#) [Simulate policy](#)

```

1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Action": [
7         "logs:CreateLogGroup",
8         "logs:CreateLogStream",
9         "logs:PutLogEvents"
10      ],
11       "Resource": "*"
12     }
13   ]
14 }
```

And only AWS Lambda is allowed to assume this role:



[Roles](#) > [cloud9-HelloWorld-HelloWorldRole-11HH2LFWGK063](#)

## Summary

Role ARN	arn:aws:iam::612457436284:role/cloud9-HelloWorld-HelloWorldRole-11HH2LFWGK063
Role description	<a href="#">Edit</a>
Instance Profile ARNs	<a href="#">Edit</a>
Path	/
Creation time	2018-07-04 20:59 UTC+0200
Maximum CLI/API session duration	1 hour <a href="#">Edit</a>

Permissions

Trust relationships

Access Advisor

Revoke sessions

You can view the trusted entities that can assume the role and the access conditions for the role. [Show policy document](#)

[Edit trust relationship](#)

**Trusted entities**  
The following trusted entities can assume this role.  

**Trusted entities**  
The identity provider(s) `lambda.amazonaws.com`

**Conditions**  
The following conditions define how and when trusted entities can assume the role.  
There are no conditions associated with this role.

**Return to the AWS HelloWorld Lambda function and create a new test event by using the Test button:**

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

MyHelloWorldEvent

```
1 {  
2   "name": "SynTouch Colleagues",  
3   "array": ["This", "is", "a", "structured", "object"],  
4   "error": false  
5  
6 }
```

**Save the event and test!**

Lambda > Functions > HelloWorld

ARN - arn:aws:lambda:eu-west-1:612457436284:function:HelloWorld

HelloWorld

Throttle

Qualifiers ▼

Actions ▼

MyHelloWorldEvent ▼

Test

Save

ⓘ

This function belongs to the CloudFormation stack `cloud9-HelloWorld`. Visit the [CloudFormation console](#) to manage this stack.

×



The execution logs provide you with the actual output, but also with useful information about the duration, the duration you will be billed for (rounded up to 100 ms) and the actual memory consumed. **Examine the log output for the actual execution in the cloud.**

Lambda > Functions > HelloWorld

ARN - arn:aws:lambda:eu-west-1:612457436284:function:HelloWorld

ThrottleQualifiersActionsMyHelloWorldEventTestSave

Execution result: succeeded (logs)

Details

The area below shows the result returned by your function execution.

"Hello SynTouch Colleagues!"

Summary

Code SHA-256	+bmQ55md2ISY6IFTwR6pwyVx1WnQN7ZEKhkPQWtNB8=	Request ID	2f13072f-7fbe-11e8-82cf-df068de8b3bb
Duration	0.64 ms	Billed duration	100 ms
Resources configured	128 MB	Max memory used	22 MB

Log output

The area 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.

```

{
  "object":
},
  "error": false
}
Execution context - 14999 ms remaining
Execution context: Function HelloWorld version $LATEST, size 128 mb
Logging to log group /aws/lambda/HelloWorld and log stream 2018/07/04:[$LATEST]d9c71e52eb4c4b4dbf849de2228bdcc4
We gaan groeten ... Hello SynTouch Colleagues!
END RequestId: 2f13072f-7fbe-11e8-82cf-df068de8b3bb
REPORT RequestId: 2f13072f-7fbe-11e8-82cf-df068de8b3bb  Duration: 0.64 ms    Billed Duration: 100 ms    Memory Size: 128 MB    Max Memory Used: 22 MB

```

The monitoring tab returns miscellaneous statistics on your function, but also offers access to the cloudwatch log files where this time period was logged:

[illegible]

## Expose HelloWorld as a WebAPI

Now that we have defined the mandatory HelloWorld artefact as a serverless function in AWS Lambda, let's go the extra mile and expose this to the GBI ("Grote Boze Internet") as an WebAPI to be called over HTTP.

**Make sure that you're still in the eu-west-1 (Ireland) region**, as this is where your Lambda function lives.

### Define a new API in API Gateway:



Amazon API Gateway APIs > Create

### Create new API

In Amazon API Gateway, an API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.

☒ New API ☐ Import from Swagger ☐ Example API

### Settings

Choose a friendly name and description for your API.

API name\*

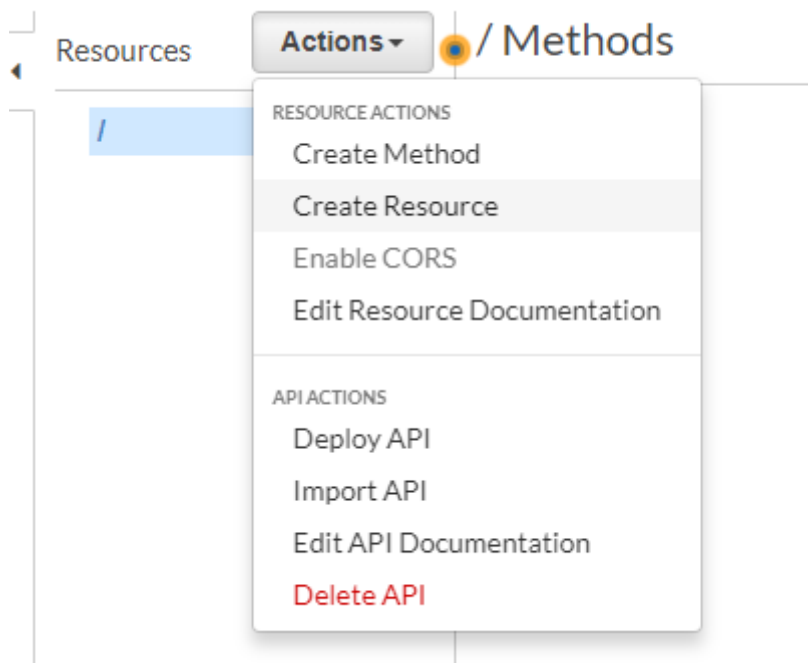
Description

Endpoint Type

\* Required

We'll define the API as a GET on resource **helloworld** where the name is to be provided as a query parameter ...

**From the drop down, choose "Create Resource":**



**Create the /helloworld resource:**

Resources Actions ▾

New Child Resource

Use this page to create a new child resource for your resource.

Configure as [proxy resource](#) ☐

Resource Name\*

Resource Path\*

You can add path parameters using brackets. For example, the resource path **(username)** represents a path parameter called 'username'. Configuring **/(proxy+)** as a proxy resource catches all requests to its sub-resources. For example, it works for a GET request to /foo. To handle requests to /, add a new ANY method on the / resource.

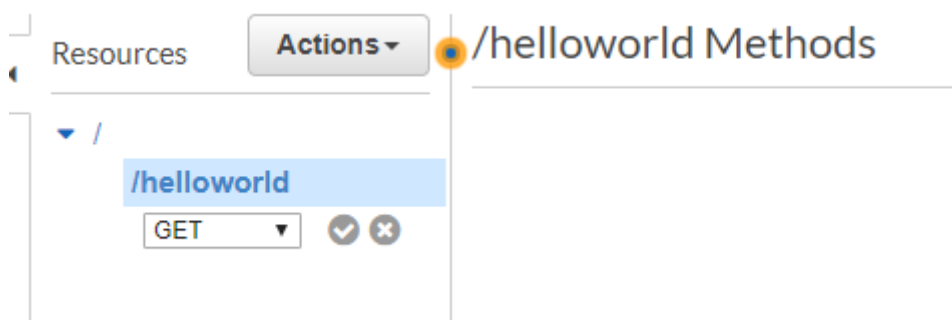
Enable API Gateway CORS ☐


\* Required

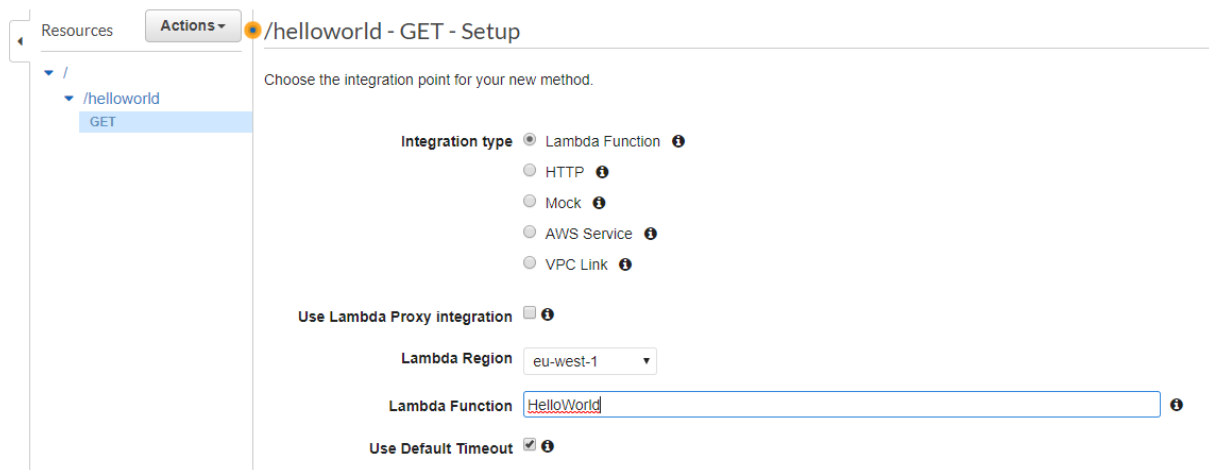
Cancel Create Resource



After creation of the resource, **make sure the new resource is selected** and **choose “Create Method”** from the dropdown list; select the **GET** HTTP method:

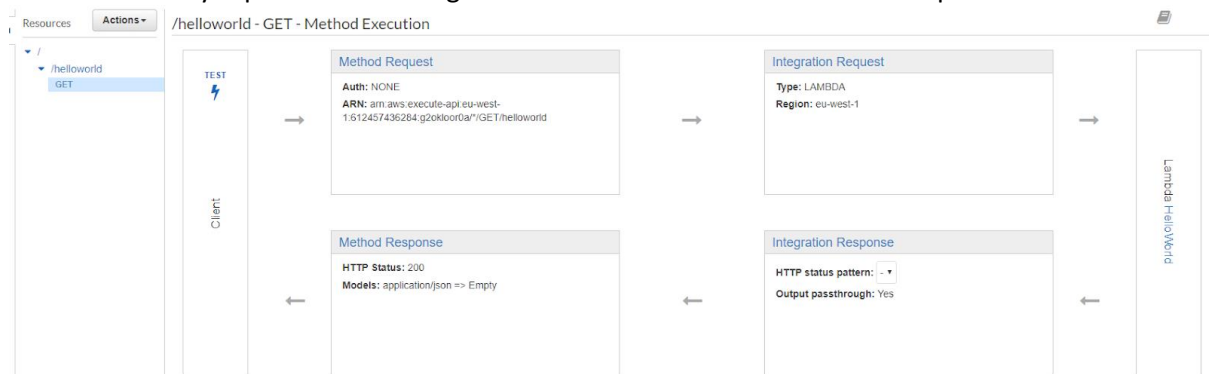


If you **press the “Okay” action** , you’re presented with a configuration form to connect your exposed endpoint. Of course **we’ll be connecting this to the HelloWorld lambda function**:



**When asked, authorize API Gateway to invoke your Lambda function.**

The API Gateway represents the integration with the backend as a number of phases:



In the first stage (Method Request), the expected payload structure, headers and query string parameters can be defined. The request can be rejected if it does not satisfy the defined requirement. For our current purpose, this is not relevant.



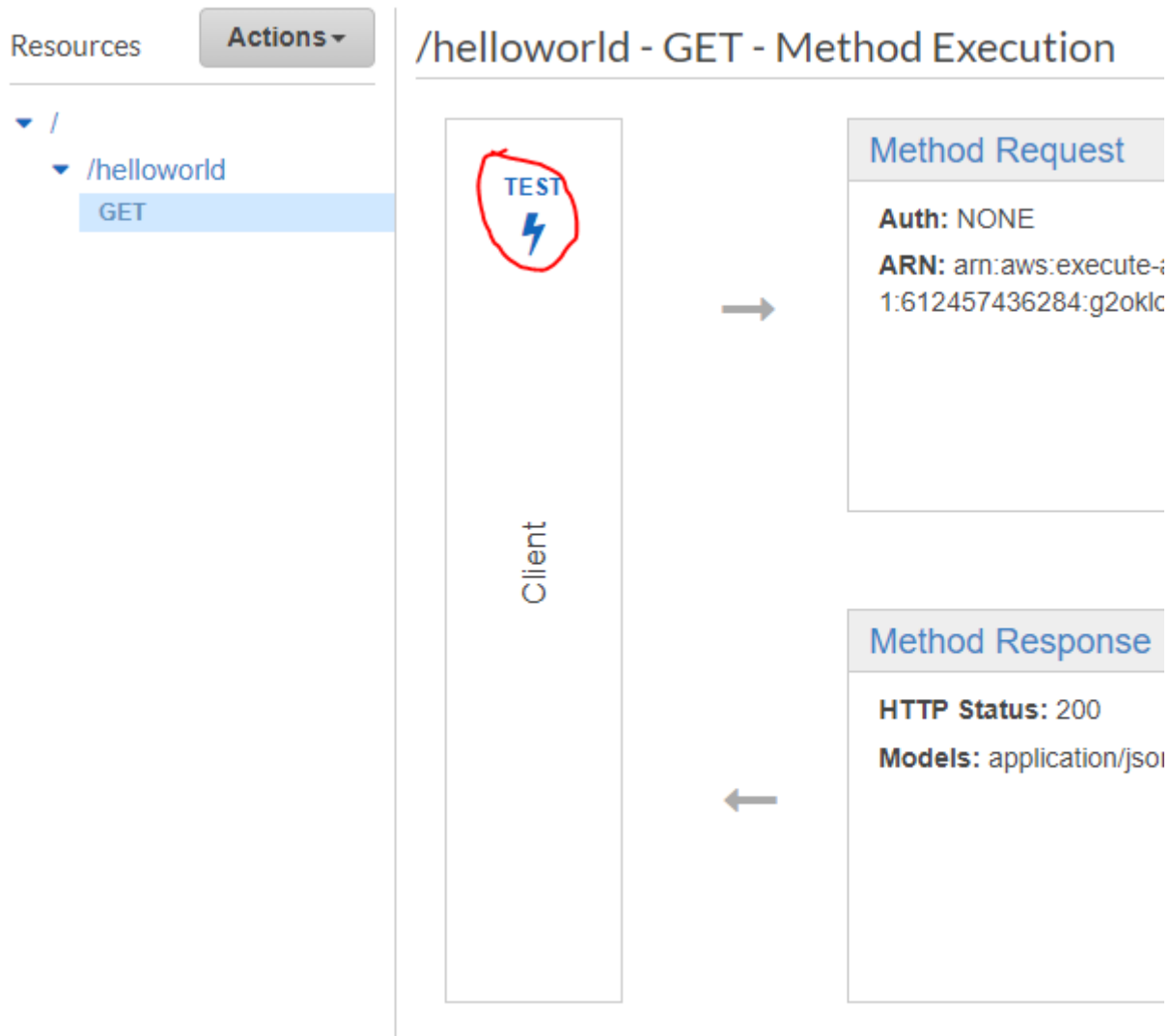
The next step, Integration Request, is used to transform the received request to the request the backend (in this case our HelloWorld lambda function) will receive.

Here, we need to find a way to transfer the contents of the name querystring parameter to a name element in the JSON payload for the backend service. API Gateway supports the Apache Velocity templating engine for performing these tasks. For an overview of the available mapping functions, see <https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-mapping-template-reference.html>

**Define a new template to be used for application/json content types; insert your expression replacing your-mapping-expression-here-to-transfer-the-querystring-param-nam-value:**

The screenshot shows the AWS API Gateway console. On the left, the 'Resources' pane shows a tree structure with a root '/' and a resource '/helloworld' with a 'GET' method selected. The main pane is titled 'Actions' and contains three sections: 'URL Query String Parameters', 'HTTP Headers', and 'Mapping Templates'. The 'Mapping Templates' section is expanded, showing a table with one row for 'application/json'. The 'Request body passthrough' options are set to 'When no template matches the request Content-Type header'. Below the table, there is a text input field containing 'application/json' and a 'Generate template' dropdown menu. The template editor shows a JSON object with a single key 'name' and a value that is a placeholder for a mapping expression: `{ "name" : your-mapping-expression-here-to-transfer-the-querystring-param-nam-value }`.

**After completing your mapping template, navigate back to your API, select the GET method on helloworld and invoke the TEST button on the client:**



**Next, provide the query string parameter name and set to a value of your choice. TEST!**

APIs > HelloWorldAPI (g2okl0r0a) > Resources > /helloworld (ykjv34) > GET

Show all hints ?

**Resources**   **Actions** ▾

▾ /helloworld

GET

**Method Execution /helloworld - GET - Method Test**

Make a test call to your method with the provided input

**Path**

No path parameters exist for this resource. You can define path parameters by using the syntax **{myPathParam}** in a resource path.

**Query Strings**

**{helloworld}**

name=World And Your Mama

**Headers**

**{helloworld}**

Use a colon (:) to separate header name and value, and new lines to declare multiple headers. eg. Accept: application/json.

**Stage Variables**

No **Stage variables** exist for this method.

**Request Body**

Request Body is not supported for GET methods.

**Request:** /helloworld?name=World And Your Mama

**Status:** 200

**Latency:** 51 ms

**Response Body**

"Hello World And Your Mama!"

**Response Headers**

{ "X-Amzn-Trace-Id": "Root=1-5b4a594f-686f9043bd5999f5949eeab;Sampled=0", "Content-Type": "application/json" }

**Logs**

Execution log for request Affc0c63-87a2-11e8-9d62-5949cfc677d7

Sat Jul 14 20:13:03 UTC 2018 : Starting execution for request: Affc0c63-87a2-11e8-9d62-5949cfc677d7

Sat Jul 14 20:13:03 UTC 2018 : HTTP Method: GET, Resource Path: /helloworld

Sat Jul 14 20:13:03 UTC 2018 : Method request path: {}

Sat Jul 14 20:13:03 UTC 2018 : Method request query string: {name=World And Your Mama}

Sat Jul 14 20:13:03 UTC 2018 : Method request headers: {}

Sat Jul 14 20:13:03 UTC 2018 : Method request body before transformations:

Sat Jul 14 20:13:03 UTC 2018 : Endpoint request URI: https://lambda.eu-west-1.amazonaws.com/2015-03-31/functions/arn:aws:lambda:eu-west-1:612457436284:function:HelloWorld/invocations

Sat Jul 14 20:13:03 UTC 2018 : Endpoint request headers: {x-amzn-lambda-integration-tag=Affc0c63-87a2-11e8-9d62-5949cfc677d7, Authorization=\*\*\*\*\*}

\*\*\*\*\*

\*\*\*\*\*469546, X-Amz-Date=20180714T201303Z, x-amzn-apigateway-api-id=g2okl0r0a, X-Amz-Source-Arn=arn:aws:execute-api:eu-west-1:612457436284:g2okl0r0a/test-invoke-stage/GET/helloworld, Accept=application/json, User-Agent=AmazonAPIGateway\_g2okl0r0a, X-Amz-Security-Token=FQcDYXZzEhWdNlqt2LtgLFPImBdIK3A64zrJqf52AjxRuT6NOK0ly9j0XhDv9a35YwRucZmKaXbq+TnShBaY9vPzegDIIMH4eep5mqdHm4gLTsvHpgdqY0xmerV1VDFcg/1ZDg14pqa+1HAlVPP37F0MPq4tkcmUjxuk3UBAVS+htZp8PKPYEHR98huuxEAsP7Vid7dCUMAl12ZUF1oz0rpXIGCZc+1foQH/AUEzqtySrmHUIubkcvkCZPDYT8d4T1e

**Test**



As you can **see from the logs** (and the response), the query string parameter is mapped to the name element in the payload of the event. *However, the response is simply returned as plain old text - since the function simply returns text.*

This can be easily solved using the **Integration Response mapping template**, in a similar fashion we have applied to the request:

Resources Actions ▾

Method Execution /helloworld - GET - Integration Response

First, declare response types using Method Response. Then, map the possible responses from the backend to this method's response types.

Lambda Error Regex	Method response status	Output model	Default mapping
-	200		Yes

Map the output from your Lambda function to the headers and output model of the 200 method response.

Lambda Error Regex

Content handling

Header Mappings

Mapping Templates

Content-Type
application/json

[Add mapping template](#)

[Add integration response](#)

Cancel Save

**Add a mapping template to take the entire response body and assign this to the JSON message element; the resulting response should be like:**

```
{ "message" : "The-actual-response-from-your-lambda-should-go-here..." }
```

**Test the GET method again, providing a proper query parameter (and possibly some other query parameters than only name):**



Resources

Actions ▾

▼ /

▼ /helloworld

GET

← Method Execution

/helloworld - GET - Method Test

Make a test call to your method with the provided input

**Path**

No path parameters exist for this resource. You can define path parameters by using the syntax **{myPathParam}** in a resource path.

**Query Strings**

**{helloworld}**

name=Lionel Richie

**Headers**

**{helloworld}**

Use a colon (:) to separate header name and value, and new lines to declare multiple headers. eg.  
Accept:application/json.

**Stage Variables**

No [stage variables](#) exist for this method.

**Request Body**

Request Body is not supported for GET methods.

⚡ Test

**Inspect the response message to verify the API is now properly responding JSON (response content + header)!**

```
{
  "message": "Hello Lionel Richie!"
}
```

#### Response Headers

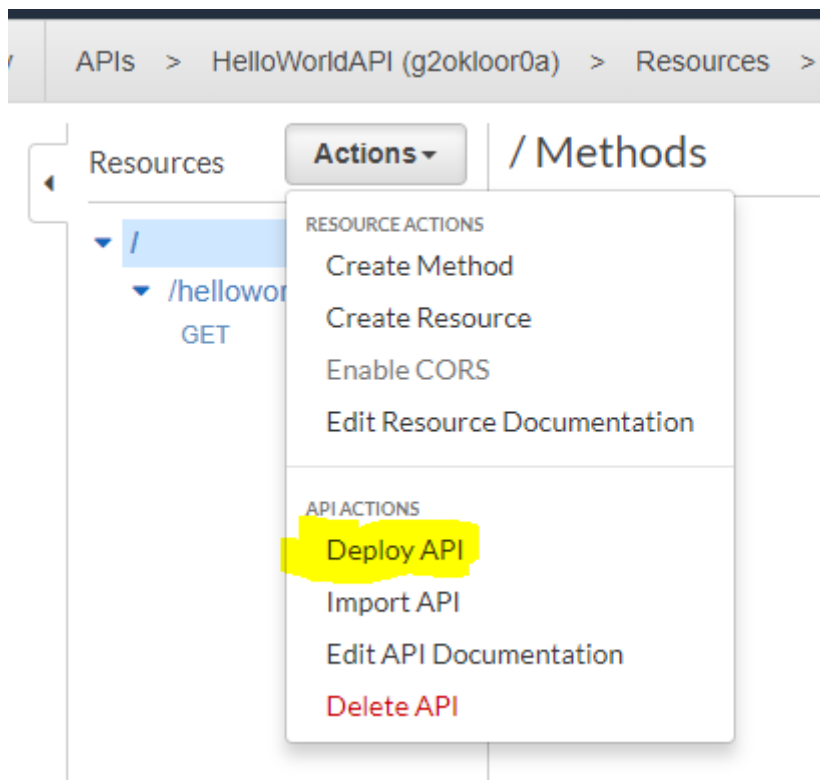
```
{"X-Amzn-Trace-Id": "Root=1-5b4a5ba8-7adcdf64f418131abe48e20;Sampled=0", "Content-Type": "application/json"}
```

**Check the Logs for the steps executed upon receiving the request!**

Making the API accessible

**The final step in the process is actually deploying the API to the world:**





**Specify a stage** (becomes a *path step* in the URL) for your API and **perform the deploy**:

Deploy API

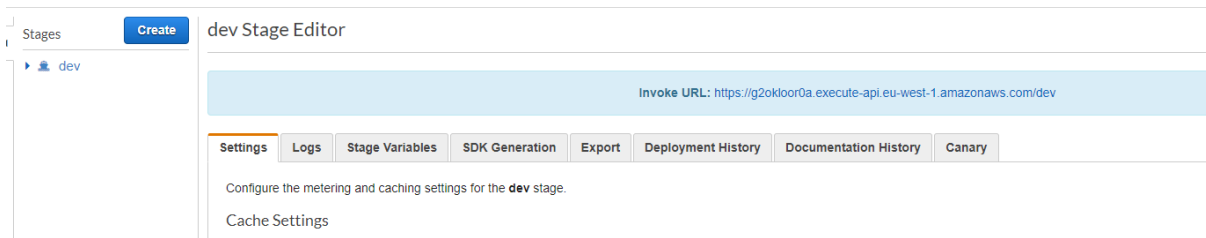
Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.

Deployment stage	<div>[New Stage]</div>
Stage name*	<div>dev</div>
Stage description	<div>Development</div>
Deployment description	<div>Vooruit met de geit!</div>

Cancel

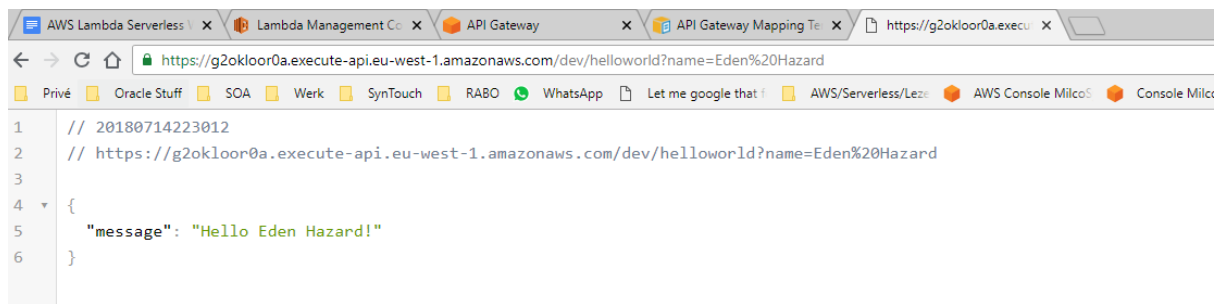
Deploy

**Examine the confirmation:**



As you can see, here the stage “dev” is exposed at “<https://g2okloor0a.execute-api.eu-west-1.amazonaws.com/dev>”. The **resource** for our API is helloworld and the GET request requires a name parameter, so you can test this api by going to <https://g2okloor0a.execute-api.eu-west-1.amazonaws.com/dev/helloworld?name=Eden%20Hazard>.

**Test your own API using a GET request from a web browser to verify it is reachable and works as expected:**



(The rendering in the screenshot above was down by my Chrome browser plugin “JSON Viewer”).

**Well done!**