# Pre-requisite steps for creating your serverless application

## Create your AWS free trial account

1. Visit <https://aws.amazon.com/free/start-your-free-trial/>
2. Click **Create your account** button.
3. Follow the instructions on the screen to create your account.
4. When you are asked for the account usage type, select **Personal**, unless you plan to reuse this account for Professional reasons.

Your account should complete set up within five minutes.

**[If this does not happen, please pair with someone to do this workshop]**

# Building your Serverless Application

During this workshop you will be building an application that allows you to play a trivia game with one or more players. The layout and gameplay is similar to Jeopardy™ , where players’ scores increases as they answer questions correctly and decrease when they answer questions incorrectly.

The static files for the website will be stored on Amazon S3. The APIs necessary will be created via Amazon API Gateway and powered by AWS Lambda. The data for the sessions will be stored in Amazon DynamoDB.

## Downloading website files

1. **Option 1 (if you have Git installed):** Open your terminal (on Mac) or use Windows Powershell (on Windows), and choose the directory under which you will place the files related to this workshop. Then either:
   1. Clone the github repository to the folder of your choice. We will use the **Documents** folder.

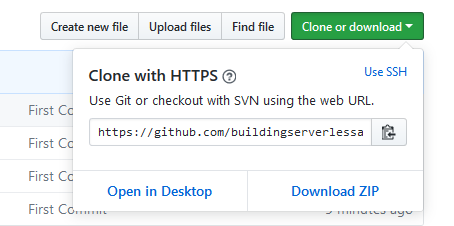
8c85903a4bb6:~ agathao$ cd Documents/

8c85903a4bb6:Documents agathao$ git clone https://github.com/buildingserverlessapps/TriviaApplication.git

* 1. There should be a **TriviaApplication-master** folder created. Go into it and then change into the **front-end** folder. This folder holds all the static files for the website which we will be uploading to S3.

8c85903a4bb6:TriviaApplication agathao$ cd front-end/

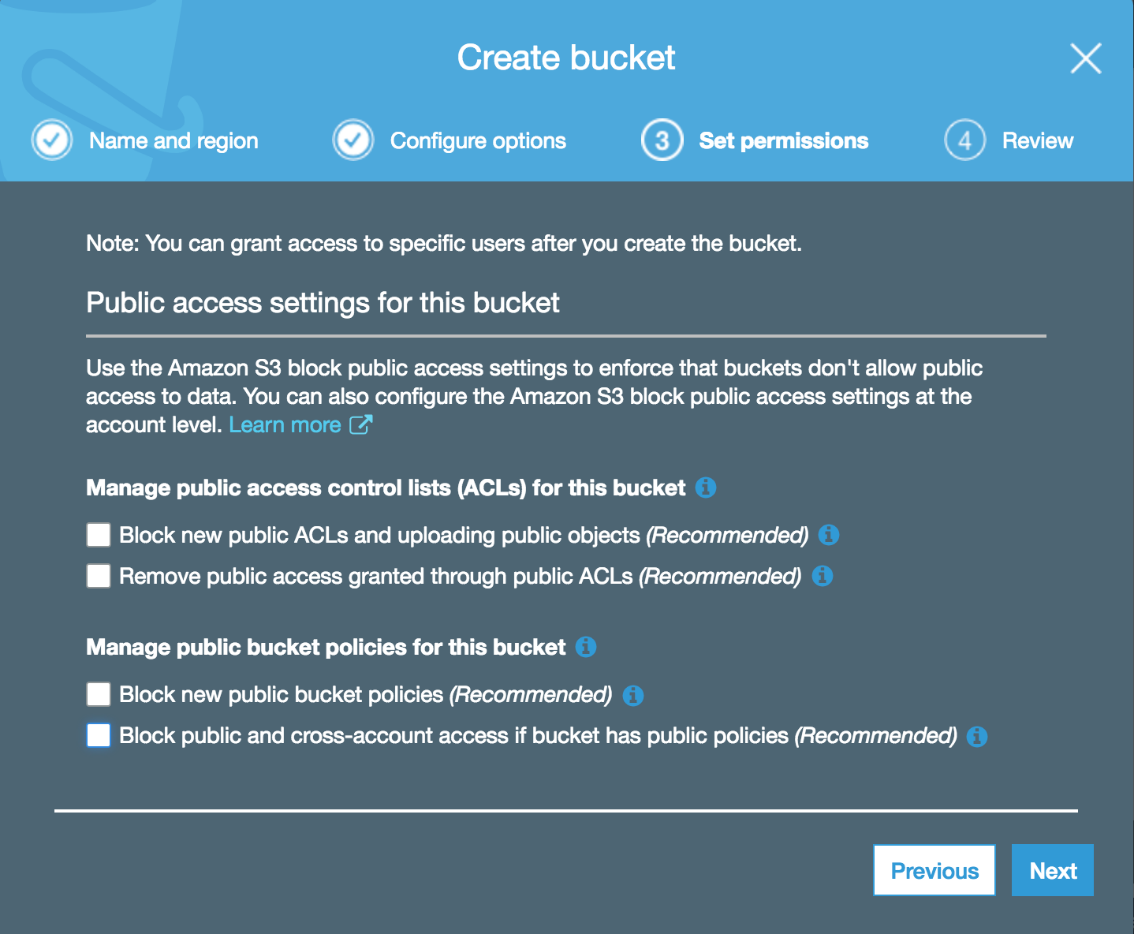
1. **Option 2 (Git is not installed):** If you don’t have git on your computer, you can download the repository as a zip file from Github. Then unpack it on your folder of choice. The repository link is: https://github.com/buildingserverlessapps/TriviaApplication



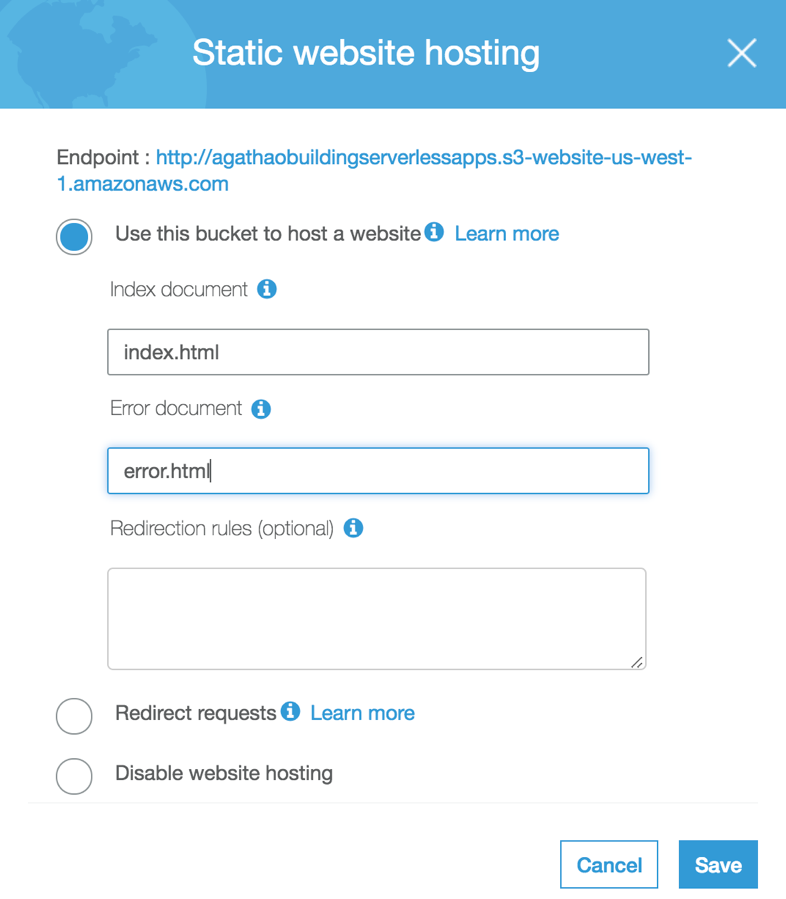
* 1. Unzip the downloaded file. There should be a **TriviaApplication-master** folder created. Using any file explorer application, go into it and then go into the **front-end** folder. This folder holds all the static files for the website which we will be uploading to S3.

## Storing the files on Amazon S3

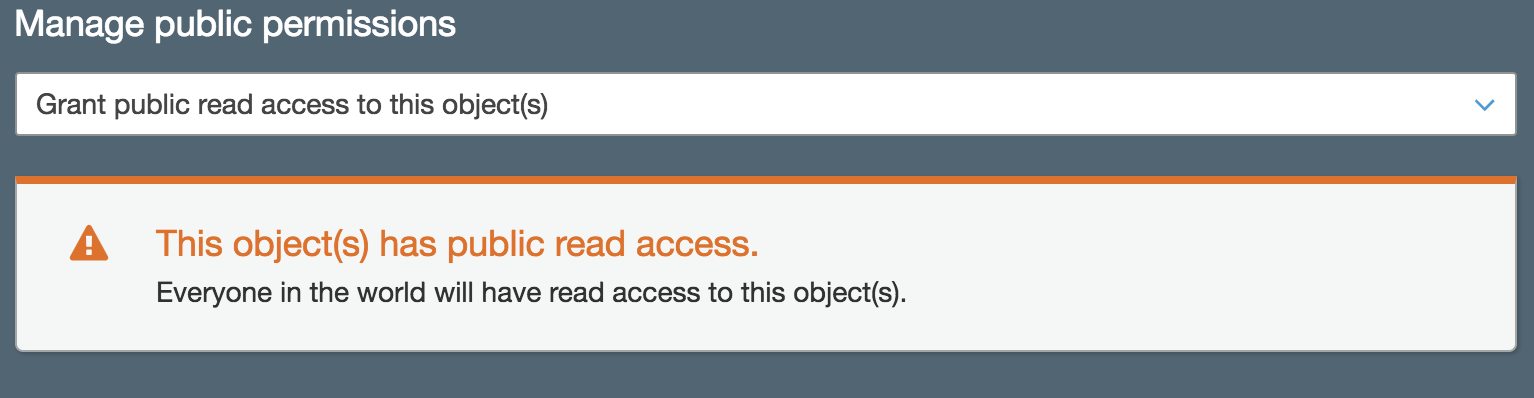
1. Now we will upload the contents of the **staticFiles** folder to Amazon S3. Go to the S3 console at <https://s3.console.aws.amazon.com/>. You will need to sign in if you haven’t already.
2. Click on **Create bucket** button.
   1. Choose a bucket name. Do not forget that it should be unique across all of AWS. We suggest <name>buildingserverlessapps. E.g.: agathaoliveirabuildingserverlessapps
   2. For Region, select **US West (N. California)**.
      1. A region is a separate geographic area.
      2. Within a region, there are multiple Availability Zones (AZ). These are isolated locations connected to each other via low latency links.
      3. When your data is stored in a region, it is replicated to at least 3 AZs. This means that even if 2 AZs go down (which is very unlikely), your data is still available from the 3rd.
   3. Click on **Next**. For this workshop we will not be setting any of the properties on this screen. However, you can click on the **Learn more** links next to them to understand what they do.
   4. Click on **Next**. For the permissions, we want the files on the bucket to have public permissions since they will be accessible to anyone. Therefore, **unselect all checkboxes on this screen**.



* 1. Click on **Next** and then on **Create Bucket.**
  2. Now that your bucket is created, click on your bucket name and then go to the **Properties** tab at the top.
  3. Click on **Static website hosting** and select **Use this bucket to host a website**. Then configure the index and the error documents as per the below. Also take note of the Endpoint that you are given. Lastly, click on save:



1. There are many ways of uploading your files. We will use the simplest of them: Go back to the **Overview** tab and click on **Upload**.
   1. On your computer, select all files inside the **front-end** folder (from step 1) and drag them to the upload screen. (*Do not copy the folder itself. You should copy the files that are within the folder to the upload screen*)
   2. Click **Next**. Under **Manage public permissions**, select **Grant public read access to this object(s).**



* 1. Click **Next**. Leave the Standard option selected.
  2. Click on **Next** again and then on **Upload**.

*If you uploaded the files without giving them public read access, your website will not run. You can still add public access after uploading the files by doing the following: select all of the files, click the “Actions” button, click “Make public” and then select the “Make Public” button in the dialog box.*

1. When your upload is complete, your files will be accessible via the S3 bucket link. If you don’t remember your endpoint link, you can access it again by going to the **Properties** tab and clicking on **Static** **Website hosting** to see it.
2. Open your browser and go to the endpoint URL to see the site running.



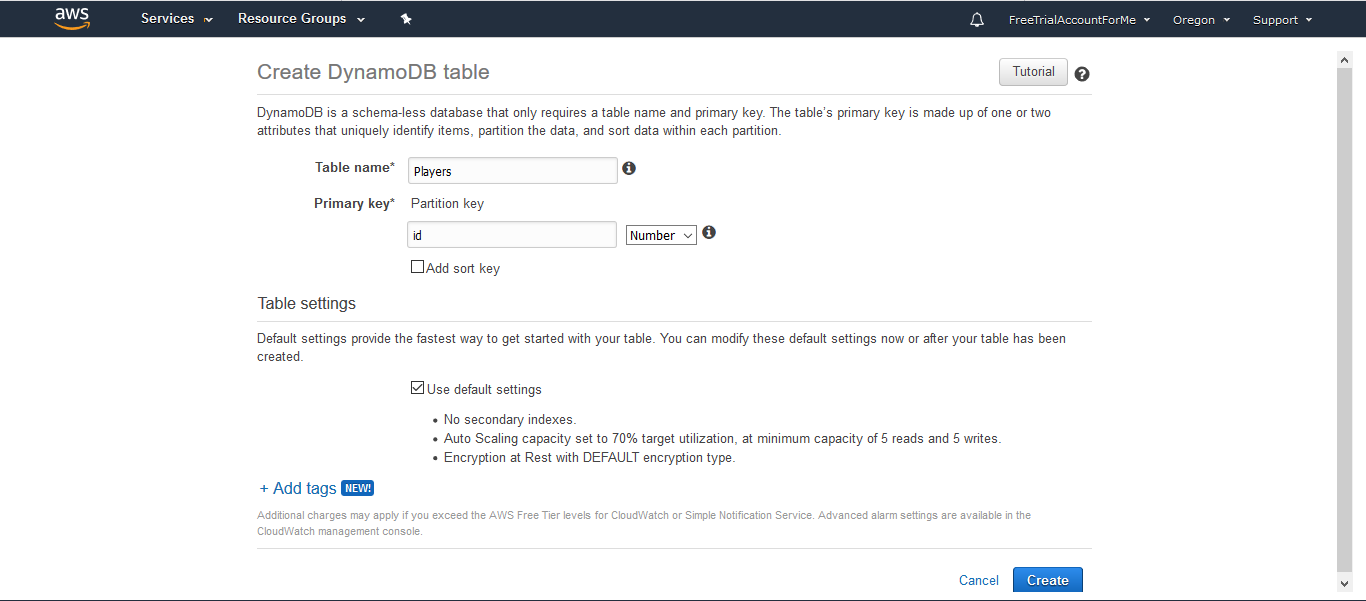
This initial version displays the site using mocked data with just the board for the trivia application. You will be able to see the categories and tiles to select from, but you won’t be able to click them to see the questions.

## DynamoDB setup

You will now create two DynamoDB tables for your application. The first table will store all the player information and the second table will store all the question information for the application. Below are the steps.

### Players Table

1. Go to the Dynamo DB console at <https://us-west-1.console.aws.amazon.com/dynamodb>. Confirm at the top right that your region is displayed as **N. California.**
2. On the DynamoDB homepage, go ahead and read about DynamoDB and click **Create table** when you are ready.
3. Create a table with the below details [fields are **case-sensitive**]:
   1. **Table name**: Players
   2. **Primary key\***: id
   3. On the drop down next to it, select **Number**
   4. Leave **Add sort key** as unchecked.



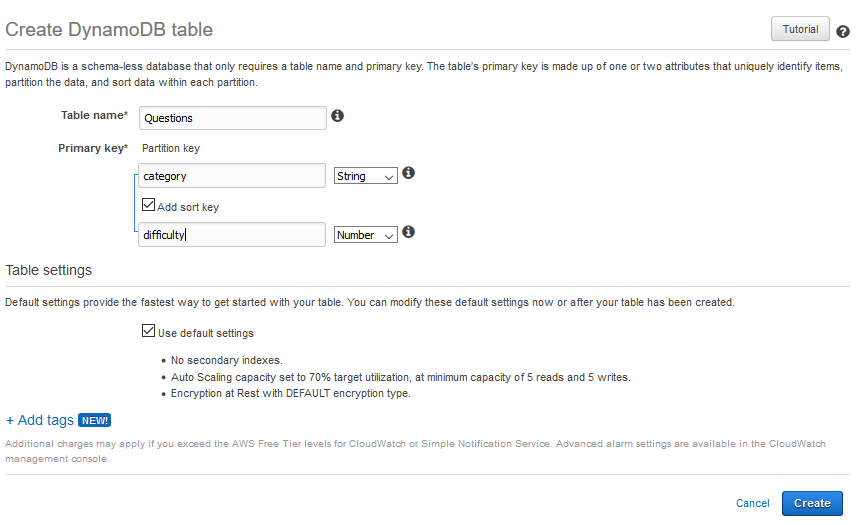
\*Note: If you created the table with a primary key that is not **Number**, the Lambda to populate the table will not work. You will need to delete and re-create the table. Make sure your Primary Key is **Number.**

1. Leave everything else on the page as-is and scroll to the bottom. Click **Create** and wait for DynamoDB to create your table. This may take a few minutes.

Congratulations! You have created your first DynamoDB table. We will now create the second DynamoDB table.

### Questions Table

1. On the left side of the screen, click the **Create table** button.
2. Create a table with the below details [fields are **case-sensitive**]:
   1. **Table name**: Questions
   2. **Primary key\*:** category
   3. On the drop down next to it, select **String** (this is the default value)
   4. Check the **Add sort key** box
   5. Type into the box that appears for the sort key: difficulty
   6. On the drop down next to it, select **Number** (NOT String)
   7. In the Table settings section, keep the **Use default settings** box as checked.



\*Note: If you created the table without both parts of the primary key, the Lambda function to populate the table will not work. You will need to delete and re-create the table. Make sure your Primary Key is set using the instructions below, identical to the image above.

1. Leave everything else on the page as-is and scroll to the bottom. Click **Create** and wait for DynamoDB to create your table. This may take a few minutes.

Next, we will create our AWS Lambda functions.

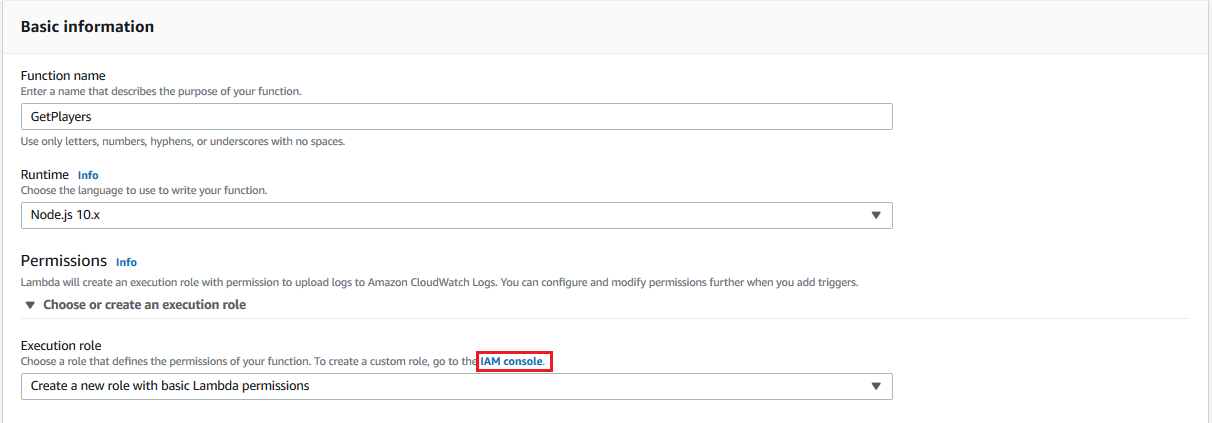
## AWS Lambda setup

For this setup, make sure you have any pop-up blockers turned off. You will create four AWS Lambda functions through the console.

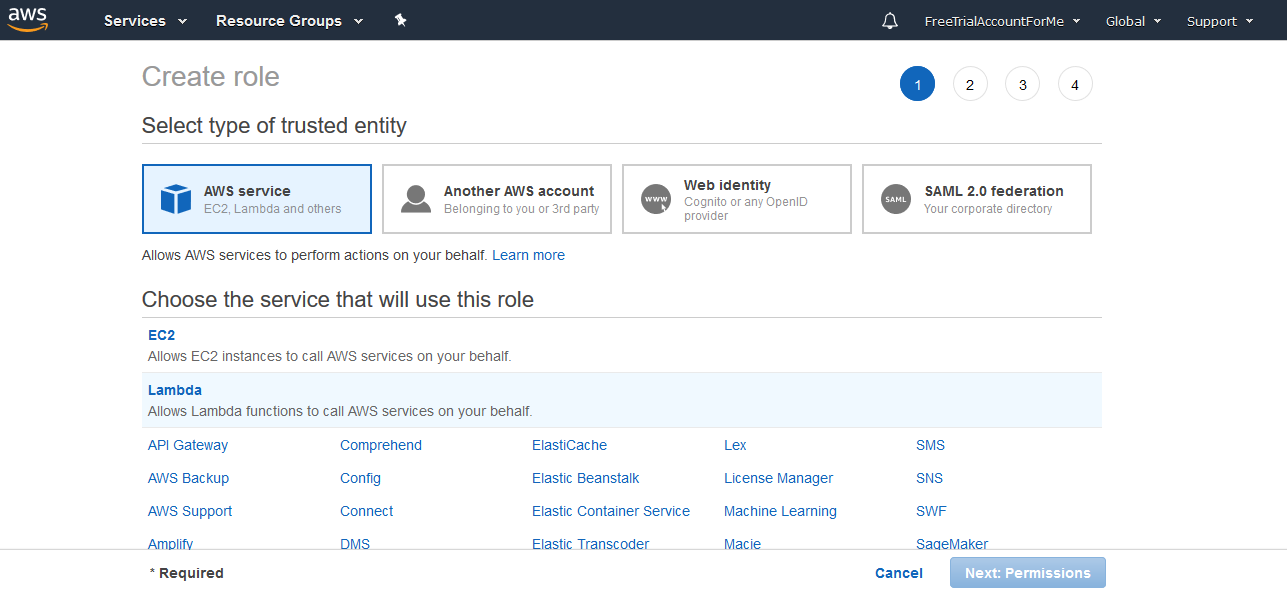
### GetPlayers

The first Lambda you create will contain the code to fetch all the player data from the Players DynamoDB. This function and the all of the functions in this exercise will use a **Node.js 10.x** runtime environment.

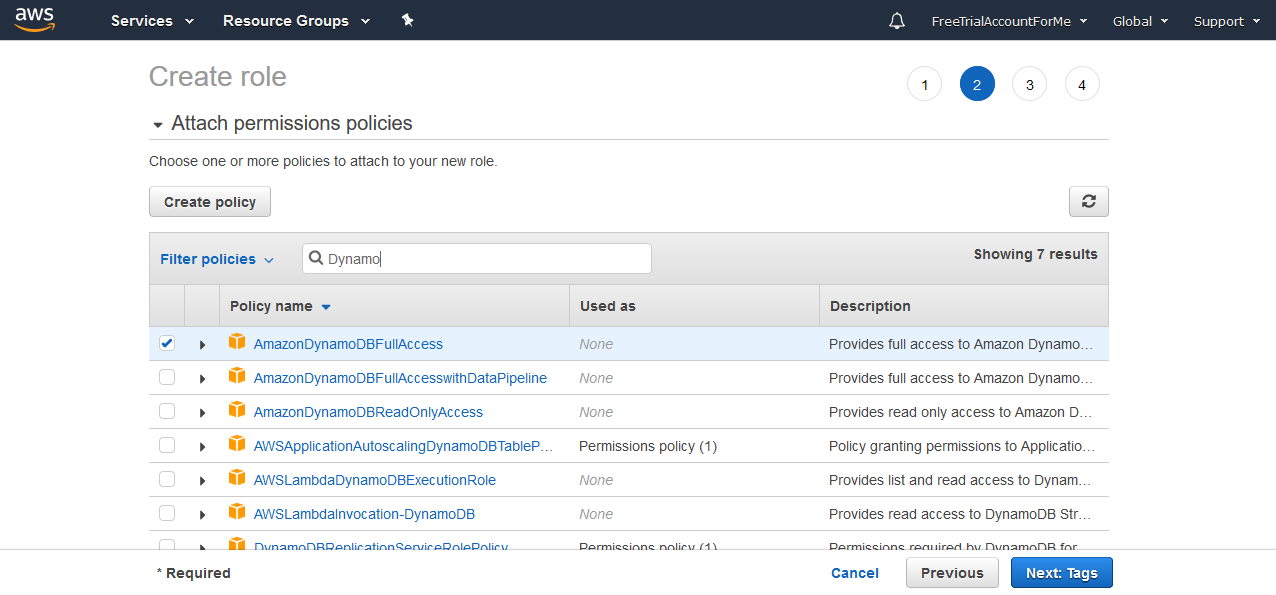
1. Open the AWS Lambda console at https://us-west-1.console.aws.amazon.com/lambda/home?region=us-west-1 Confirm at the top right that your region is displayed as **N. California.**
2. Click the **Create Function** button.
3. Select the **Author from scratch** card.
4. Fill out the form for the Lambda with the following details:
   1. **Name**: GetPlayers
   2. **Runtime**: Node.js10.x*(not 6.10 and not 8.10)*
5. Below **Execution role,** you will see the statement, “**Choose a role that defines the permissions of your function. To create a custom role, go to the** [**IAM console**](https://us-west-2.console.aws.amazon.com/iam/home?#/roles$new).”
6. Select the **IAM console** link in that line to create a new IAM role that we will use for this lambda



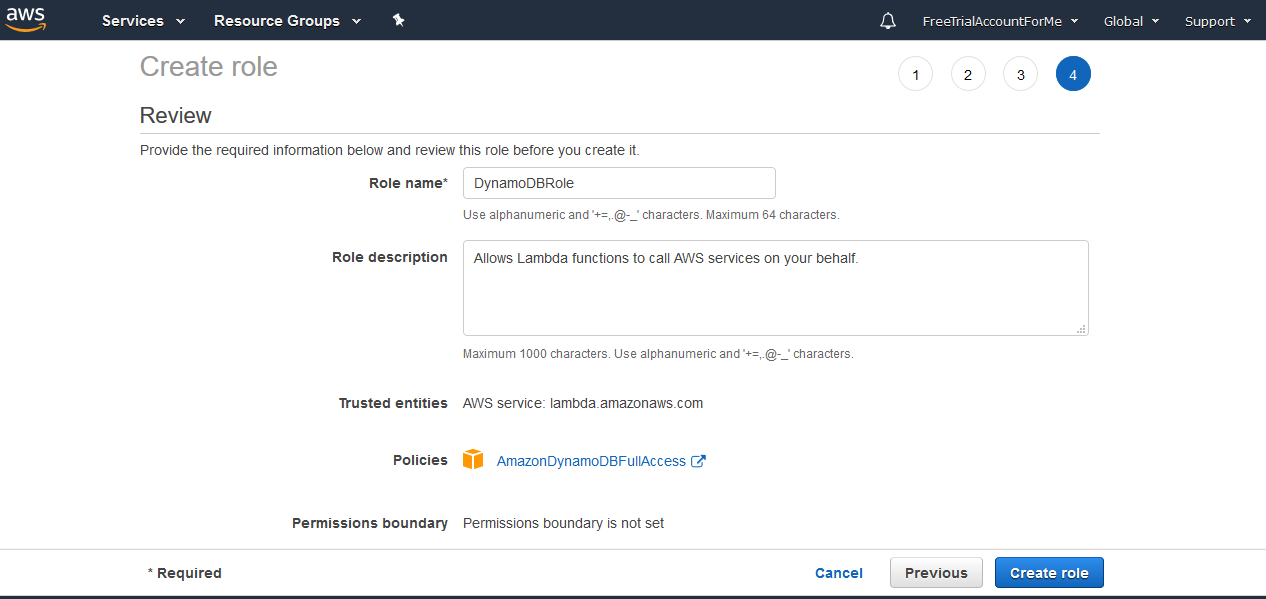
Your IAM (Identity Access Management) Role is what defines the permissions for your lambda, such as which AWS services your lambda will be able to access. You see the following window open in your browser:



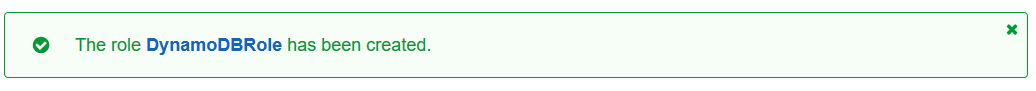
1. In the **Choose the service that will use this role,** select **Lambda** and click the **Next: Permissions** button
2. In the Search bar next to **Filter policies,** type Dynamo to filter for the DynamoDB policy that we need
3. Select the first policy- **AmazonDynamoDBFullAccess.** This will give our lambda function full permissions to DynamoDB to create, read, update and delete items in the tables that we created.



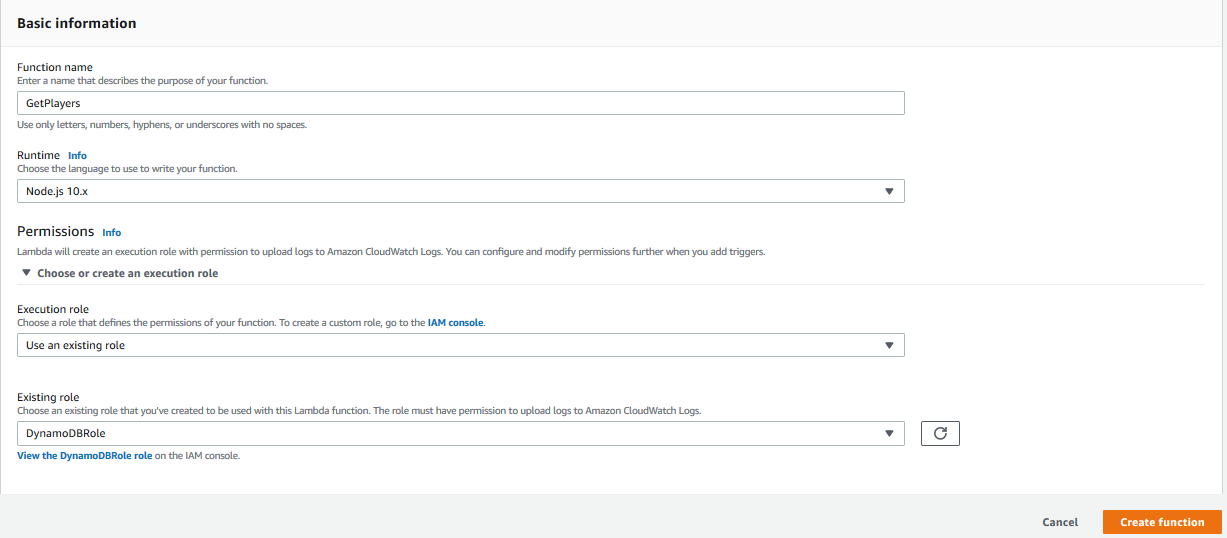
1. Select the **Next:Tags** button. Leave everything on this page the same and click the **Next:Review** button.
2. Fill in the below details and click **Create Role** to complete the creation of your IAM role.
   1. Role Name: DynamoDBRole
   2. Role description: Allows Lambda functions to call AWS services on your behalf**.** (leave as-is)



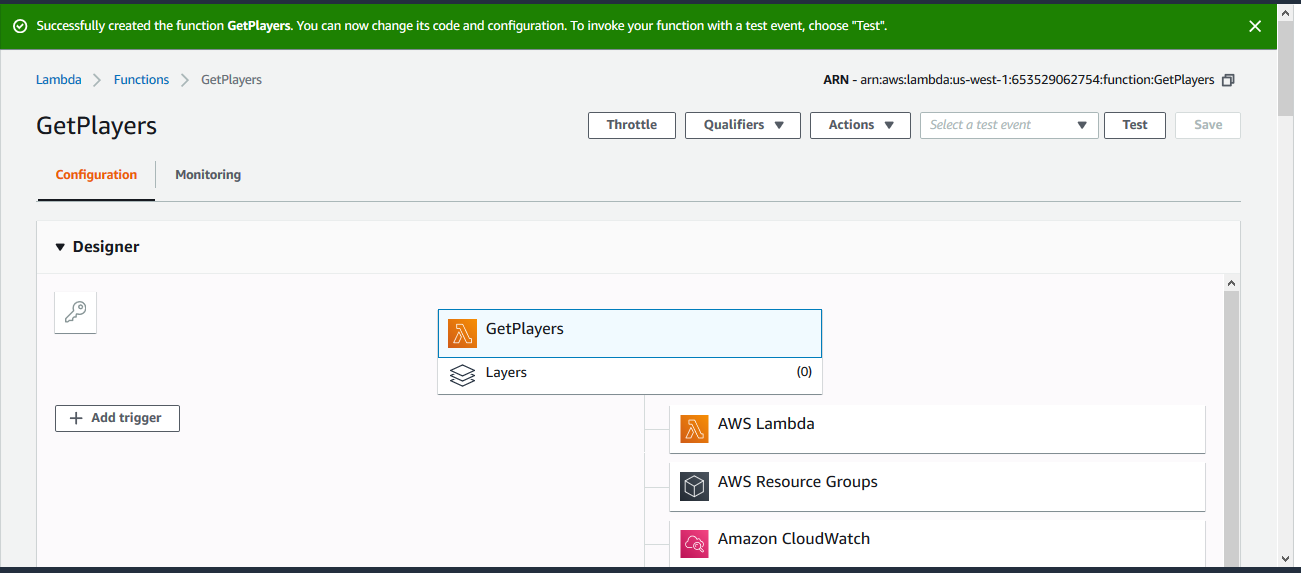
1. You should see a confirmation that your role was created at the top of the screen.



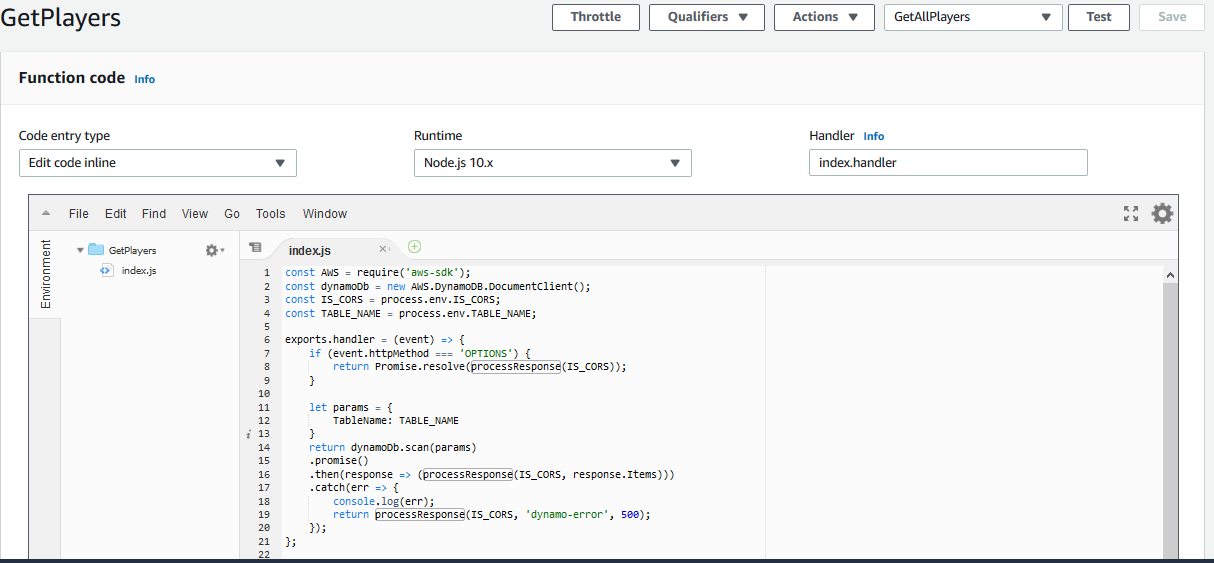
1. You can exit this window and navigate back to the window where you were creating your Lambda function.
2. On the lambda creation screen, in the **Execution Role** dropdown, you can select **Use an existing role**
3. In the **Existing Role** dropdown, select **DynamoDBRole,** which we just created in the steps above



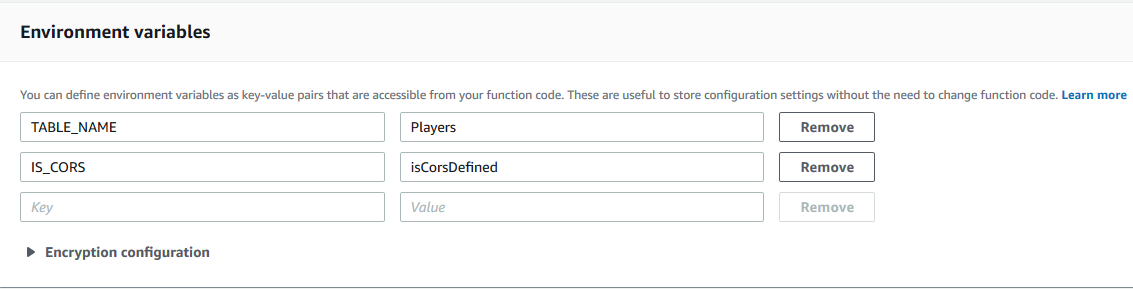
1. Go ahead and continue the Lambda setup by clicking **Create Function**. You should now see your Lambda’s dashboard page.



1. Scroll down to the Function code section. You will be editing the index.js file that is already open. Go ahead and delete any code that is already pre-filled in the editor.
2. On a new tab on your browser, navigate to <https://github.com/buildingserverlessapps/TriviaApplication/blob/master/lambda/functions/getPlayers/index.js>  
   The code at this link contains the logic to retrieve all the session data in Players table through a DynamoDB scan call. The scan operation returns one or more items and item attributes by accessing every item in a table. The code sends a success HTTP response along with all the data, once the scan is complete.
3. Copy the code from the file and paste it into the AWS Lambda function code editor
4. Go to **File > Save** to save your changes.



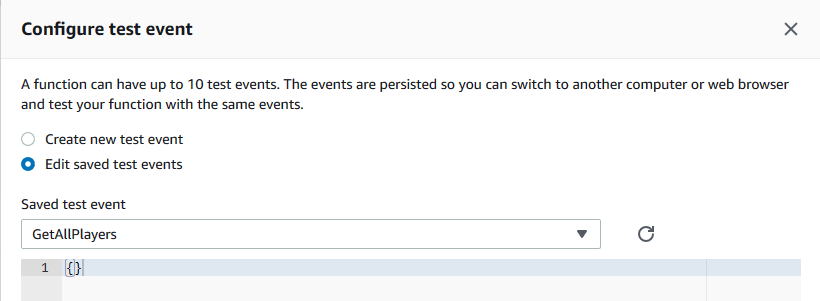
1. Scroll down to the **Environment variables** section on the Lambda detail page. You want to add two Environment variables.
   1. **Key**: TABLE\_NAME, **Value**: Players
   2. **Key**: IS\_CORS, **Value**: isCorsDefined



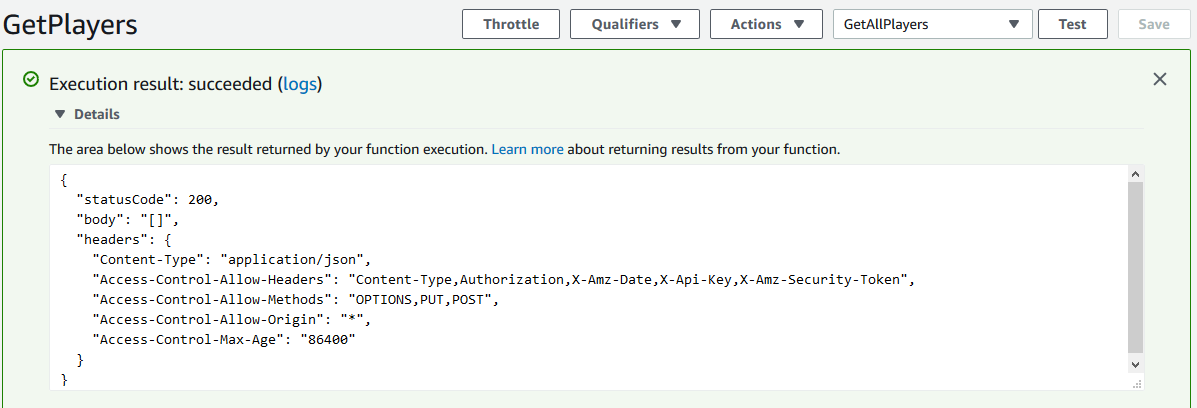
1. In **Basic settings** increase the timeout from 3 seconds to 30 seconds. This will give your function more time to run and get all the sessions from the DynamoDB table.
2. Keep everything else as-is, and then click the Orange **Save** button at the top of the screen.

You have now finished creating your lambda function. We will quickly test that the function works by creating a test event.

1. Click on the **Test** button on the top right.
2. From this screen, you can send test requests to a Lambda function.
   1. In **Event name** type GetAllPlayers.
   2. You can use the **Hello World** Event template. Replace all the content on the textbox below with **{}**. This is because for our case, the lambda function does not need any inputs to execute.



1. Click on **Create**.
2. You should see your test name appear in the dropdown next to the Test button. Click on **Test** to run your test event and trigger the code. Since we have not created players yet, in the response body, will see an empty response.



Congratulations, you created your first lambda function!

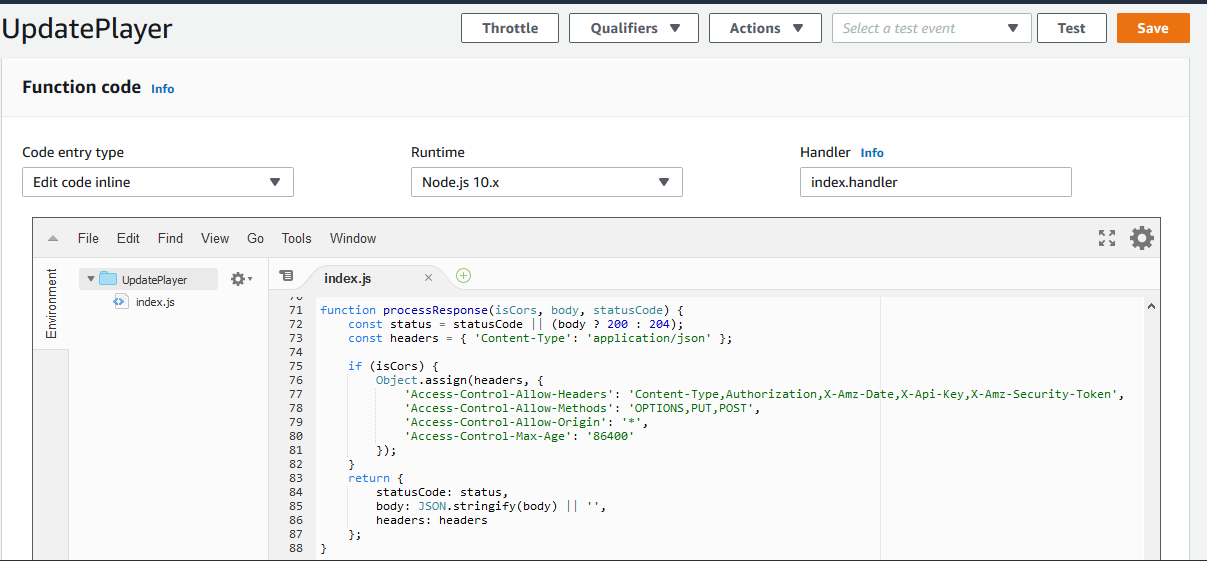
### UpdatePlayer

This second Lambda will be used to update player data in the DynamoDB **Players** table.

1. Open the AWS Lambda console at <https://us-west-1.console.aws.amazon.com/lambda/home?region=us-west-1#/functions> . Confirm at the top right that your region is displayed as **N. California**.
2. Click the **Create Function** button.
3. Select the **Author from scratch** card.
4. Fill out the form for the Lambda with the following details:
   1. **Name**: UpdatePlayer
   2. **Runtime**: Node.js 10.x*(not 6.10 and not 8.10)*
   3. **Execution role**: Use an existing role
   4. **Existing role**: DynamoDBRole
5. Continue the Lambda setup by clicking **Create Function**.
6. Scroll down to the Function code section. You will be editing the index.js file that is already open. Go ahead and delete any code that is already pre-filled in the editor.
7. On a new tab on your browser, navigate to <https://github.com/buildingserverlessapps/TriviaApplication/blob/master/lambda/functions/updatePlayer/index.js>

The code at this link contains the logic to update an existing item in the **Players** table. A DynamoDB update call is performed in order to indicate that an item should be updated. The code sends a success HTTP response once the player item has been updated in DynamoDB.

1. Copy the code from the file and paste it into the AWS Lambda function code editor.
2. Go to **File > Save** to save your changes.



1. Setup the **UpdatePlayer** function with the same **Environment Variables** and the same **Basic Settings** as **GetAllSessions**:
   1. **Key**: TABLE\_NAME, **Value**: Players
   2. **Key**: IS\_CORS, **Value**: isCorsDefined
2. In **Basic settings** increase the timeout from 3 seconds to 30 seconds.
3. Then, click the orange **Save button** at the top.

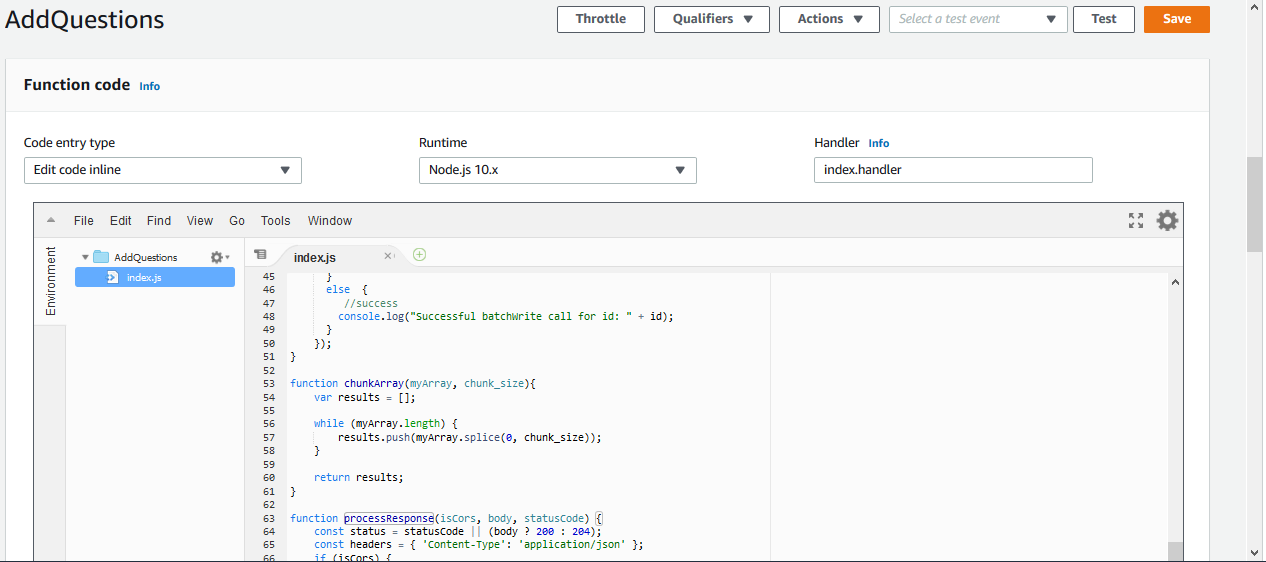
The next Lambda you create will contain the code to table populate the **Questions** DynamoDB table that you created.

### AddQuestions

1. Open the AWS Lambda console at <https://us-west-1.console.aws.amazon.com/lambda>. Confirm at the top right that your region is displayed as **N. California**
2. Select **Create a function**
3. Select the first card titled **Author from scratch**
4. Fill out the form for the Lambda with the following details:
   1. **Name**: AddQuestions
   2. **Runtime**: Node.js 10.x*(not 6.10 and not 8.10)*
   3. **Execution role**: Use an existing role
   4. **Existing role**: DynamoDBRole
5. Continue the Lambda setup by clicking **Create Function**.
6. Scroll down to the Function code section. You will be editing the index.js file that is already open in the editor. Go ahead and delete any code that is already pre-filled in the editor.
7. On a new tab on your browser, navigate to this link- <https://github.com/buildingserverlessapps/TriviaApplication/blob/master/lambda/functions/addQuestions/index.js> .

The code at this link contains the logic to populate data into your DynamoDB **Questions** table via batch-write calls. The batch-write operation puts multiple items into a table. The code sends a success HTTP response once the process is complete.

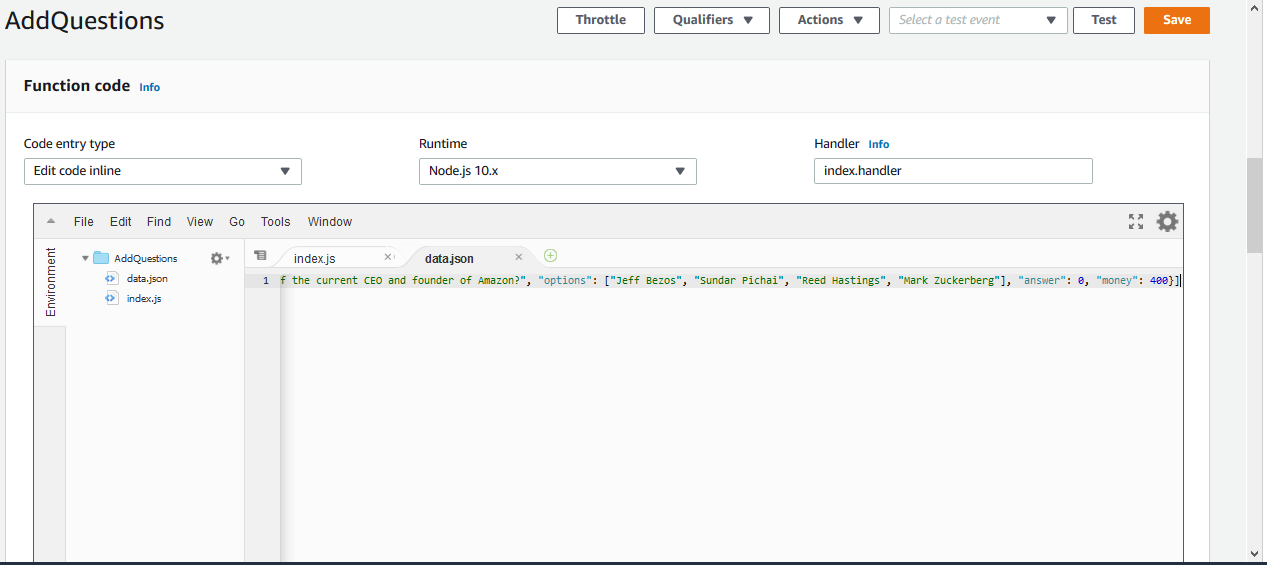
1. Copy the code from the file and paste it into the AWS Lambda function code editor.
2. In the editor, go to **File > Save** to save your changes.



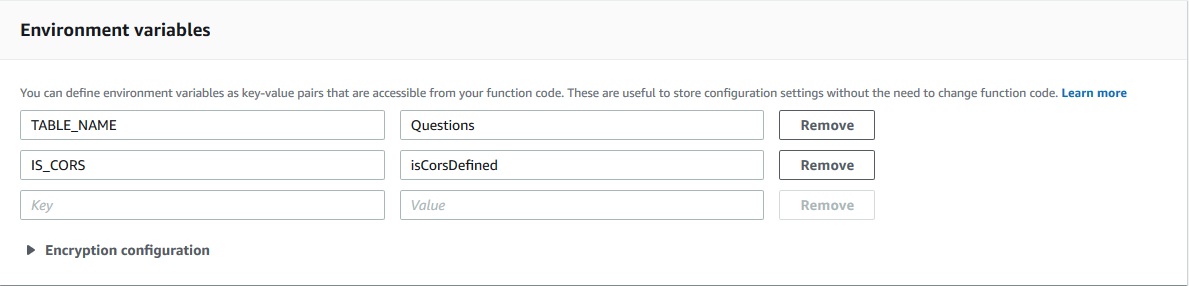
1. You will also create a new file in the AddQuestions directory. At the top of the Lambda editor, go to **File > New File**.
2. On a new tab on your browser, navigate to <https://github.com/buildingserverlessapps/TriviaApplication/blob/master/lambda/functions/addQuestions/data.json>.

The file at this link contains all the data for each question that the application will use in JSON format.

1. Copy the data from that file and paste it into the AWS Lambda function code editor for the new file that you have open.
2. In the editor, go to **File > Save** to save your changes. Use the below details to save the file:
   1. **Filename**: data.json
   2. Make sure **AddQuestions** is selected
   3. **Folder**: /
3. Your Function code section should look similar to the below



1. Scroll down to the **Environment variables** section on the Lambda detail page. You want to add two Environment variables:
   1. **Key**: TABLE\_NAME, **Value**: Questions
   2. **Key**: IS\_CORS, **Value**: isCorsDefined

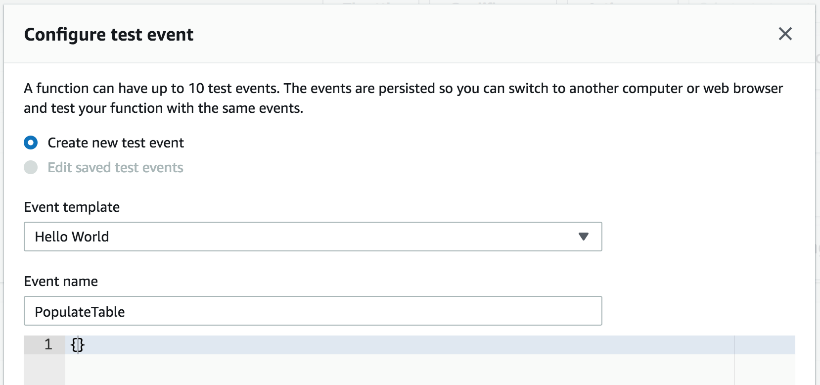


1. Scroll to the **Basic settings** section and **increase the timeout** from 3 seconds to 30 seconds. This will give your function more time to run and populate the DynamoDB table.
2. Keep everything else as-is, and then click the Orange **Save** button at the top of the screen.

And that’s all! You have created your first Lambda function.

Since this function is purely for populating your table and it is not exposed to users, we will execute it right from the AWS Lambda console.

1. Click on the **Test** button on the top right.
2. From this screen, you can send test requests to your Lambda function.
   1. On **Event name** type **PopulateTable.**
   2. You can use the **Hello World** Event template. Replace all the content on the textbox below with {}. This is because for our case the lambda function does not need any inputs to execute. Your screen should look as per below:



1. Click on **Create.**
2. You should see your test name “PopulateTable” appear in the dropdown next to the Test button. Click on the **Test** button to run your test event and execute the code.

Your table should be populated now. You can verify that by going visiting your DynamoDB table: https://us-west-1.console.aws.amazon.com/dynamodb/home?region=us-west-1#tables:selected=Questions;tab=items

Let’s create the last lambda function for our application.

### GetQuestion

The last lambda you create will contain the code to a retrieve data from the **Questions** DynamoDB table.

1. Open the AWS Lambda console at https://us-west-1.console.aws.amazon.com/lambda/home?region=us-west-1 Confirm at the top right that your region is displayed as **N. California.**
2. Click the **Create Function** button.
3. Select the **Author from scratch** card.
4. Fill out the form for the Lambda with the following details:
   1. **Name**: GetQuestion
   2. **Runtime**: Node.js 10.x*(not 6.10 and not 8.10)*
   3. **Execution role**: Use an existing role
   4. **Existing role**: DynamoDBRole
5. Continue the Lambda setup by clicking **Create Function**. You should now see the Lambda’s detail page.
6. Scroll down to the Function code section. You will be editing the index.js file that is already open. Go ahead and delete any code that is already pre-filled in the editor.
7. On a new tab on your browser, navigate to <https://github.com/buildingserverlessapps/TriviaApplication/blob/master/lambda/functions/getQuestion/index.js>  
   The code at this link contains the logic to retrieve a question from the Questions table through a DynamoDB query call. The query operation returns one or more items and item attributes that matches the passed-in category and difficulty value. The code sends a success HTTP response along with all the data, once the query is complete.
8. Copy the code from the file and paste it into the AWS Lambda function code editor
9. Go to **File > Save** to save your changes.
10. Scroll down to the **Environment variables** section on the Lambda detail page. You want to add two Environment variables.
    1. **Key**: TABLE\_NAME, **Value**: Questions
    2. **Key**: IS\_CORS, **Value**: isCorsDefined
11. In **Basic settings** increase the timeout from 3 seconds to 30 seconds. This will give your function more time to run and get all the sessions from the DynamoDB table.
12. Keep everything else as-is, and then click the Orange **Save** button at the top of the screen.

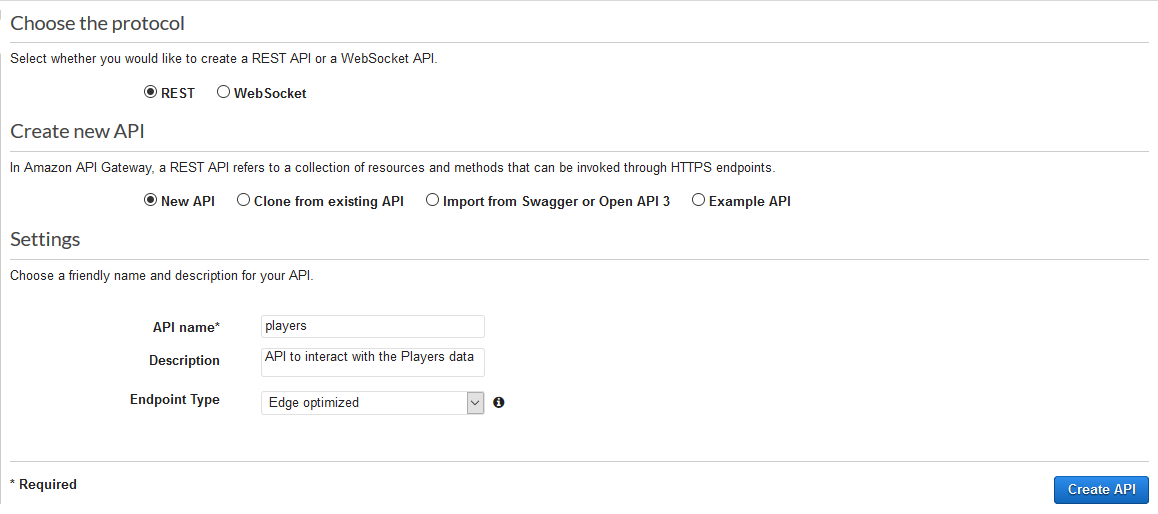
You have finished creating your lambda functions.

## Amazon API Gateway

Now that you have your Lambdas, you will now create the 2 APIs that will trigger the functions.

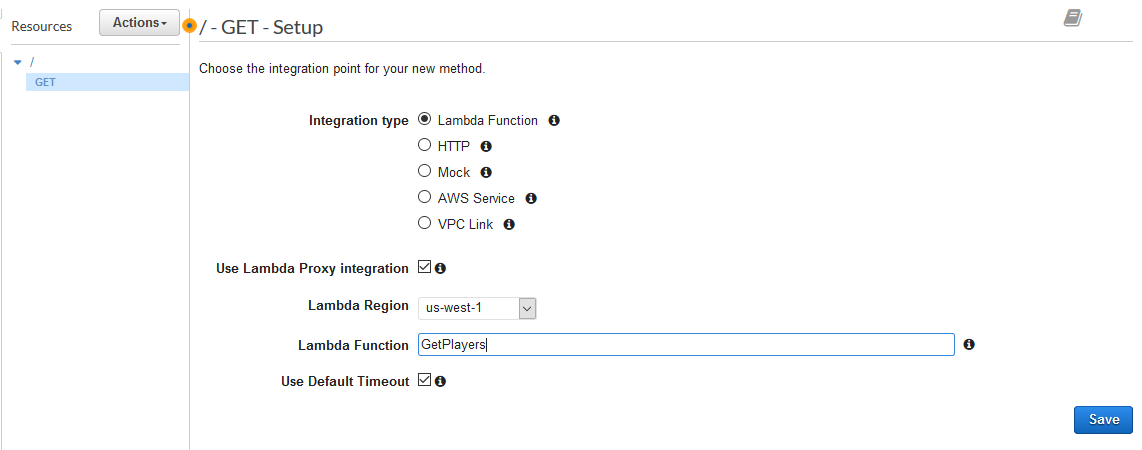
### Get Players API

1. Open the Amazon API Gateway console at <https://us-west-1.console.aws.amazon.com/lambda/home?region=us-west-1#/functions>. Confirm at the top right that your region is displayed as **N. California**.
2. Click the **Get Started** button to start building your API.
3. Keep the **REST** protocol selected
4. Select the **New API** radio button.
5. Enter the below details for your API and then click on **Create API**:
   1. **API name**: players
   2. **Description**: API to interact with the Players data
   3. **Endpoint Type**: Edge optimized



You will be taken to the API dashboard where you can define your methods and resources for your API.

1. With **/** selected, click the **Actions** Button and select **Create Method**.
2. You will see that a dropdown menu element gets added to the tree below **/**. In that dropdown, select **GET**, and then click the Check mark next to it.
3. Here is where we set up the API integration with Lambda. Fill out the below details:
   1. Select the **Lambda Function** radio button as the **Integration Type**
   2. Select the **Use Lambda Proxy integration** checkbox
   3. Select **us-west-1** as the **Lambda Region**
   4. In **Lambda Function**, enter the name of the function that we created earlier that gets all the sessions in the table – GetPlayers
   5. Leave **Use Default Timeout** as selected



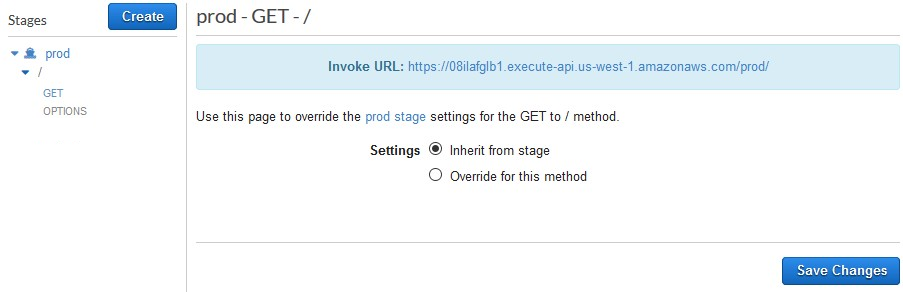
1. Click **Save**.
2. When you see the **Add Permission to Lambda Function** pop-up, select **Ok**.
3. With **GET** (or **/**) selected in your Resources tree, go ahead and click the **Actions** button again.
4. Click **Enable CORS:**
   1. Leave all the options as-is on the **Enable CORS** page
   2. Click E**nable CORS and replace existing CORS headers**
   3. Click **Yes, replace existing values** on the next dialog



1. Lastly, you will deploy your API. Click on **Actions** then, **Deploy API**. Fill in the details for your deployment stage in the dialog box:
   1. **Deployment stage**: [New Stage]
   2. **Stage name**: prod
   3. **Stage description**: Production deployment stage
   4. **Deployment description**: Initial deployment
2. Click **Deploy.**

You have now created your first API!

While you are in the Stages page for your API, if you select the small arrow next to **prod**, you should be able to see your Resource Tree you created earlier. Go ahead and click the **GET** Method that you created. You should then be able to see the **Invoke URL** shaded in blue.



Now go ahead, and open that link in a new tab on your browser to test your API. You should be able to see a JSON with all the players returned (this will be empty for now).

### Update players resource

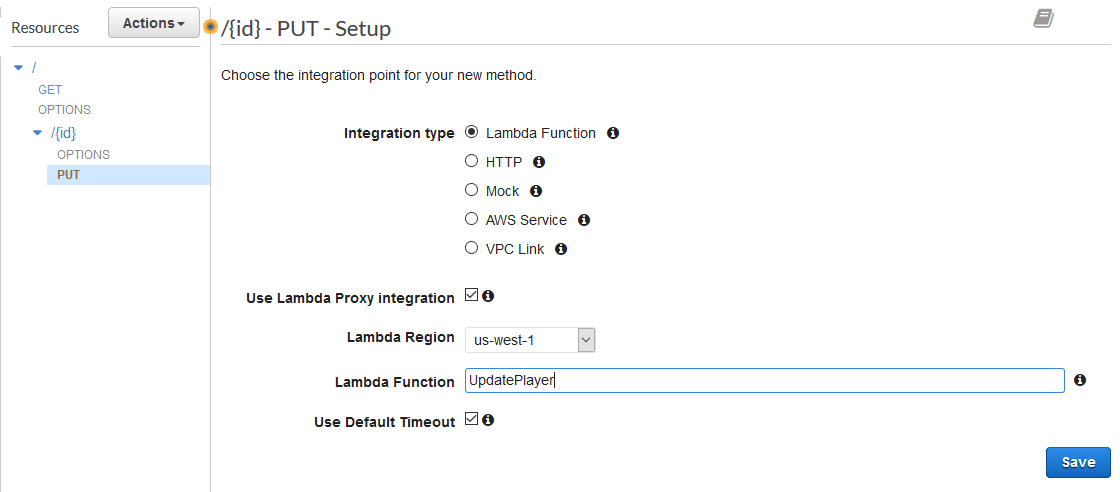
You will now create another resource to update an item in the **Players** table.

1. Click on the **players** API on the left (beneath the **API** heading).
2. Click the **Actions** Button and on the dropdown, select **Create Resource**
3. Supply the following details:
   1. Leave **Configure as proxy resource** unchecked.
   2. **Resource Name**: id
   3. **Resource Path**\*: {id}
   4. Select the **Enable API Gateway CORS** checkbox.

*\*Include the curly brackets around* ***id*** *in the Resource Path*

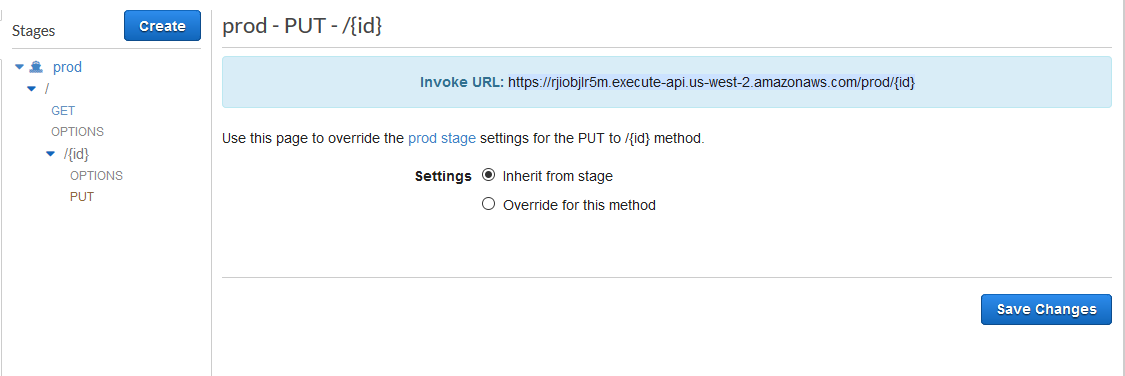


1. Click the **Create Resource** button.
2. With **id** selected in the Resources tree, click the **Actions** button again, and select **Create Method**
3. You will see that a dropdown menu element gets added to the tree below **OPTIONS**. In that dropdown, select **PUT**, and then click the Check mark next to it.
4. Here is where we set up the API integration with Lambda. Fill out the below details:
   1. Select the **Lambda Function** radio button as **Integration Type.**
   2. Select the **Use Lambda Proxy integration** checkbox
   3. Select the **Lambda Region** as us-west-1
   4. In **Lambda Function**, enter the name of the first function that we created earlier that gets all the sessions in the table – UpdatePlayer
   5. Leave **Use Default Timeout** as selected



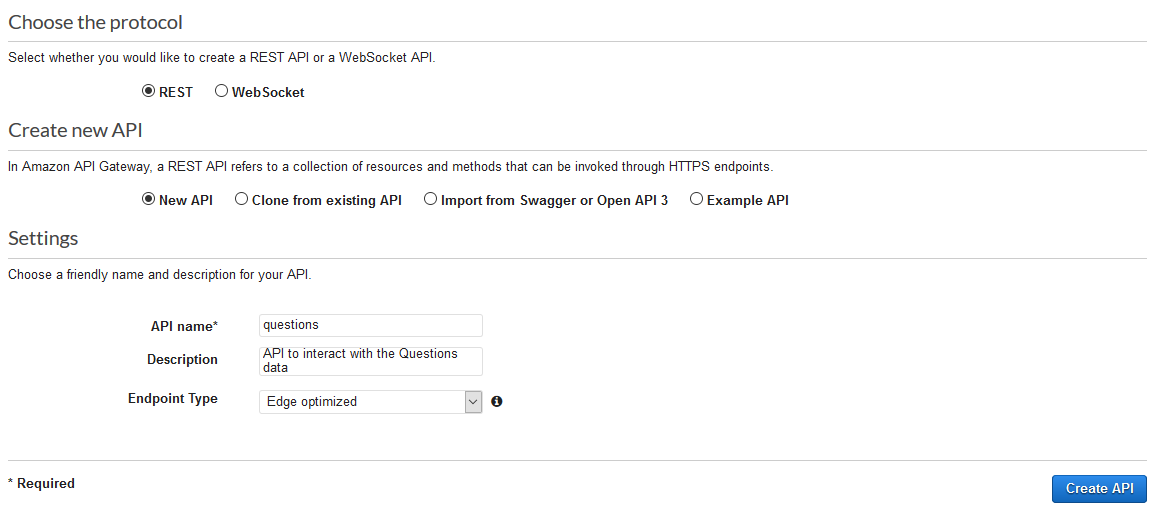
1. Click **Save**.
2. When you see the **Add Permission to Lambda Function** pop-up, select **Ok**.
3. With **PUT** (or **{id}**) selected in your Resources tree, go ahead and click the **Actions** button again.
4. Click **Enable CORS:**
   1. Leave all the options as-is on the **Enable CORS** page
   2. Click **Enable CORS and replace existing CORS headers**
   3. Click **Yes, replace existing values** on the next dialog and you should see the configuration confirmation
5. Lastly, you will deploy your API. Click on **Actions** then, **Deploy API**. Fill in the details for your deployment stage in the dialog box:
   1. **Deployment stage**: prod
   2. **Deployment description**: Adding update players functionality
6. Click **Deploy.**

While you are in the Stages page for your API, if you select the small arrow next to **prod**, you should be able to see your Resource Tree you created earlier. Go ahead and click the **PUT** Method that you created. You should then be able to see the **Invoke URL** shaded in blue.



### Questions API

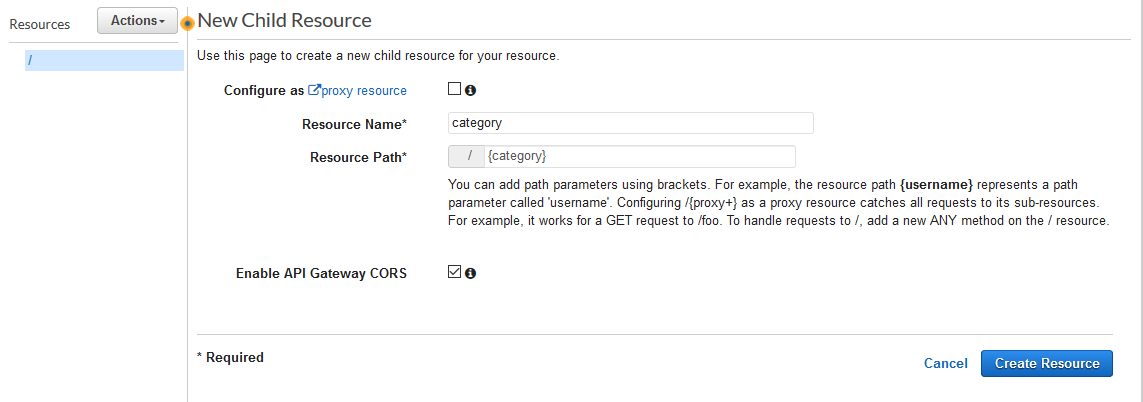
1. Open the Amazon API Gateway console at <https://us-west-1.console.aws.amazon.com/lambda/home?region=us-west-1#/functions>. Confirm at the top right that your region is displayed as **N. California**.
2. Click the **Get Started** button to start building your API.
3. Keep the **REST** protocol selected
4. Select the **New API** radio button.
5. Enter the below details for your API and then click on **Create API**:
   1. **API name**: questions
   2. **Description**: API to interact with the Questions data
   3. **Endpoint Type**: Edge optimized



You will be taken to the API dashboard where you can define your methods and resources for your API.

1. Click the **Actions** Button and on the dropdown, select **Create Resource**
2. Supply the following details:
   1. Leave **Configure as proxy resource** unchecked.
   2. **Resource Name**: category
   3. **Resource Path**\*: {category}
   4. Select the **Enable API Gateway CORS** checkbox.

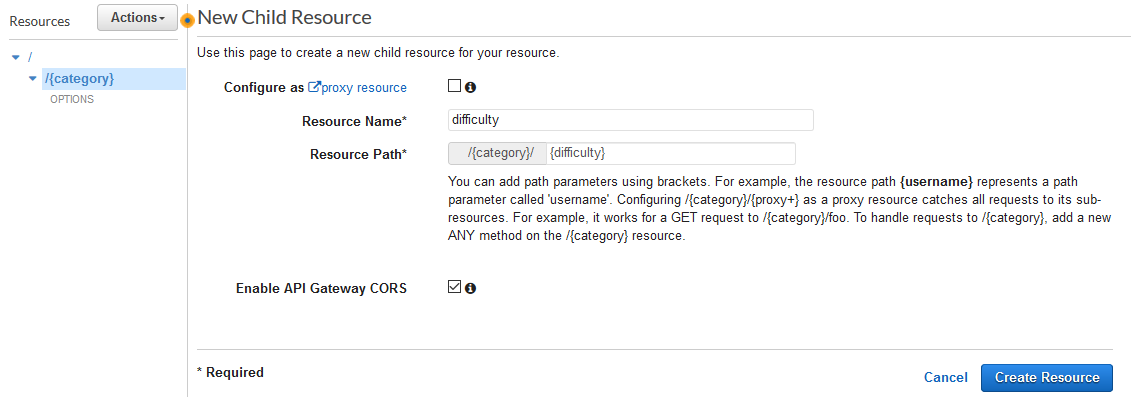
*\*Include the curly brackets around* ***category*** *in the Resource Path*



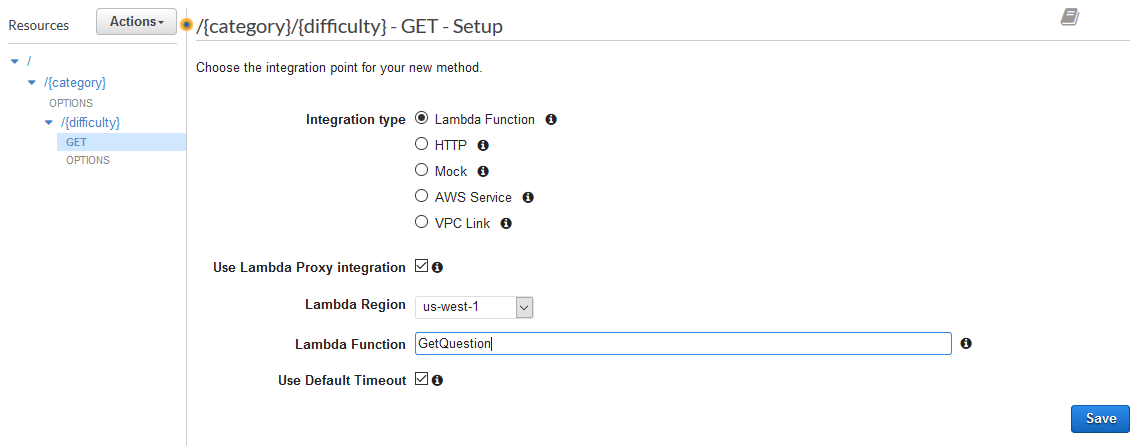
1. With {category} highlighted on the left (it should automatically be highlighted, since it was just created), click the **Actions** dropdown menu and clickthe **Create Resource** button.
2. Supply the following details:
   1. Leave **Configure as proxy resource** unchecked.
   2. **Resource Name**: difficulty
   3. **Resource Path**\*: {difficulty}
   4. Select the **Enable API Gateway CORS** checkbox.

*\*Include the curly brackets around* ***difficulty*** *in the Resource Path*

1. With {difficulty} highlighted on the left (it should automatically be highlighted, since it was just created), click the **Actions** dropdown menu and clickthe **Create Resource** button.
2. With **id** selected in the Resources tree, click the **Actions** button again, and select **Create Method**
3. You will see that a dropdown menu element gets added to the tree below **OPTIONS**. In that dropdown, select **GET**, and then click the Check mark next to it.

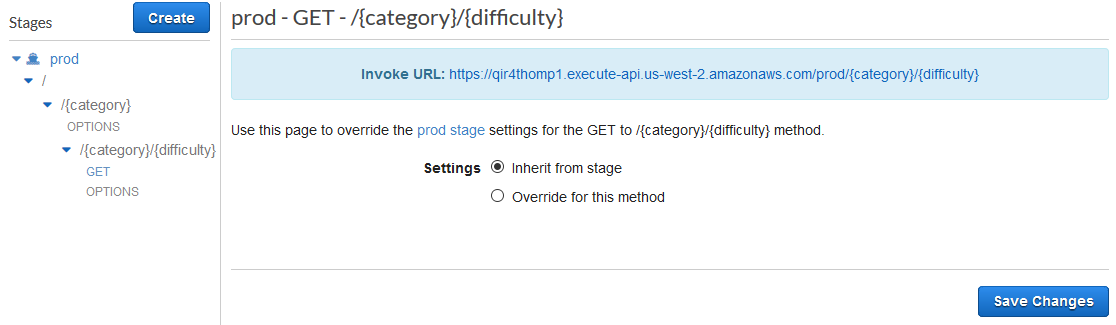


1. Here is where we set up the API integration with Lambda. Fill out the below details:
   1. Select the **Lambda Function** radio button as **Integration Type.**
   2. Select the **Use Lambda Proxy integration** checkbox
   3. Select the **Lambda Region** as us-west-1
   4. In **Lambda Function**, enter the name of the first function that we created earlier that gets all the sessions in the table – GetQuestion
   5. Leave **Use Default Timeout** as selected



1. Click **Save**.
2. When you see the **Add Permission to Lambda Function** pop-up, select **Ok**.
3. With **GET** (or **{difficulty}**) selected in your Resources tree, go ahead and click the **Actions** button again.
4. Click **Enable CORS:**
   1. Leave all the options as-is on the **Enable CORS** page
   2. Click **Enable CORS and replace existing CORS headers**
   3. Click **Yes, replace existing values** on the next dialog and you should see the configuration confirmation
5. Lastly, you will deploy your API. Click on **Actions** then, **Deploy API**. Fill in the details for your deployment stage in the dialog box:
   1. **Deployment stage**: [New Stage]
   2. **Stage name**: prod
   3. **Stage description**: Production deployment stage
   4. **Deployment description**: Initial deployment
6. Click **Deploy.**

While you are in the Stages page for your API, if you select the small arrow next to **prod**, you should be able to see your Resource Tree you created earlier. Go ahead and click the **GET** Method that you created. You should then be able to see the **Invoke URL** shaded in blue.



You have created all the APIs for your Trivia application. Congratulations!

# Final Integration

Now that you have created your APIs, you can integrate them with your application.

1. On your machine, under the **front-end** folder with your front end code, open the **main-es5.js** file on any text editor that you have.
   1. Search for the statement: //TODO1: REPLACE THE URL BELOW WITH THE ONE FOR YOUR QUESTIONS API, FROM SECTION 3, ITEM III

It is right above the **questionsUrl** variable declaration. This is the URL that the code uses to retrieve questions data. Change its assignment to be the URL that was created when you create the Amazon API GET method in Section 3, Item iii. The URL is located on the **Questions** API Gateway dashboard on the **Stages** category on the left. In the Stages tree, select the GET method under the **prod** stage to see the URL shaded in blue.

private questionsUrl = 'https://xijlfuiy35.execute-api.us-west-1.amazonaws.com/prod/';

* 1. Next, search for the statement: //TODO2: REPLACE THE URL BELOW WITH THE ONE FOR YOUR PLAYERS API, FROM SECTION 3, ITEM I

It is right above the **playersUrl** variable declaration. This is the URL that the code uses to create new players, retrieve their current information (such as name and score) and update it when necessary (e.g. a player answers a question correctly). Assign to it the URL for the **GET** method in the **GetPlayers** API that you created on Section 3, item i. The URL is located on the **Players** API Gateway dashboard on the **Stages** category on the left. In the Stages tree, select the GET method under the **prod** stage (and under {id}) to see the URL shaded in blue.

private playersUrl = 'https://xijlfuiy35.execute-api.us-west-1.amazonaws.com/prod/';

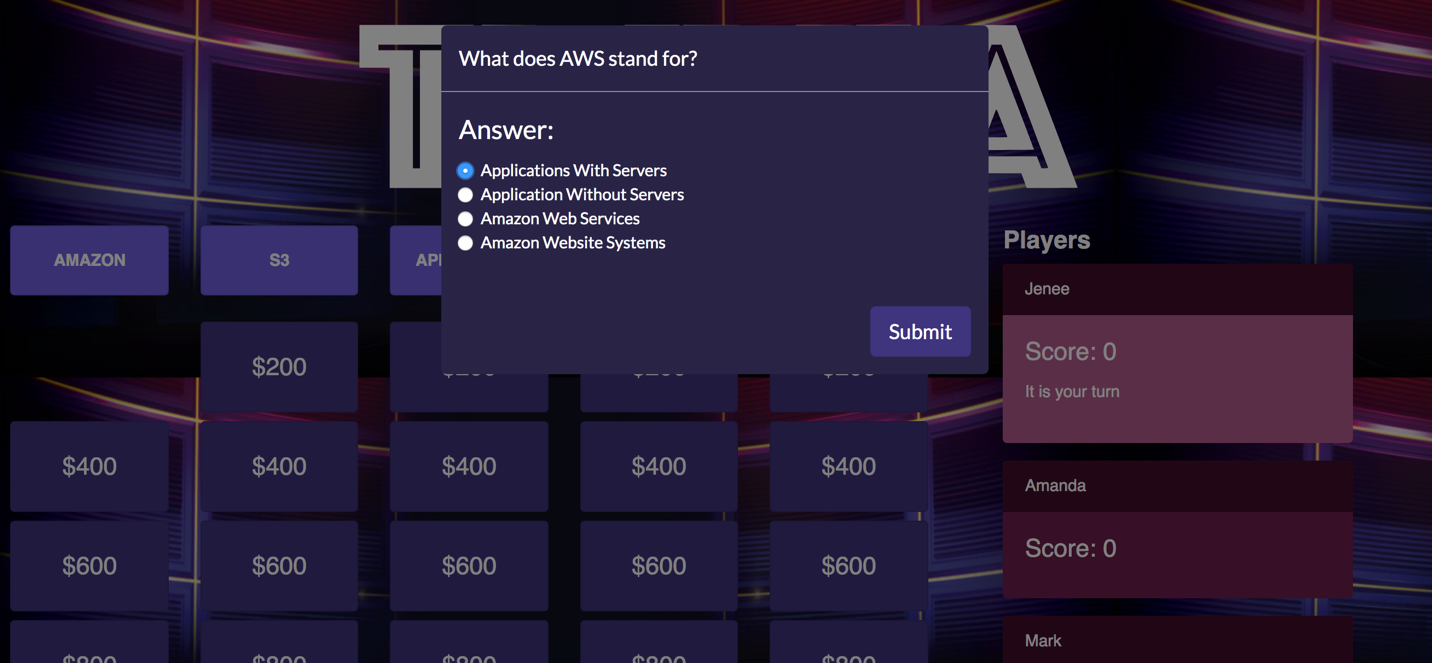
* 1. Next, search for the statement: //TODO3: REMOVE PLAYERS FROM THE OBJECT BELOW WHEN YOUR API IS READY

Remove players from the statement below it. This will stop the application from using mocked data for the players. Your final statement should look as below:

return { state };

* 1. Save your file and then upload it to your S3 bucket with by following what we did on Step 3 of Section 2b.

1. Open the link to your S3 bucket’s website (the link is in your S3 bucket’s Properties tab when you go to the **Static** **Website Hosting** section) and verify that your application works. Try to create a couple players and answer some of the questions.



Congratulations! You have built a serverless application on AWS. You can go ahead and continue to interact with it.

After you are done, don’t forget to delete the resources you have created in your AWS account to avoid any unwanted charges.

**How does the front-end work?**

The front end was built using Angular. Angular is a platform that makes it easy to build applications with the web. Angular combines declarative templates, dependency injection, end to end tooling, and integrated best practices to solve development challenges.

The application has different components to display the trivia questions information and the list of players. It also defines different services to communicate with the back end and retrieve the relevant information. It is using an in-memory database to retrieve the mocked data for the application (such as the initial players). If there is no in memory data present, it continues on to try and querying via the APIs defined.

To keep this workshop simple and within the time frame proposed, we have implemented the logic previously mentioned, and provided you directly with the output file generated with Angular with all of the code for the application combined. On your real application, you would need to write the front-end code yourself.