# Building Private REST API with AWS Lambda and Amazon API Gateway

Created by[in28Minutes Official](https://deloittedevelopment.udemy.com/user/in28minutes/)

**AWS Workspace**

**90-180 minutes**

In this lab, you will be an engineer at a company using AWS as their cloud platform. You need to build a REST API for Todo management. Your architecture team wants to try out a serverless approach. You are planning to use Amazon API Gateway, AWS Lambda and Amazon DynamoDB. Your boss wants you to build a proof-of-concept. You want to host the REST API privately until you are ready for a public release.

We are planning to build a REST API for our enterprise Todo management platform. Our architecture team recommends trying out a serverless approach to keep our operations and costs to a minimum. We will start with a proof of concept building a private REST API exposing the following operations on a todo resource - add, delete, update and retrieve. Here's the recommended starting architecture: REST API will be exposed using an Amazon API Gateway. Business logic will be written in AWS Lambda. Amazon DynamoDB will be the data store.

Your system will provide a REST API for managing todos. You would want to be able to add, delete, modify and retrieve todos. You want to build an auto scaling and highly resilient serverless REST API using Amazon API Gateway, AWS Lambda and Amazon DynamoDB.

**How you'll work**

Your project has been broken into a set of tasks. To complete these tasks, use the provided workspace. You can launch your workspace by clicking below or using the button in the top right of the screen.

Launch workspace

TASK

1-Create a hello world Lamda Function

2-Create a Private API Gateway

3-Integrate hello world Resource GET Method with Lamda Function

4-Create a DynamoDB to store todos

5-Create IAM Role for Lambda Functions

6-Create todos REST API to retrieve detail of specific todos(GET method)

7-Create Todo REST API to retrieve details of specific todo(GET method)

8-Add Resource to delete a todo(DELETE nethod)

9-ADD Resource Method to create a todo(POST method)

10-Add Resource Method to update a todo(PUT method)

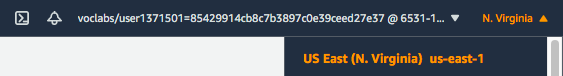
11-Deploy and test the REST API

12-Clean up resources

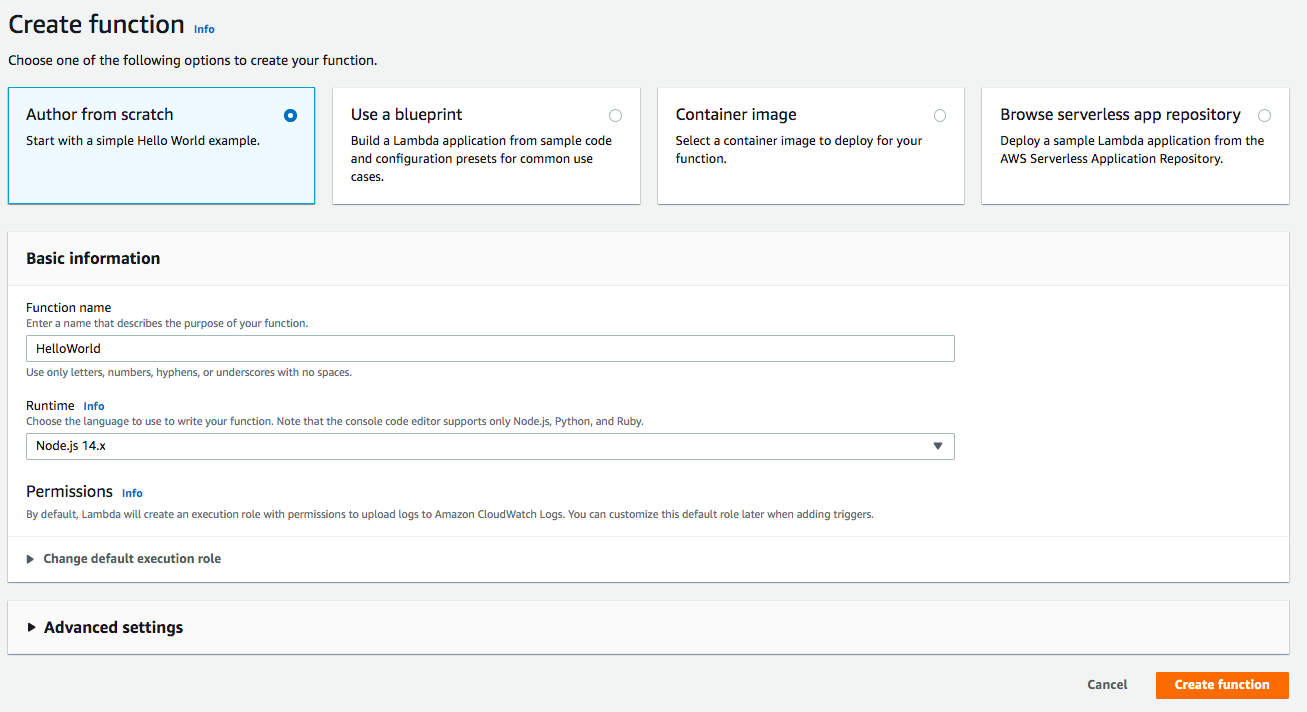
Create a hello world Lamda Function

Your task is to create a hello world Lambda Function returning a JSON response. Example JSON Response: *{"statusCode": 200, "body": "SOME\_JSON"}*.

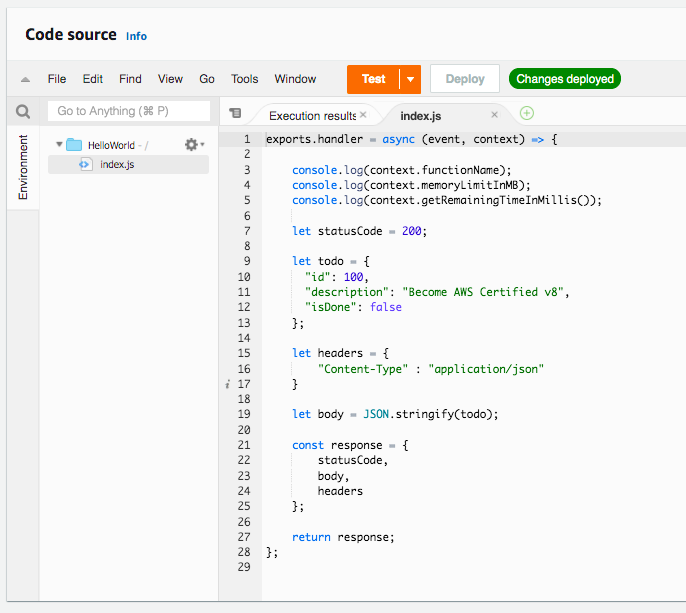
1. Select **region** US East (N. Virginia) - us-east-1. All our resources in this lab will be created in us-east-1.



1. In the **AWS Console**, from **Services** menu - choose **Lambda**. Click **Create function**.
2. Choose to **Author from scratch**. Enter the Function**name** as *HelloWorld*.
3. Choose **Runtime** as Node.js 14.x. Choose defaults for the rest of the options.



1. Click **Create function**
2. On the **Lambda Source Code** screen, Under **Code** > **Code Source** > Select the file **index.js** under the folder **HelloWorld**. Copy code from file lambda-function-01-hello-world.js. Your screen should look like:

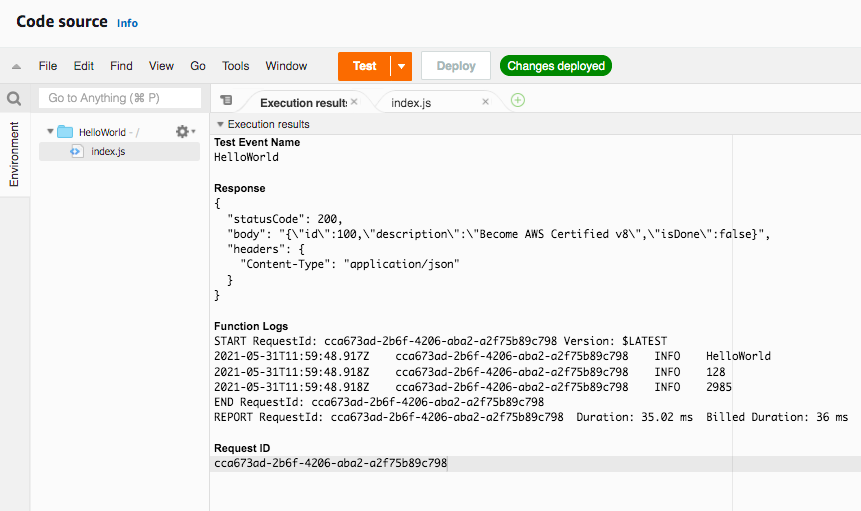


1. Click **Deploy**

Your hello world Lambda function is ready

1. Click **Test**
2. On the **Configure test event** screen, select**Create new test event**.
3. Choose **Event template** hello-world. Enter the Event name as HelloWorld and click **Create**.
4. Click **Test** again

Result of execution is shown below:



Lambda function returns a *JSON response* containing the fields - *statusCode*, *body* and *headers*. This will be mapped to the REST API response from API Gateway in the subsequent steps.

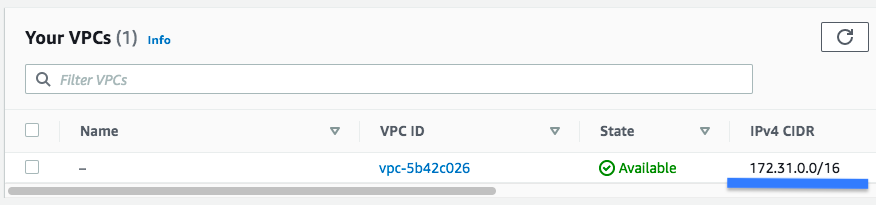
Ref lamda-function-01-hello-world.js

<https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-lambda-proxy-integrations.html#api-gateway-simple-proxy-for-lambda-output-format>

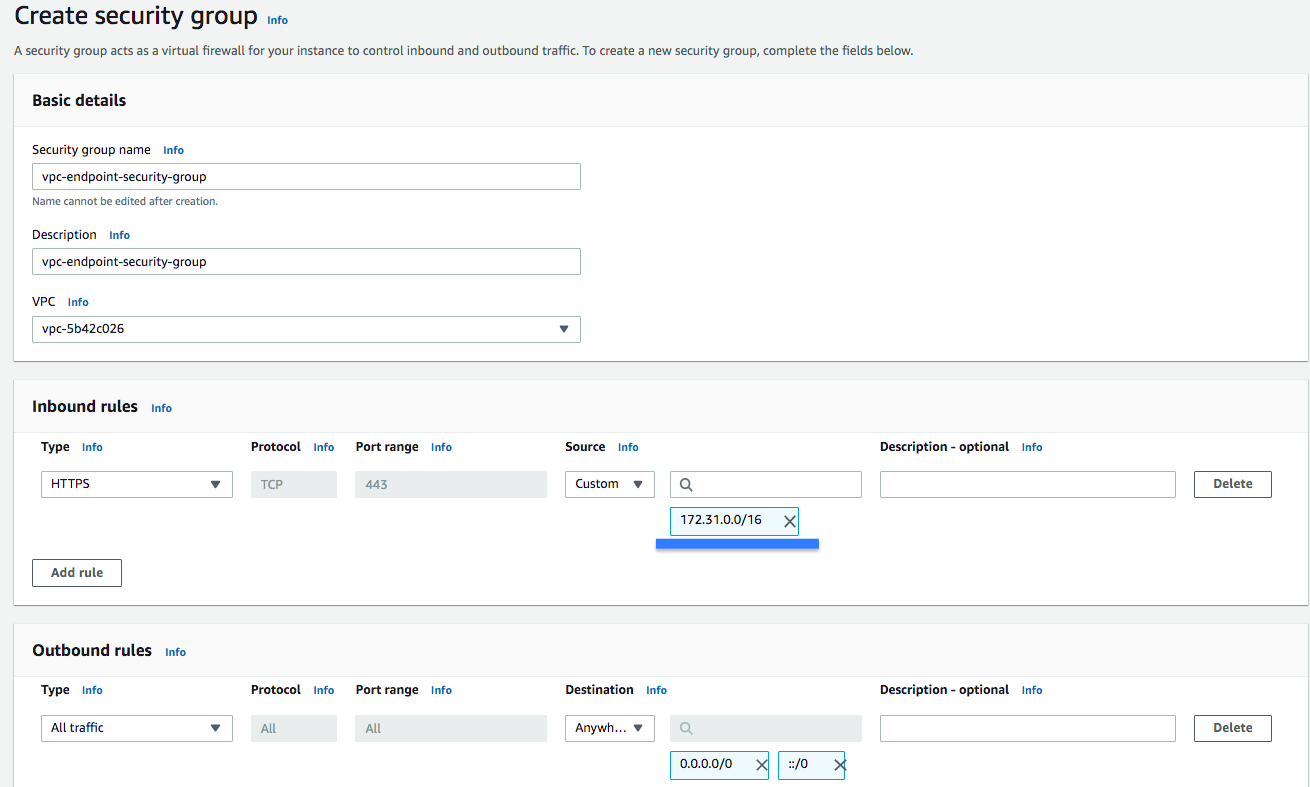
Create a Private API Gateway

Your task is to create a Private API Gateway. Prerequisites to create a Private API Gateway are a Security Group and a VPC Endpoint . You will start with creating a Security Group that allows inbound traffic from the default VPC on port 443. Next, you will create a VPC Endpoint connected to the Security Group. Lastly, you will create the Private API Gateway of type REST API.

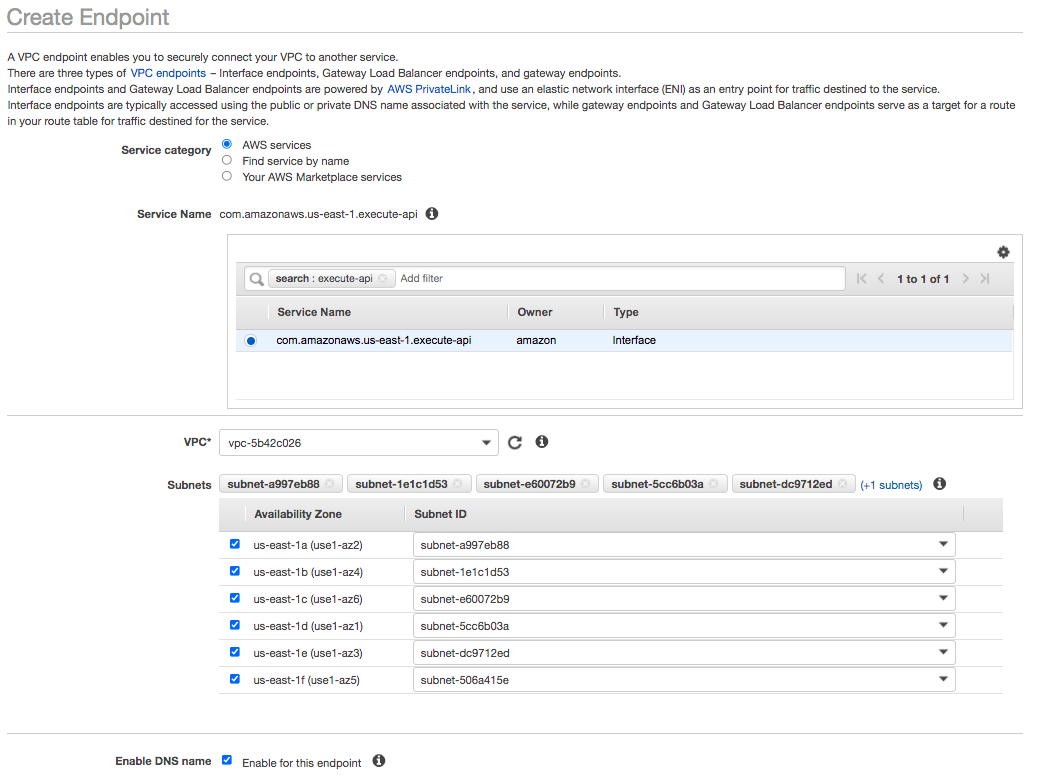
1. In the AWS Console, search for VPC in the search bar. From **Services**, select **VPC**. Click the **VPCs** link on the next screen. You should see one VPC on the **Your VPCs** page. Note down the **VPC ID** as VPC-ID and **IPv4 CIDR** as IPV4-CIDR. Both these values will be used when creating the Security Group later.



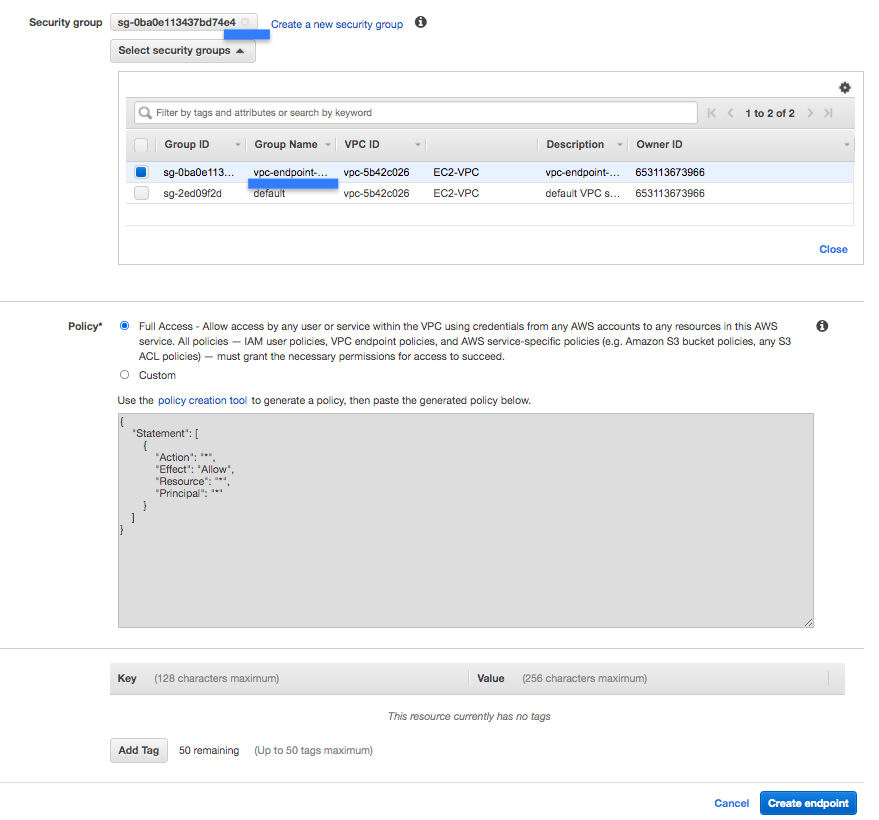
1. In the **AWS Console**, search for **Security groups** in the search bar. From **Features**, select **Security groups - VPC feature**.
2. Click **Create security group**
3. Under **Basic details**, enter *Security group name* as vpc-endpoint-security-group. Enter Description as vpc-endpoint-security-group.
4. Choose the **default VPC** as the VPC
5. Under **Inbound rules**, click **Add rule**. Choose **Type** as HTTPS. Enter **Destination** as the CIDR Range of the default VPC - 172.31.0.0/16 (You noted this down earlier - IPV4-CIDR).
6. Under **Outbound rules**, click **Add rule**. Choose **Type** as All Traffic and **Destination** as Anywhere.



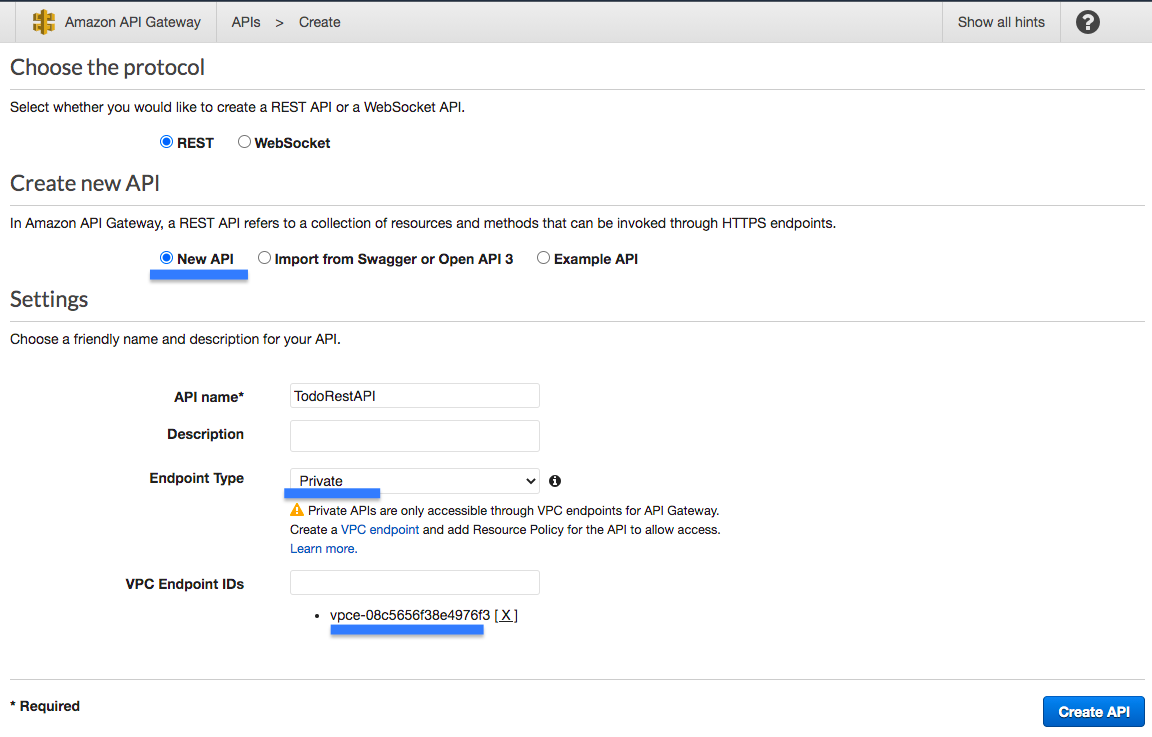
1. Click **Create security group**
2. In the **AWS Console**, search for **VPC Endpoints** in the search bar
3. From **Features**, select **Endpoints - VPC feature**
4. Click **Create Endpoint**
5. On the **Create Endpoint** screen, choose **Service category** as AWS services
6. Under **Service Name** search for execute-api and choose com.amazonaws.us-east-1.execute-api
7. Select ***default*** VPC (make sure that the ID matches VPC-ID noted down earlier)
8. Choose all 6 subnets from default VPC
9. Enable (check) the check box **Enable DNS name** **- Enable for this endpoint**



1. You need to select vpc-endpoint-security-group as the **Security Group**. Delete the suggestion to use the default security group by clicking the X button beside it. From the list of security groups displayed, choose the **Security Group** with name vpc-endpoint-security-group.
2. Under **Policy**, choose Full Access and click **Create Endpoint**



1. *Note down* the id of the VPC Endpoint as VPC-ENDPOINT-ID for use in the next step
2. In the **AWS Console**, from **Services** menu - choose **API Gateway**
3. On the welcome screen, choose the **API type** - REST API Private. Click **'Build'**.
4. Click '**OK'** on the welcome popup
5. Choose the protocol **REST**
6. Under **Create new API** choose New API
7. Under **Settings** enter **API name** as TodoRestAPI and Choose **Endpoint Type** as Private.
8. Choose **VPC Endpoint IDs** as the VPC Endpoint you created earlier (Type in vpce and you should see the endpoint listed)

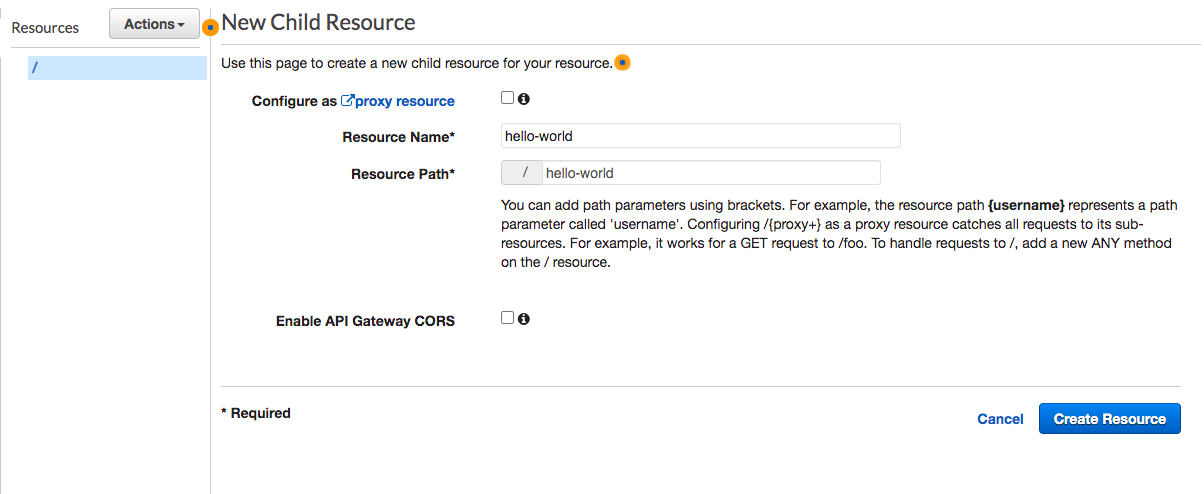


1. Click **Create API**

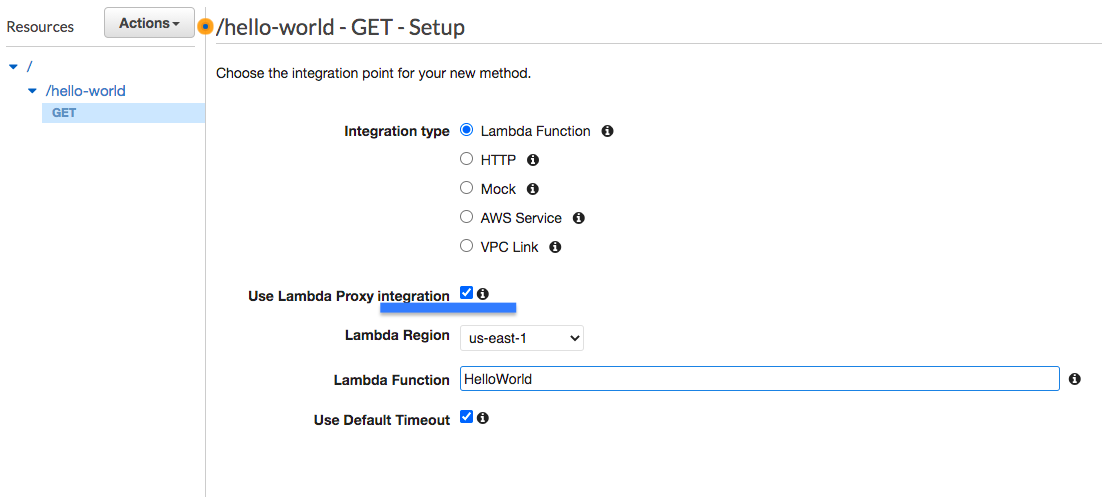
Integrate hello world Resource GET Method with Lamda Function

Your task is to integrate the private REST API Gateway with the hello world lambda function you created earlier. You will start with creating a hello world resource in the API Gateway. After that you will create a GET method and integrate it with the hello world Lambda function.

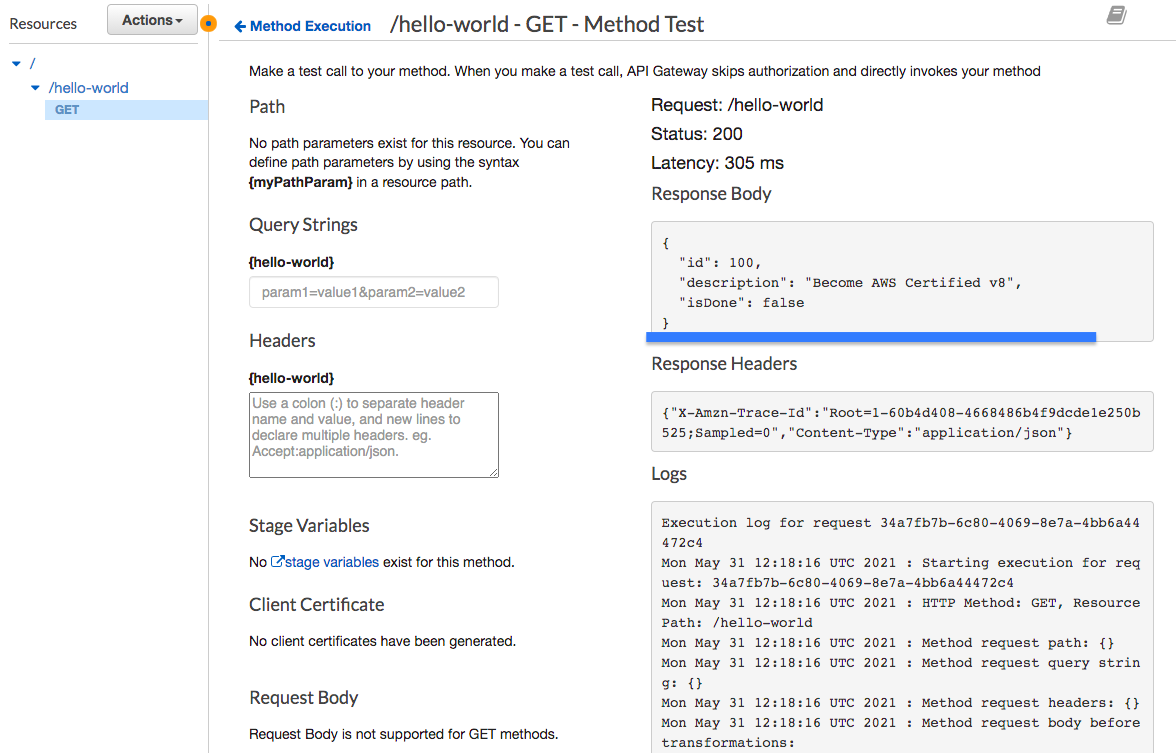
1. In the **Actions** dropdown beside **Resources**, Choose **Create Resource**
2. On the **New Child Resource** screen, leave **Configure as a proxy resource** as unchecked
3. Enter **Resource Name** as hello-world. Enter ***Resource Path*** as hello-world.



1. Choose defaults for other options and click **Create Resource**
2. Select */hello-world* resource and select **Create Method** from **Actions**
3. Choose **GET** and click the **check** icon beside it
4. On the **/hello-world - GET - Setup** screen, choose **Integration type** - Lambda Function.
5. Check the checkbox **Use Lambda Proxy integration**
6. Choose the Lambda Region us-east-1
7. Choose the Lambda Function HelloWorld



1. Choose defaults for other options and click Save
2. Click **'OK'** on the Add Permission to Lambda Function popup
3. Your Hello World REST API Resource and Resource Methods are now ready
4. Click **Test** inside **/hello-world - GET - Method Execution** (Top left corner)
5. On the **Method execution** screen, scroll down and click **Test**.  
   You should see a response as shown:



1. You have now integrated your API Gateway with a Lambda function

<https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-lambda-proxy-integrations.html#api-gateway-simple-proxy-for-lambda-output-format>

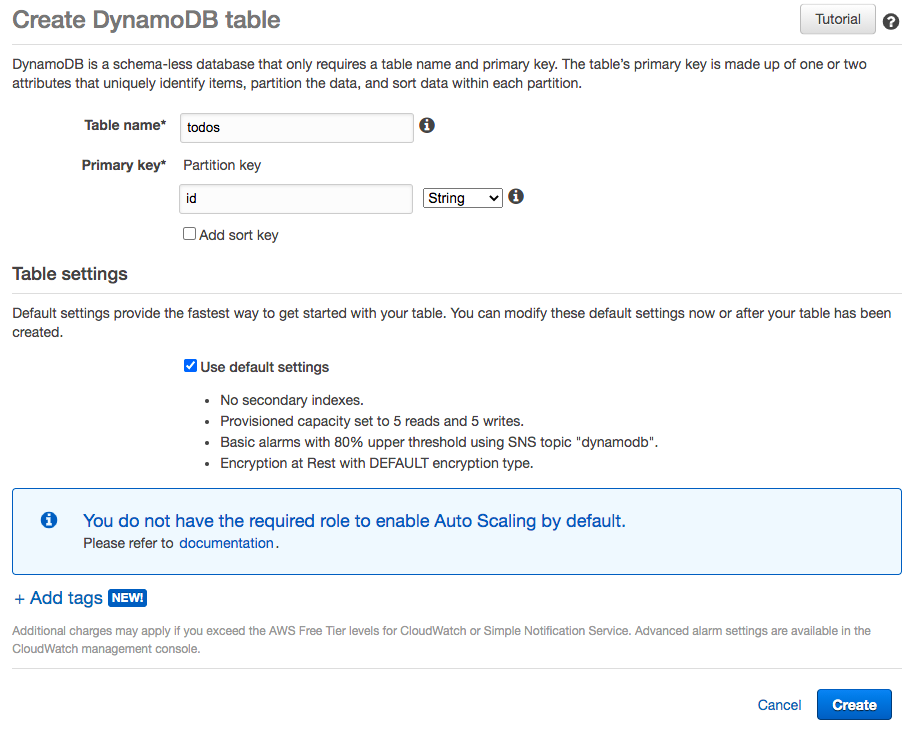
<https://docs.aws.amazon.com/lambda/latest/dg/nodejs-context.html>

<https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-lambda-proxy-integrations.html#api-gateway-create-api-as-simple-proxy>

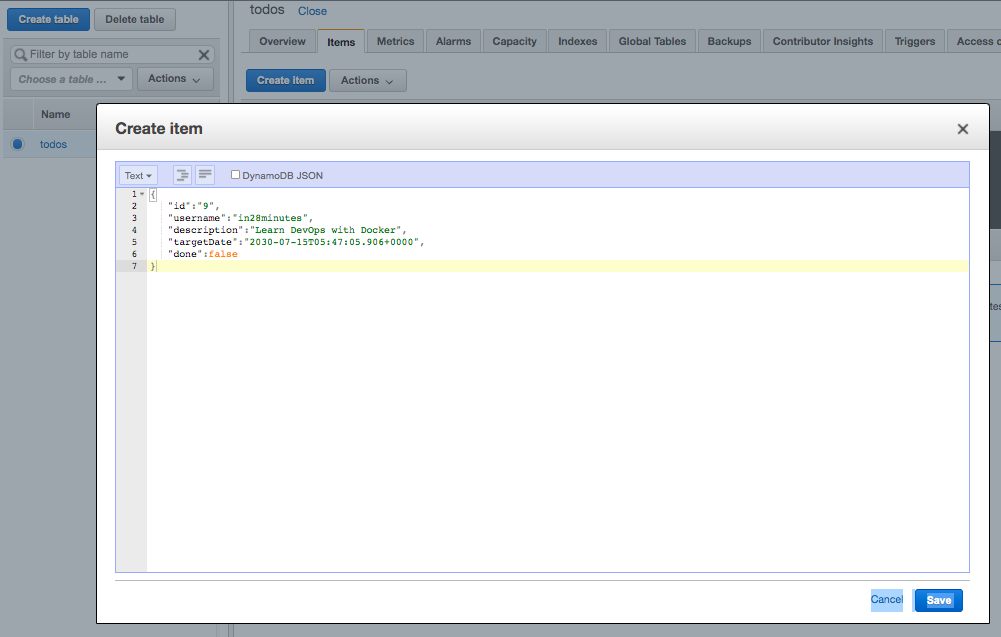
Create a DynamoDB to store todos

Your task is to create a DynamoDB table, named *todos*, to store your todos. You will also create a couple items in the *todos* table.

1. In the **AWS Console**, from **Services** menu - choose **DynamoDB**
2. Click **Create table** and then set the **Table name** to todos
3. Enter id as **Primary key** with **type** as String
4. Take the defaults for all other settings and click **Create**



1. Select table todos in DynamoDB
2. Choose the **Items** tab
3. Click **Create Item**
4. Choose the **Text** option from the dropdown in the **Create Item** screen
5. Copy item JSON content from resource todo-item-1.json
6. Click **Save**



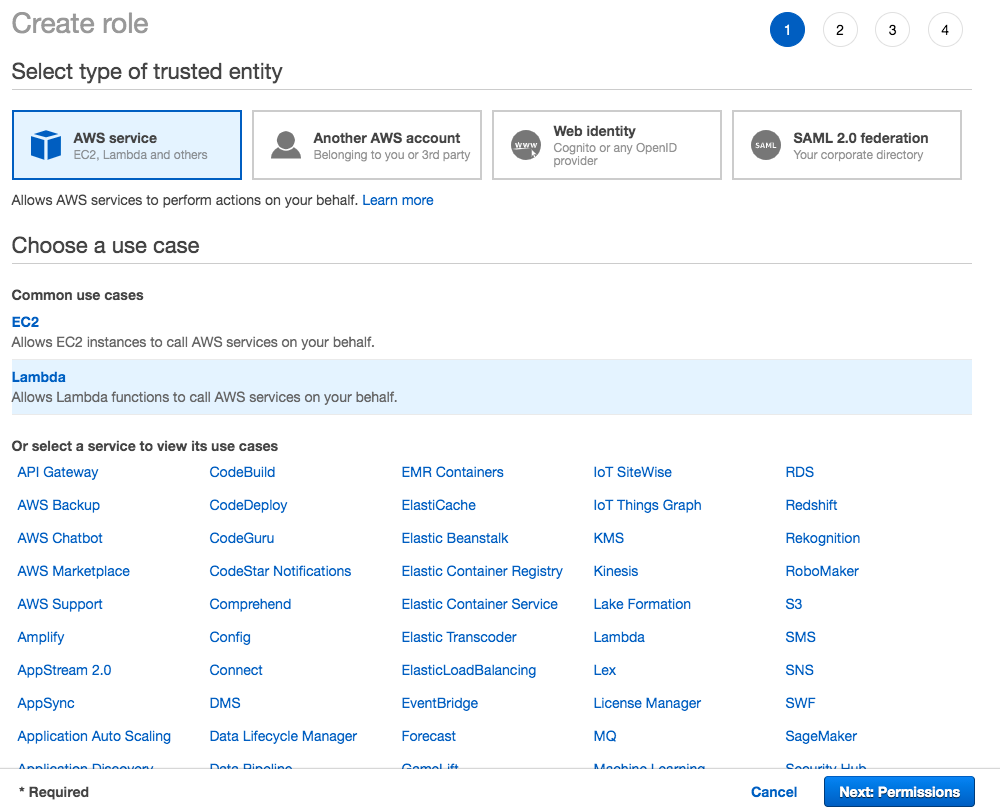
1. Create another item using JSON content from resource todo-item-2.json

Ref todo-item-1.json, todo-item-2.json

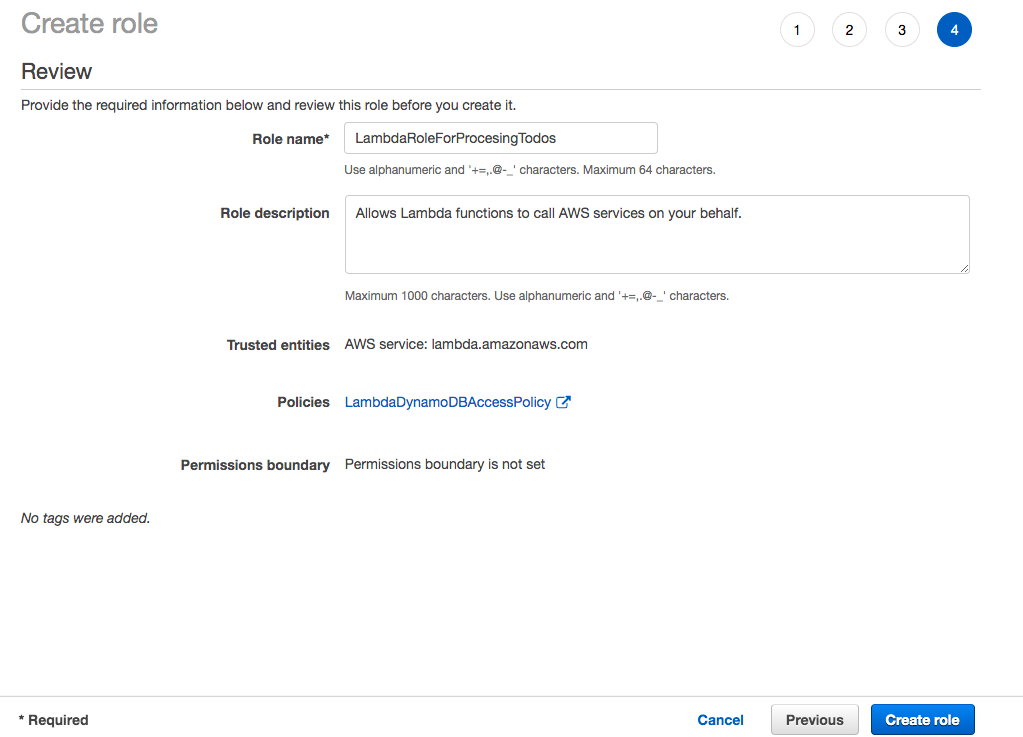
Create IAM Role for Lambda Functions

Your task is to create a IAM Role LambdaRoleForProcessingTodos with the permissions to insert, delete, update and retrieve items from the DynamoDB Table. You will create an IAM Policy LambdaDynamoDBAccessPolicy with right permissions and assign it to the IAM Role LambdaRoleForProcessingTodos.

1. In the **AWS Console**, from **Services** menu - choose **IAM**
2. Choose **Policies** from the left hand side menu
3. Click **Create policy**
4. On the **Create policy** screen, click the **JSON** tab
5. Copy JSON from resource iam-role-lambda-function.json into the JSON editor
6. Click **Next: Tags**
7. Click **Next: Review**
8. On the **Create policy** screen, Enter **Name** as *LambdaDynamoDBAccessPolicy*
9. Take the defaults for all other settings and click **Create policy**
10. Choose **Roles** from the **IAM** left hand side menu
11. Click **Create role**
12. Choose the **type of trusted entity** as **AWS service**
13. **Choose a use case** Lambda. Your screen should look like:



1. Click **Next: Permissions**
2. **Search** and choose the policy *LambdaDynamoDBAccessPolicy*
3. Click **Next: Tags**
4. Click **Next: Review**
5. Enter the **Role name** as *LambdaRoleForProcessingTodos*



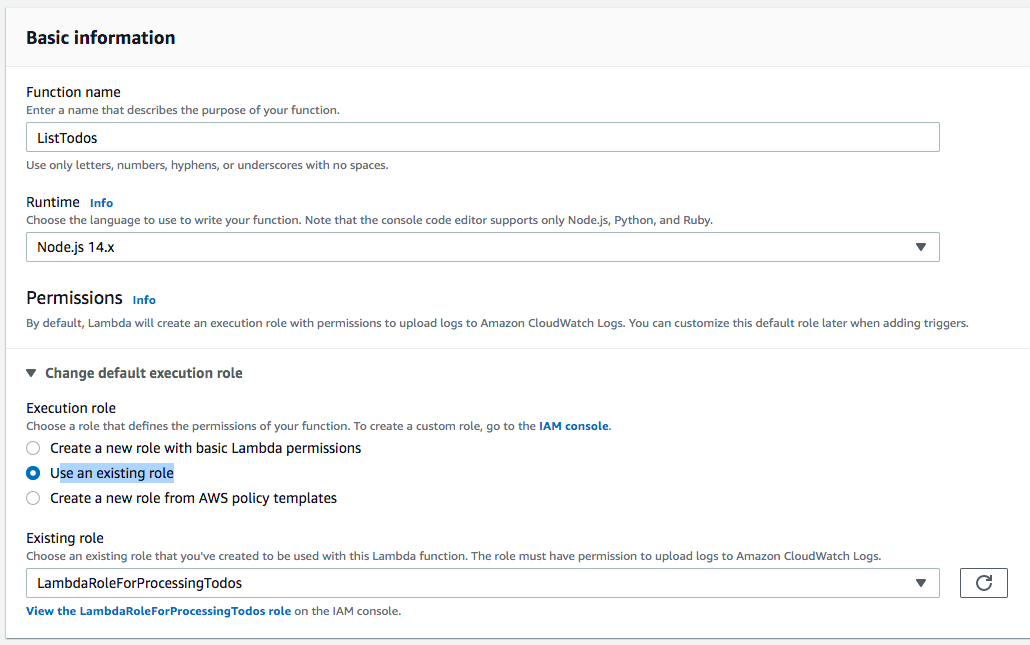
1. Click **Create role**
2. You should see a message ***The role LambdaRoleForProcessingTodos has been created***

Ref Iam-role-lambda-function.json

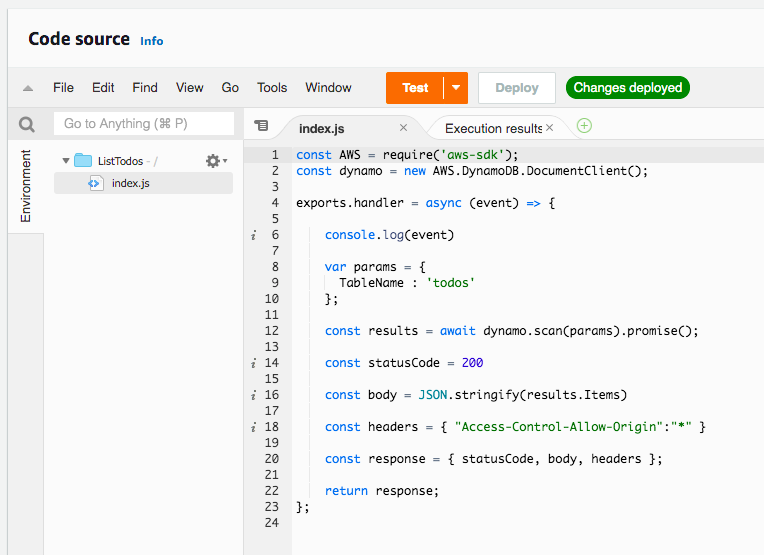
Create todos REST API to retrieve detail of specific todos(GET method)

Your task is to create a REST API to retrieve all the todos from the DynamoDB table. You will start with creating a Lambda function, named *ListTodos*, to retrieve all items from the todos DynamoDB table. You will create a Todo resource in the API Gateway and create a GET method invoking the Lambda function *ListTodos*.

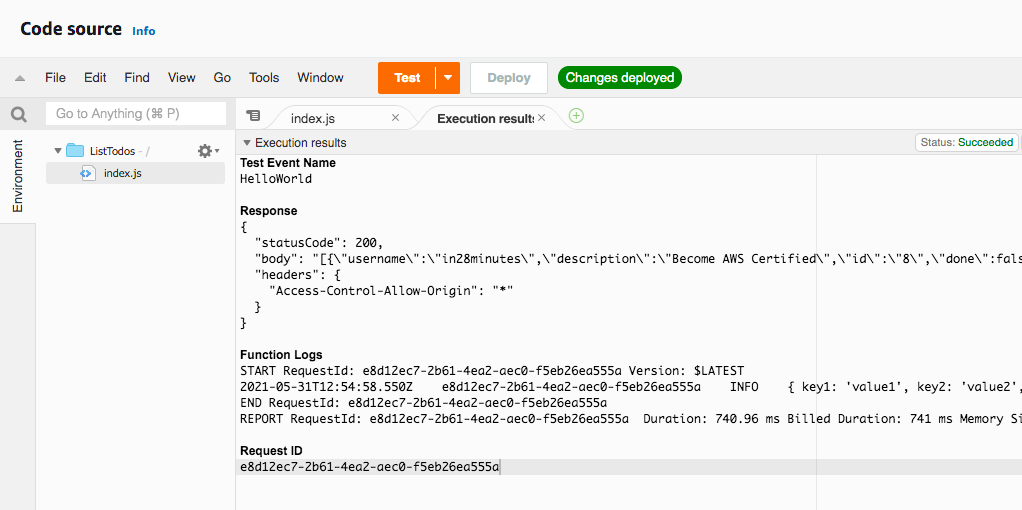
1. In the **AWS Console**, from **Services** menu - choose **Lambda**
2. Click **Create function**
3. Choose to **Author from scratch**. Enter **Function name** as ListTodos
4. Choose **Runtime** as Node.js 14.x
5. In **Permissions** expand **Change default execution role**. Choose **Use an existing role**.
6. Choose Existing role as *LambdaRoleForProcessingTodos*



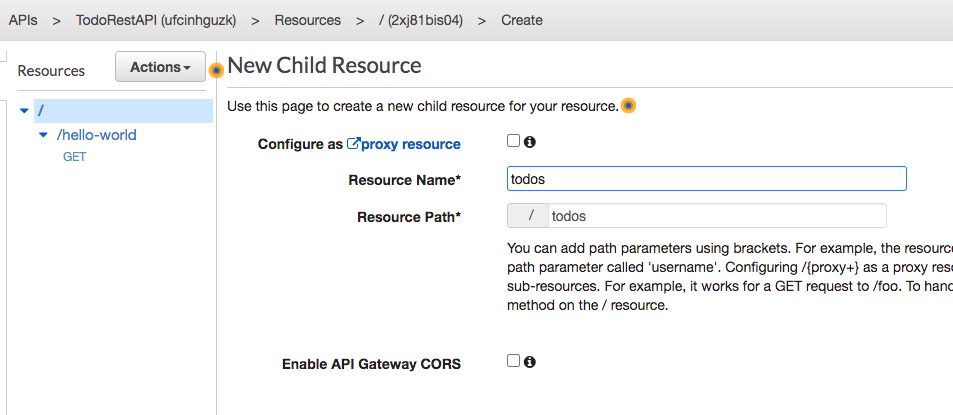
1. Click **Create function**
2. On the **Lambda Source Code** screen, Under Code > Code Source > Select the file index.js under the folder ListTodos
3. **Copy** the code from the file lambda-function-03-list-todos.js
4. Click **Deploy**



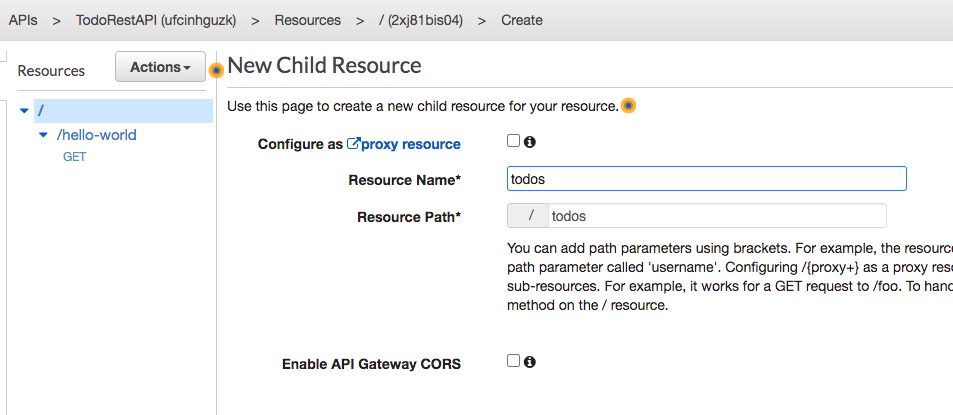
1. Click **Test**
2. On the **Configure test event** screen, select **Create new test event**
3. Choose **Event template** hello-world
4. Enter the **Event name** as HelloWorld and click **Create**
5. Click **Test** again. You should be able to see the result of execution as shown below:



1. In the **AWS Console**, from **Services** menu - choose **API Gateway**
2. Select the API Gateway you created earlier - *TodoRestAPI*
3. **Select**/ as a resource.
4. In the **Actions** dropdown beside **Resources**, Choose **Create Resource**
5. On the **New Child Resource** screen, leave **Configure as a proxy resource** as unchecked
6. Enter **Resource Name** as todos
7. Enter **Resource Path** as *todos*

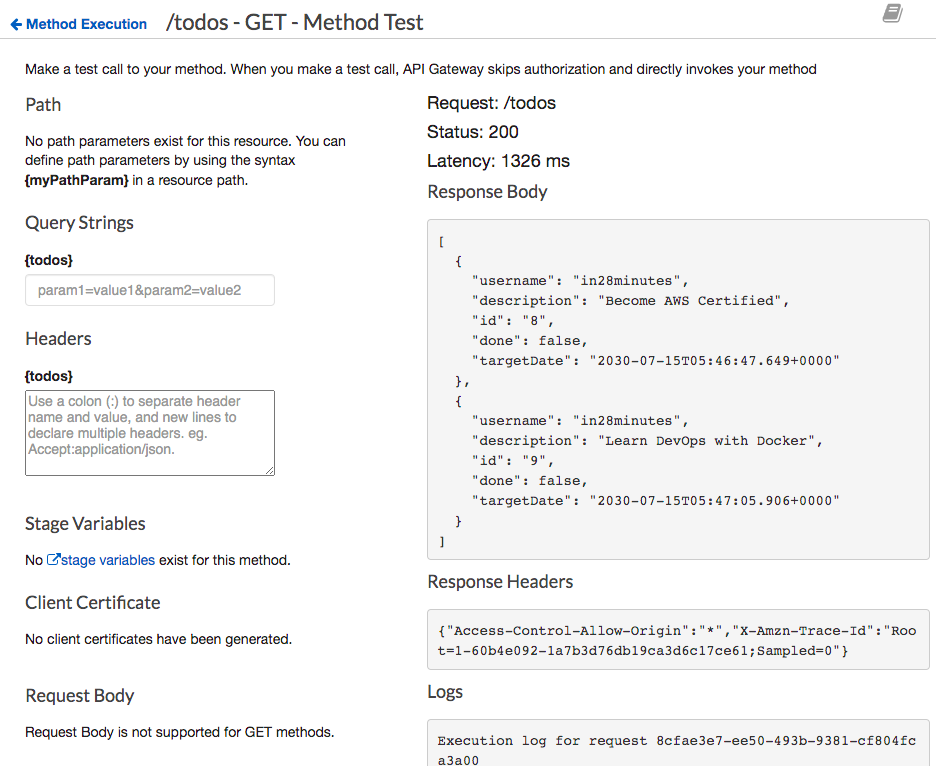


1. Choose defaults for other options and click **Create Resource**
2. **Select** /todos resource and select **Create Method** from **Actions** dropdown
3. Choose **GET** and click the **check**icon beside it
4. On the **/todos - GET - Setup** screen, choose **Integration type** - Lambda Function
5. Check the checkbox **Use Lambda Proxy integration**
6. Choose the **Lambda Region** us-east-1
7. Choose the **Lambda Function** *ListTodos*



1. Choose defaults for other options and click **Save**
2. Click **'OK'** on the **Add Permission to Lambda Function** popup
3. Click **Test**
4. On the '**Method execution**' screen, scroll down and click **Test**.

You should see a response as shown below:

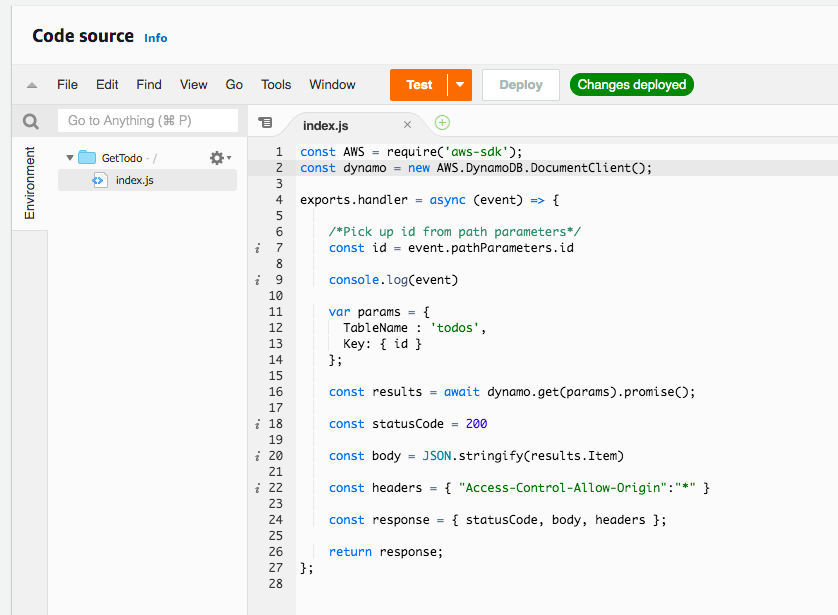


Ref Lamda-functio-03-list-todos.js

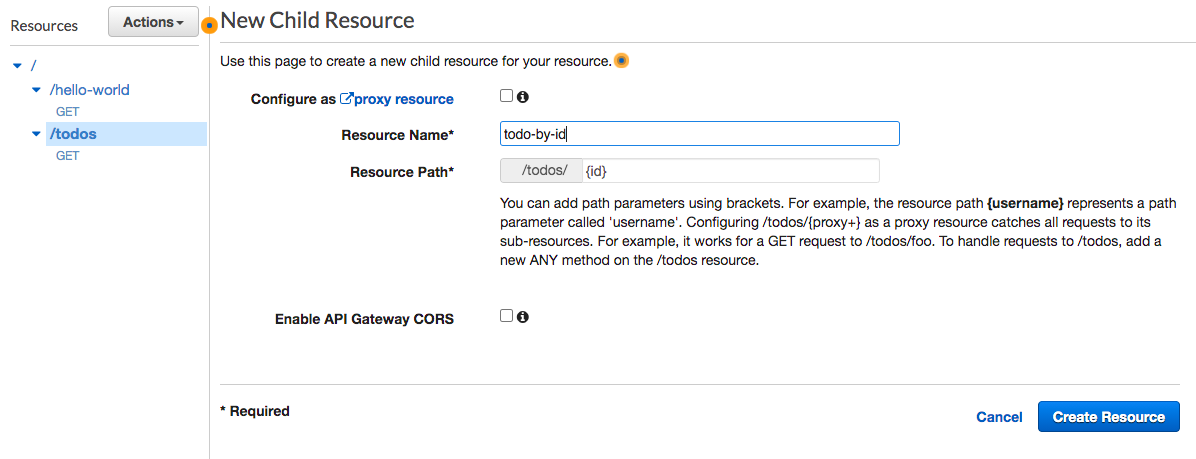
Create Todo REST API to retrieve details of specific todo(GET method)

Your task is to create a REST API to retrieve the details of a specific todo. You will pass todo id as a path parameter. You will start with creating a Lambda function, GetTodo, to retrieve the details of a specific todo id from the DynamoDB todos table. After that, you will integrate the API Gateway with the Lambda Function GetTodo.

1. In the **AWS Console**, from **Services** menu - choose **Lambda**
2. Click **Create function**
3. Choose to **Author from scratch**
4. Enter the Function**name** as *GetTodo*
5. Choose **Runtime** as Node.js 14.x
6. In **Permissions** expand **Change default execution role**
7. Choose **Use an existing role**
8. Choose Existing role as *LambdaRoleForProcessingTodos*
9. Click **Create function**
10. On the **Lambda Source Code** screen, Under **Code** > **Code Source** > Select the file index.js under the folder *GetTodo*
11. **Copy** the code from the file lambda-function-04-get-todo.js
12. Click **Deploy**



1. In the **AWS Console**, from **Services** menu - choose **API Gateway**
2. Select the API Gateway you created earlier - TodoRestAPI
3. **Select** /todos as resource
4. In the **Actions** dropdown beside **Resources**, Choose **Create Resource**
5. On the **New Child Resource** screen, leave **Configure as a proxy resource** as unchecked
6. Enter **Resource Name** as todo-by-id
7. Enter **Resource Path** as {id} to represent an id parameter

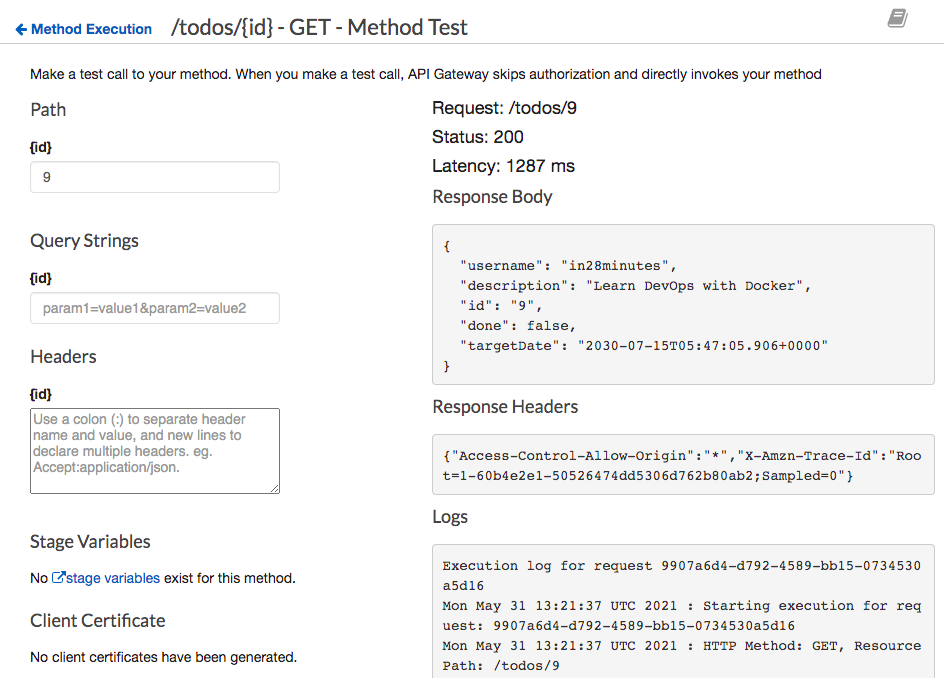


1. Choose defaults for other options and click **Create Resource**
2. **Select** /todos/{id} resource and select **Create Method** from action
3. Choose **GET** and click the **check** icon beside it
4. On the **/todos/{id} - GET - Setup** screen, choose **Integration type** - Lambda Function
5. Check the checkbox **Use Lambda Proxy integration**
6. Choose the **Lambda Region** us-east-1
7. Choose the Lambda Function GetTodo



1. Choose defaults for other options and click **Save**
2. Click **'OK'** on the **Add Permission to Lambda Function** popup
3. Click **Test**
4. On the '**Method execution'** screen, scroll down and click **Test**
5. In Path {id} enter 9 (You can pick any id of items which are present in DynamoDB todos table)

You should see a response as shown below:

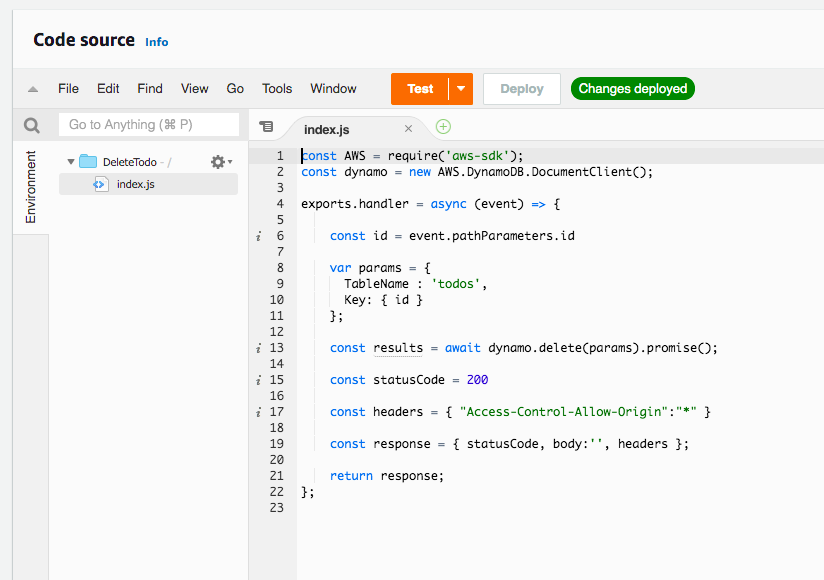


Ref Lambda-function-04-get-todo.js

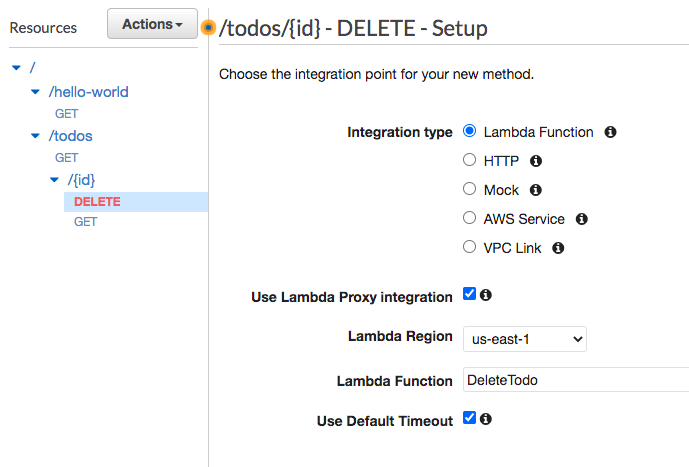
Add Resource to delete a todo(DELETE nethod)

Your task is to create a REST API to delete a specific todo. You will pass todo id as a path parameter. You will start with creating a Lambda function, DeleteTodo, to delete a todo with a given id from the DynamoDB todos table. After that, you will integrate the API Gateway with the Lambda Function DeleteTodo.

1. In the **AWS Console**, from **Services** menu - choose **Lambda**
2. Click **Create function**
3. Choose to **Author from scratch**
4. Enter the Function**name** as *DeleteTodo*
5. Choose **Runtime** as Node.js 14.x
6. In **Permissions** expand **Change default execution role**
7. Choose **Use an existing role**
8. Choose Existing role as *LambdaRoleForProcessingTodos*
9. Click **Create function**
10. On the Lambda Source Code screen, Under **Code**> **Code Source** > Select the file index.js under the folder *DeleteTodo*
11. Copy the code from the file lambda-function-05-delete-todo.js
12. Click **Deploy**



1. In the **AWS Console**, from **Services** menu - choose **API Gateway**
2. Select the API Gateway you have created earlier - TodoRestAPI
3. **Select** /todos > /{id} resource and select **Create Method** from actions
4. Choose **DELETE** and click the **check** icon beside it
5. On the**/todos/{id} - DELETE - Setup** screen, choose **Integration type** - Lambda Function
6. Check the checkbox **Use Lambda Proxy integration**
7. Choose the **Lambda Region** us-east-1
8. Choose the Lambda Function *DeleteTodo*



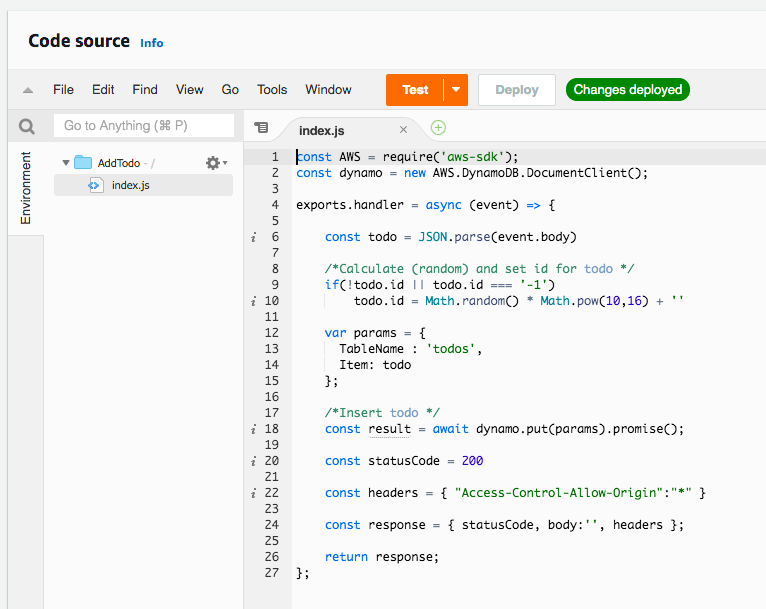
1. Choose defaults for other options and click **Save**
2. Click 'OK' on the **Add Permission to Lambda Function** popup
3. Click **Test**
4. On the '**Method execution**' screen, scroll down and click **Test**
5. In Path {id} enter 9 (You can pick any id of items which are present in DynamoDB todos table)
6. You should see a successful response

Ref lamda-function-05-delete-todo.js

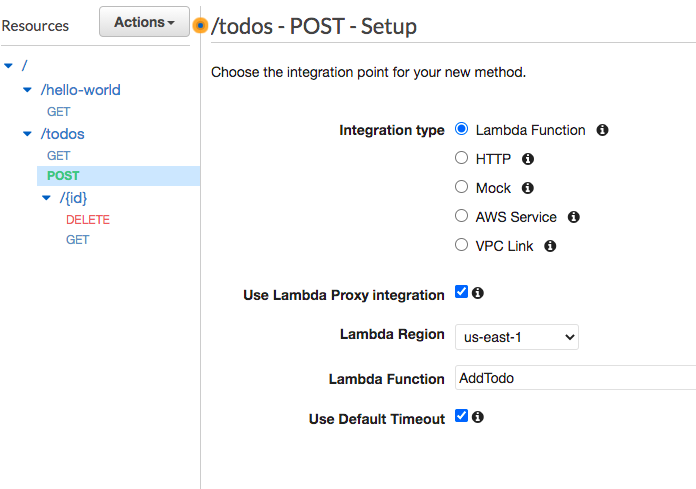
ADD Resource Method to create a todo(POST method)

Your task is to create a REST API to create a new todo. You will pass todo details as body of the request. You will start with creating a Lambda function, AddTodo, to add a todo item into the DynamoDB todos table. After that, you will integrate the API Gateway with the Lambda Function AddTodo.

1. In the **AWS Console**, from **Services** menu - choose **Lambda**
2. Click **Create function**
3. Choose to **Author from scratch**
4. Enter the Function**name** as *AddTodo*
5. Choose **Runtime** as Node.js 14.x
6. In **Permissions** expand **Change default execution role**
7. Choose **Use an existing role**
8. Choose Existing role as *LambdaRoleForProcessingTodos*
9. Click **Create function**
10. On the Lambda Source Code screen, Under **Code**> **Code Source** > Select the file index.js under the folder AddTodo
11. **Copy** the code from the file lambda-function-06-create-todo.js
12. Click **Deploy**

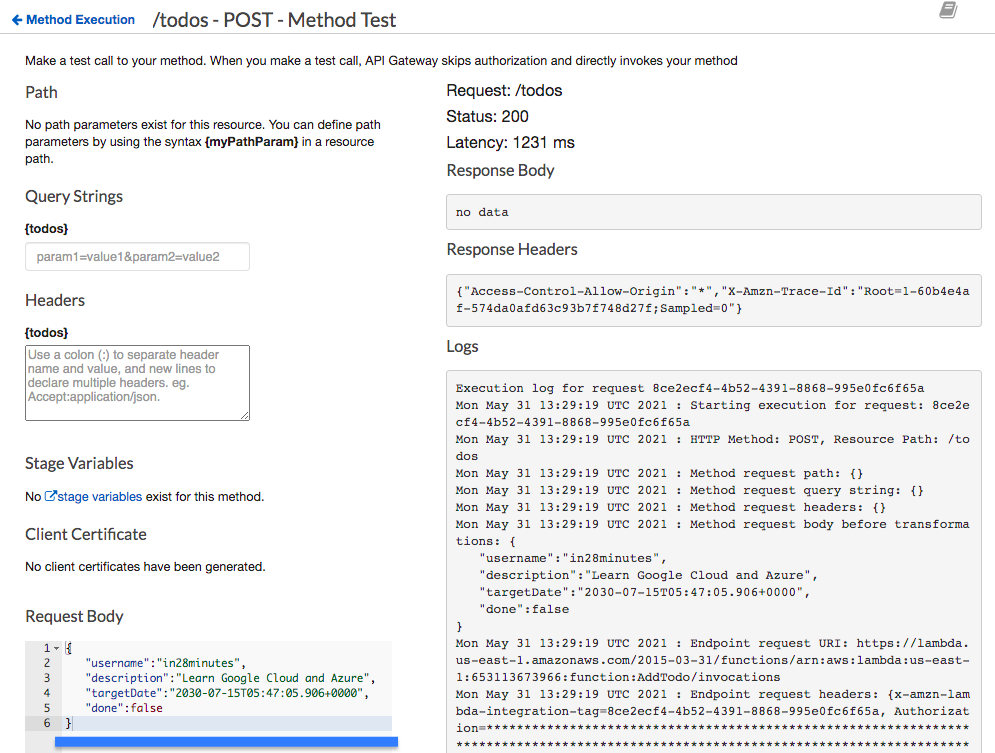


1. In the **AWS Console**, from **Services** menu - choose **API Gateway**
2. Select the API Gateway you created earlier - TodoRestAPI
3. **Select** /todos as a resource and select **Create Method** from actions
4. Choose **POST** and click the **check** icon beside it.
5. On the **/todos - POST - Setup** screen, choose **Integration type** - Lambda Function
6. Check the checkbox **Use Lambda Proxy integration**
7. Choose the **Lambda Region** us-east-1
8. Choose the **Lambda Function***AddTodo*



1. Choose defaults for other options and click **Save**
2. Click **'OK'** on the Add Permission to *Lambda Function* popup
3. Click **Test**
4. On the '**Method execution**' screen, enter **Request Body** as text from the resource lambda-function-06-create-todo-request.json, scroll down and click **Test.**

You should see a response as shown below:



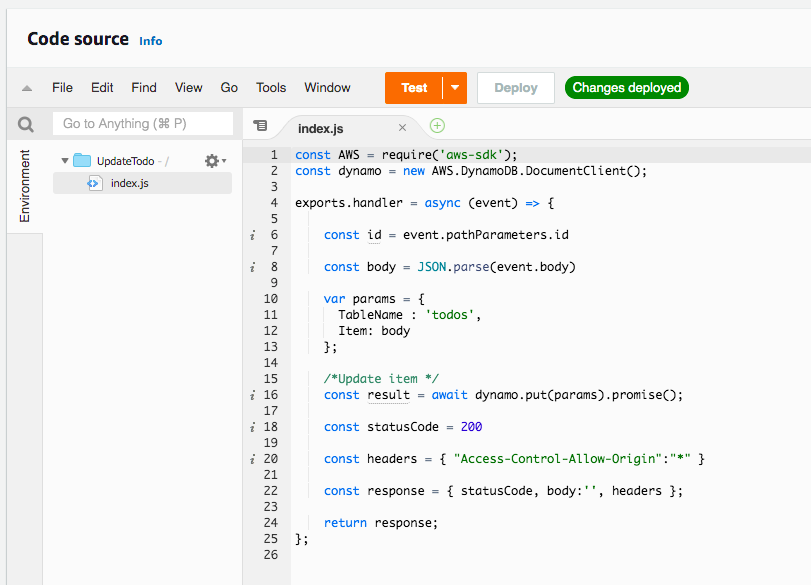
Ref lambda-function-06-create-todo.js

lamda-function-06-create-request.js

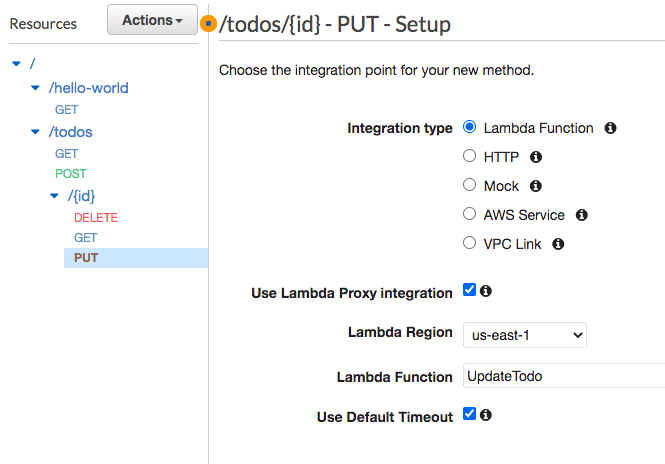
Add Resource Method to update a todo(PUT method)

Your task is to create a REST API to update a specific todo. You will pass todo id as a path parameter. You will start with creating a Lambda function, UpdateTodo, to update a todo with a given id in the DynamoDB todos table. After that, you will integrate the API Gateway with the Lambda Function UpdateTodo.

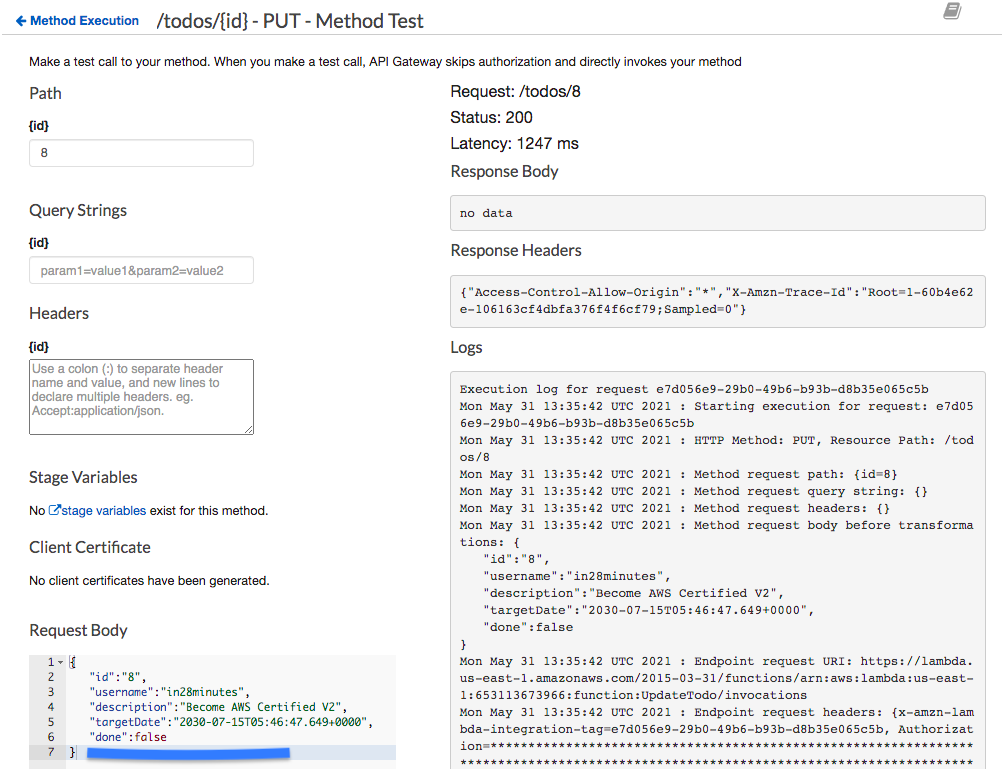
1. In the **AWS Console**, from **Services** menu - choose **Lambda**
2. Click **Create function**
3. Choose to **Author from scratch**
4. Enter the Function**name** as *UpdateTodo*
5. Choose **Runtime** as Node.js 14.x
6. In **Permissions** expand **Change default execution role**
7. Choose **Use an existing role**
8. Choose Existing role as *LambdaRoleForProcessingTodos*
9. Click **Create function**
10. On the **Lambda Source Code** screen, Under **Code** > **Code Source** > Select the file index.js under the folder *UpdateTodo*
11. **Copy** the code from the file lambda-function-07-update-todo.js
12. Click **Deploy**



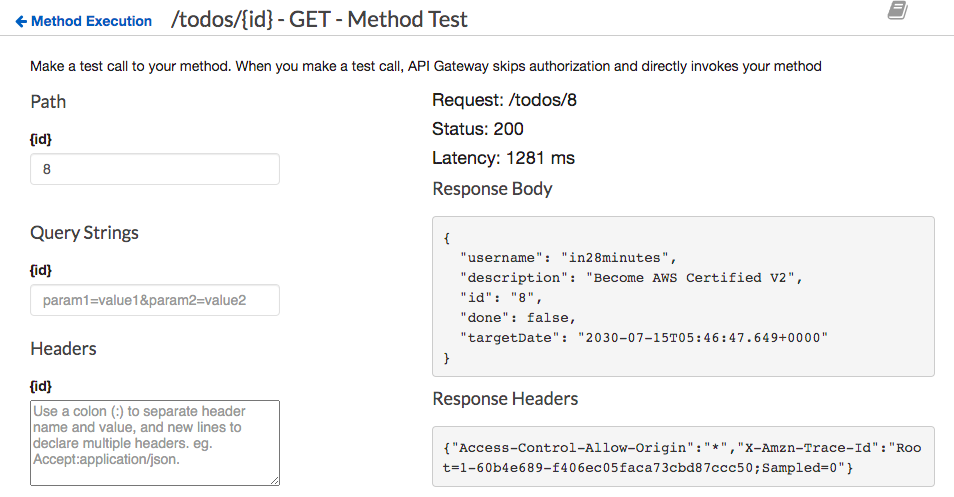
1. In the **AWS Console**, from **Services** menu - choose **API Gateway**
2. Select the API Gateway you created earlier - *TodoRestAPI*
3. Select **/todos > /{id}** resource and select **Create Method** from **Actions**
4. Choose **PUT** and click the **check** icon beside it
5. On the /todos/{id} - PUT - Setup screen, choose **Integration type** - Lambda Function
6. Check the checkbox **Use Lambda Proxy integration**
7. Choose the **Lambda Region** us-east-1
8. Choose the Lambda Function *UpdateTodo*



1. Choose defaults for other options and click **Save**. Click **'OK'** on the ***Add Permission to Lambda Function*** popup.
2. Click **Test**. On the '**Method execution**' screen, scroll down and click **Test**
3. In Path {id} enter 9 (You can pick any id of items which are present in DynamoDB todos table)
4. Enter **Request Body** as content from lambda-function-07-update-todo-request-body.json. You should see a response as shown below:



1. Execute a test for the GET method of /todos/{id} with id 8 to test if the update is successful



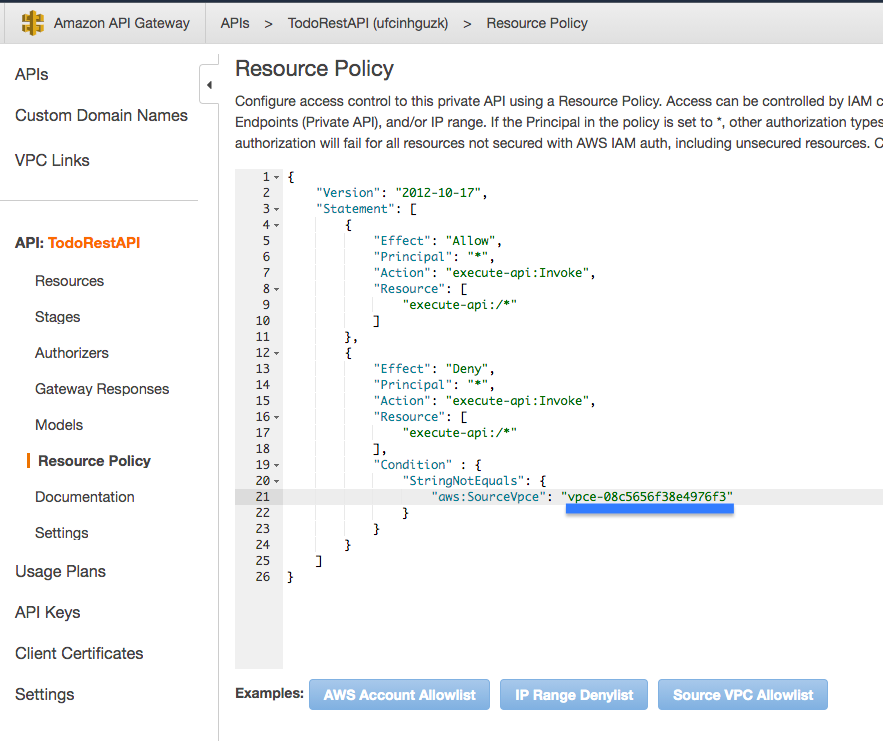
Ref lambda-function-07-update-todo.js

lamda-function-07-update-todo-request-body.js

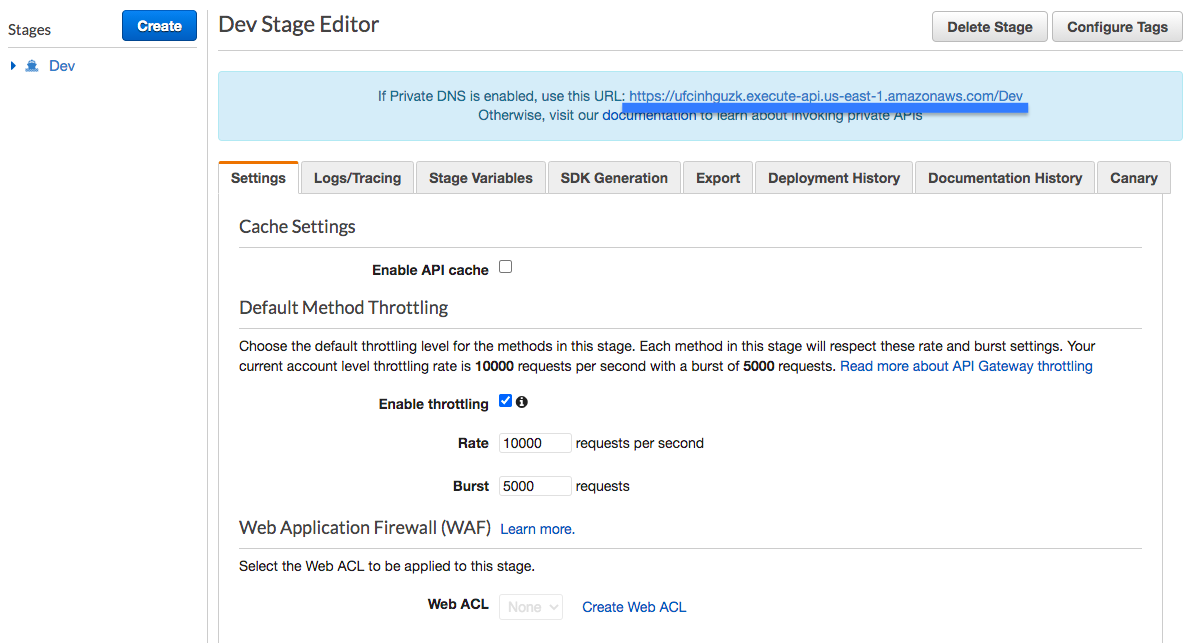
Deploy and test the REST API

Your task is to deploy the REST API to an environment and test it. To deploy a private REST API Gateway, you need to create a Resource Policy. You will start with creating a Resource Policy allowing access to API Gateway from the default VPC through the VPC endpoint you created earlier. After that, you will deploy the API Gateway resources to a stage named Dev. At the end, you will test the private REST API from an EC2 instance created in the default VPC.

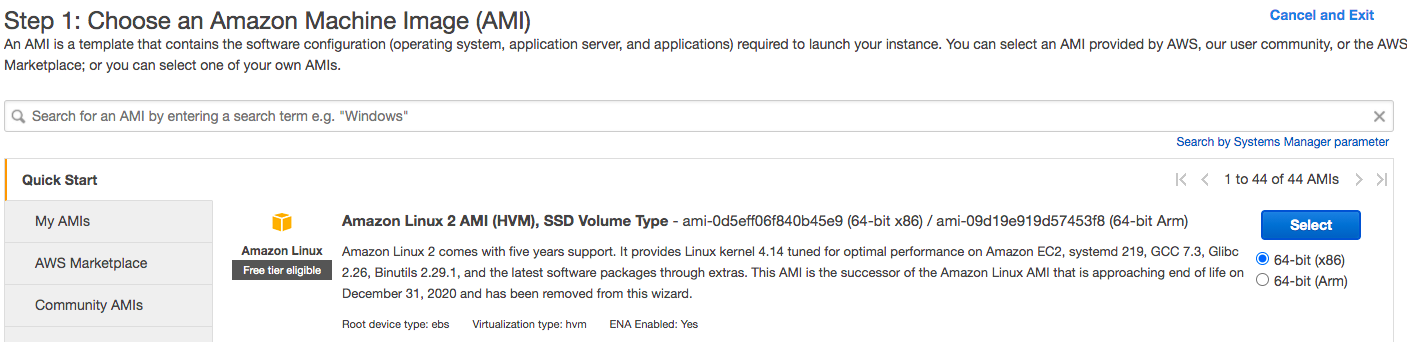
1. In the **AWS Console**, from **Services** menu - choose **API Gateway**
2. Select the API Gateway you created earlier - TodoRestAPI
3. Go to **Resource Policy**
4. In the **Resource Policy** screen, configure policy using the content from *api-gateway-resource-policy.json*
5. **Make sure** that you replace *{{VPC-ENDPOINT-ID}}* with the id of the VPC endpoint (you noted it down as VPC-ENDPOINT-ID) created earlier.



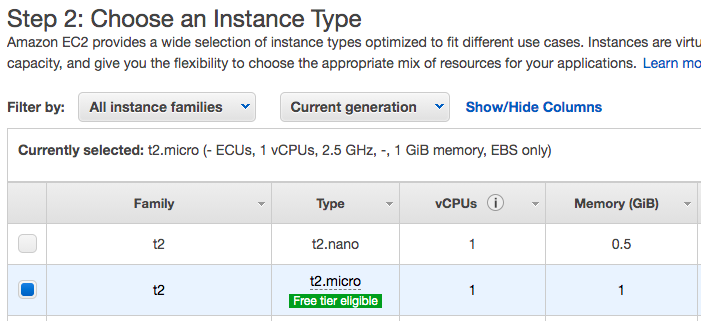
1. Click **Save**
2. Go to **Resources**
3. **Select** the *root* / resource
4. In **Actions** dropdown, Choose **Deploy API**
5. On the Deploy API screen, Choose **Deployment Stage** as [New Stage] and enter the **Stage name** as Dev
6. Click **Deploy**
7. Note down the URL of the Dev stage (as API\_GATEWAY\_STAGE\_URL)



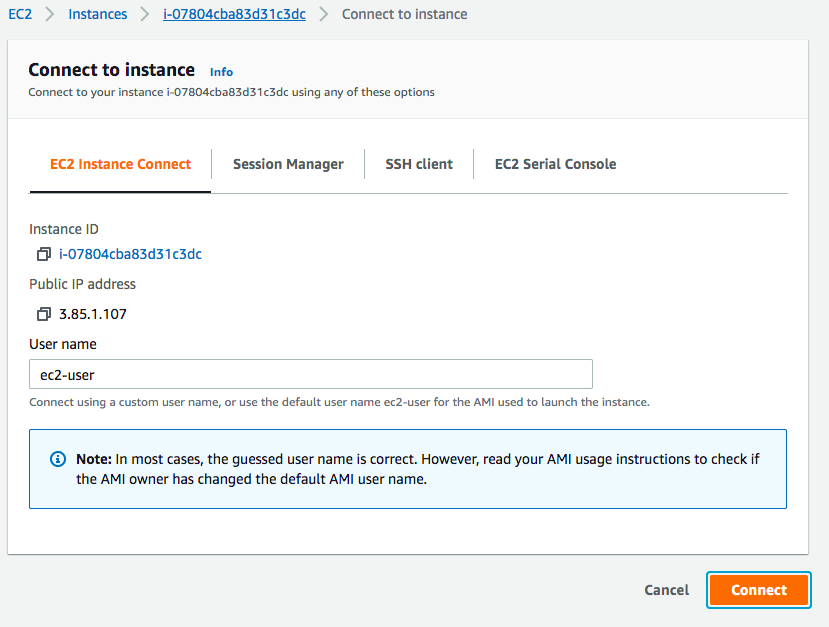
1. In the **AWS Console**, from **Services** menu - choose **EC2**
2. Click**Launch Instance** to start creation of EC2 instance
3. In **Step 1: Choose an Amazon Machine Image (AMI)** choose Amazon Linux 2 AMI (HVM) and click **Select**



1. On **Step 2: Choose an Instance Type** choose t2.micro instance and click **Review and Launch**



1. On **Step 7: Review Instance Launch**, check Network under Instance Details to ensure that you are launching the EC2 instance into the default VPC
2. Click **Launch**
3. Choose default options on **Select a Key Pair** popup. Enable (check) the check box I acknowledge that I have access to the selected private key file ....
4. Click **Launch Instances**
5. Wait until your EC2 instance is ready
6. Once the instance is launched, click the instance link
7. Select the instance in the **Instances** screen and click **Connect**
8. On the **Connect to instance** page, choose the defaults and click **Connect**



1. You can send requests to the REST API from the EC2 instance. Replace *API\_GATEWAY\_STAGE\_URL* with the path to the API Gateway Dev stage you noted down earlier.

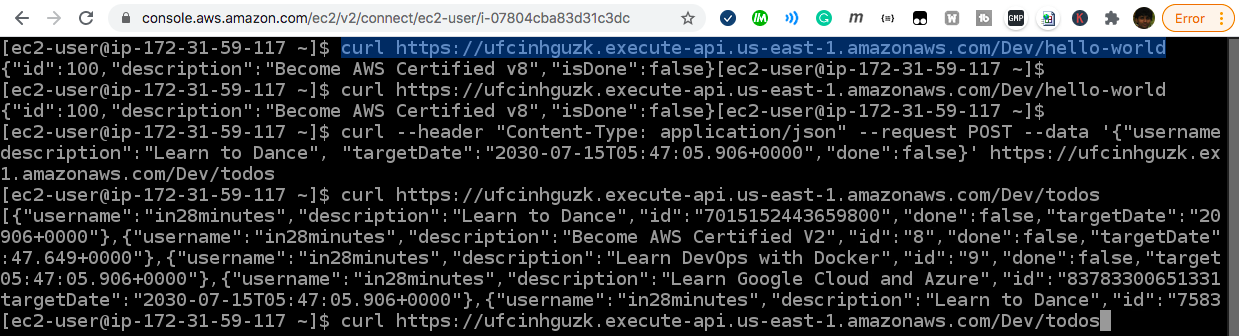
*curl API\_GATEWAY\_STAGE\_URL/hello-world*

*curl API\_GATEWAY\_STAGE\_URL/todos*

*curl --header "Content-Type: application/json" --request POST --data '{"username":"in28minutes", "description":"Learn to Dance", "targetDate":"2030-07-15T05:47:05.906+0000","done":false}' API\_GATEWAY\_STAGE\_URL/todos*

***Examples are shown below:***  
curl <https://ufcinhguzk.execute-api.us-east-1.amazonaws.com/Dev/hello-world>curl <https://ufcinhguzk.execute-api.us-east-1.amazonaws.com/Dev/todos>curl --header "Content-Type: application/json" --request POST --data '{"username":"in28minutes", "description":"Learn to Dance", "targetDate":"2030-07-15T05:47:05.906+0000","done":false}' <https://ufcinhguzk.execute-api.us-east-1.amazonaws.com/Dev/todos>

Result of execution is shown below:



Ref api-gateway-resource-policy.json

Clean up resources

You are done with your POC. Your task is to clean up all the resources you have created for your proof of concept. Ensure that you delete them in the right order.

1. Delete EC2 instance first. In the **AWS Console**, from **Services** menu - choose **EC2**. In the navigation pane, choose **Instances**. Select the instance, and choose **Actions**, **Instance State**, **Terminate**. Choose **Yes**, **Terminate** when prompted for confirmation.
2. Delete API Gateway - TodoRestAPI. In the **AWS Console**, from **Services** menu - choose **API Gateway**. On the **APIs** page, select the API. Choose **Actions**, and then choose **Delete**. Choose **Delete**. Confirm your selection on the confirmation dialog.
3. Delete all Lambda functions. In the **AWS Console**, from **Services** menu - choose **AWS Lambda**. On the **Functions** page, select all the functions. Choose **Actions**, and then choose **Delete**. Choose **Delete**. Confirm your selection on the confirmation dialog.

Delete DynamoDB items Table. In the **AWS Console**, from **Services** menu - choose **DynamoDB**. In the navigation pane, choose **Tables**.In the list of tables, choose ***todos***. Choose **Delete Table**. Confirm your selection on the confirmation dialog