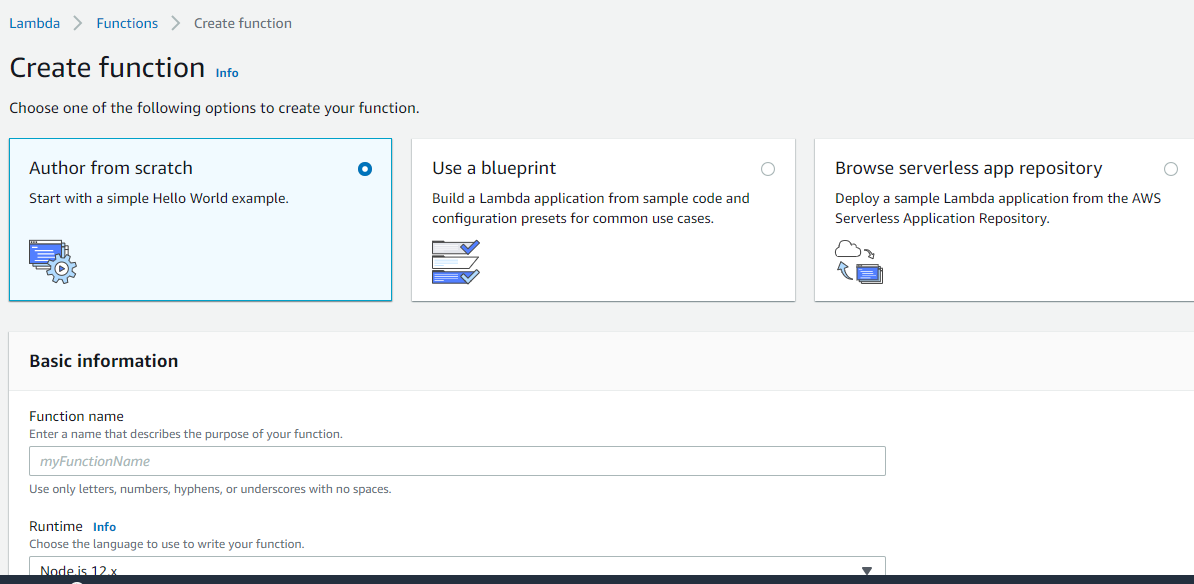
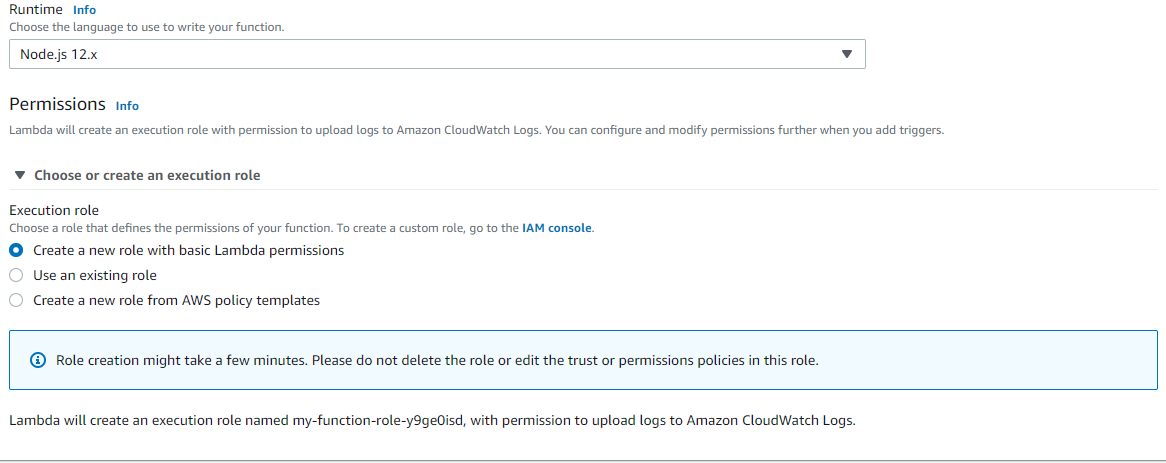
Create a Lambda Function with the Console

In this Getting Started exercise you create a Lambda function using the AWS Lambda console. Next, you manually invoke the Lambda function using sample event data. AWS Lambda executes the Lambda function and returns results. You then verify execution results, including the logs that your Lambda function created and various CloudWatch metrics.

**To create a Lambda function:**

1. **Open** <https://us-east-2.console.aws.amazon.com/lambda/home?region=us-east-2#/functions>
2. Choose **Create a function**.

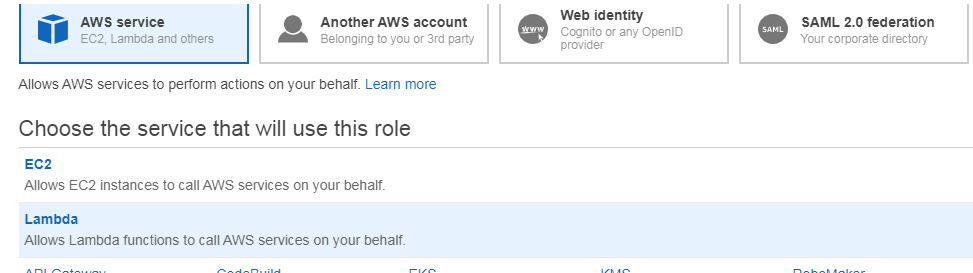




*Lambda creates a Node.js function and an execution role that grants the function permission to upload logs. Lambda assumes the execution role when you invoke your function, and uses it to create credentials for the AWS SDK and to read data from event sources.*

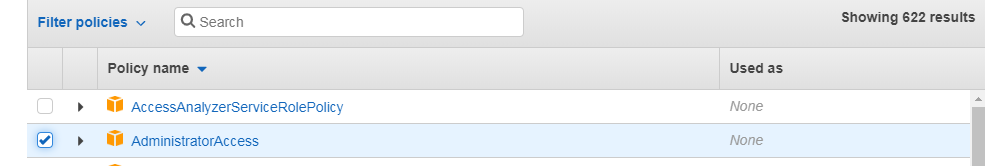
***Choose or create an execution role:***

1. Go to <https://console.aws.amazon.com/iam/home?#/roles$new?step=type>
2. Selecting Service which will be permitted:

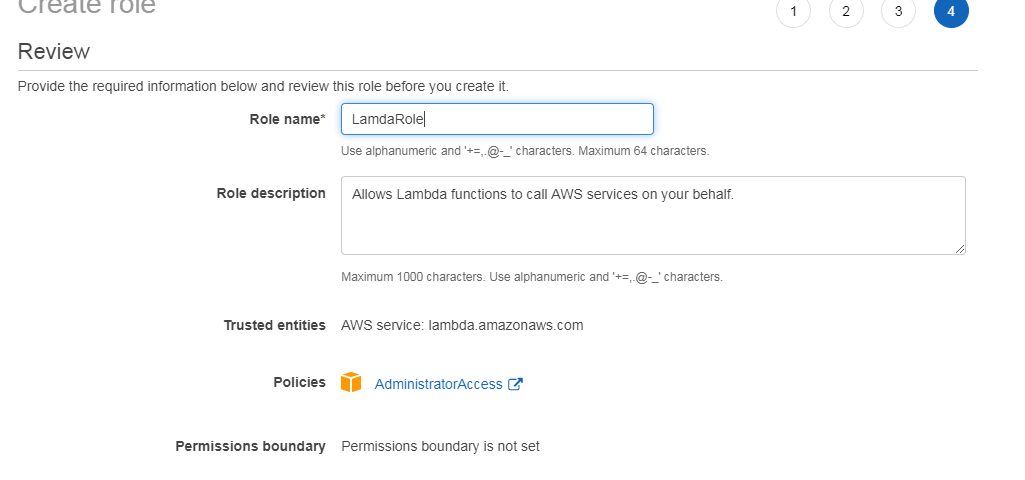


Here Lambda is the service

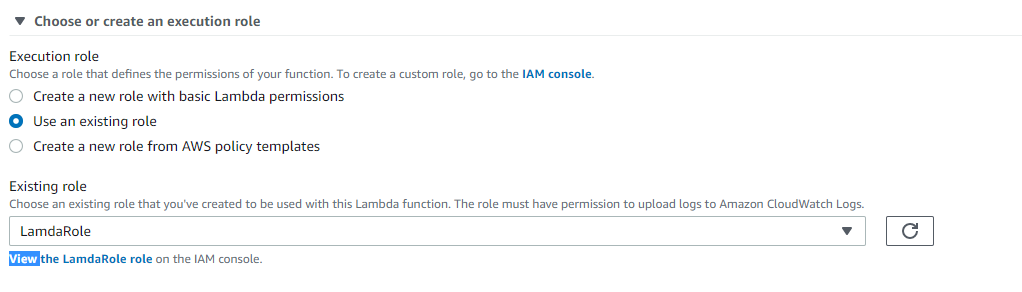
1. Selecting existing Policy “**AdministratorAccess**” .



1. click on **Next:Tag**
2. click on **Next:Review**
3. **Create Role**

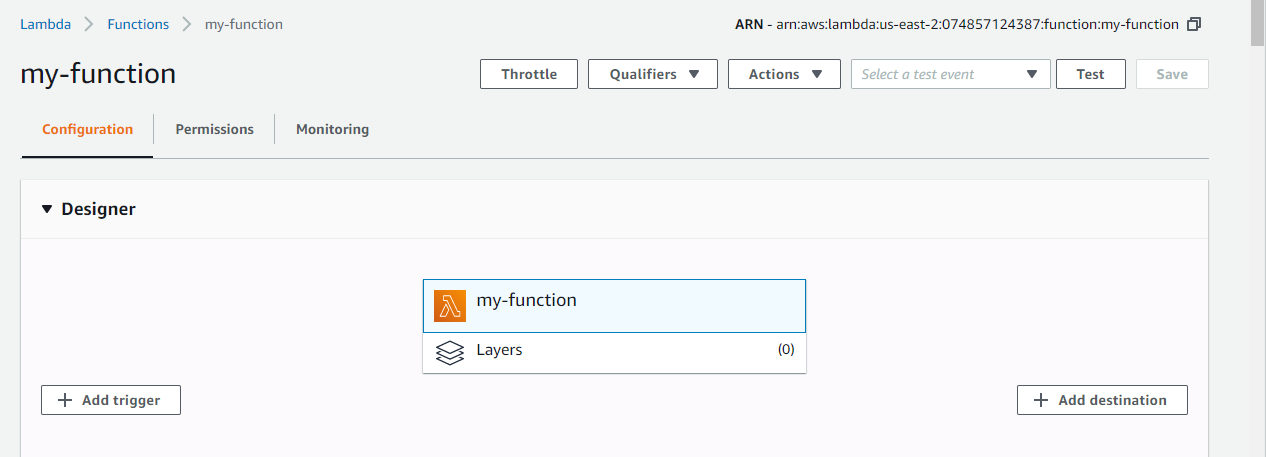


1. Selecting existing role.



***Selecting existing role here “LambdaRole”***

1. After clicking **Create function** below will be details:



1. Default Code :

exports.handler = async (event) => {

// TODO implement

const response = {

statusCode: 200,

body: JSON.stringify('Hello from Lambda!'),

};

return response;

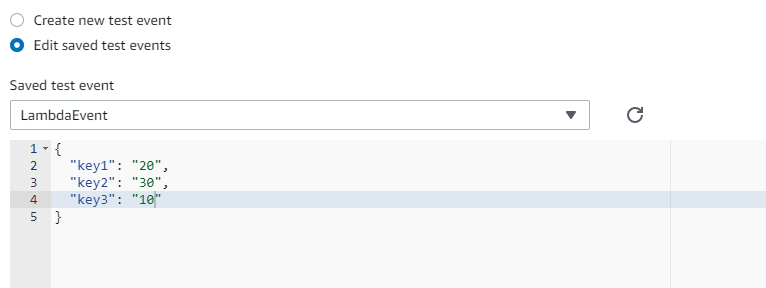
};

## **Invoke the Lambda Function**

Invoke your Lambda function using the sample event data provided in the console.

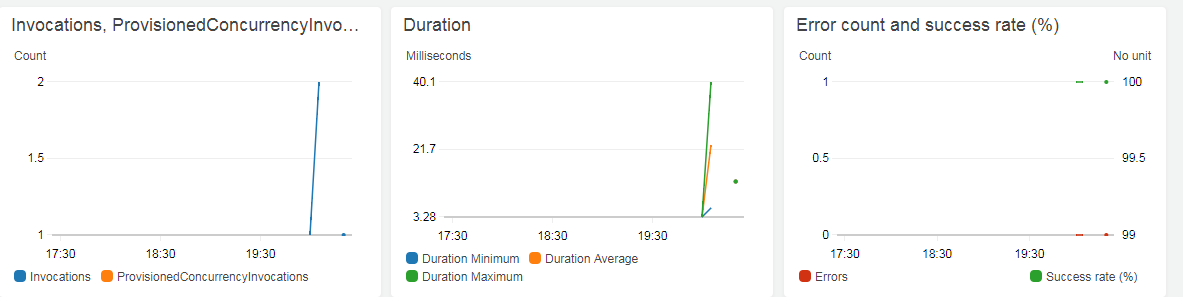
**To invoke a function**

1. In the upper right corner, choose **Test**.
2. In the **Configure test event** page, choose **Create new test event** and in **Event template**, leave the default **Hello World** option. Enter an **Event name** and note the following sample event template:



1. Click on test to execute the event
2. Monitoring event in graph through **cloudwatch**:

After clicking Monitoring tab:



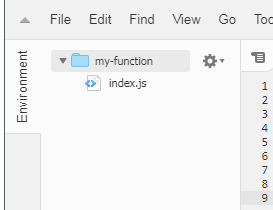
# Creating Functions Using the AWS Lambda Console Editor

The code editor in the AWS Lambda console enables you to write, test, and view the execution results of your Lambda function code.

The code editor includes the *menu bar*, *windows*, and the *editor pane*.

## **Working with Files and Folders**

You can use the **Environment** window in the code editor to create, open, and manage files for your function.



**To show or hide the Environment window**, choose the **Environment** button. If the **Environment** button is not visible, choose **Window, Environment** on the menu bar.

# AWS Lambda Features

AWS Lambda provides a management console and API for managing and invoking functions. It provides runtimes that support a standard set of features so that you can easily switch between languages and frameworks, depending on your needs. In addition to functions, you can also create versions, aliases, layers, and custom runtimes.

## **Programming Model**

Authoring specifics vary between runtimes, but all runtimes share a common programming model that defines the interface between your code and the runtime code. You tell the runtime which method to run by defining a **handler** in the function configuration, and the runtime runs that method. The runtime passes in objects to the handler that contain the invocation **event** and the **context**, such as the function name and request ID.

If your function exits without error, the runtime sends it another event. The function's class stays in memory, so clients and variables that are declared outside of the handler method in **initialization code** can be reused. Your function also has access to local storage in the /tmp directory. Instances of your function that are serving requests remain active for a few hours before being recycled.

The runtime captures **logging** output from your function and sends it to Amazon CloudWatch Logs. You can use the standard logging functionality of your programming language. If your function throws an **error**, the runtime returns that error to the client.

# Using AWS Lambda with Amazon API Gateway

*You can invoke AWS Lambda functions over HTTPS. You can do this by defining a custom REST API and endpoint using*[*Amazon API Gateway*](https://aws.amazon.com/api-gateway/)*, and then mapping individual methods, such as GET and PUT, to specific Lambda functions. Alternatively, you could add a special method named ANY to map all supported methods (****GET****,****POST****,****PATCH****,****DELETE****) to your Lambda function. When you send an HTTPS request to the API endpoint, the Amazon API Gateway service invokes the corresponding Lambda function.*

*Amazon API Gateway invokes your function*[*synchronously*](https://docs.aws.amazon.com/lambda/latest/dg/invocation-sync.html)*with an event that contains details about the HTTP request that it received.*

**Amazon API Gateway also adds a layer between your application users and your app logic that enables the following:**

* Ability to throttle individual users or requests.
* Protect against Distributed Denial of Service attacks.
* Provide a caching layer to cache response from your Lambda function.

# Tutorial: Using AWS Lambda with Amazon API Gateway

In this example you create a simple API using Amazon API Gateway. An Amazon API Gateway is a collection of resources and methods. For this tutorial, you create one resource (DynamoDBManager) and define one method (POST) on it. The method is backed by a Lambda function (LambdaFunctionOverHttps). That is, when you call the API through an HTTPS endpoint, Amazon API Gateway invokes the Lambda function.

The **POST** method on the **DynamoDBManager** resource supports the following DynamoDB operations:

* Create, update, and delete an item.
* Read an item.
* Scan an item.
* Other operations (echo, ping), not related to DynamoDB, that you can use for testing.

**The request payload you send in the POST request identifies the DynamoDB operation and provides necessary data.**

The following is a sample request payload for a DynamoDB

**Create Item** **operation**:

{

"operation": "create",

"tableName": "lambda-apigateway",

"payload": {

"Item": {

"id": "1",

"name": "Bob"

}

}

}

The following is a sample request payload for a DynamoDB

**Read Item Operation:**

{

"operation": "read",

"tableName": "lambda-apigateway",

"payload": {

"Key": {

"id": "1"

}

}

}

The following is a sample request payload for an **echo** operation. You send an HTTP POST request to the endpoint, using the following data in the request body.

{

"operation": "echo",

"payload": {

"somekey1": "somevalue1",

"somekey2": "somevalue2"

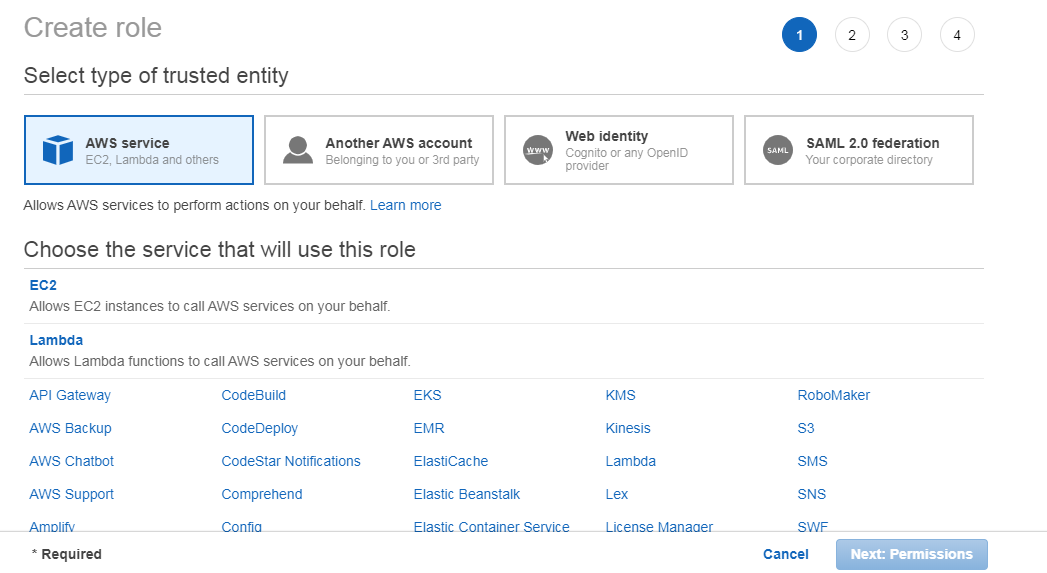
}

}

**Create the Execution Role:**

Create the [execution role](https://docs.aws.amazon.com/lambda/latest/dg/lambda-intro-execution-role.html) that gives your function permission to access AWS resources.

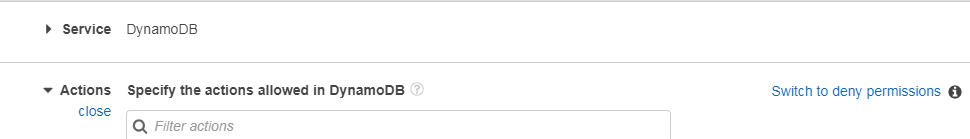
1. Go to <https://console.aws.amazon.com/iam/home?#/roles>
2. Choose **Create role**.



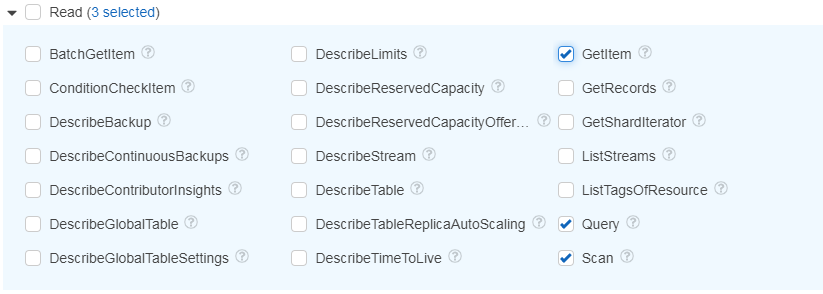
1. Create a role with the following properties.

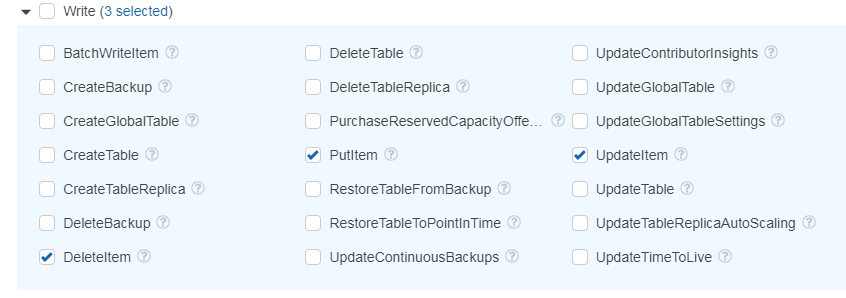
* **Trusted entity** – Lambda.
* **Role name** – **lambda-apigateway-role**.
* **Permissions** – Custom policy with permission to DynamoDB and CloudWatch Logs.
* Select Lambda as Service and click on **Next:Permissions**
* Creating Custom Policy after clicking on **Create Policy:**

**Selecting Dynamo DB access with specific details:**

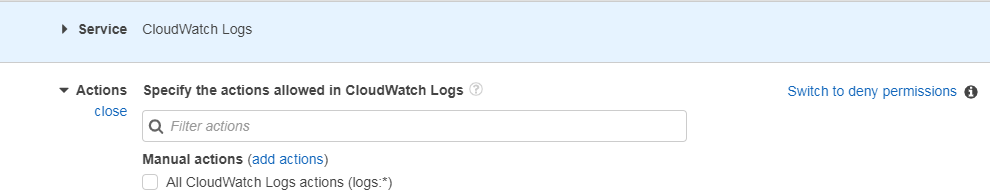


**After expanding:**

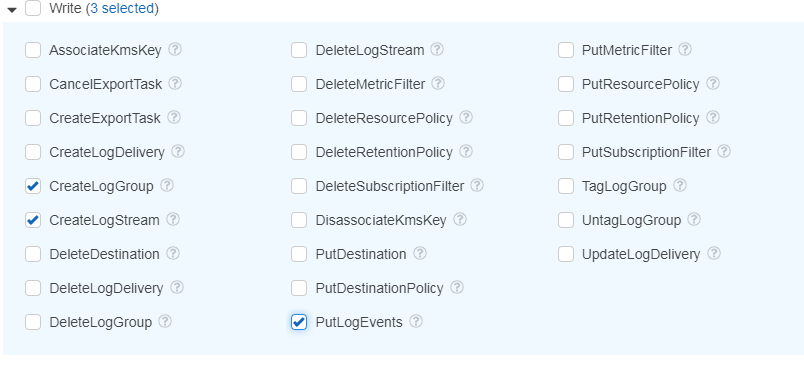




**Selecting CloudWatch Logs access:**



**After Expanding:**



**JSON Format of Custom Policy:**

{

"Version": "2012-10-17",

"Statement": [

{

"Sid": "VisualEditor0",

"Effect": "Allow",

"Action": [

"logs:CreateLogStream",

"dynamodb:PutItem",

"dynamodb:DeleteItem",

"dynamodb:GetItem",

"dynamodb:Scan",

"dynamodb:Query",

"dynamodb:UpdateItem",

"logs:CreateLogGroup",

"logs:PutLogEvents"

],

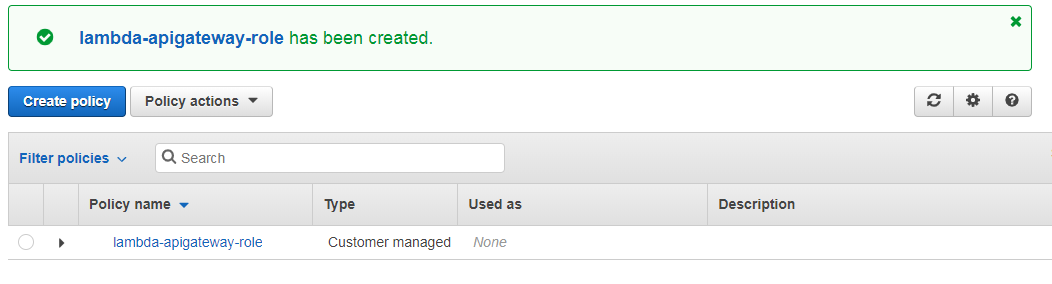
"Resource": "\*"

}

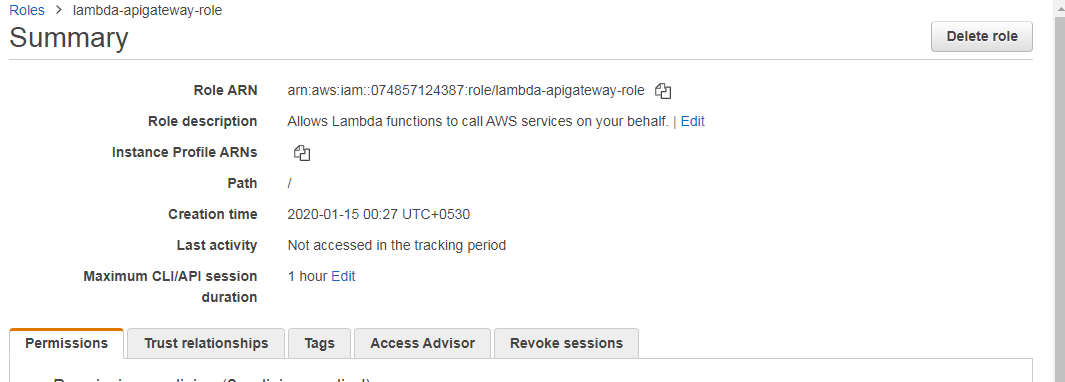
]

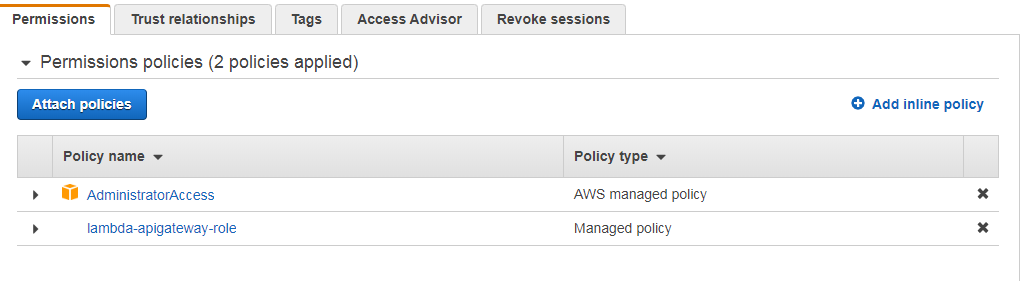
}

**After Creating Custom Policy:**



**After Creating Role with Custom Policy:**





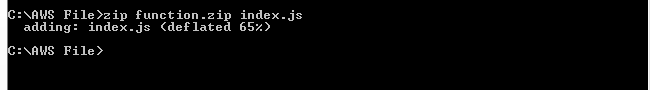
**To create the function**

1. Copy the sample code into a file named index.js.



1. Create a deployment package.

**Command**: *zip function.zip index.js*



1. Create a Lambda function with the create-function command.

**Command:**

*aws* *lambda* *create-function* --function-name **LambdaFunctionOverHttps** --zip-file **fileb://C:\AWSFile\function.zip** --handler **index.handler** --runtime nodejs12.x --role **arn:aws:iam::074857124387:role/lambda-apigateway-role**

**Role ARN :** arn:aws:iam::074857124387:role/lambda-apigateway-role

**Output:**

*{*

*"FunctionName": "LambdaFunctionOverHttps",*

*"FunctionArn": "arn:aws:lambda:us-west-2:074857124387:function:LambdaFunctionOve*

*rHttps",*

*"Runtime": "nodejs12.x",*

*"Role": "arn:aws:iam::074857124387:role/lambda-apigateway-role",*

*"Handler": "index.handler",*

*"CodeSize": 611,*

*"Description": "",*

*"Timeout": 3,*

*"MemorySize": 128,*

*"LastModified": "2020-01-14T19:00:44.315+0000",*

*"CodeSha256": "uqNjoz8QDyo+Wy9lOc0QWzxxmCs6IJ+Ebp2m/MUgWKM=",*

*"Version": "$LATEST",*

*"TracingConfig": {*

*"Mode": "PassThrough"*

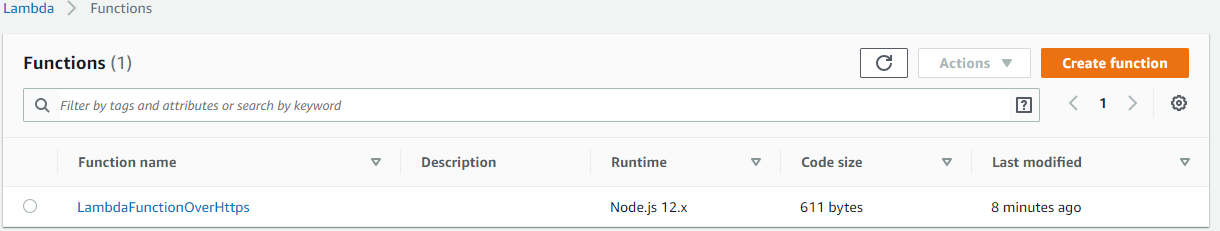
*},*

*"RevisionId": "ae78c0eb-216a-4502-a6f0-3f43412b628b"*

*}*

**Result of GUI:**

***Make sure to select proper region which default mention here: us-west-2***



1. Execute the following invoke command:

**Command:**

*aws lambda invoke --function-name LambdaFunctionOverHttps --payload fileb://C:\AWSFile\input.txt outputfile.txt*

***Test Data:***

******

**Output:**

{

"StatusCode": 200,

"ExecutedVersion": "$LATEST"

}

**Create an API Using Amazon API Gateway**

In this step, you associate your Lambda function with a method in the API that you created using Amazon API Gateway and test the end-to-end experience. That is, when an HTTP request is sent to an API method, Amazon API Gateway invokes your Lambda function.

First, you create an API (DynamoDBOperations) using Amazon API Gateway with one resource (DynamoDBManager) and one method (POST). You associate the POST method with your Lambda function. Then, you test the end-to-end experience.

**Create the API:**

Run the following create-rest-api command to create the DynamoDBOperations API for this tutorial.

**Command:**

*aws apigateway create-rest-api --name DynamoDBOperations*

**Output:**

*{*

*"id": "****z1bi9yi1s6****",*

*"name": "DynamoDBOperations",*

*"createdDate": 1579107676,*

*"apiKeySource": "HEADER",*

*"endpointConfiguration": {*

*"types": [*

*"EDGE"*

*]*

*}*

*}*

Save the API ID=*"****z1bi9yi1s6****",* for use in further commands. You also need the ID of the API root resource. To get the ID, run the get-resources command.

**Command:**

*aws apigateway get-resources --rest-api-id z1bi9yi1s6*

**Output:**

*{*

*"items": [*

*{*

*"id": "****8eif65luyg****",*

*"path": "/"*

*}*

*]*

*}*

At this time you only have the root resource, but you add more resources in the next step.

***Create a Resource in the API***

Run the following create-resource command to create a resource (DynamoDBManager) in the API that you created in the preceding section.

Parent ID: ***8eif65luyg***

**Command:**

*aws apigateway create-resource --rest-api-id z1bi9yi1s6 --path-part DynamoDBManager --parent-id 8eif65luyg*

**Output:**

{

"id": "**6felrn**",

"parentId": "8eif65luyg",

"pathPart": "DynamoDBManager",

"path": "/DynamoDBManager"

}

***Create POST Method on the Resource***

Run the following put-method command to create a POST method on the DynamoDBManager resource in your API.

**Command:**

*aws apigateway put-method --rest-api-id z1bi9yi1s6 --resource-id 6felrn --http-method POST --authorization-type NONE*

**Output:**

{

"httpMethod": "POST",

"authorizationType": "NONE",

"apiKeyRequired": false

}

### Set the Lambda Function as the Destination for the POST Method

Run the following command to set the Lambda function as the integration point for the POST method. This is the method Amazon API Gateway invokes when you make an HTTP request for the POST method endpoint. This command and others use ARNs that include your account ID and region. Save these to variables (you can find your account ID in the role ARN that you used to create the function).

**Command:**

*aws apigateway put-integration --rest-api-id z1bi9yi1s6 --resource-id 6felrn --http-method POST --type AWS --integration-http-method POST --uri arn:aws:apigateway:us-west-2:lambda:path/2015-03-31/functions/arn:aws:lambda:us-west-2:074857124387:function:LambdaFunctionOverHttps/invocations*

**Output:**

*{*

*"type": "AWS",*

*"httpMethod": "POST",*

*"uri": "arn:aws:apigateway:us-west-2:lambda:path/2015-03-31/functions/arn:aws:lambda:us-west-2:074857124387:function:LambdaFunctionOverHttps/invocations",*

*"passthroughBehavior": "WHEN\_NO\_MATCH",*

*"timeoutInMillis": 29000,*

*"cacheNamespace": "6felrn",*

*"cacheKeyParameters": []*

*}*

--integration-http-method is the method that API Gateway uses to communicate with AWS Lambda. --uri is unique identifier for the endpoint to which Amazon API Gateway can send request.

Set content-type of the POST method response and integration response to JSON as follows:

* Run the following command to set the POST method response to JSON. This is the response type that your API method returns.

**Command:**

aws apigateway put-method-response --rest-api-id z1bi9yi1s6 --resource-id 6felrn --http-method POST --status-code 200 --response-models application/json=Empty

**Output:**

*{*

*"statusCode": "200",*

*"responseModels": {*

*"application/json": "Empty"*

*}*

*}*

Run the following command to set the POST method integration response to JSON. This is the response type that Lambda function returns.

**Command:**

aws apigateway put-integration-response --rest-api-id z1bi9yi1s6 --resource-id 6felrn --http-method POST --status-code 200 --response-templates application/json=""

**Output:**

*{*

*"statusCode": "200",*

*"responseTemplates": {*

*"application/json": null*

*}*

*}*

***Deploy the API***

*In this step, you deploy the API that you created to a stage called prod.*

**Command:**

aws apigateway create-deployment --rest-api-id z1bi9yi1s6 --stage-name prod

**Output:**

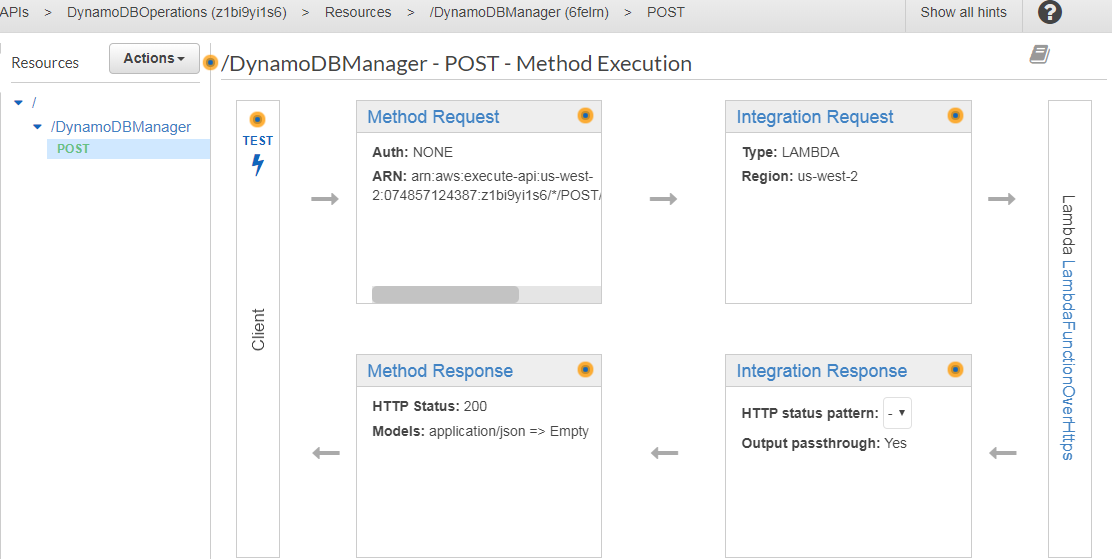
{

"id": "z8iaqg",

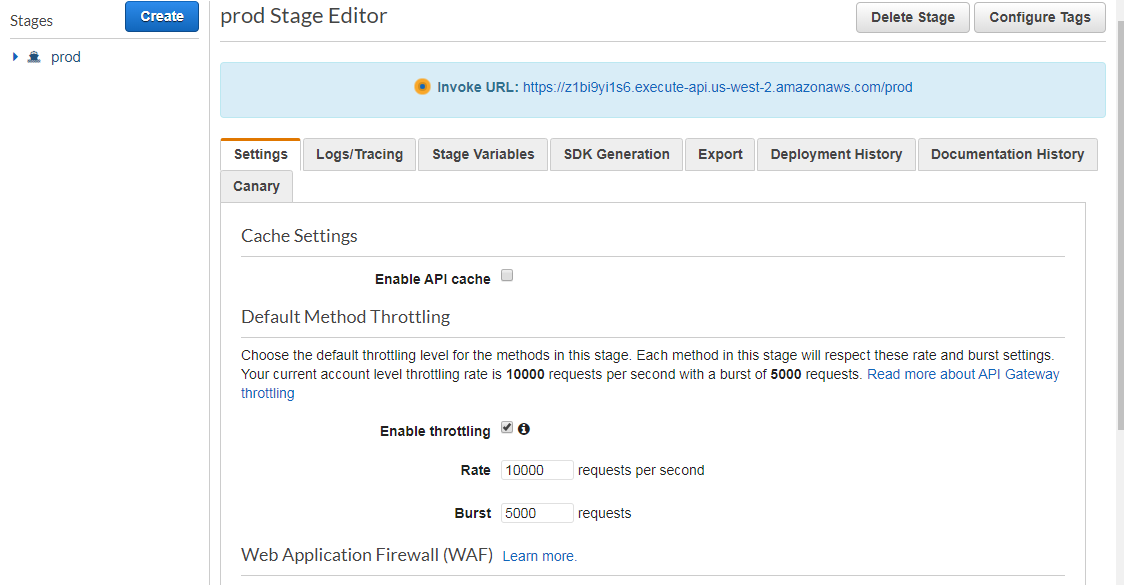
"createdDate": 1579289709

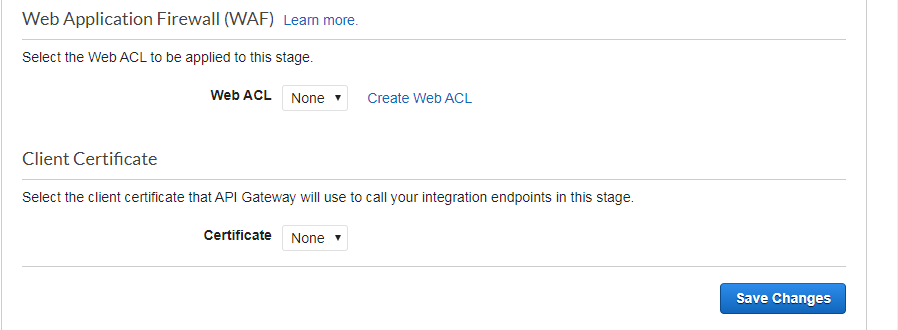
}

**GUI Details:**



**Deployment details:**





**Invoke URL:**

**Invoke URL:** <https://z1bi9yi1s6.execute-api.us-west-2.amazonaws.com/prod>

**Grant Invoke Permission to the API**

Now that you have an API created using Amazon API Gateway and you've deployed it, you can test. First, you need to add permissions so that Amazon API Gateway can invoke your Lambda function when you send HTTP request to the POST method.

To do this, you need to add a permission to the permissions policy associated with your Lambda function. Run the following add-permission AWS Lambda command to grant the Amazon API Gateway service principal (apigateway.amazonaws.com) permissions to invoke your Lambda function (LambdaFunctionOverHttps).

**Command:**

*aws lambda add-permission --function-name LambdaFunctionOverHttps --statement-id apigateway-test-2 --action lambda:InvokeFunction --principal apigateway.amazonaws.com --source-arn "arn:aws:execute-api:us-west-2:074857124387:z1bi9yi1s6/\*/POST/DynamoDBManager"*

**Output:**

*{*

*"Statement": "{\"Sid\":\"apigateway-test-2\",\"Effect\":\"Allow\",\"Principal\":{\"Service\":\"apigateway.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction\",\"Resource\":\"arn:aws:lambda:us-west-2:074857124387:function:LambdaFunctionOverHttps\",\"Condition\":{\"ArnLike\":{\"AWS:SourceArn\":\"arn:aws:execute-api:us-west-2:074857124387:z1bi9yi1s6/\*/POST/DynamoDBManager\"}}}"*

*}*

You must grant this permission to enable testing (if you go to the Amazon API Gateway and choose **Test** to test the API method, you need this permission). Note the --source-arn specifies a wildcard character (\*) as the stage value (indicates testing only). This allows you to test without deploying the API.

**Note**

If your function and API are in different regions, the region identifier in the source ARN must match the region of the function, not the region of the API.

Now, run the same command again, but this time you grant to your deployed API permissions to invoke the Lambda function.

**Command:**

*aws lambda add-permission --function-name LambdaFunctionOverHttps --statement-id apigateway-prod-2 --action lambda:InvokeFunction --principal apigateway.amazonaws.com --source-arn "arn:aws:execute-api:us-west-2:074857124387:z1bi9yi1s6/****prod****/POST/DynamoDBManager"*

**Output:**

*{*

*"Statement": "{\"Sid\":\"apigateway-prod-2\",\"Effect\":\"Allow\",\"Principal\":{\"Service\":\"apigateway.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction\",\"Resource\":\"arn:aws:lambda:us-west-2:074857124387:function:LambdaFunctionOverHttps\",\"Condition\":{\"ArnLike\":{\"AWS:SourceArn\":\"arn:aws:execute-api:us-west-2:074857124387:z1bi9yi1s6/prod/POST/DynamoDBManager\"}}}"*

*}*

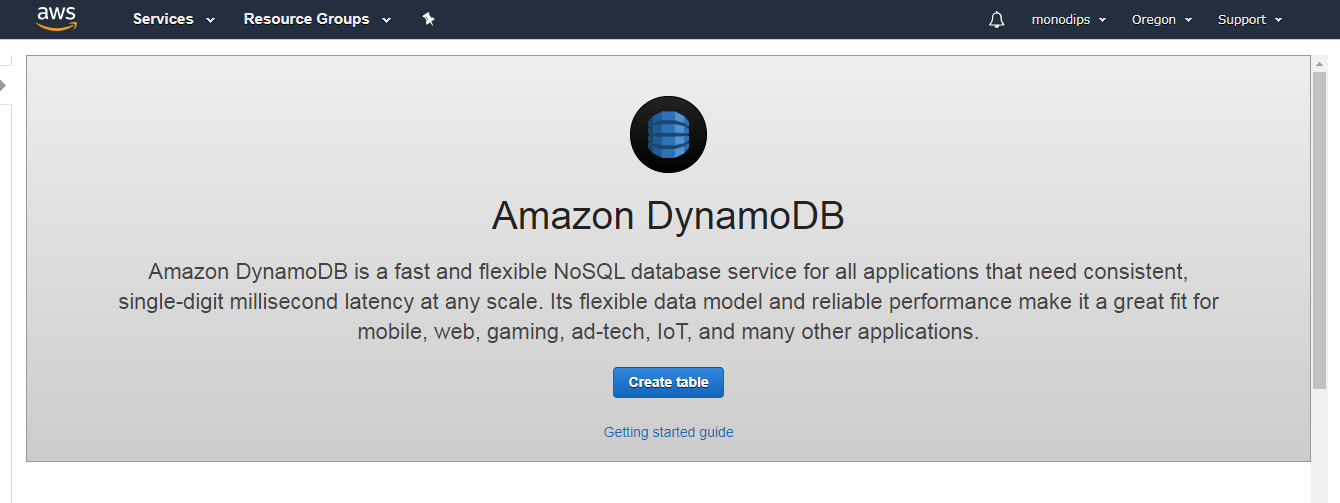
You grant this permission so that your deployed API has permissions to invoke the Lambda function. Note that the --source-arn specifies a prod which is the stage name we used when deploying the API.

**Create a Amazon DynamoDB Table**

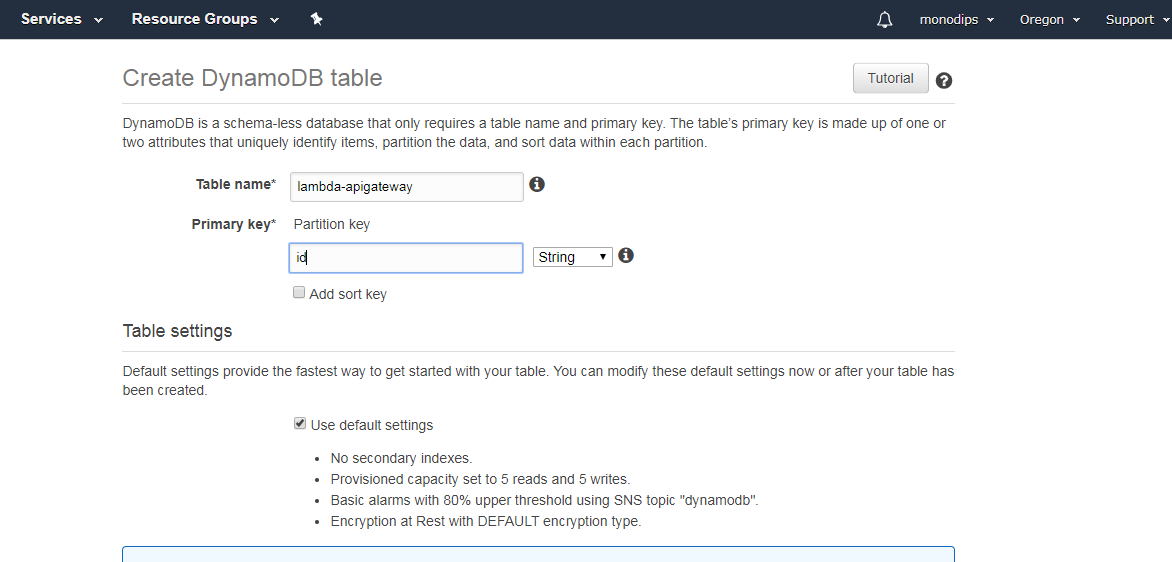
Create the DynamoDB table that the Lambda function uses.

**To create a DynamoDB table**

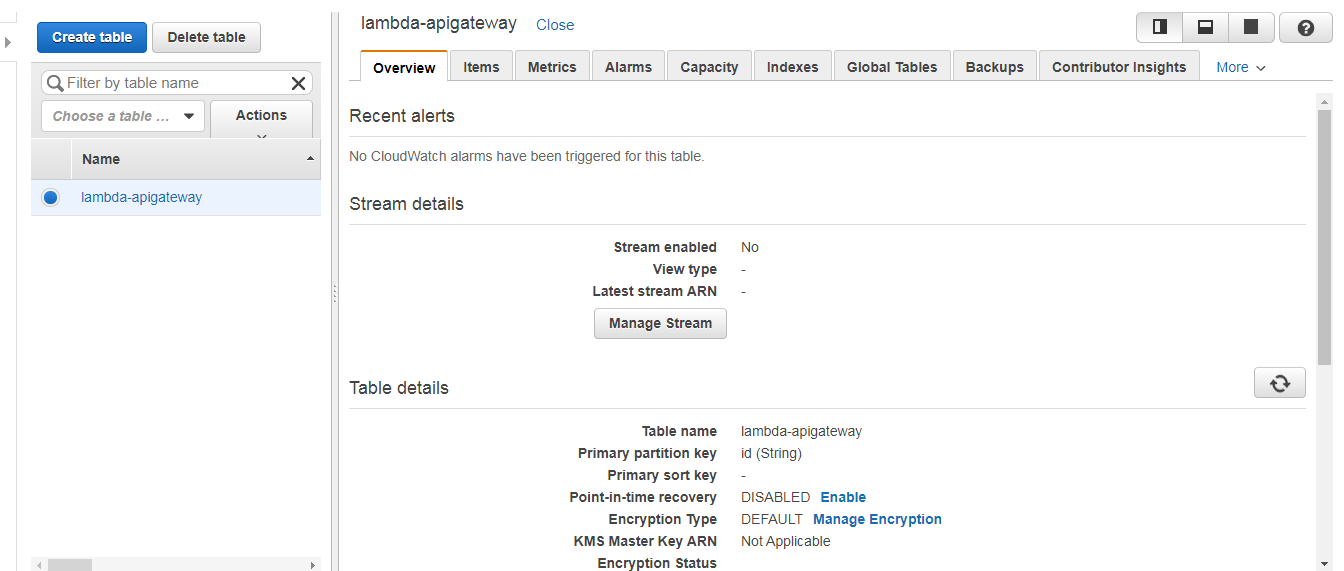
1. Open the [DynamoDB console](https://console.aws.amazon.com/dynamodb).



1. Choose **Create table**.
2. Create a table with the following settings.
   * **Table name** – **lambda-apigateway**
   * **Primary key** – **id** (string)



1. Choose **Create**.



## **Trigger the Function with an HTTP Request**

In this step, you are ready to send an HTTP request to the POST method endpoint. You can use either Curl or a method (test-invoke-method) provided by Amazon API Gateway.

You can use Amazon API Gateway CLI commands to send an HTTP POST request to the resource (DynamoDBManager) endpoint. Because you deployed your Amazon API Gateway, you can use Curl to invoke the methods for the same operation.

The Lambda function supports using the create operation to create an item in your DynamoDB table. To request this operation, use the following JSON:

**Example create-item.json**

{

"operation": "create",

"tableName": "lambda-apigateway",

"payload": {

"Item": {

"id": "1234ABCD",

"number": 5

}

}

}

Save the test input to a file named create-item.json. Run the test-invoke-method Amazon API Gateway command to send an HTTP POST method request to the resource (DynamoDBManager) endpoint.

**Command For TEST:**

aws apigateway test-invoke-method --rest-api-id z1bi9yi1s6 --resource-id 6felrn --http-method POST --path-with-query-string "" --body file://C:\AWSFile\create-item.json

**Output:**

*{*

*"status": 200,*

*"body": "{}",*

*"headers": {*

*"Content-Type": "application/json",*

*"X-Amzn-Trace-Id": "Root=1-5e220d4a-e2045bae94e9201837c60588;Sampled=0"*

*},*

*"log": "Execution log for request e0581ccf-6b08-4603-9572-c219bba51040\nFri Jan 17 19:38:50 UTC 2020 : Starting execution for request: e0581ccf-6b08-4603-9572-c219bba51040\nFri Jan 17 19:38:50 UTC 2020 : HTTP Method: POST, Resource Path: /DynamoDBManager\nFri Jan 17 19:38:50 UTC 2020 : Method request path: {}\nFri Jan 17 19:38:50 UTC 2020 : Method request query string: {}\nFri Jan 17 19:38:50 UTC 2020 : Method request headers: {}\nFri Jan 17 19:38:50 UTC 2020 : Method request body before transformations: {\n \"operation\": \"create\",\n \"tableName\": \"lambda-apigateway\",\n \"payload\": {\n \"Item\": {\n \"id\": \"1234ABCD\",\n \"number\": 5\n }\n }\n}\nFri Jan 17 19:38:50 UTC 2020 : Endpoint request URI: https://lambda.us-west-2.amazonaws.com/2015-03-31/functions/arn:aws:lambda:us-west-2:074857124387:function:LambdaFunctionOverHttps/invocations\nFri Jan 17 19:38:50 UTC 2020 : Endpoint request headers: {x-amzn-lambda-integration-tag=e0581ccf-6b08-4603-9572-c219bba51040, Authorization=\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*cdca30, X-Amz-Date=20200117T193850Z, x-amzn-apigateway-api-id=z1bi9yi1s6, X-Amz-Source-Arn=arn:aws:execute-api:us-west-2:074857124387:z1bi9yi1s6/test-invoke-stage/POST/DynamoDBManager, Accept=application/json, User-Agent=AmazonAPIGateway\_z1bi9yi1s6, X-Amz-Security-Token=IQoJb3JpZ2luX2VjECMaCXVzLXdlc3QtMiJIMEYCIQCKupuzPbWt5ic00qxDdqNdB9cokzdY4WPVsDDx3QcFZwIhAKvsr8FPhZcpqcV9Bj5MYQLutSrrVcwlB+C/3N3ci0egKr0DCKz//////////wEQARoMMTA5MzUxMzA5NDA3Igyq6Sx8Pu4HlFDQDwkqkQNtSjTwax/u5KmEdyVWW7KoRgOnwZhPMxcHeUuPqP25kGhm6+M+yO97Sh2aHDLstB1fVrSW77F2gwAp6JJA3ninkER2ov0+thGwfI/ftxfXKbpMnWusX18Hg5tbMazGD+5XftqFxKXV [TRUNCATED]\nFri Jan 17 19:38:50 UTC 2020 : Endpoint request body after transformations: {\n \"operation\": \"create\",\n \"tableName\": \"lambda-apigateway\",\n \"payload\": {\n \"Item\": {\n \"id\": \"1234ABCD\",\n \"number\": 5\n }\n }\n}\nFri Jan 17 19:38:50 UTC 2020 : Sending request to https://lambda.us-west-2.amazonaws.com/2015-03-31/functions/arn:aws:lambda:us-west-2:074857124387:function:LambdaFunctionOverHttps/invocations\nFri Jan 17 19:38:51 UTC 2020 : Received response. Status: 200, Integration latency: 1129 ms\nFri Jan 17 19:38:51 UTC 2020 : Endpoint response headers: {Date=Fri, 17 Jan 2020 19:38:51 GMT, Content-Type=application/json, Content-Length=2, Connection=keep-alive, x-amzn-RequestId=c490df88-4e80-47eb-9451-b0d99526ecfe, x-amzn-Remapped-Content-Length=0, X-Amz-Executed-Version=$LATEST, X-Amzn-Trace-Id=root=1-5e220d4a-e2045bae94e9201837c60588;sampled=0}\nFri Jan 17 19:38:51 UTC 2020 : Endpoint response body before transformations: {}\nFri Jan 17 19:*

*38:51 UTC 2020 : Method response body after transformations: {}\nFri Jan 17 19:38:51 UTC 2020 : Method response headers: {X-Amzn-Trace-Id=Root=1-5e220d4a-e2045bae94e9201837c60588;Sampled=0, Content-Type=application/json}\nFri Jan 17 19:38:51 UTC 2020 : Successfully completed execution\nFri Jan 17 19:38:51 UTC 2020 : Method completed with status: 200\n",*

*"latency": 1131*

*}*

**Command for Prod:**

curl -X POST -d "{\"operation\":\"create\",\"tableName\":\"lambda-apigateway\",\"payload\":{\"Item\":{\"id\":\"1\",\"name\":\"Bob\"}}}" https://z1bi9yi1s6.execute-api.us-west-2.amazonaws.com/prod/DynamoDBManager

To send request for the echo operation that your Lambda function supports, you can use the following request payload:

**Example echo.json**

{

"operation": "echo",

"payload": {

"somekey1": "somevalue1",

"somekey2": "somevalue2"

}

}

Save the test input to a file named echo.json. Run the test-invoke-method Amazon API Gateway CLI command to send an HTTP POST method request to the resource (DynamoDBManager) endpoint using the preceding JSON in the request body.

**Command of echo For TEST:**

aws apigateway test-invoke-method --rest-api-id z1bi9yi1s6 --resource-id 6felrn --http-method POST --path-with-query-string "" --body file://C:\AWSFile\echo.json

**Output:**

*{*

*"status": 200,*

*"body": "\"Success\"",*

*"headers": {*

*"Content-Type": "application/json",*

*"X-Amzn-Trace-Id": "Root=1-5e22114e-c03505d732d2da599880077f;Sampled=0"*

*},*

*"log": "Execution log for request 2f8cfec1-a0bf-4828-840a-f3845f21a450\nFri Jan 17 19:55:58 UTC 2020 : Starting execution for request: 2f8cfec1-a0bf-4828-840a-f3845f21a450\nFri Jan 17 19:55:58 UTC 2020 : HTTP Method: POST, Resource Path: /DynamoDBManager\nFri Jan 17 19:55:58 UTC 2020 : Method request path: {}\nFri Jan 17 19:55:58 UTC 2020 : Method request query string: {}\nFri Jan 17 19:55:58 UTC 2020 : Method request headers: {}\nFri Jan 17 19:55:58 UTC 2020 : Method request body before transformations: {\n \"operation\": \"echo\",\n \"payload\": {\n \"somekey1\": \"somevalue1\",\n \"somekey2\": \"somevalue2\"\n }\n}\nFri Jan 17 19:55:58 UTC 2020 : Endpoint request URI: https://lambda.us-west-2.amazonaws.com/2015-03-31/functions/arn:aws:lambda:us-west-2:074857124387:function:LambdaFunctionOverHttps/invocations\nFri Jan 17 19:55:58 UTC 2020 : Endpoint request headers: {x-amzn-lambda-integration-tag=2f8cfec1-a0bf-4828-840a-f3845f21a450, Authorization=\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*cf1f87, X-Amz-Date=20200117T195558Z, x-amzn-apigateway-api-id=z1bi9yi1s6, X-Amz-Source-Arn=arn:aws:execute-api:us-west-2:074857124387:z1bi9yi1s6/test-invoke-stage/POST/DynamoDBManager, Accept=application/json, User-Agent=AmazonAPIGateway\_z1bi9yi1s6, X-Amz-Security-Token=IQoJb3JpZ2luX2VjECMaCXVzLXdlc3QtMiJIMEYCIQCKupuzPbWt5ic00qxDdqNdB9cokzdY4WPVsDDx3QcFZwIhAKvsr8FPhZcpqcV9Bj5MYQLutSrrVcwlB+C/3N3ci0egKr0DCKz//////////wEQARoMMTA5MzUxMzA5NDA3Igyq6Sx8Pu4HlFDQDwkqkQNtSjTwax/u5KmEdyVWW7KoRgOnwZhPMxcHeUuPqP25kGhm6+M+yO97Sh2aHDLstB1fVrSW77F2gwAp6JJA3ninkER2ov0+thGwfI/ftxfXKbpMnWusX18Hg5tbMazGD+5XftqFxKXV [TRUNCATED]\nFri Jan 17 19:55:58 UTC 2020 : Endpoint request body after transformations: {\n \"operation\": \"echo\",\n \"payload\": {\n \"somekey1\": \"somevalue1\",\n \"somekey2\": \"somevalue2\"\n }\n}\nFri Jan 17 19:55:58 UTC 2020 : Sending request to https://lambda.us-west-2.amazonaws.com/2015-03-31/functions/arn:aws:lambda:us-west-2:074857124387:function:LambdaFunctionOverHttps/invocations\nFri Jan 17 19:55:58 UTC 2020 : Received response. Status: 200, Integration latency: 179 ms\nFri Jan 17 19:55:58 UTC 2020 : Endpoint response headers: {Date=Fri, 17 Jan 2020 19:55:58 GMT, Content-Type=application/json, Content-Length=9, Connection=keep-alive, x-amzn-RequestId=52644da2-907a-4c0d-ab24-bc668699cac2, x-amzn-Remapped-Content-Length=0, X-Amz-Executed-Version=$LATEST, X-Amzn-Trace-Id=root=1-5e22114e-c03505d732d2da599880077f;sampled=0}\nFri Jan 17 19:55:58 UTC 2020 : Endpoint response body before transformations: \"Success\"\nFri Jan 17 19:55:58 UTC 2020 : Method response body after transformations: \"Success\"\nFri Jan 17 19:55:58 UTC 2020 : Method response headers: {X-Amzn-Trace-*

*Id=Root=1-5e22114e-c03505d732d2da599880077f;Sampled=0, Content-Type=application/json}\nFri Jan 17 19:55:58 UTC 2020 : Successfully completed execution\nFri Jan 17 19:55:58 UTC 2020 : Method completed with status: 200\n",*

*"latency": 184*

*}*

**Command for echo in Prod stage:**

curl -X POST -d "{\"operation\":\"echo\",\"payload\":{\"somekey1\":\"somevalue1\",\"somekey2\":\"somevalue2\"}}" <https://z1bi9yi1s6.execute-api.us-west-2.amazonaws.com/prod/DynamoDBManager>

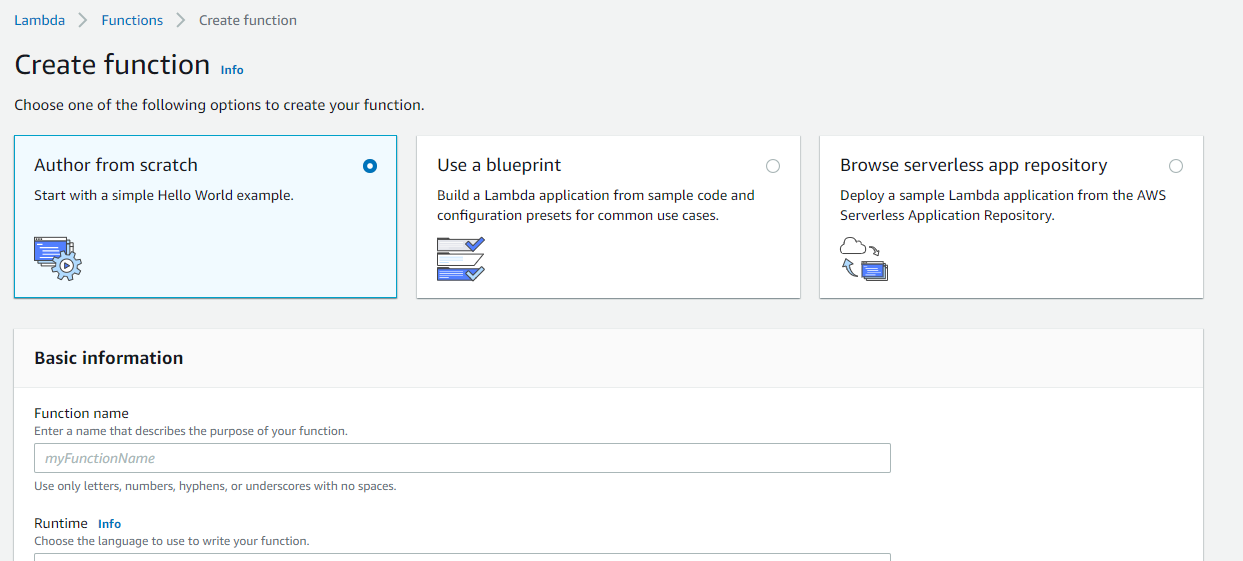
**Tutorial: Using AWS Lambda with Scheduled Events**

*In this tutorial, you do the following:*

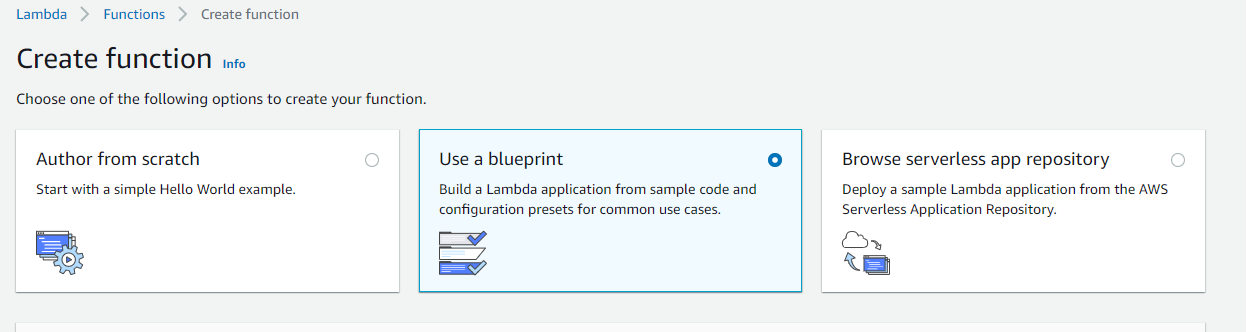
* *Create a Lambda function using the****lambda-canary****blueprint. You configure the Lambda function to run every minute. Note that if the function returns an error, AWS Lambda logs error metrics to* ***CloudWatch****.*
* *Configure a* ***CloudWatch*** *alarm on the Errors metric of your Lambda function to post a message to your Amazon SNS topic when AWS Lambda emits error metrics to* ***CloudWatch****. You subscribe to the Amazon SNS topics to get email notification. In this tutorial, you do the following to set this up:*
  + *Create an Amazon SNS topic.*
  + *Subscribe to the topic so you can get email notifications when a new message is posted to the topic.*
  + *In Amazon* ***CloudWatch****, set an alarm on the Errors metric of your Lambda function to publish a message to your SNS topic when errors occur.*

**Create a Lambda Function:**

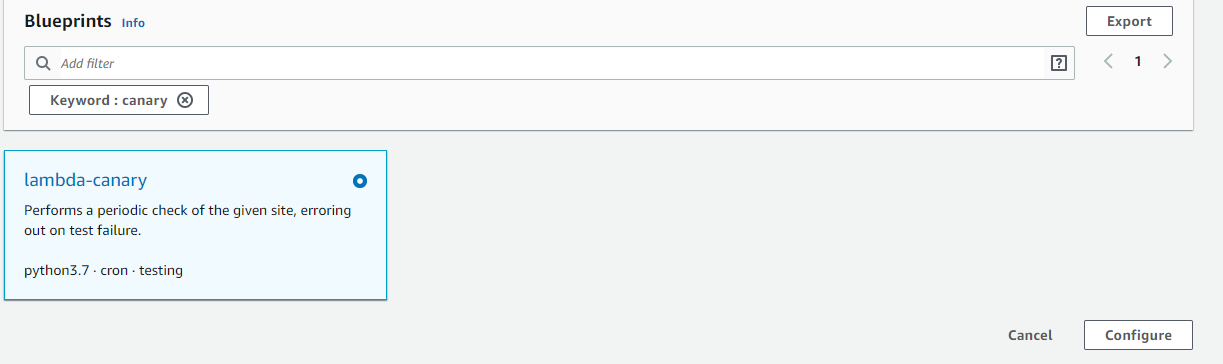
* + 1. Sign in to the AWS Management Console and open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
    2. Choose **Create function**.



* + 1. Choose **Blueprints**.



* + 1. enter **canary** in the search bar. Choose the **lambda-canary** blueprint and then choose **Configure**.



* + 1. Configure the following settings.
* **Name** – **lambda-canary**.
* **Role** – Create a new role with basic Lambda permissions
* **Role name** – **lambda-apigateway-role**.
* **Rule** – **Create a new rule**.
* **Rule name** – **CheckWebsiteScheduledEvent**.
* **Rule description** – **CheckWebsiteScheduledEvent trigger**.
* **Schedule expression** – **rate(1 minute)**.
* **Enabled** – True (checked).
* **Environment variables**
  + **site** – **https://docs.aws.amazon.com/lambda/latest/dg/welcome.html**
  + **expected** – **What Is AWS Lambda?**

