# 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.

## Verify that you have npm installed on your computer

1. Open your terminal (Windows PowerShell on PC or Terminal on Mac) and run the following command:

8c85903a4bb6:~ agathao$ npm -v

6.2.0

1. If a version of npm is returned, you can proceed to Section 2. If instead, you get an error that npm is not recognized, then you will need to install it.
   1. On a mac:
      1. Run the command: brew -v

8c85903a4bb6:~ agathao$ brew -v

Homebrew 1.7.2

Homebrew/homebrew-core (git revision 41d26; last commit 2018-08-31)

* + 1. If you have a version returned, that means that you have Homebrew installed. If you receive an error instead, then install it by running:

8c85903a4bb6:~ agathao$ ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"

* + 1. Once the installation is complete, run the command: brew install node

8c85903a4bb6:~ agathao$ brew install node

* 1. On a pc:
     1. Download the windows installer from <https://nodejs.org/en/>
     2. Run the installer and follow the prompts.
     3. Restart your computer.

To confirm that everything has been installed properly, run the command: npm -v one more time and verify that a version is returned.

# Building your Serverless Application

During this workshop you will be building an application that allows you to see and add sessions being held at GHC to your application’s calendar. The application will have a user interface that display cards with the different sessions held at GHC. You can click on the cards to see all the details of the session and to add the session to your calendar. Once you add a session to your calendar, the application will identify any sessions that have conflicts with your selected one and will update the cards and the calendar to indicate them.

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.

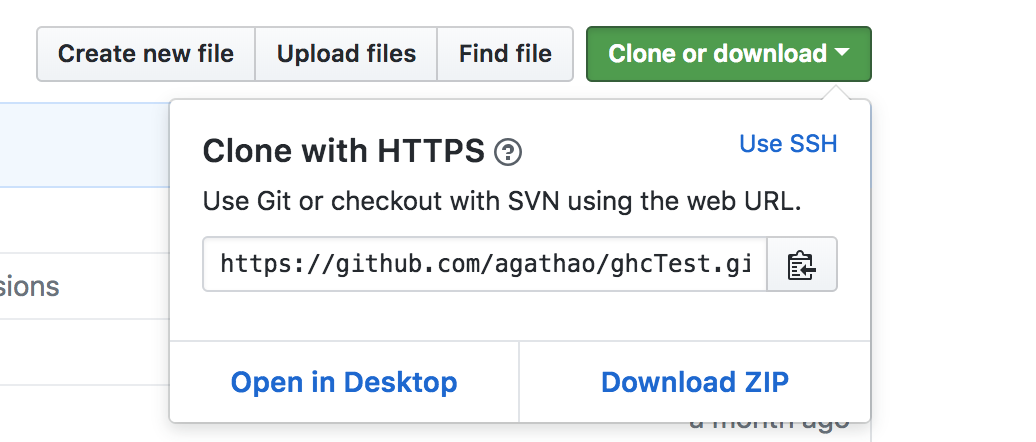
* 1. Downloading website files

1. Open your terminal, 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/GHC2018-BuildingServerlessApps/GHCSessionsScheduler.git

* 1. 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/GHC2018-BuildingServerlessApps/GHCSessionsScheduler>



1. There should be a **GHCSessionsScheduler/** folder created. Go into it and change into the **front-end** folder. Run npm install. This will install all the packages necessary for your site to run.

8c85903a4bb6:Documents agathao$ cd GHCSessionsScheduler/front-end/

8c85903a4bb6:front-end agathao$ npm install

1. Once the installation is complete, run the command npm start. The application will start under http://localhost:4200/. Navigate to it and see the application running:

8c85903a4bb6:front-end agathao$ npm start

> agenda@0.0.0 start /Users/agathao/Documents/GHC/GHCSessionsScheduler/front-end

> ng serve

\*\* Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ \*\*

This initial version displays the site using mocked data with a subset of all GHC sessions. You will be able to see the sessions available and their details, but not to add or remove them from your calendar.

When you try to start the npm, if you are having issues resolving the ng serve command, you may not have an up-to-date version of angular installed on your machine. Run npm install -g @angular/cli@latest to get the latest version. Then, try the npm start command again.

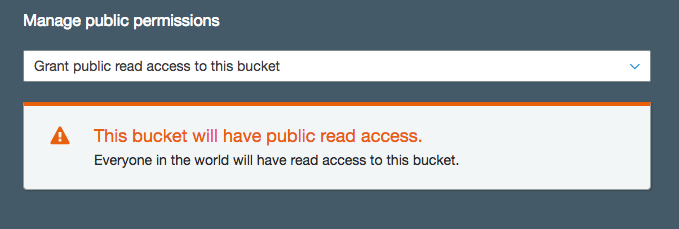
* 1. Storing the files on Amazon S3

1. Let’s prepare the files to be downloaded to S3. Open your terminal and under the **front-end** folder run the build command below. Once the command executes you will see that you have a /dist folder under front-end and inside of it, there will be a folder called ghcSessionsScheduler.

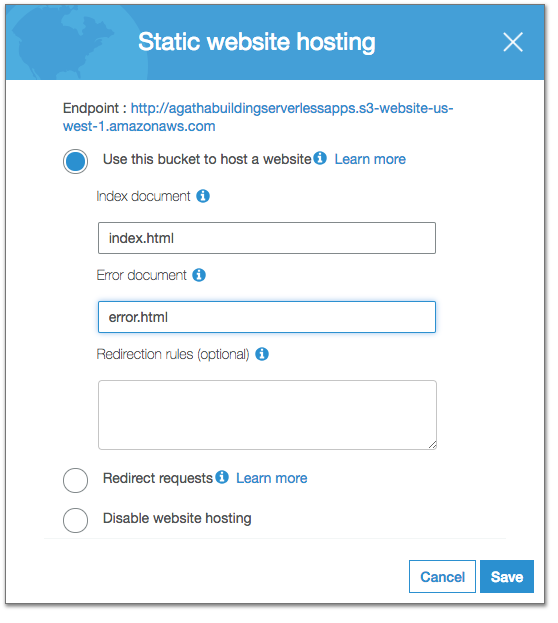
8c85903a4bb6:front-end agathao$ npm run-script build

8c85903a4bb6:front-end agathao$ cd dist/ghcSessionsScheduler/

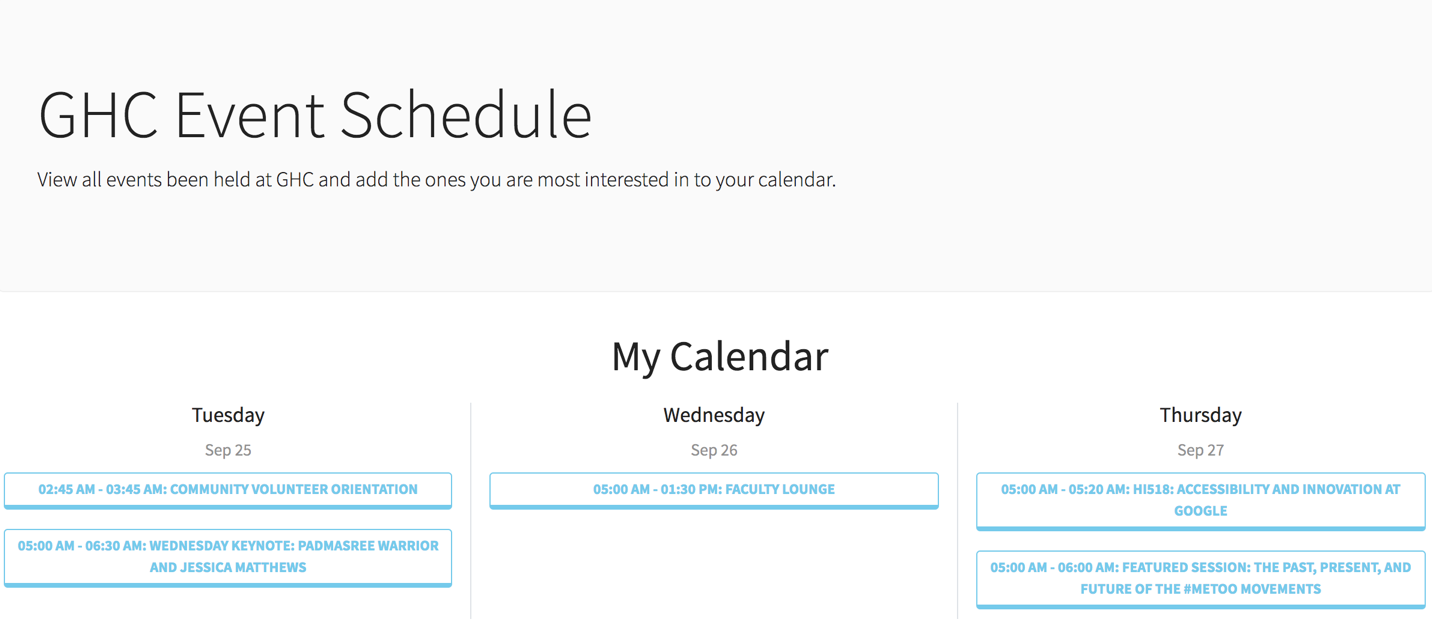
1. Now we will upload the contents of the **ghcSessionsScheduler** folder to Amazon S3. Go to the S3 console at <https://s3.console.aws.amazon.com/>. You will need to sign in if you haven’t already.
2. Click on **Create bucket** on the top right.
   1. Choose a bucket name. Do not forget that it should be unique across all of AWS. We suggest <name>buildingserverlessapps. E.g.: agathaoliveirabuildingserverlessapps
   2. For Region, select **US East (Ohio)**.
      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, you can see that your account has already been given permissions. However, we want the contents of the application to be accessible to anyone. Under **Manage public permissions** select **Grant public access to this bucket**



* 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:



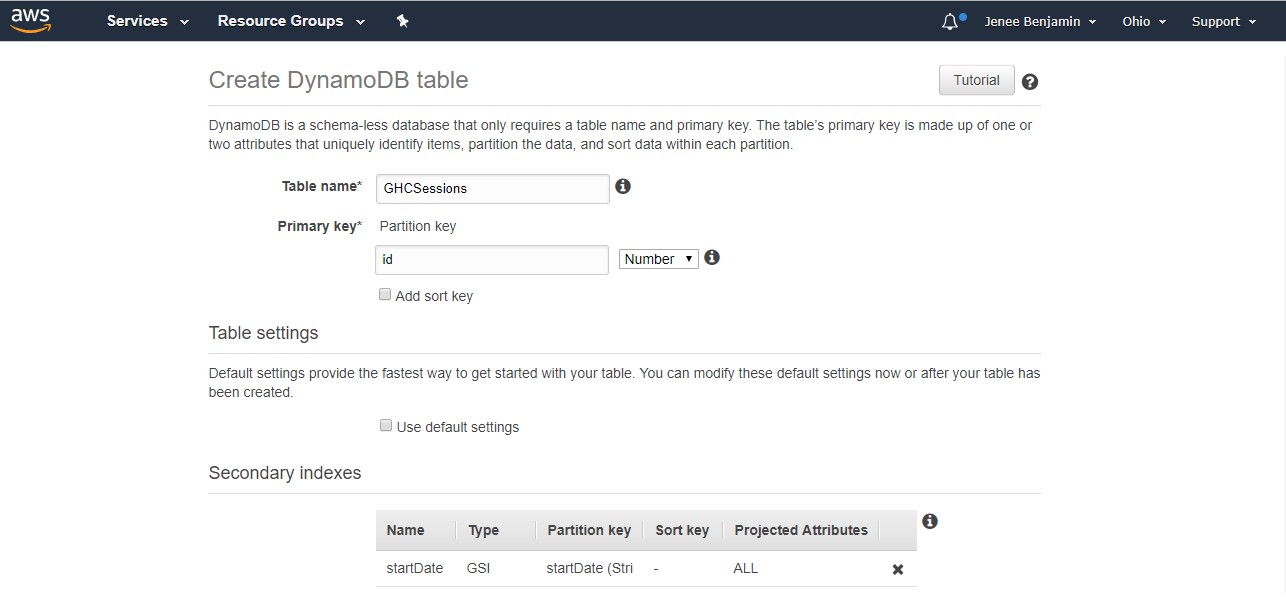
1. There are many ways of uploading your files. We will use the simplest of them: Go back to the **Overview** tab and click on **Upload**.
   1. Select all files under the **ghcSessionsScheduler** folder (from step 1) and drag them to the upload screen. (Do not copy the folder itself. You should copy 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).**
   3. Click **Upload**.
2. When your upload is complete, your files will be accessible via the S3 bucket link. If you don’t remember your link, you can go to the **Properties** tab and click on **Website hosting** to see it.
3. Open your browser and go to the S3 URL to see the site running.



## DynamoDB setup

You will now create the DynamoDB table for your application. This table will store all the GHC session information. Below are the steps:

1. Go to the Dynamo DB console at <https://us-east-2.console.aws.amazon.com/dynamodb>. Confirm at the top right that your region is displayed as **Ohio.**
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:
   1. **Table name**: GHCSessions
   2. **Primary key**: id
   3. On the drop down next to it, select **Number**
   4. Leave **Add sort key** as unchecked.
   5. In the Table settings section, unselect the **Use default settings** box. You will be creating a Secondary Index on this table.
      1. Secondary Indexes allow efficient access to data with attributes other than the primary key.
   6. Under **Secondary indexes**, click on **+ Add index**. Then, enter the below details:
      1. **Primary key**\*: startDate
      2. On the drop down next to it, select **String**
      3. Your index name should be auto-filled: startDate-index
      4. Leave **Add sort key** as unchecked and leave **Projected attributes** as **All**
      5. Click the **Add Index** button to create your Secondary Index.



\*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.

In the meantime, we will create our AWS Lambda functions.

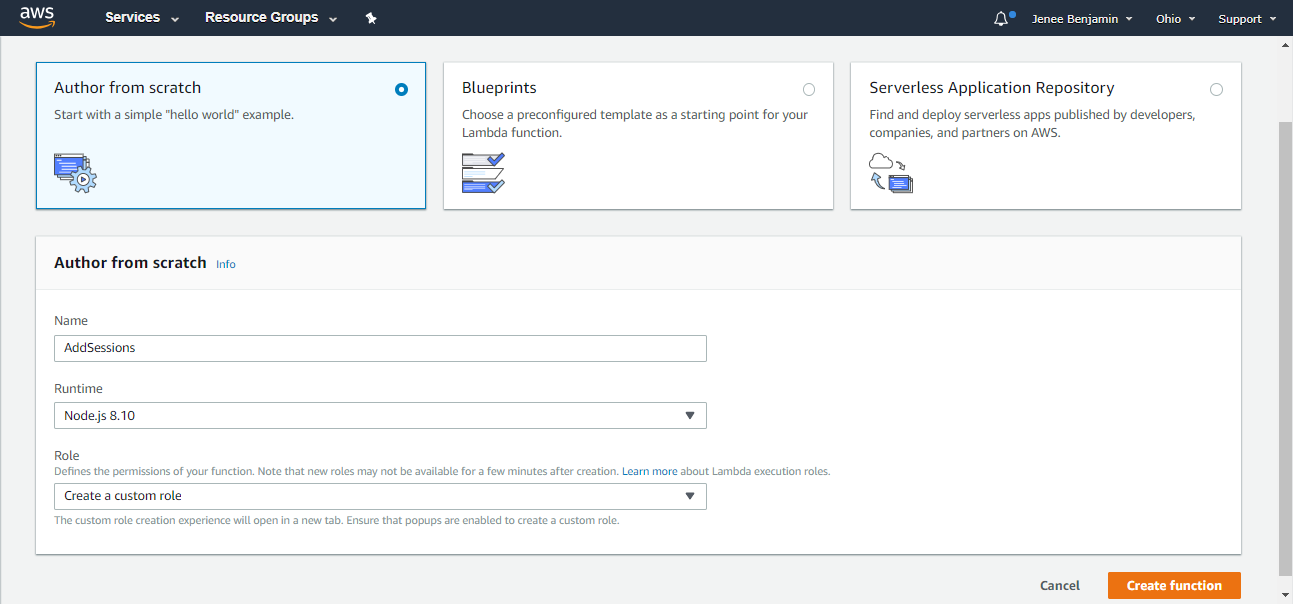
## AWS Lambda setup

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

### AddSessions

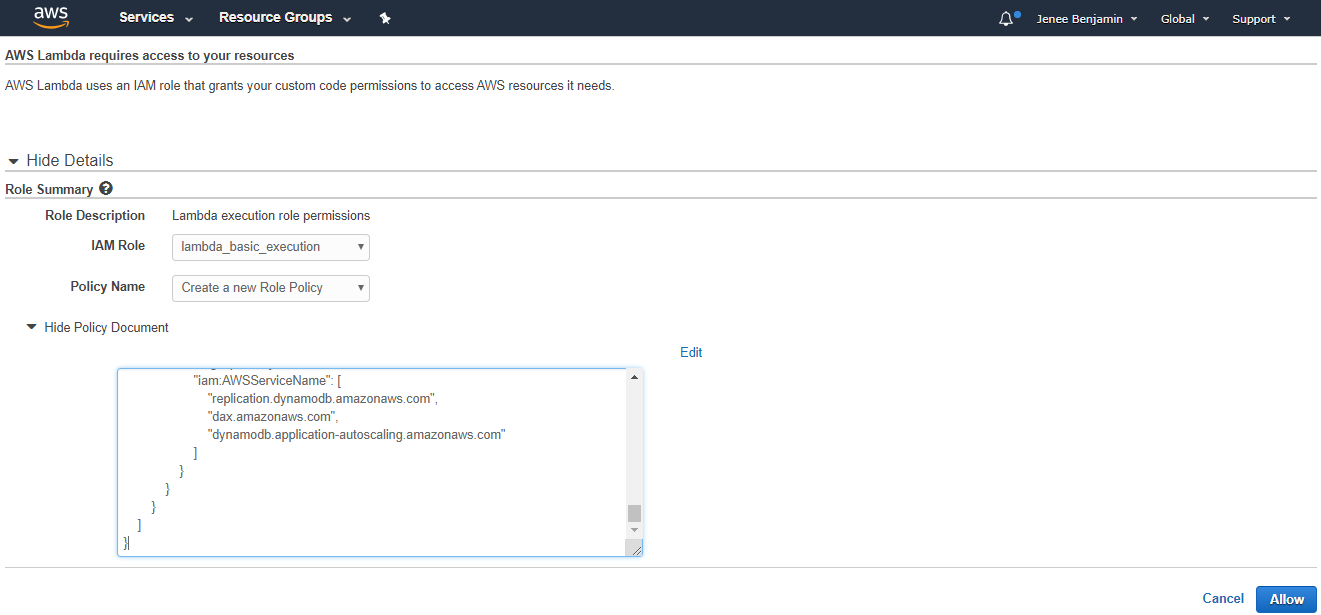
The first Lambda you create will contain the code to populate the DynamoDB table that you just created. This function and the all of the functions in this exercise will use a **Node.js 8.10** runtime environment.

1. Open the AWS Lambda console at <https://us-east-2.console.aws.amazon.com/lambda/home?region=us-east-2#/functions>. Confirm at the top right that your region is displayed as **Ohio**
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**: AddSessions
   2. **Runtime**: Node.js8.10*(not 6.10)*
   3. **Role**: Create a custom role

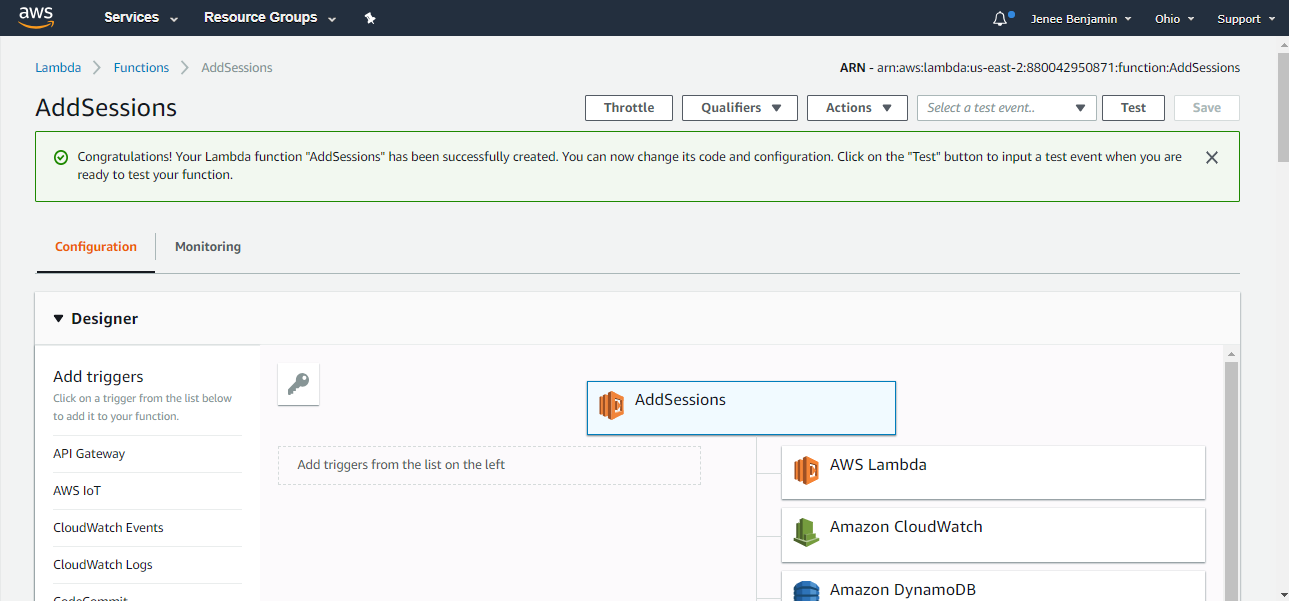


Your IAM (Identity Access Management) role is what defines the permissions for your lambda, such as which AWS services your lambda will be able to access.

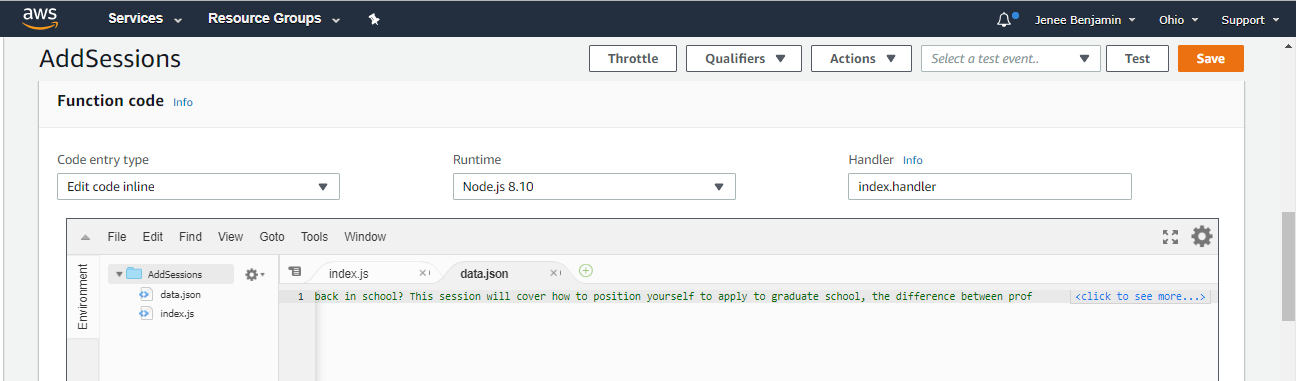
1. In the pop-up window that opens, you will create a new role for this function. You will use this role for this Lambda and the others you create. In the window that opens, enter the following details to create your role.
   1. We will leave the role description, role and name unchanged.
   2. Click on **View Policy Document** and then on **Edit**. The Policy Document textbox should become modifiable.
   3. On the GitHub repository, open the iam\_role.txt file located at https://github.com/GHC2018-BuildingServerlessApps/GHCSessionsScheduler/blob/master/lambda/iam/dynamodb\_full\_access.txt
   4. Copy the contents of the file and paste into the Policy Document textbox on the IAM page.
   5. Click the **Allow** Button at the bottom and the window will close.



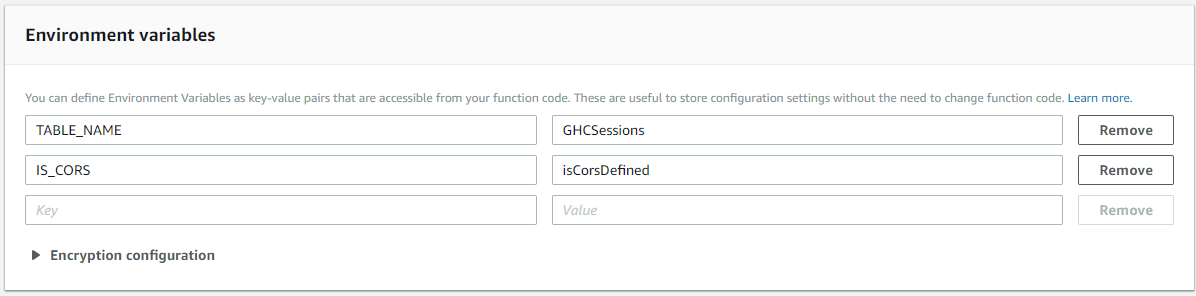
1. The role you created should show up under Existing role. Go ahead and continue the Lambda setup by clicking **Create Function**. You should now see the Lambda’s detail page.



1. Scroll down to the Function code section. You will be editing the index.js file that is already open in the editor.
2. On a new tab on your browser, navigate to this link- <https://github.com/GHC2018-BuildingServerlessApps/GHCSessionsScheduler/blob/master/lambda/functions/addSessions/index.js>. The code at this link contains the logic to populate data into your DynamoDB GHCSessions table via batch-write calls. The batch-write operation puts multiple items in one or more tables. The code sends a success HTTP response once the process is complete.
3. Copy the code from the file and paste it into the AWS Lambda function code editor.
4. In the editor, go to **File > Save** to save your changes.
5. You will also create a new file in the AddSessions directory. Go to **File > New File**.
6. On a new tab on your browser, navigate to <https://github.com/GHC2018-BuildingServerlessApps/GHCSessionsScheduler/blob/master/lambda/functions/addSessions/data.json> . The file at this link contains all the data for each GHC session occurring (from Wednesday to Friday) in JSON format.
7. Copy the data from that file and paste it into the AWS Lambda function code editor for the new file that you have open.
8. 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 **AddSessions** is selected
   3. **Folder**: /
9. 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**: GHCSessions
   2. **Key**: IS\_CORS, **Value**: isCorsDefined

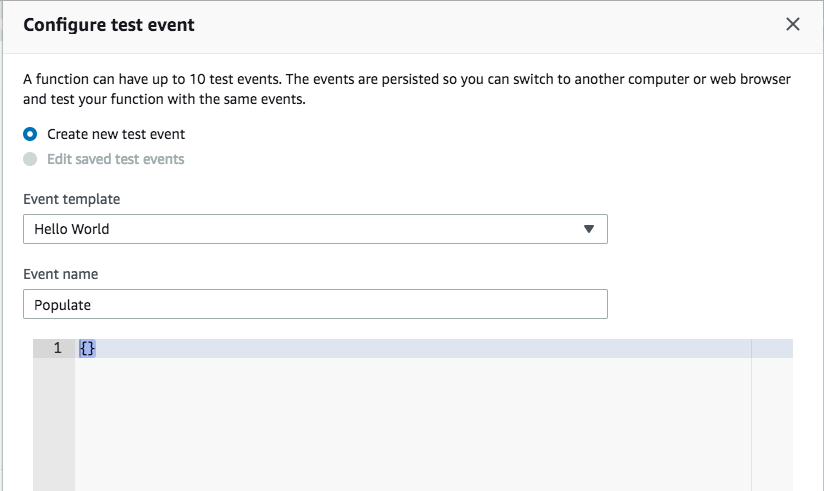


1. Scroll to the **Basic settings** section and increase the timeout from 3 seconds to 30 seconds. This will give your function more time to run and populate the DynamoDB table.
2. Keep everything else as-is, and then click the Orange **Save** button at the top of the screen.

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 appear in the dropdown next to the Test button. Click on the **Test** button to run your test event and execute the code.
3. Your table should be populated now. You can verify that by going visiting your DynamoDB table: <https://us-east-2.console.aws.amazon.com/dynamodb/home?region=us-east-2#tables:selected=GHCSessions;tab=items>

Let’s create the next lambda function for our application, which will have a simpler setup!

### GetAllSessions

The next Lambda you create will contain the code to fetch all the sessions from the GHCSessions DynamoDB table.

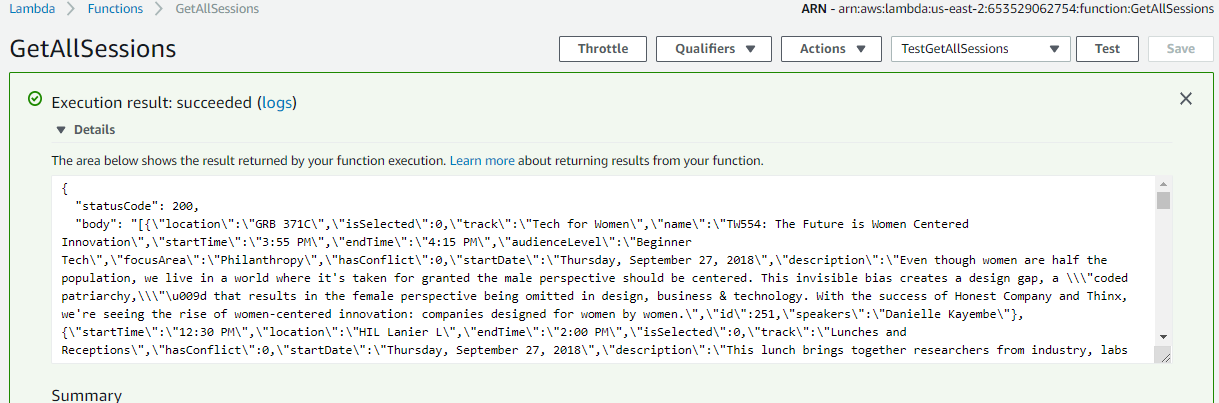
1. Open the AWS Lambda console at <https://us-east-2.console.aws.amazon.com/lambda/home?region=us-east-2#/functions>. Confirm at the top right that your region is displayed as **Ohio.**
2. Click the **Create Function** button.
3. Select the **Author from scratch** card.
4. Fill out the form for the Lambda with the following details:
   1. **Name**: GetAllSessions
   2. **Runtime**: Node.js 8.10*(not 6.10)*
   3. **Role**: Choose an existing role
   4. **Existing role**: lambda\_basic\_execution
5. Continue the Lambda setup by clicking **Create Function**.

You should now see the Lambda’s detail page.

1. Scroll down to the Function code section. You will be editing the index.js file that is already open.
2. On a new tab on your browser, navigate to <https://github.com/GHC2018-BuildingServerlessApps/GHCSessionsScheduler/blob/master/lambda/functions/getAllSessions/index.js> . The code at this link contains the logic to retrieve all the session data in GHCSessions 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.
5. Scroll down to the **Environment variables** section on the Lambda detail page. You want to add two Environment variables.
   1. **Key**: TABLE\_NAME, **Value**: GHCSessions
   2. **Key**: IS\_CORS, **Value**: isCorsDefined
6. 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.
7. Keep everything else as-is, and then click the Orange **Save** button at the top of the screen.

You have now created your second Lambda function. You can test your function 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 **TestGetAllSessions**.
   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.
3. Click on **Create**.
4. 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. In the response body, you should see the unformatted list of sessions returned.



### UpdateSession

This last Lambda will be used to update existing sessions in the DynamoDB **GHCSessions** table.

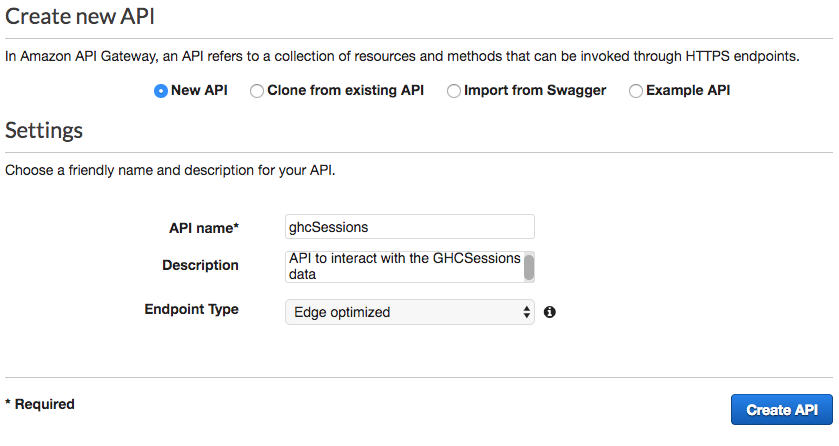
1. Open the AWS Lambda console at <https://us-east-2.console.aws.amazon.com/lambda/home?region=us-east-2#/functions>. Confirm at the top right that your region is displayed as **Ohio**.
2. Click the **Create Function** button.
3. Select the **Author from scratch** card.
4. Fill out the form for the Lambda with the following details:
   1. **Name**: UpdateSession
   2. **Runtime**: Node.js 8.10 (not 6.10)
   3. **Role**: Choose an existing role
   4. **Existing role**: lambda\_basic\_execution
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.
7. On a new tab on your browser, navigate to <https://github.com/GHC2018-BuildingServerlessApps/GHCSessionsScheduler/blob/master/lambda/functions/updateSession/index.js>. The code at this link contains the logic to update an existing item in the GHCSessions table. A DynamoDB update call is performed in order to indicate that an attribute on the item- for our application’s case, the isSelected attribute- has changed. This is the code that gets invoked when an item is added to your GHC calendar. Once the update completes, then the code will query the secondary index- startDate-index- that we created when we created the DynamoDB table. It will query all the sessions that occur the same day as the session, and of those, it will find and set any overlapping sessions as conflicts using a hasConflict flag. The code sends a success HTTP response once this whole process 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. Setup the **UpdateSession** function with the same **Environment Variables** and the same **Basic Settings** as **GetAllSessions**:
    1. **Key**: TABLE\_NAME, **Value**: GHCSessions
    2. **Key**: IS\_CORS, **Value**: isCorsDefined
11. In **Basic settings** increase the timeout from 3 seconds to 30 seconds.
12. Then, click **Save**.

## Amazon API Gateway

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

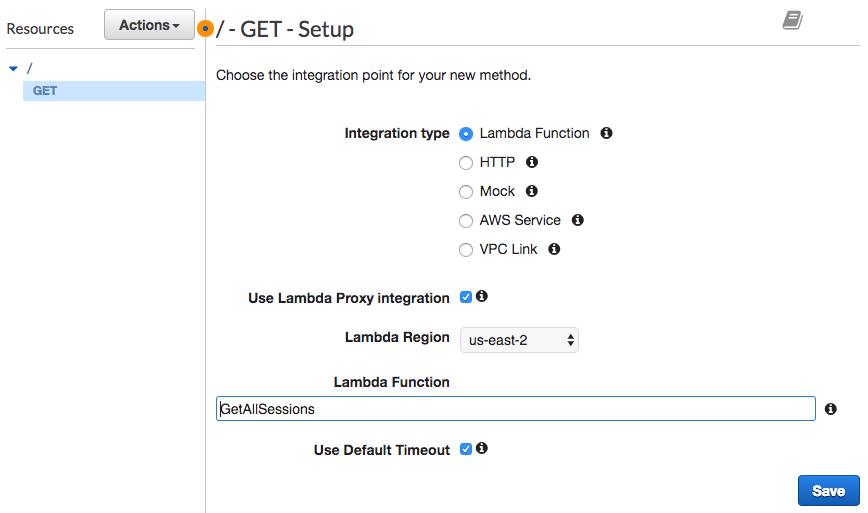
## Retrieving GHC sessions

1. Open the Amazon API Gateway console at <https://us-east-2.console.aws.amazon.com/apigateway/home?region=us-east-2#/apis> . Confirm at the top right that your region is displayed as **Ohio**.
2. Click the **Get Started** button to start building your API.
3. Select the **New API** radio button.
4. Enter the below details for your API and then click on **Create API**:
   1. **API name**: ghcSessions
   2. **Description**: API to interact with the GHCSessions 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-east-2** 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 – GetAllSessions
   5. Leave **Use Default Timeout** as selected



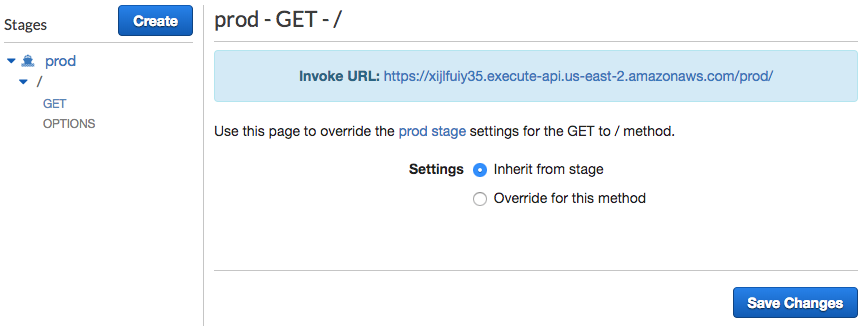
1. Click **Save**.
2. When you see the **Add Permission to Lambda Function** pop-up, select **Ok**.
3. With **GET** (or **/**) selected in your Resources tree, go ahead and click the **Actions** button again.
4. Click **Enable CORS:**
   1. Leave all the options as-is on the **Enable CORS** page
   2.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

## update-session

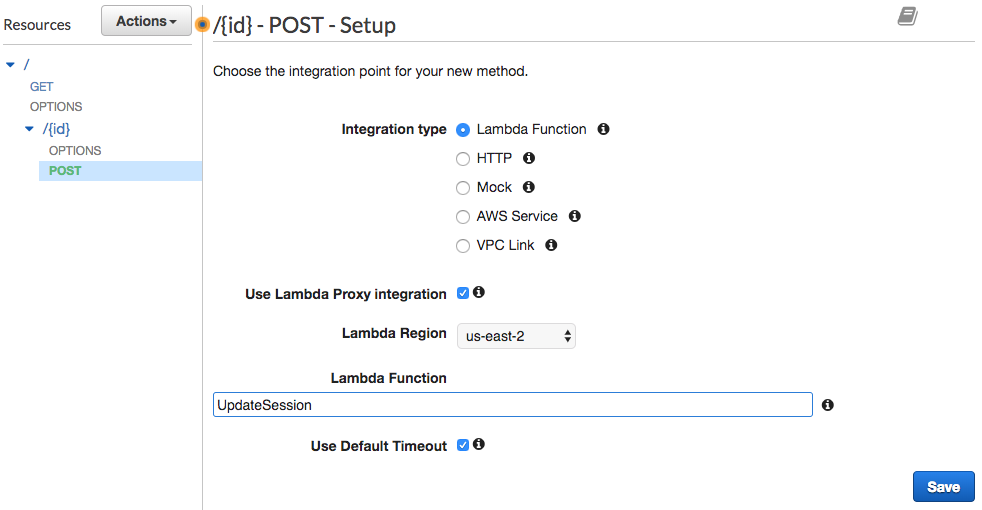
You will now create another resource to update an item in the **GHCSessions** table.

1. Click on the **ghcSessions** API on the left column under APIs.
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 **POST**, 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-east-2
   4. In **Lambda Function**, enter the name of the first function that we created earlier that gets all the sessions in the table – UpdateSession
   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 **POST** (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 session functionality
6. Click **Deploy.**

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

* + - 1. Final Integration

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

1. Under the **front-end** folder with your front end code, open the **src > app** folder and then edit the **sessions.service.ts** file with any text editor that you have.
   1. The sessions.service.ts file contains the code responsible for retrieving and updating session data so that it can be used by the rest of the application.
   2. At the top of the file, you will see a declaration for the **getSessionsUrl**. Assign to it the URL for the **get-all-sessions** API that you created on Section 5.a. The URL was located on the **get-all-sessions** API Gateway dashboard on the **Stages** page (select the GET method under the **prod** stage to see the URL shaded in blue).

private getSessionsUrl = 'https://liks2l9a0b.execute-api.us-east-1.amazonaws.com/Stage/ghcsessions';

* 1. On the next line, you will see a declaration for the **updateSessionsBaseUrl**. Assign to it the URL for the **update-session** API that you created on Section 5.b. The URL was located on the **get-all-sessions** API Gateway dashboard on the **Stages** page (select the POST method under the **prod** stage to see the URL shaded in blue).

private updateSessionsBaseUrl = 'https://q4fhvualcc.execute-api.us-east-1.amazonaws.com/Stage/ghcsessions/';

* 1. Save your changes and regenerate your distribution files by following what we did on Step 1 of section 2b.
  2. Next, replace the files on your S3 bucket with the new files that were created by following what we did on Step 4 of section 2b.

1. Open the link to the S3 bucket and verify that your application works. Try to add and remove some events from the calendar and see the app update.

Congratulations! You have built a Serverless Application on AWS. You can go ahead and continue to play with it.

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