# Build document: geoCore geospatial library

## Table of Contents

[Document version history](#_heading=h.3ijfejlqb2zn) 3

[Summary](#_heading=h.x3ht4quu5ys2) 4

[Architecture](#_heading=h.nofhud9n7p38) 5

[Technical Instructions](#_heading=h.iwbelgwd2clj) 7

[Creating the temporary update mechanism](#_heading=h.6gn3a95s43o) 7

[Creating the s3 bucket for GeoNetwork JSON files](#_heading=h.x0mnfdudyltq) 7

[Creating the s3 bucket for geoCore metadata storage](#_heading=h.uoflh4imkemr) 8

[Creating the harvesting function](#_heading=h.iojsq6x2fhzt) 8

[The translation function](#_heading=h.b6nok17kh8or) 9

[Understanding the geoCore format](#_heading=h.4cbs5t1i2yb1) 9

[Creating the translation Lambda function](#_heading=h.p9mr0gs0byjz) 14

[Creating the geoCore geospatial library](#_heading=h.vkawbvldmmfw) 15

[Creating the s3 logging bucket](#_heading=h.g4416e614f0x) 15

[Creating AWS Athena Database and AWS Glue Table.](#_heading=h.x0shb2gxrwt2) 15

[Creating the Query function](#_heading=h.ctrjovbed7qh) 21

[Create an API using AWS API Gateway](#_heading=h.6fys4slxsmyr) 23

[Creating an AWS Web Application Firewall](#_heading=h.6rt2yhbwchuy) 27

[Creating supporting query functions for the Web Presence](#_heading=h.3ho6mmhymvex) 29

[Creating DynamoDB tagging system](#_heading=h.dri5lrk7sdg) 29

[Creating the Search by ID endpoint](#_heading=h.8dwv34ru2xbt) 32

[Creating supporting query API endpoints](#_heading=h.yz0void40k9e) 35

## Document version history

| Name | Reversion Type | Date | Comments |
| --- | --- | --- | --- |
| Chris Melnick-MacDonald | Created | 2021-04-20 | Initial document creation |
| Chris Melnick-MacDonald and Bo Lu | Additions | 2021-10-04 | Additions to Harvest and Analytics |

## 

## Summary

This build document describes the procedures to build the geoCore geospatial library. It is built using a cloud-first strategy for search, discovery, and dissemination of geospatial content. It is built using Amazon Web Services’s managed services to reduce overall cost of development and maintenance.

In this document, we will be building and configuring multiple AWS services to create the geospatial library system. The system stores the information in the geoCore format and has been built to easily query the system through the geoCore APIs.

Before we build the geoCore system, here is an overview of the architecture.

## Architecture

The temporary update mechanism is the infrastructure needed to harvest the metadata records from the current GeoNetwork catalog, convert them to the geoCore format, and store them in the library.

*Note: Names used in the images are for illustrative purposes only.*

## Technical Instructions

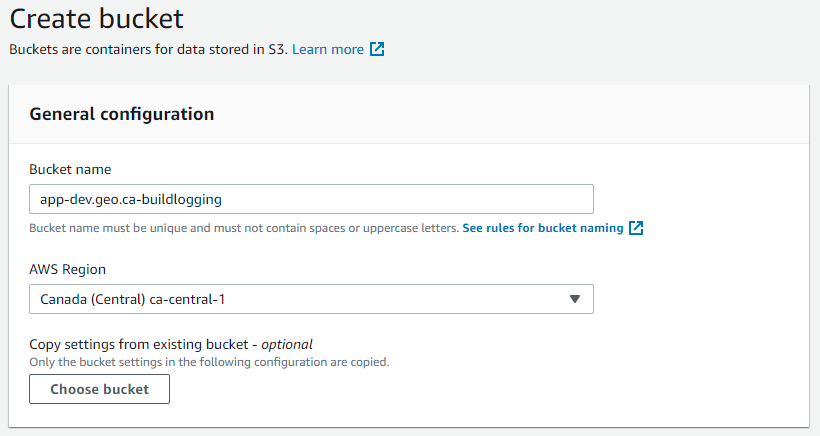
## Creating the temporary update mechanism

The temporary update mechanism is created with the public access API of GeoNetwork. An AWS lambda function has been created to schedule the harvest of the metadata from GeoNetwork. From here the JSON files from the API are saved to an s3 bucket. Once these files are saved to this location, another AWS lambda function takes each JSON file and translates it into the geoCore format and saves it into the geospatial library.

### Creating the s3 bucket for GeoNetwork JSON files

To begin with setting up the temporary update mechanism we will have to create two s3 buckets. One is used to store GeoNetwork JSON files, and one is for the geoCore metadata storage.

1. In the AWS Management console, navigate to the s3 interface.
2. Click Create s3 Bucket
3. For the naming convention, please include ‘json’ and the environment (i.e., dev, staging, prod) for this first bucket to easily identify the bucket for use in later stages.
4. Choose the Canada (Central) ca-central-1 region



1. Leave all other options as default.
2. Click Create Bucket.

### Creating the s3 bucket for geoCore metadata storage

To create the next bucket, please follow steps 1 to 6 again. This time, for the naming convention, please use ‘geoCore’ and the environment (i.e., dev, staging, prod) in the naming of the bucket to be used at a later stage.

### Creating the harvesting function

To create the lambda function for the harvesting of the HNAP JSON files from the GeoNetwork catalog, navigate to the HNAP\_JSON\_Harvest repository on GitHub under the Canadian Geospatial Platform’s organisation: https://github.com/Canadian-Geospatial-Platform/HNAP\_JSON\_Harvest

1. Create a t2.micro AWS Cloud9 VM
2. Either upload the zipped code or git clone the repository
3. Once cloned or unzipped, change directory to HNAP\_JSON\_Harvest using the shell
   1. I.e., cd HNAP\_JSON\_Harvest
4. Verify the settings in app.py and change line 28 to match the destination S3 bucket used in Step 3.
5. Build the project using the command ‘sam build’
6. Check for build errors, if none, the command ‘sam deploy’ can be run to deploy the Lambda function.
7. Verify deployment on AWS Lambda. If errors occur, consult the documentation on the github repository. If errors persist, contact [bo.lu@nrcan-rncan.gc.ca](mailto:bo.lu@nrcan-rncan.gc.ca) for assistance.
8. Delete the Cloud9 instance if desired (we are only billed $0.0116/hr if it is actively being used).

## The translation function

### Understanding the geoCore format

As the Federal Geospatial Platform deals with not only the Harmonized North American Profile (HNAP) of ISO 19115, but different metadata standards from the provinces and territories, a flexible metadata storage format was created. The geoCore format is a standardless format that is able to store various metadata fields. It is based on the geoJSON format where the properties of the geoJSON file store the metadata for each record.

The following is an example of the geoCore format:

{

"type": "FeatureCollection",

"features": [

{

"type":"Feature",

"geometry": {

"type": "Polygon",

"coordinates": [[[ west, south ], [ east, south ], [ east, north ], [ west, north ], [ west, south]]]},

"properties": {

"id": id,

"title": {

"en": title\_en,

"fr": title\_fr

},

"description": {

"en": description\_en,

"fr": description\_fr

},

"keywords": {

"en": keyword\_String\_en,

"fr": keyword\_String\_fr

},

"topicCategory": topicCategory,

"date": {

"published": {

"text": date\_text\_publication,

"date": date\_publication

},

"created": {

"text": date\_text\_creation,

"date": date\_creation

}

},

"spatialRepresentation": spatialRepresentation,

"type": type,

"geometry": boundingbox,

"temporalExtent": {

"begin": temporalExtent\_begin,

"end": temporalExtent\_end

},

"refSys": refSys,

"refSys\_version": refSys\_version,

"status": status,

"maintenance": maintenance,

"metadataStandard": {

"en": metadataStandard\_en,

"fr": metadataStandard\_fr

},

"metadataStandardVersion": metadataStandardVersion,

"graphicOverview": goArray,

"distributionFormat\_name": distributionFormat\_name,

"distributionFormat\_format": distributionFormat\_format,

"useLimits": {

"en": useLimits\_en,

"fr": useLimits\_fr

},

"accessConstraints": accessConstraints,

"otherConstraints": {

"en": otherConstraints\_en,

"fr": otherConstraints\_fr

},

"dateStamp": dateStamp,

"dataSetURI": dataSetURI,

"locale": {

"language": locale\_language,

"country": locale\_country,

"encoding": locale\_encoding

},

"language": language,

"characterSet": characterSet,

"environmentDescription": environmentDescription,

"supplementalInformation": {

"en": supplementalInformation\_en,

"fr": supplementalInformation\_fr

},

"contact": contacts\_Array,

"credits": credits\_Array,

"cited": cited\_Array,

"distributor": dist\_Array,

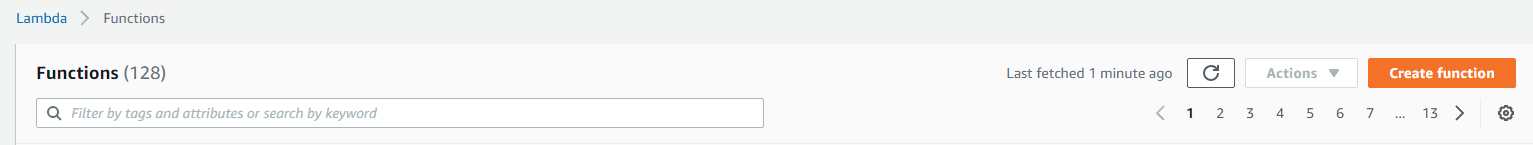
"options": options\_Array

}

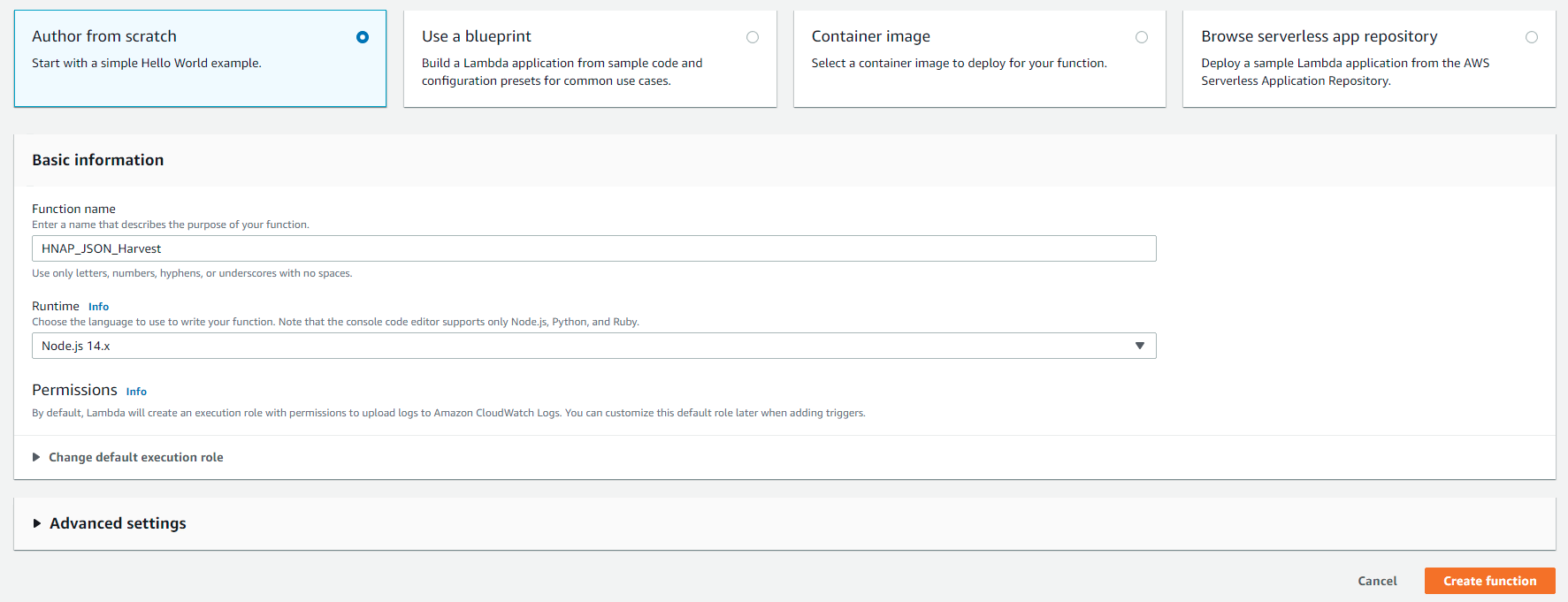
}]}

### Creating the translation Lambda function

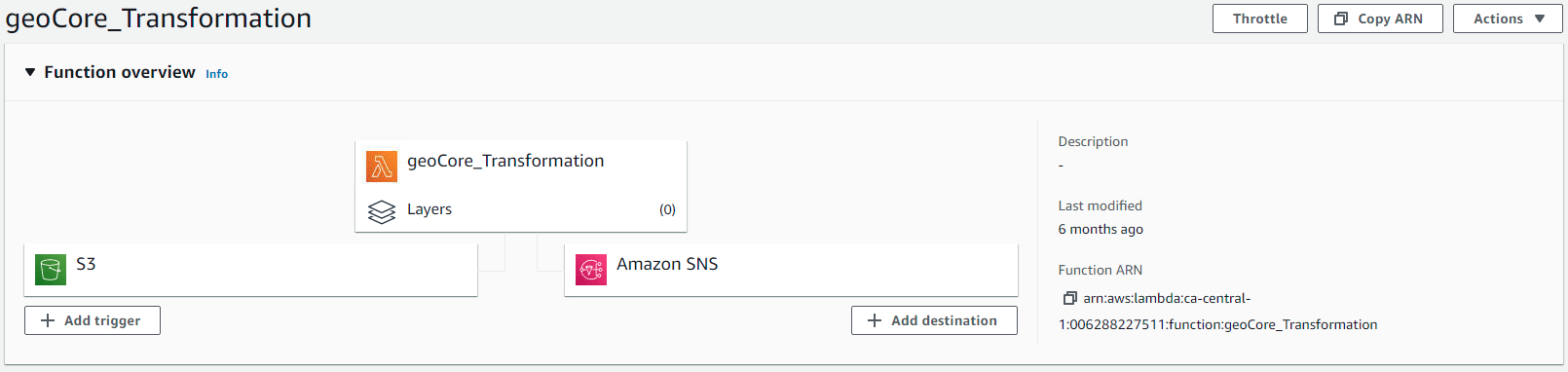
To create the lambda function for translating the HNAP JSON to the geoCore format, navigate to the geoNetwork\_to\_geoCore repository on GitHub under the Canadian Geospatial Platform’s organisation. Download the production level zip file located in this repository. Once this is available, navigate to the AWS Management Console.



1. Click Create Function
2. Make sure Author from scratch is selected
3. Name the function geoCore\_Transformation
4. Select the Nodejs runtime
5. Click Create Function



1. When the function is created, the configuration window will open.
2. Click upload, and upload the zip that you received from the GitHub repository.
3. On line 2917 change the code for bucketoutput as the name of the bucket created for the geocore files.
4. Click deploy to deploy the function.
5. Click Add trigger and add the s3 bucket created in steps 1-6 of this document. This will allow for any files add to this s3 bucket to be translated into geoCore files and stored in the geoCore geospatial library bucket.
6. Make sure the Lambda function role has permissions to place objects in an s3 bucket.



## Creating the geoCore geospatial library

The main geoCore geospatial library stack was created using Amazon Web Services managed services. This is a combination of s3 buckets, AWS glue, AWS Athena, AWS Lambda, and API gateway. The first action is to create another s3 bucket for the output logging of AWS Athena.

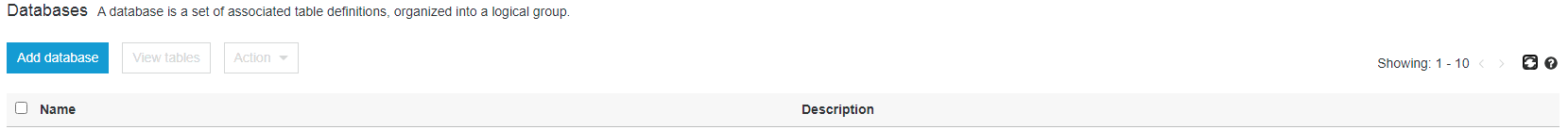
### Creating the s3 logging bucket

Following steps 1-6 in this document, create an s3 bucket for the query output of AWS Athena. For the naming convention, please include a name that is easily understandable for the purpose.

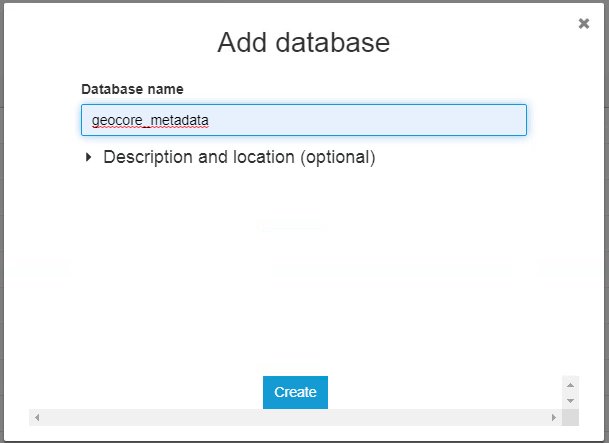
### Creating AWS Athena Database and AWS Glue Table.

AWS Athena and Glue work together to enable SQL queries against non-SQL data. To accomplish this, we will need to create an AWS Glue database, and then create an Athena table to store the expected geoJSON schema to query against.

For the first step, navigate to the AWS Glue management console.



1. Click Add database
2. For the database name, use geocore\_metadata
3. Click Create



1. Navigate to the AWS Athena console.
2. In the Database dropdown menu, select the geocore\_metadata
3. In the query window, insert the following code without alterations. When asked for an S3 bucket to store query results, select a temporary S3 bucket e.g., aws-glue-temporary-006288227511-ca-central-1

***ADD THIS CODE BLOCK TO THE ATHENA QUERY WINDOW WITHOUT ALTERATIONS***  
  
CREATE EXTERNAL TABLE IF NOT EXISTS metadata (

type STRING,

features ARRAY<struct<

type:string,

geometry:struct<

type:string,

coordinates:array<array<array<double>>>>,

properties:struct<

id:string,

title:struct<

en:string,

fr:string

>,

description:struct<

en:string,

fr:string

>,

keywords:struct<

en:string,

fr:string

>,

topicCategory:string,

date:struct<

published:struct<

text:string,

date:string

>,

created:struct<

text:string,

date:string

>>,

spatialRepresentation:string,

type:string,

geometry:string,

temporalExtent:struct<begin:string,`end`:string>,

refSys:string,

refSys\_version:string,

status:string,

maintenance:string,

metadataStandard:struct<

en:string,

fr:string

>,

metadataStandardVersion:string,

graphicOverview:array<

struct<

overviewFileName:string,

overviewFileDescription:string,

overviewFileTupe:string,

overviewFileType:string

>>,

distributionFormat\_name:string,

distributionFormat\_format:string,

useLimits:struct<

en:string,

fr:string>,

accessConstraints:string,

otherConstraints:struct<

en:string,

fr:string

>,

dateStamp:string,

dataSetURI:string,

locale:struct<

language:string,

country:string,

encoding:string

>,

language:string,

characterSet:string,

environmentDescription:string,

supplementalInformation:struct<

en:string,

fr:string>,

contact:string,

credits:array<

struct<

en:string,

fr:string

>>,

cited:array<

struct<

individual:string,

position:struct<

en:string,

fr:string

>,

organisation:struct<

en:string,

fr:string

>,

telephone:struct<

en:string,

fr:string

>,

fax:string,

address:struct<

en:string,

fr:string

>,

city:string,

pt:struct<

en:string,

fr:string

>,

postalcode:string,

country:struct<

en:string,

fr:string

>,

email:struct<

en:string,

fr:string

>,

onlineResource:struct<

onlineResource:string,

onlineResource\_Name:string,

onlineResource\_Protocol:string,

onlineResource\_Description:string

>,

hoursOfService:string,

role:string

>>,

distributor:array<

struct<

individual:string,

position:struct<

en:string,

fr:string

>,

organisation:struct<

en:string,

fr:string

>,

telephone:struct<

en:string,

fr:string

>,

fax:string,

address:struct<

en:string,

fr:string

>,

city:string,

pt:struct<

en:string,

fr:string

>,

postalcode:string,

country:struct<

en:string,

fr:string

>,

email:struct<

en:string,

fr:string

>,

onlineResource:struct<

onlineResource:string,

onlineResource\_Name:string,

onlineResource\_Protocol:string,

onlineResource\_Description:string

>,

hoursOfService:string,

role:string

>>,

options:string>>>

)

ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'

WITH SERDEPROPERTIES (

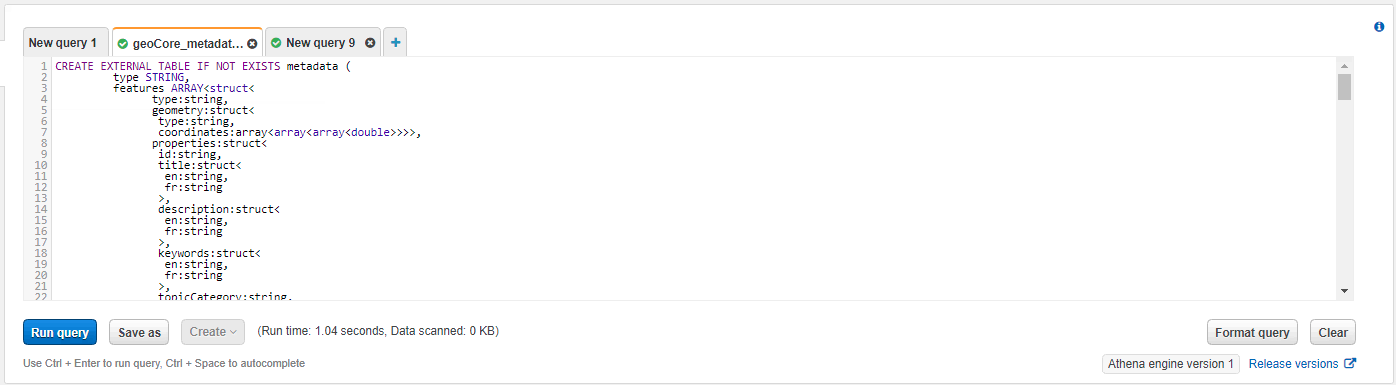
'serialization.format' = '1',

'ignore.malformed.json' = 'true'

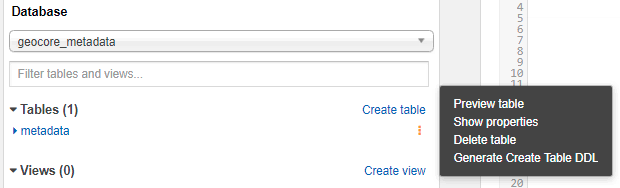
) LOCATION 's3://cgp-meta-l1-geojson-dev/'

TBLPROPERTIES ('has\_encrypted\_data'='false')

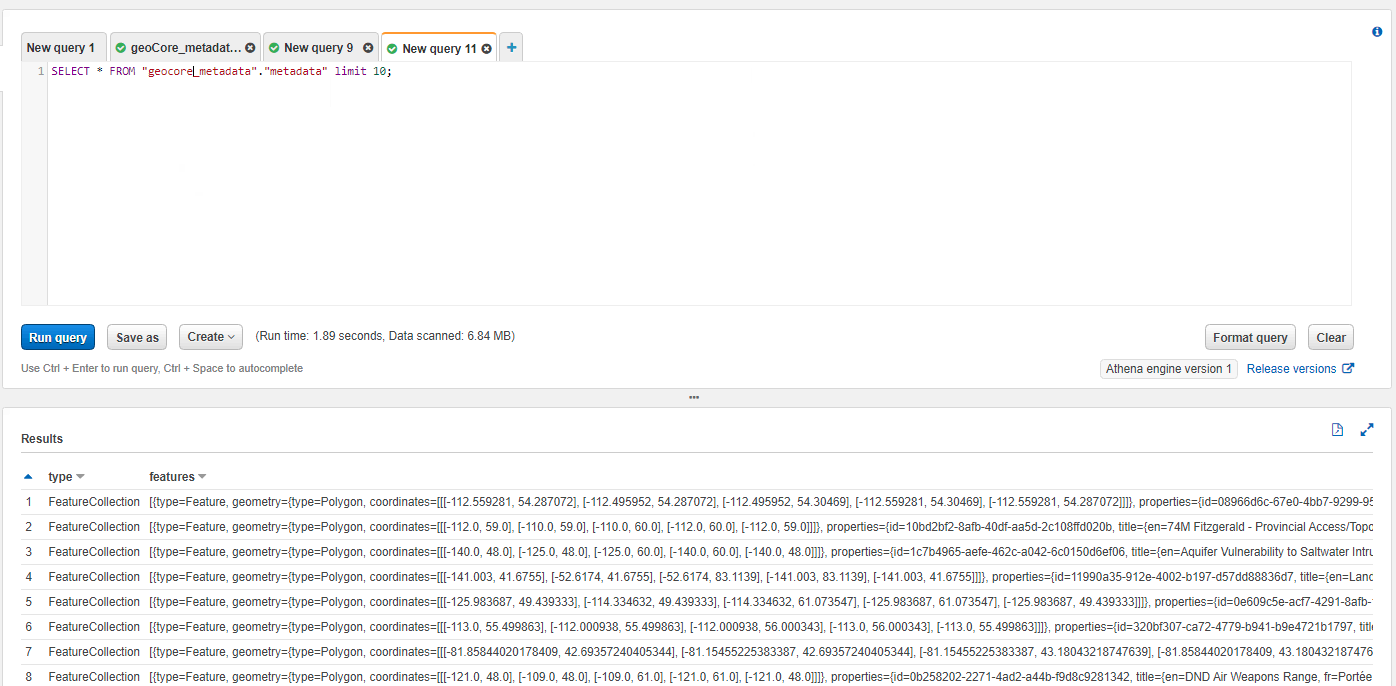
***END OF CODE BLOCK***



1. Click Run Query
2. In the left hand menu, you will now see a table named metadata.
3. Click on the three small dots on the metadata table, and click preview table



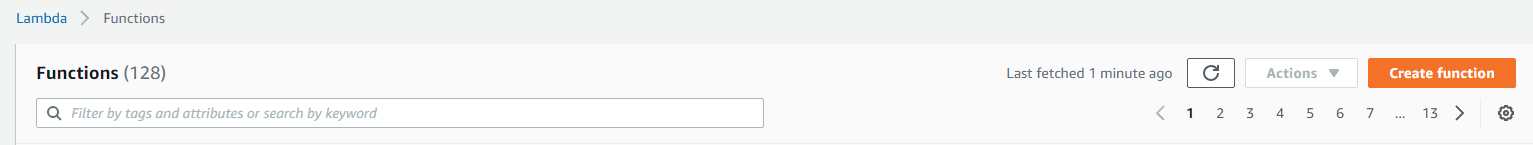
1. You will now see an output with several metadata records



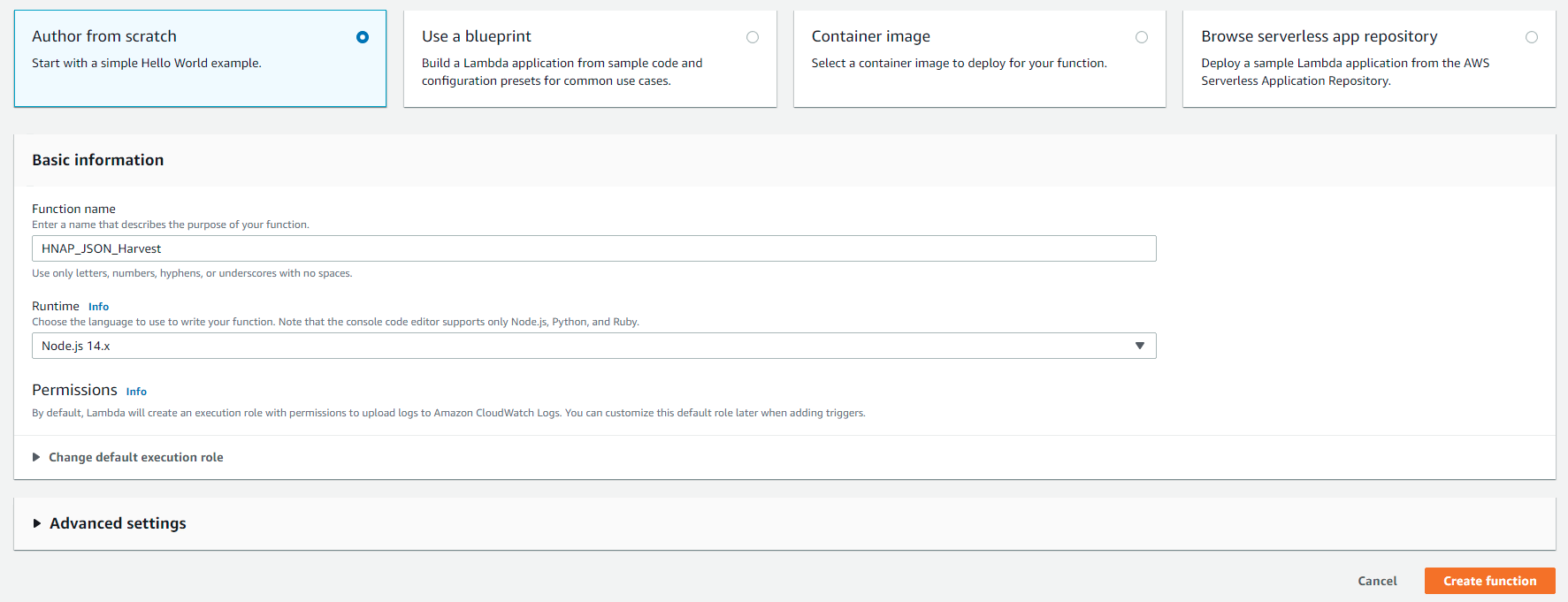
When you have accurate metadata previews as seen in the above image, your ready to move on to the next step.

### Creating the Query function

Now we will create the query function. Navigate to the Canadian Geospatial Platform’s github organisation. Download the production level zip file found under the geoCore\_searchfunction repository. Once this is available, navigate to the AWS Lambda Management Console.



1. Click Create Function
2. Make sure Author from scratch is selected
3. Name the function with search in the name to easily identify the function for later stages.
4. Select the latest Nodejs runtime
5. Click Create Function



1. When the function is created, the configuration window will open.
2. Click upload, and upload the zip that you received from the GitHub repository.
3. Click deploy to deploy the function.
4. Configure a test event with the following code
5. Make sure the Lambda function role has permissions to search Athena.

***ADD THIS CODE BLOCK TO THE TEST EVENT***

{

"north": "",

"east": "",

"south": "",

"west": "",

"keyword": "",

"keyword\_only": "true",

"lang": "en",

"theme": "",

"type": "",

"org": "",

"min": "1",

"max": "10",

"foundational": ""

}

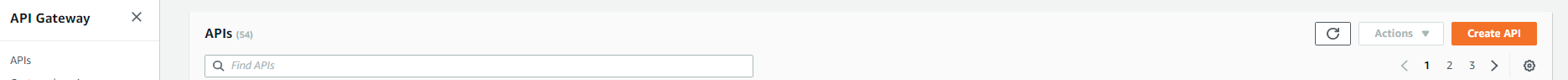
***END OF CODE BLOCK***

1. Click test, execution results should appear. If this is successful, move to the next step.

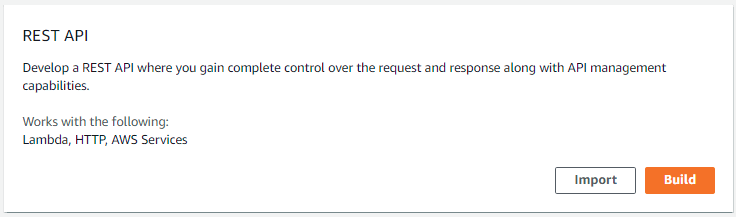
### Create an API using AWS API Gateway

Now we will create an API access point to trigger the Lambda function. Navigate to the API Gateway management console.

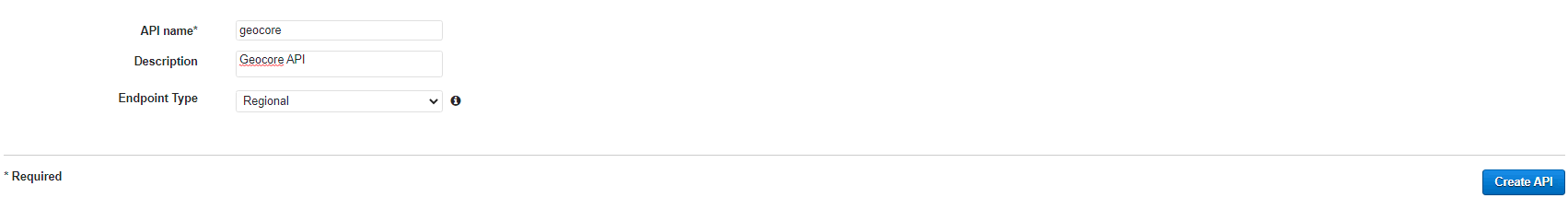
1. Click Create API



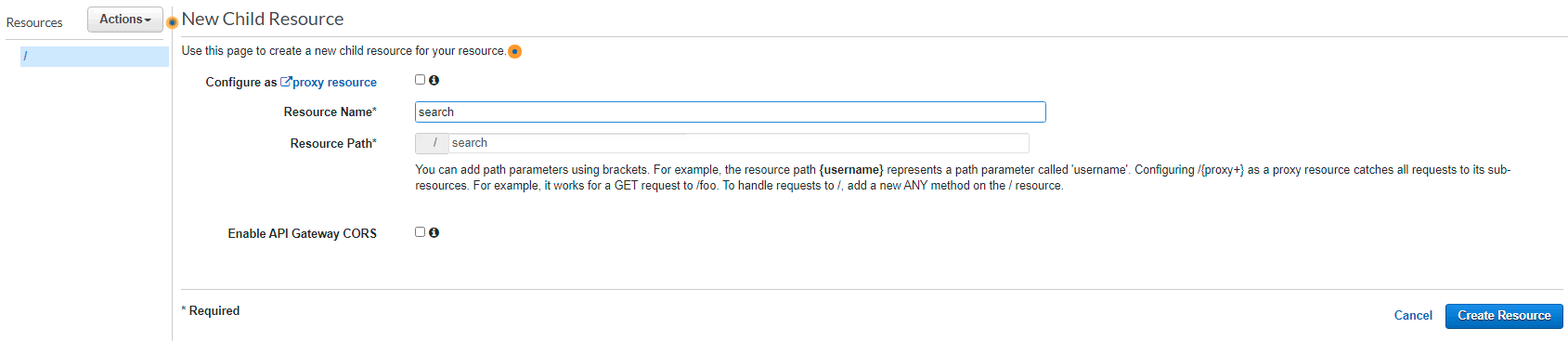
1. Find the box that says REST API and click Build



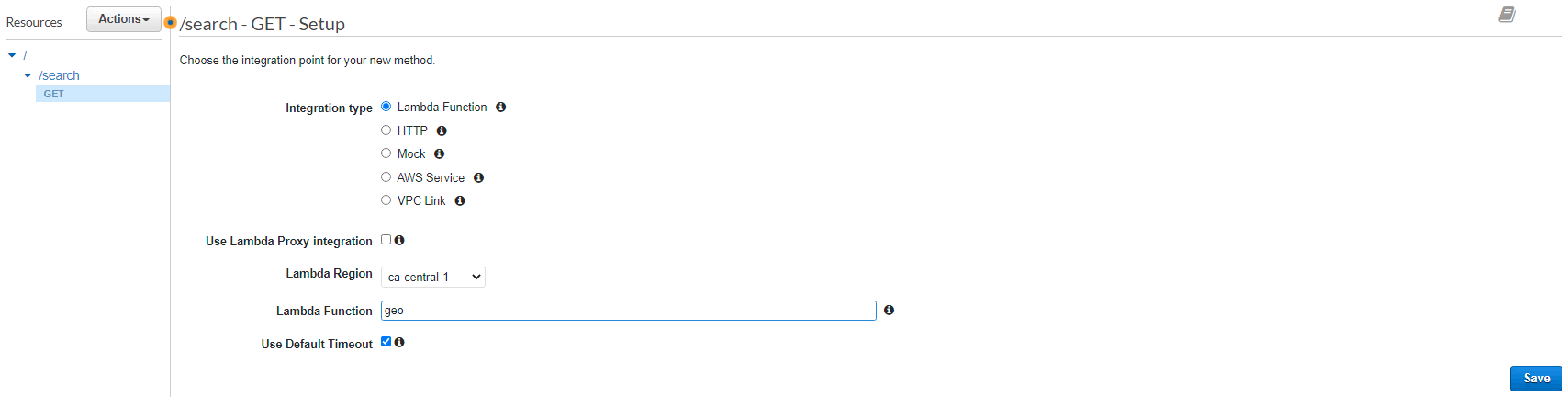
1. Name the API geocore and provide a short description



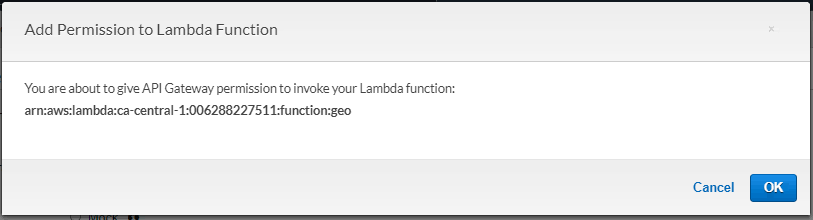
1. Click Create API
2. Once the API is created, click Action and Create Resource.
3. Add the resource name as search, and click Create Resource



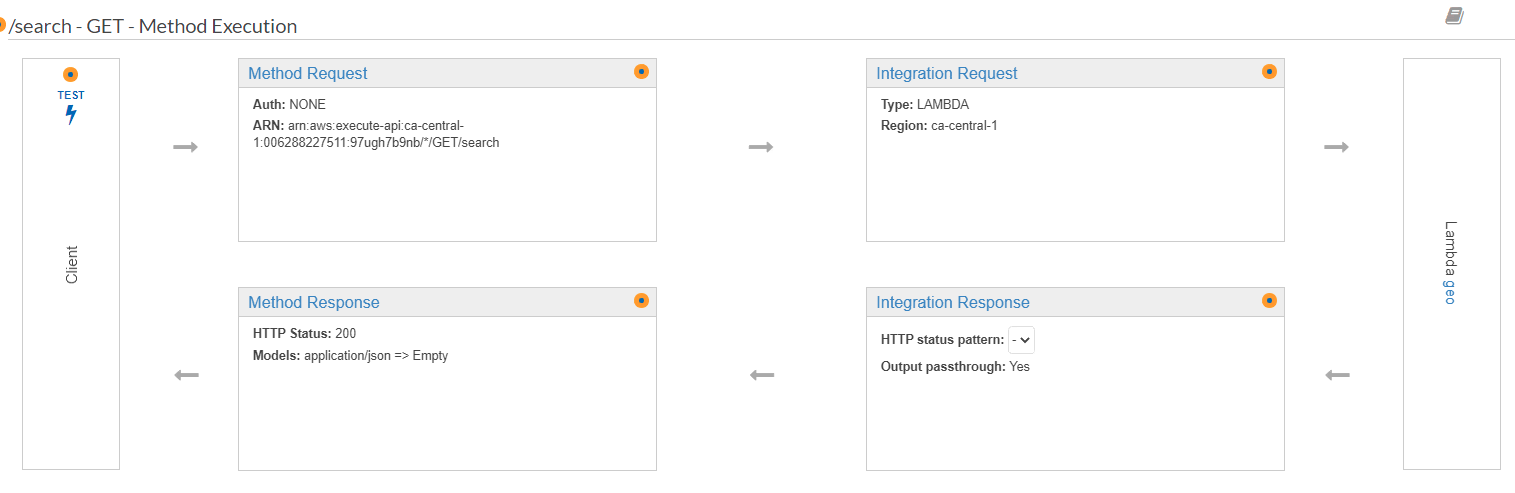
1. You will see search show up in the resources list on the left, click on search
2. Then click the action drop down and select Create Method, a small drop down will be presented below the search resource
3. Click GET in this drop down and select the checkmark next to the method
4. From here the configuration console will be displayed for this method



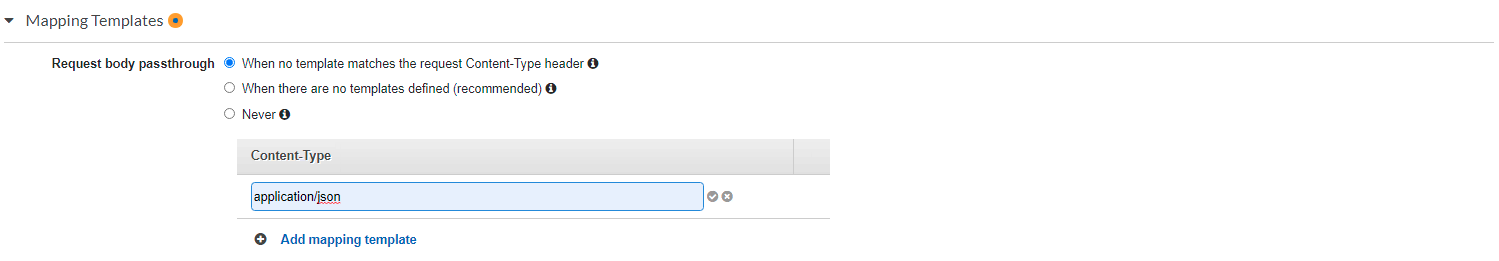
1. Select Lambda Function
2. Select the name of the search function created in an earlier step.
3. Click save.
4. A popup window will display, click OK



1. The method execution window will display, click Integration Request



1. Once the configuration is displayed, scroll down and click Mapping Template
2. Select When there are no templates defined (recommended)
3. Click add mapping template
4. Add application/json as the name and click the checkmark next to it



1. A popup will be displayed, click Yes, secure this integration
2. In the template text box add the following code

***ADD THIS CODE BLOCK TO THE TEMPLATE***

{

"north" : "$input.params('north')",

"east" : "$input.params('east')",

"south" : "$input.params('south')",

"west" : "$input.params('west')",

"keyword" : "$input.params('keyword')",

"keyword\_only" : "$input.params('keyword\_only')",

"lang" : "$input.params('lang')",

"theme" : "$input.params('theme')",

"type": "$input.params('type')",

"org": "$input.params('org')",

"min": "$input.params('min')",

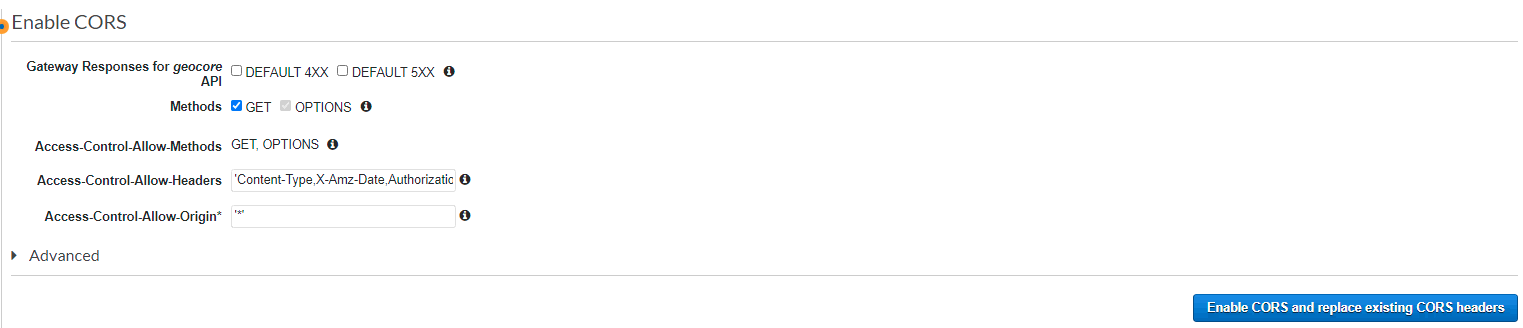
"max": "$input.params('max')",

"foundational": "$input.params('foundational')"

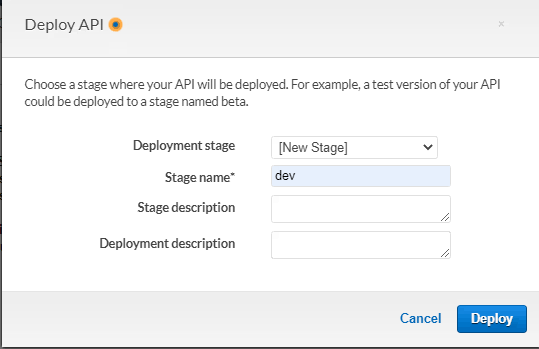
}

***END OF CODE BLOCK***

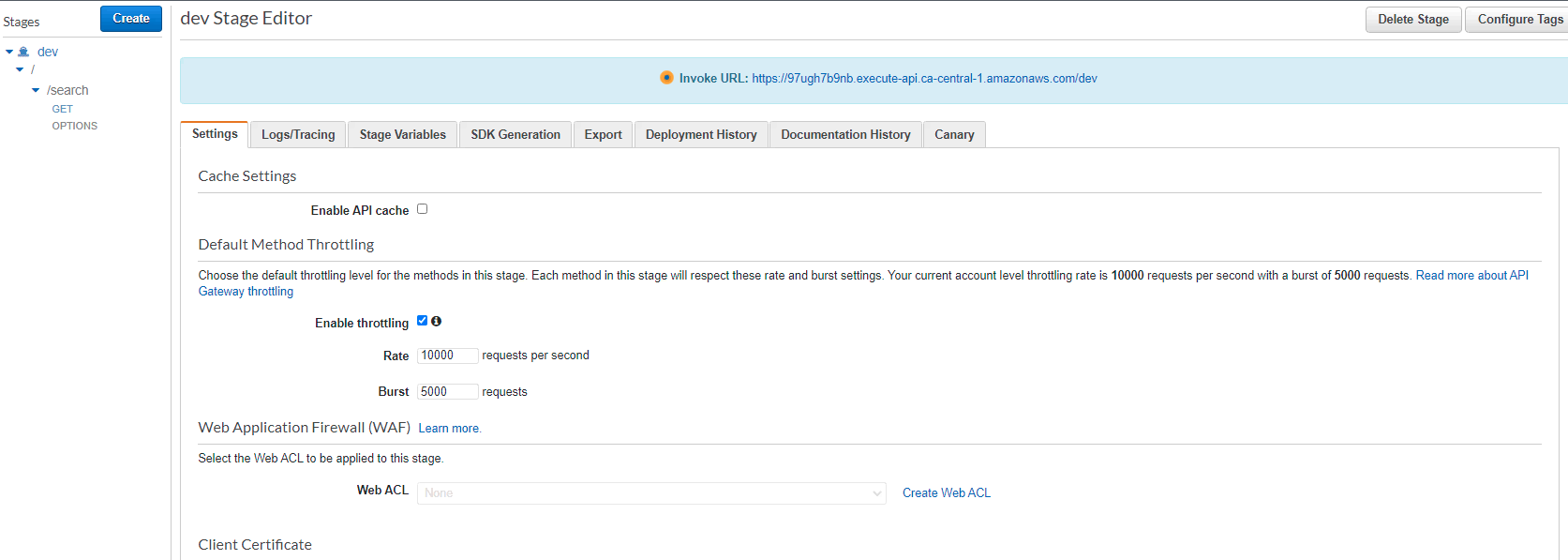
1. Click Save
2. From here, from the resources left hand panel, click Actions
3. Click Enable CORS
4. Click Enable CORS and replace existing CORS headers



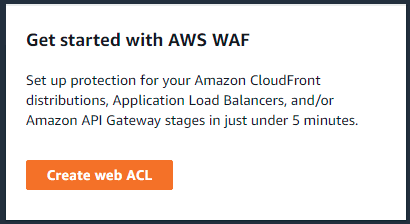
1. A popup window will be displayed, click yes, replace existing values
2. Now navigate to the Actions drop down, and click Deploy API
3. A popup window will display and select new stage.
4. Name the stage name after your environment, such as dev, staging, prod.
5. Click deploy



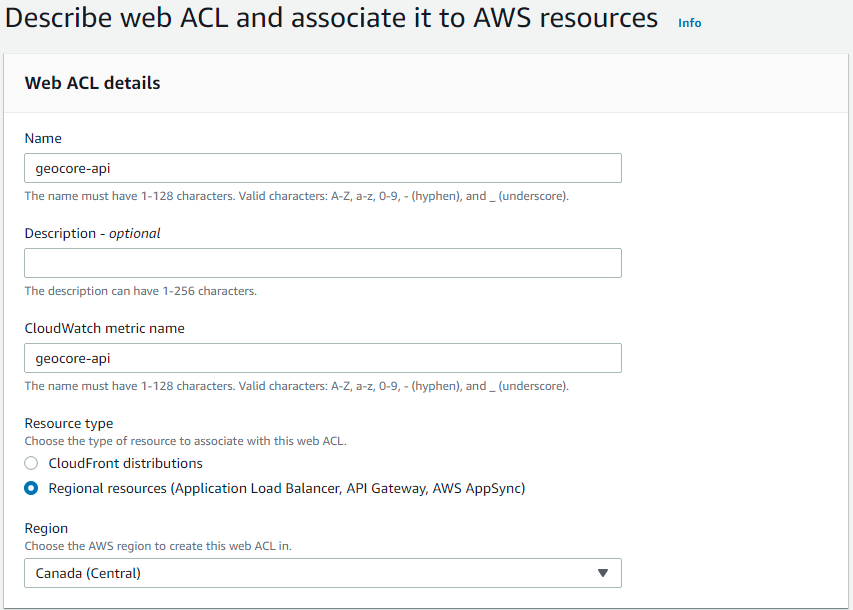
1. Navigate to the new management console, and find the search resource and the GET method underneath.
2. You can now see the Open API endpoint for use in the GEO.CA Search and Discovery Application
3. To protect this from unwanted attacks at any stage, navigate to the settings menu of the stage. Click Create Web ACL you will be taken to the Web ACL landing page. Click Create Web ACL again.



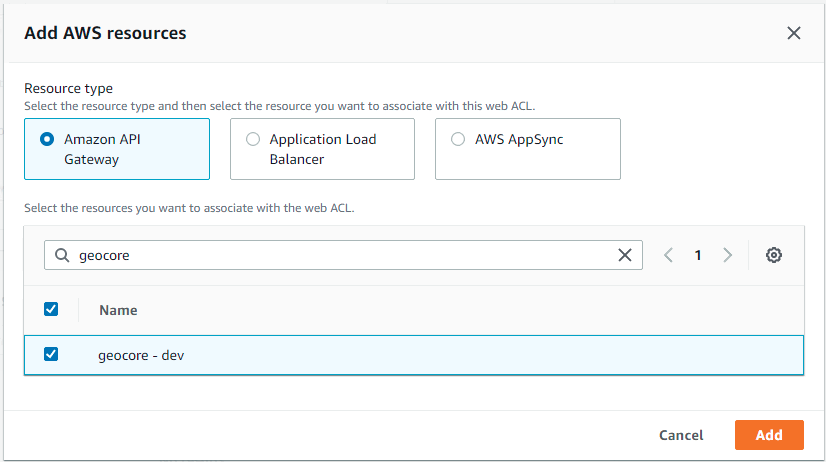
#### Creating an AWS Web Application Firewall



1. Fill in geocore-api as the name, and make sure the Canada Central region is selected.



1. Click Add AWS Resources
2. Select Amazon API Gateway
3. Select the geocore API
4. Click Add



1. Click Next
2. On the next screen, add the rules required for security.
3. Make Allow as the default action
4. Click Next on the next three screens and click Create Web ACL
5. In the left hand panel, click Rule groups.
6. Click create rule group, and make sure you are in Canada Central
7. Name the rule group, such as XSS-Blocker.
8. Click Add Rule
9. Name the new rule like XSS-Blocker-Rule
10. Select If the request matches a statement.
11. Select All query parameters
12. Select Contains XSS injection Atacks.
13. Click Block
14. Click Custom Response with response 400.
15. Type in { “error”: “XSS Blocked” }
16. Click Add Rule
17. Click Next
18. Click Next
19. Click Create rule group
20. Add this rule to the WAF.
21. Once this is completed, return to the API Gateway Stage settings
22. Select the WAF created
23. Click Save changes

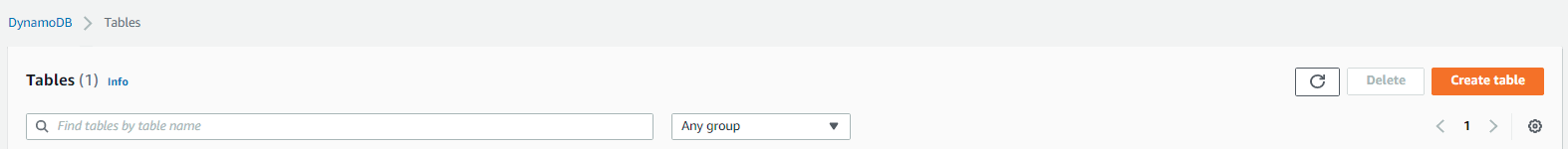
The main geocore API is now complete, the next steps are the creation of query functions to support the dynamic content presented as part of GEO.CA

### Creating supporting query functions for the Web Presence

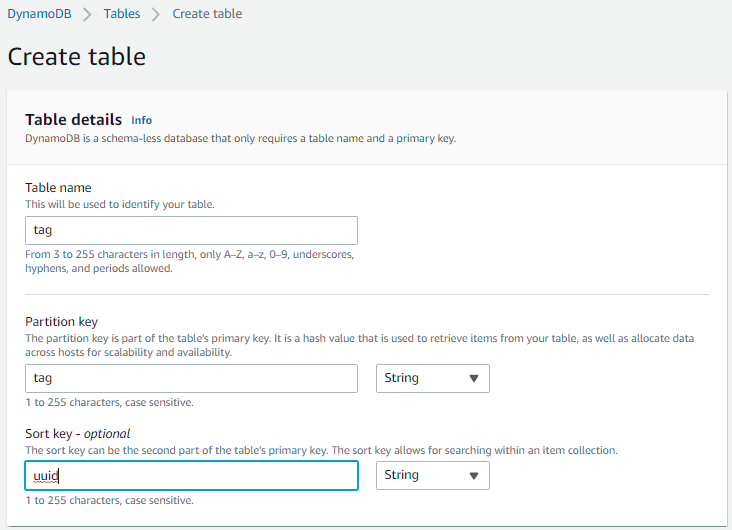
#### Creating DynamoDB tagging system

Since the metadata contributed to our system is owned by the respective organisations, we aren’t able to alter it or make additions to the core metadata. We need to add another layer of metadata to allow for additional information to be added, or tagged to each dataset. Such as to feature the content record on our web presence. This is accomplished by creating a cloud-first database to query against. For geocore’s tagging system, it’s DynamoDB.

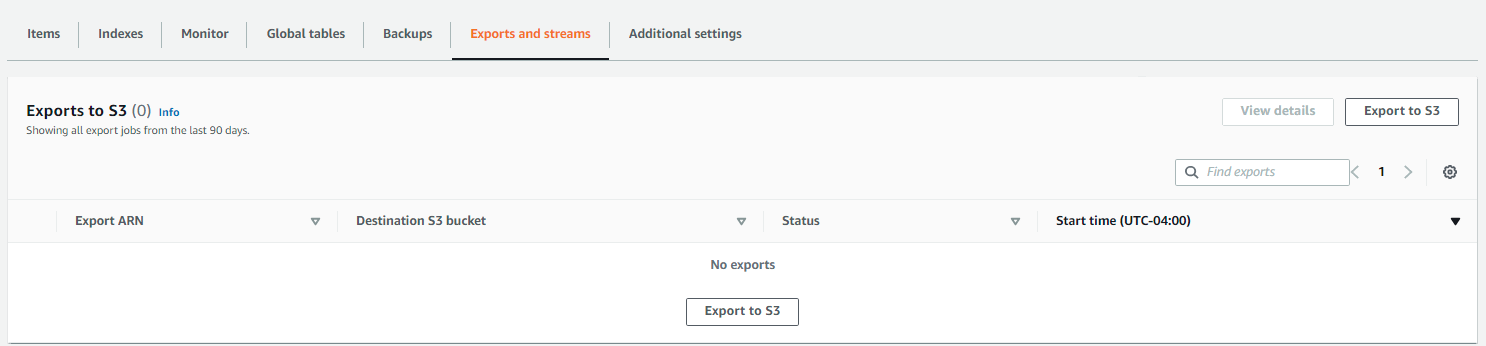
1. Navigate to the DynamoDB console.
2. Click Create Table



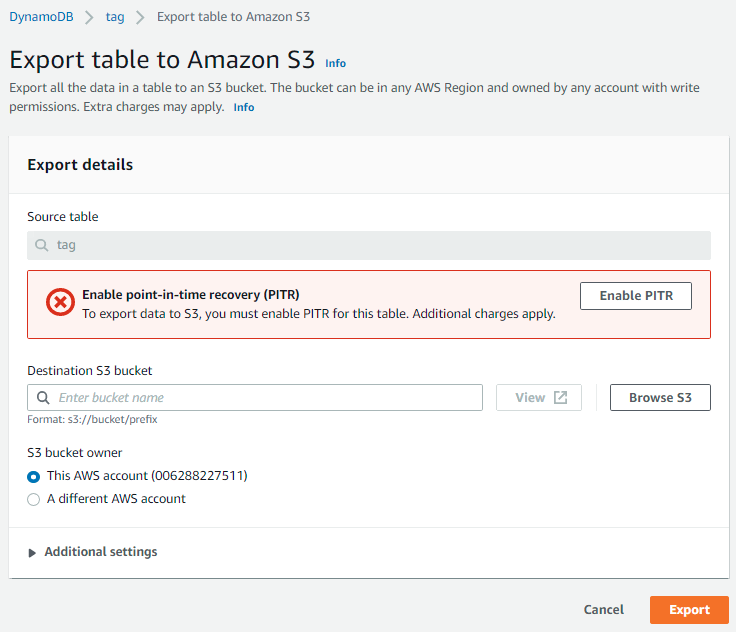
1. Enter the name of the table as tag
2. For partition key, enter the value tag
3. For sort key, enter the value uuid
4. The default settings can be used for this deployment
5. Click Create Table



1. To enable backup of the data to an s3 bucket using a mechanism called Point-in-Time Recovery, create an s3 bucket for this purpose by following steps 1-6.
2. Navigate to the Exports and Streams tab of the table configuration



1. Click Export to s3
2. Click Enable PITR

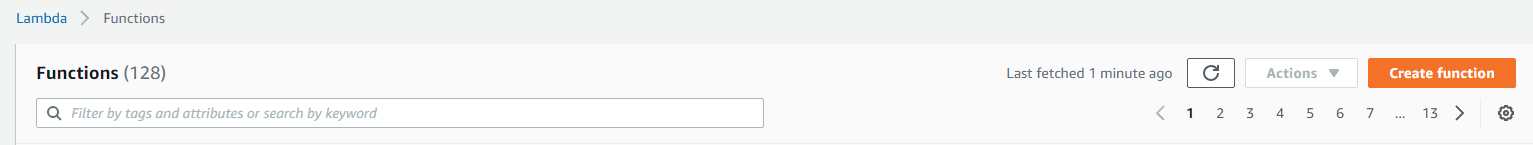


1. Select the s3 bucket created for export
2. Click Export
3. This will automatically back up your DynamoDB table.
4. If you need to import data, please follow your preferred method
5. Repeat steps 93-106 with the following information

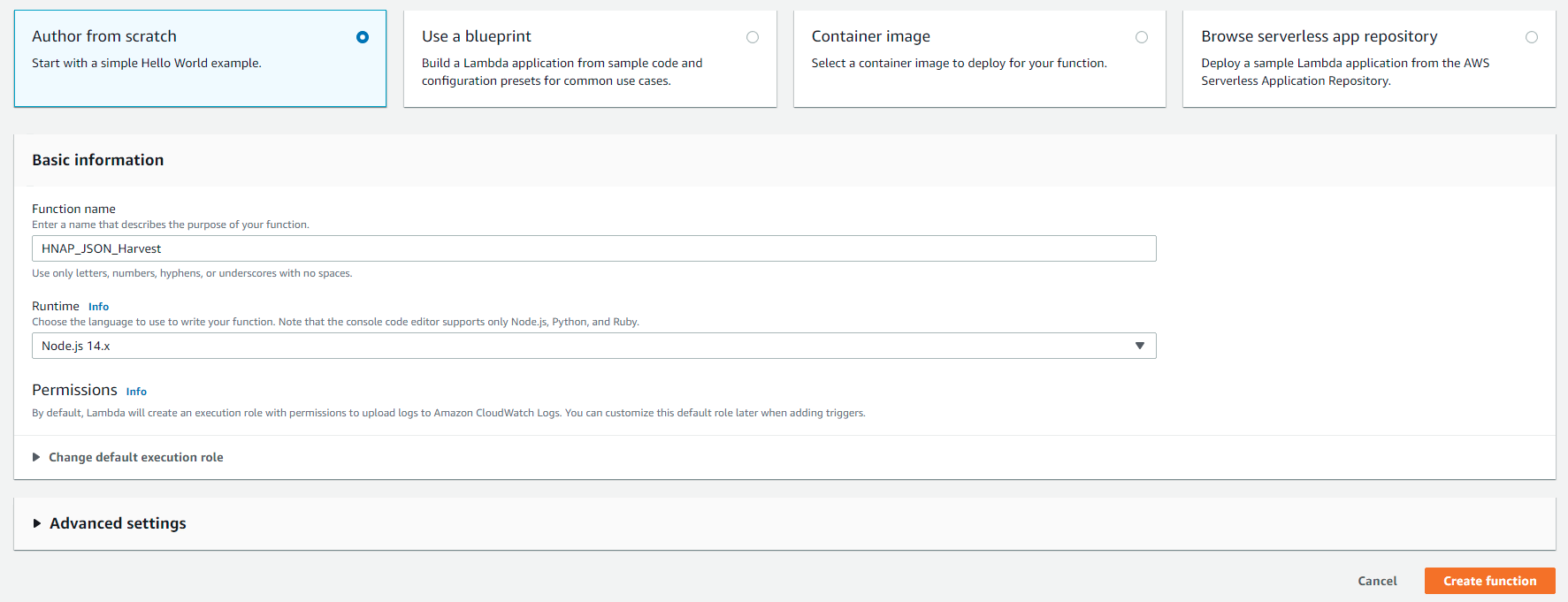
| **Table Name** | **Partition Key** | **Sort Key** |
| --- | --- | --- |
| tag *(created)* | tag (string) | uuid (string) |
| foundational | tag (string) | uuid (string) |
| theme | tag (string) | uuid (string) |
| analytics | key (string) |  |

### Creating the Search by ID endpoint

To get the information of a single metadata record, we will create another API endpoint that will allow systems to get the full metadata record of a single item. Navigate to the Canadian Geospatial Platform’s github organisation. Download the production level zip file found under the geoCore\_searchfunction repository for Search by ID. Once this is available, navigate to the AWS Lambda Management Console.



1. Click Create Function
2. Make sure Author from scratch is selected
3. Name the function with search by id in the name to easily identify the function for later stages.
4. Select the latest Nodejs runtime
5. Click Create Function



1. When the function is created, the configuration window will open.
2. Click upload, and upload the zip that you received from the GitHub repository.
3. Confirm that the Athena database in the function is the same created as part of this document.
4. Click deploy to deploy the function.
5. Configure a test event with the following code

***ADD THIS CODE BLOCK TO THE TEST EVENT***

{

"id": "c495dc2b-6988-4d1f-a6a4-6db4ac14c311",

"lang": "en"

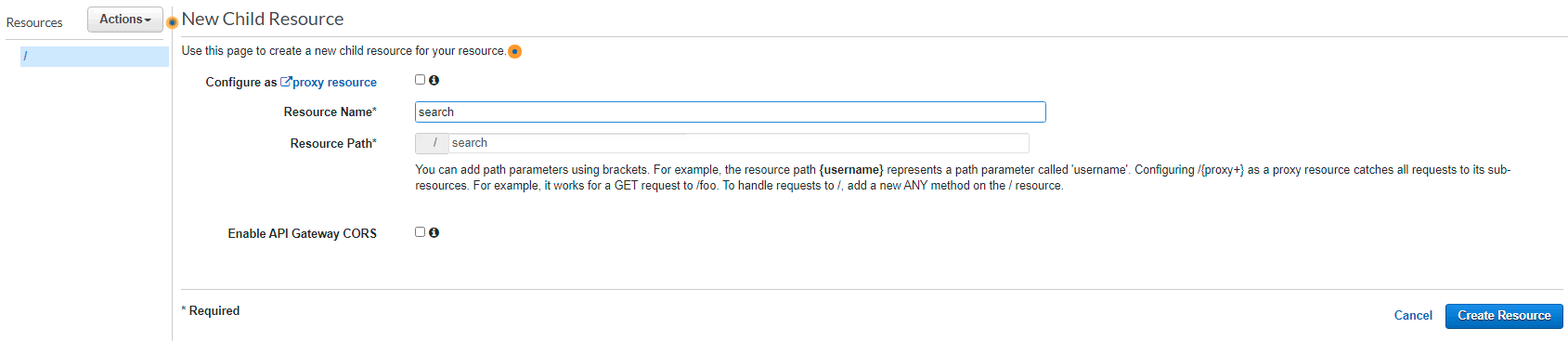
}

***END OF CODE BLOCK***

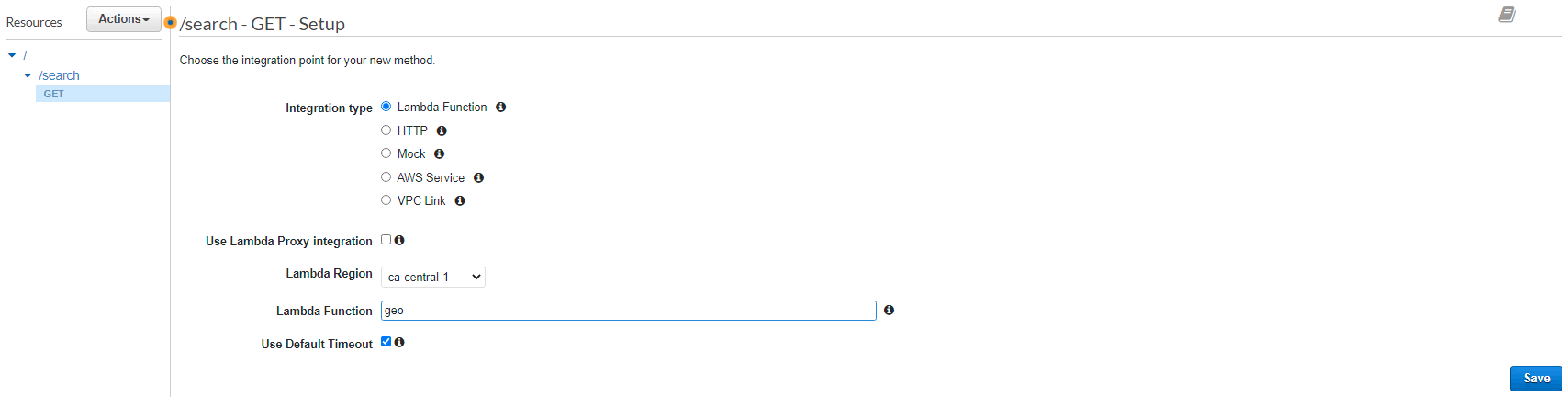
1. Click test, execution results should appear. If this is successful, move to the next step.

Now we will create an API access point to trigger the Lambda function. Navigate to the API Gateway management console.

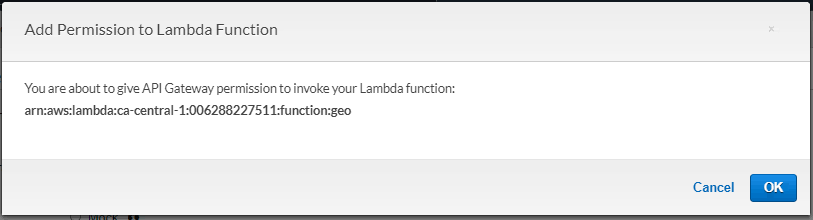
1. Click on the geocore API in the list.
2. Once at the API console, click Action and Create Resource.
3. Add the resource name as id, and click Create Resource



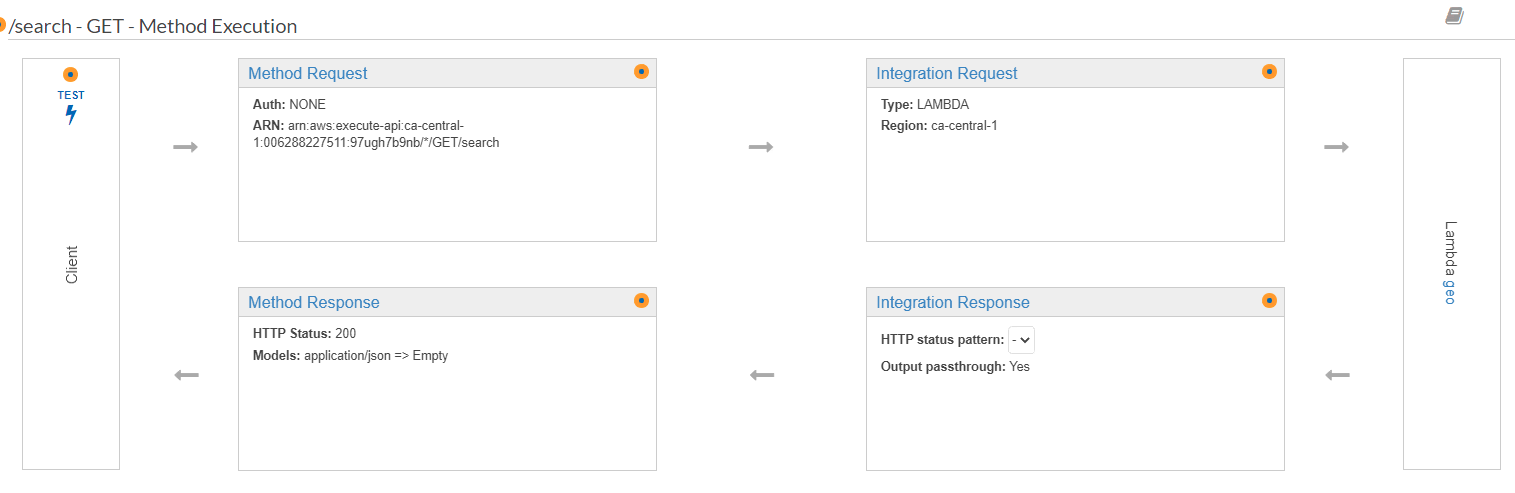
1. You will see search show up in the resources list on the left, click on search
2. Then click the action drop down and select Create Method, a small drop down will be presented below the search resource
3. Click GET in this drop down and select the checkmark next to the method
4. From here the configuration console will be displayed for this method



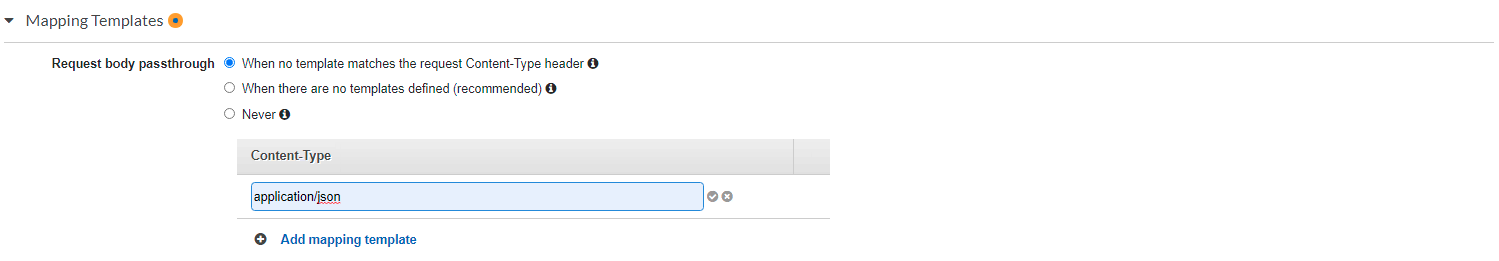
1. Select Lambda Function
2. Select the name of the search function created in an earlier step.
3. Click save.
4. A popup window will display, click OK



1. The method execution window will display, click Integration Request



1. Once the configuration is displayed, scroll down and click Mapping Template
2. Select When there are no templates defined (recommended)
3. Click add mapping template
4. Add application/json as the name and click the checkmark next to it



1. A popup will be displayed, click Yes, secure this integration
2. In the template text box add the following code

***ADD THIS CODE BLOCK TO THE TEMPLATE***

{

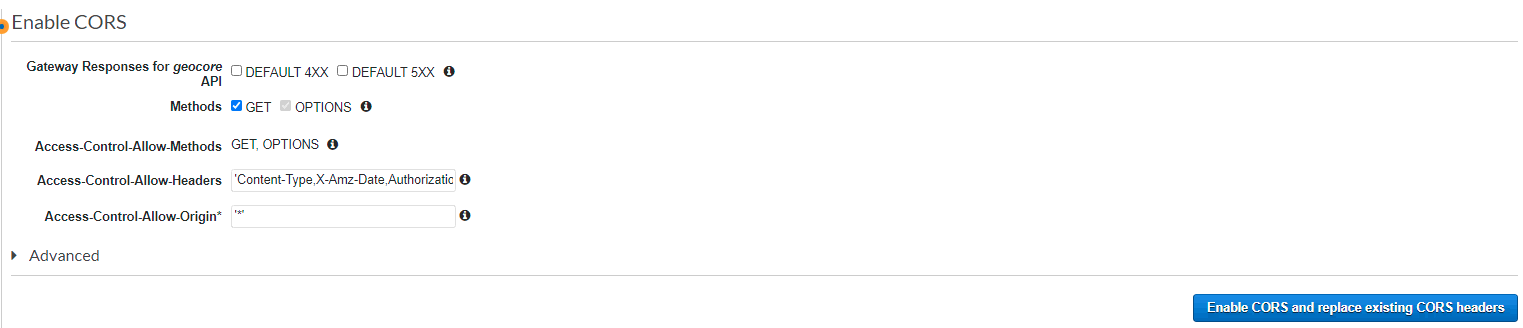
"lang" : "$input.params('lang')",

"id" : "$input.params('id')"

}

***END OF CODE BLOCK***

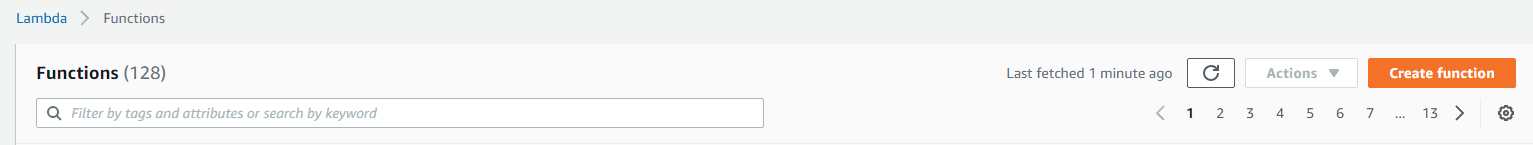
1. Click Save
2. From here, from the resources left hand panel, click Actions
3. Click Enable CORS
4. Click Enable CORS and replace existing CORS headers



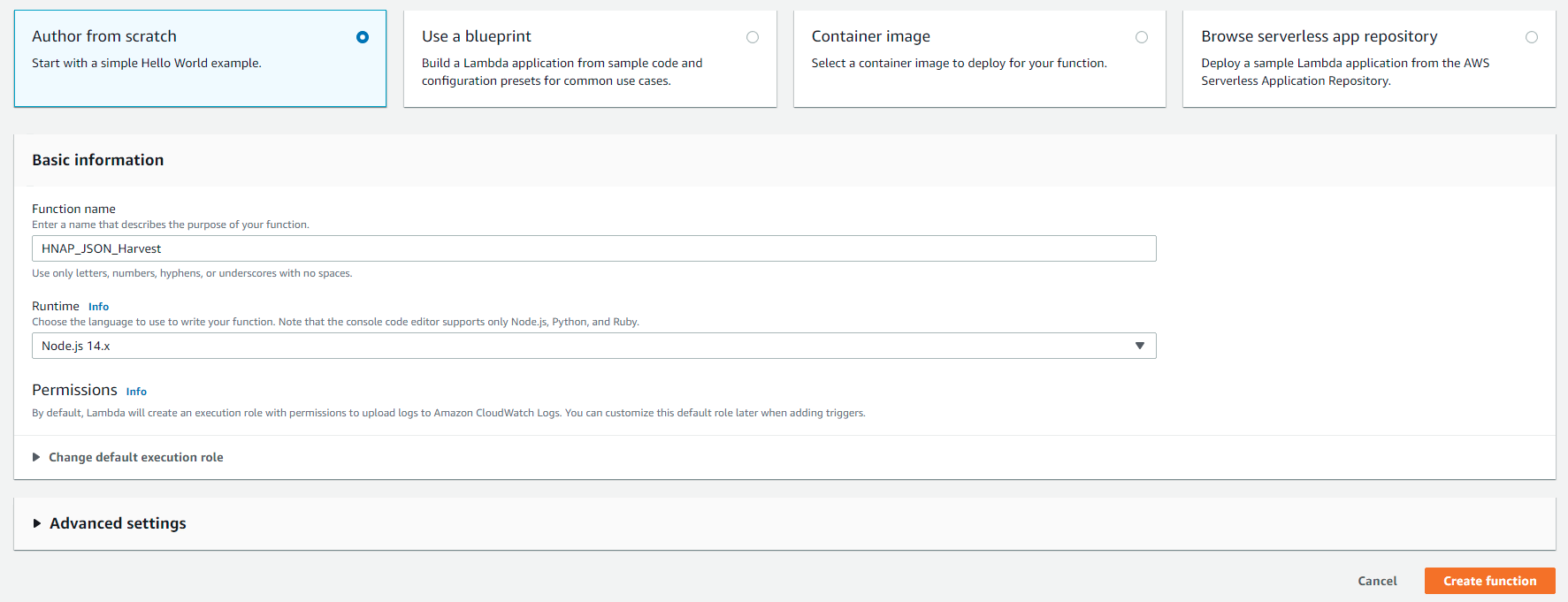
1. A popup window will be displayed, click yes, replace existing values
2. Now navigate to the Actions drop down, and click Deploy API
3. Select the stage created in an earlier step.
4. Click deploy
5. Navigate to the management console, and find the id resource and the GET method underneath.
6. You can now see the Open API endpoint for use in the GEO.CA Search and Discovery Application and is automatically protected from attacks because of the AWS WAF initially created with the geocore API.

### Creating supporting query API endpoints

There are several lambda functions that will need to be created to support the operations of the web presence at GEO.CA. The two main functions are featured and foundation\_layer.   
Navigate to the Canadian Geospatial Platform’s github organisation. Download the production level zip file found under the geoCore\_searchfunction repository for featured and foundational layers. Once this is available, navigate to the AWS Lambda Management Console.



1. Click Create Function
2. Make sure Author from scratch is selected
3. Name the function with featured in the name to easily identify the function for later stages.
4. Select the latest Nodejs runtime
5. Click Create Function



1. When the function is created, the configuration window will open.
2. Click upload, and upload the zip that you received from the GitHub repository.
3. Confirm that the DynamoDB database in the function is the same created as part of this document.
4. Click deploy to deploy the function.
5. Configure a test event with the following code

***ADD THIS CODE BLOCK TO THE TEST EVENT***

{

"theme": "administration",

"lang": "fr"

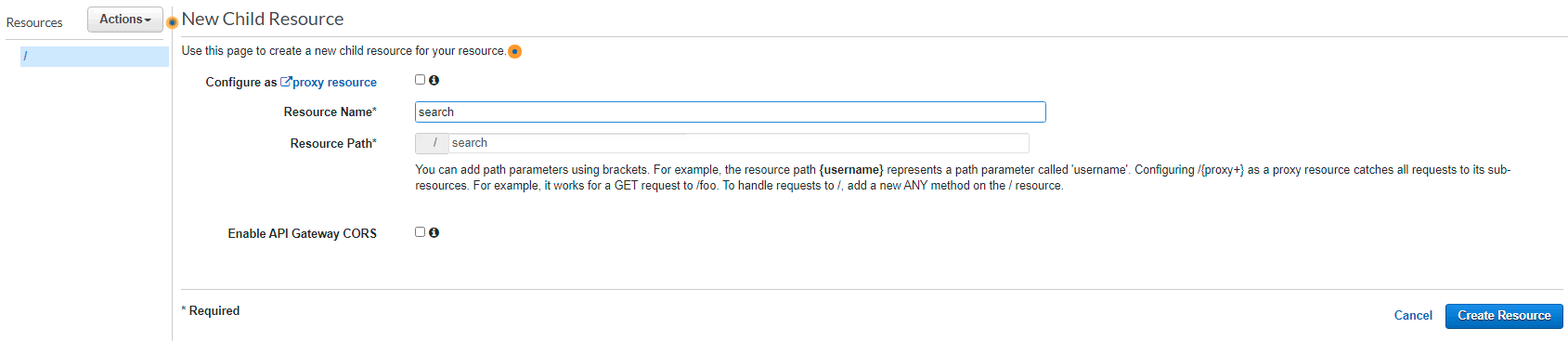
}

***END OF CODE BLOCK***

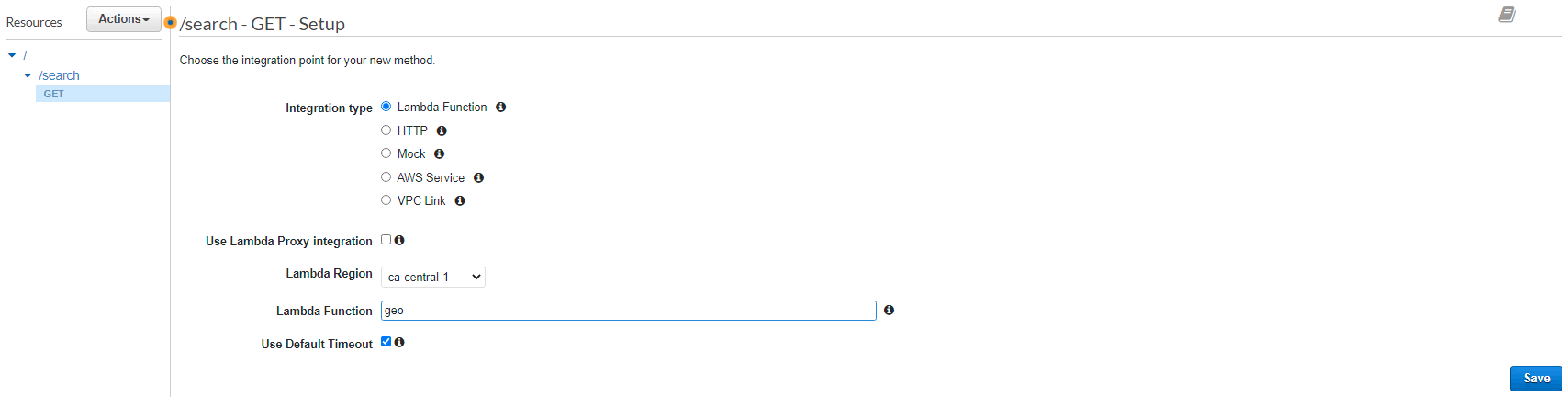
1. Click test, execution results should appear. If this is successful, move to the next step.

Now we will create an API access point to trigger the Lambda function. Navigate to the API Gateway management console.

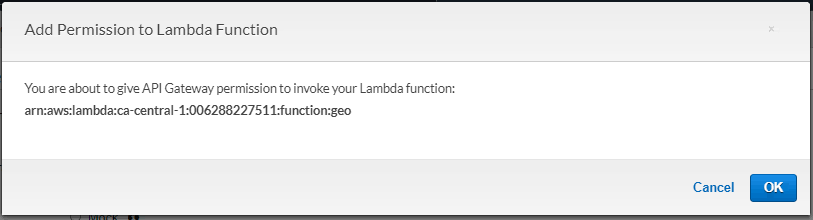
1. Click on the geocore API in the list.
2. Once at the API console, click Action and Create Resource.
3. Add the resource name as featured, and click Create Resource



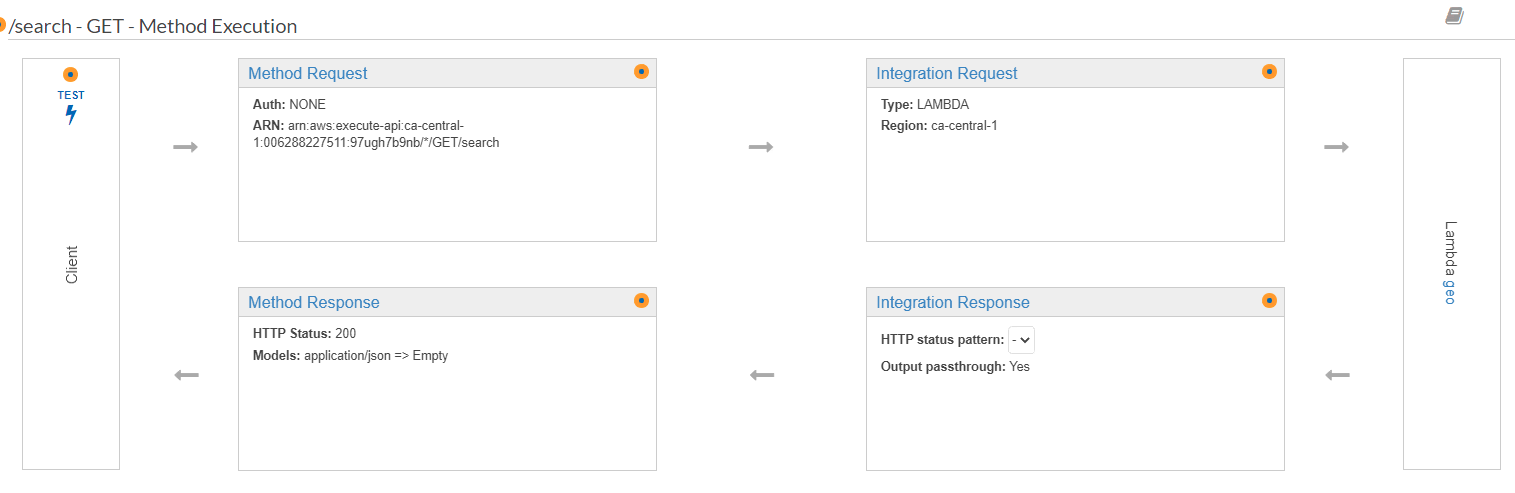
1. You will see featured show up in the resources list on the left, click on featured
2. Then click the action drop down and select Create Method, a small drop down will be presented below the search resource
3. Click GET in this drop down and select the checkmark next to the method
4. From here the configuration console will be displayed for this method



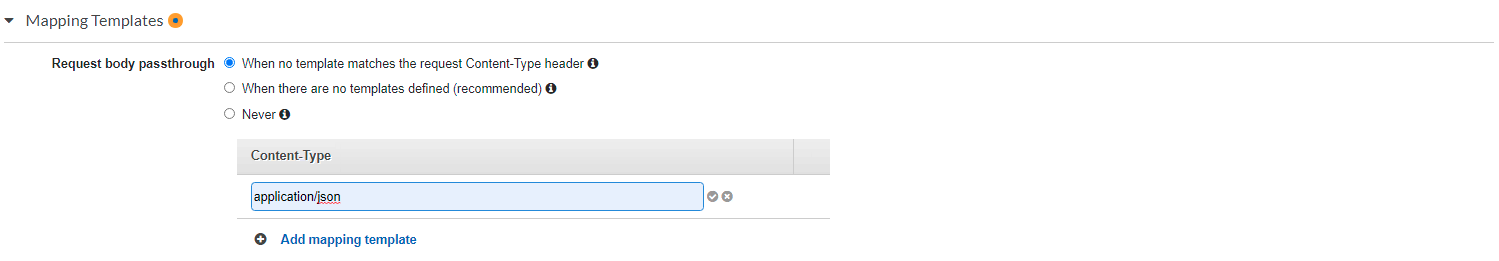
1. Select Lambda Function
2. Select the name of the search function created in an earlier step.
3. Click save.
4. A popup window will display, click OK



1. The method execution window will display, click Integration Request



1. Once the configuration is displayed, scroll down and click Mapping Template
2. Select When there are no templates defined (recommended)
3. Click add mapping template
4. Add application/json as the name and click the checkmark next to it



1. A popup will be displayed, click Yes, secure this integration
2. In the template text box add the following code

***ADD THIS CODE BLOCK TO THE TEMPLATE***

{

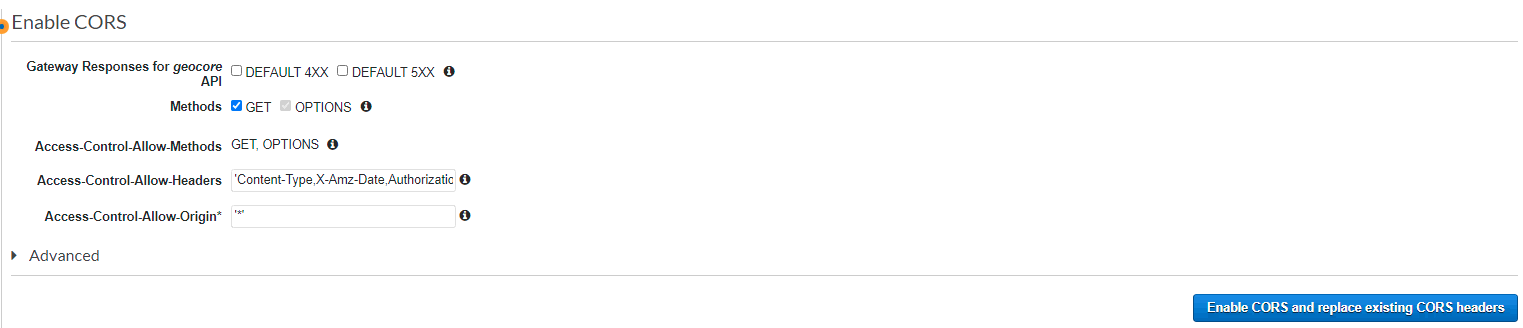
"theme" : "$input.params('theme')",

"lang" : "$input.params('lang')"

}

***END OF CODE BLOCK***

1. Click Save
2. From here, from the resources left hand panel, click Actions
3. Click Enable CORS
4. Click Enable CORS and replace existing CORS headers



1. A popup window will be displayed, click yes, replace existing values
2. Now navigate to the Actions drop down, and click Deploy API
3. Select the stage created in an earlier step.
4. Click deploy
5. Navigate to the management console, and find the id resource and the GET method underneath.
6. You can now see the Open API endpoint for use in the GEO.CA Search and Discovery Application and is automatically protected from attacks because of the AWS WAF initially created with the geocore API.
7. Repeat steps 145-184 with the foundational layer function.

### Creating the AWS Athena Analytics Database

To enhance the sql capabilities of dynamodb queries, we will use dynamodb as a data source for AWS Athena. For this we will need to launch a specialized cloudformation that connects DynamoDB to AWS Athena, this creates a Lambda function that allows this connection to be established.

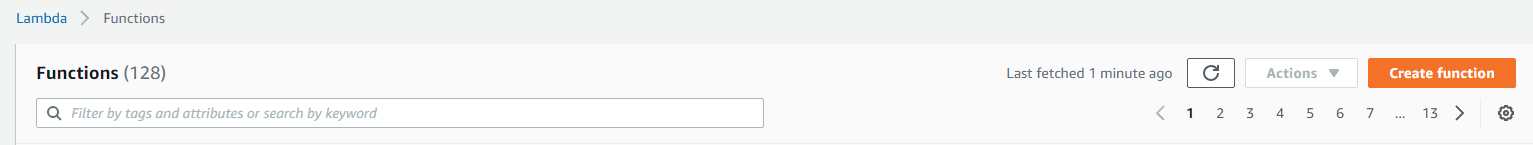
1. Navigate to the CloudFormation management console.
2. Click Create Stack with new resources
3. Upload the template file found at <https://github.com/Canadian-Geospatial-Platform/deployment_accessories/blob/main/serverlessrepo-AthenaDynamoDB.yml>
4. Click Next
5. Enter the stack name - such as AthenaDynamoDB
6. Enter the AthenaCatalogName of your choice
7. Enter the name of a SpillBucket
8. Leave all others as default
9. Click Next
10. Leave defaults and click Next.
11. Confirm information, and Click Acknowledge
12. Click Create Stack

Navigate to the AWS Athena management console

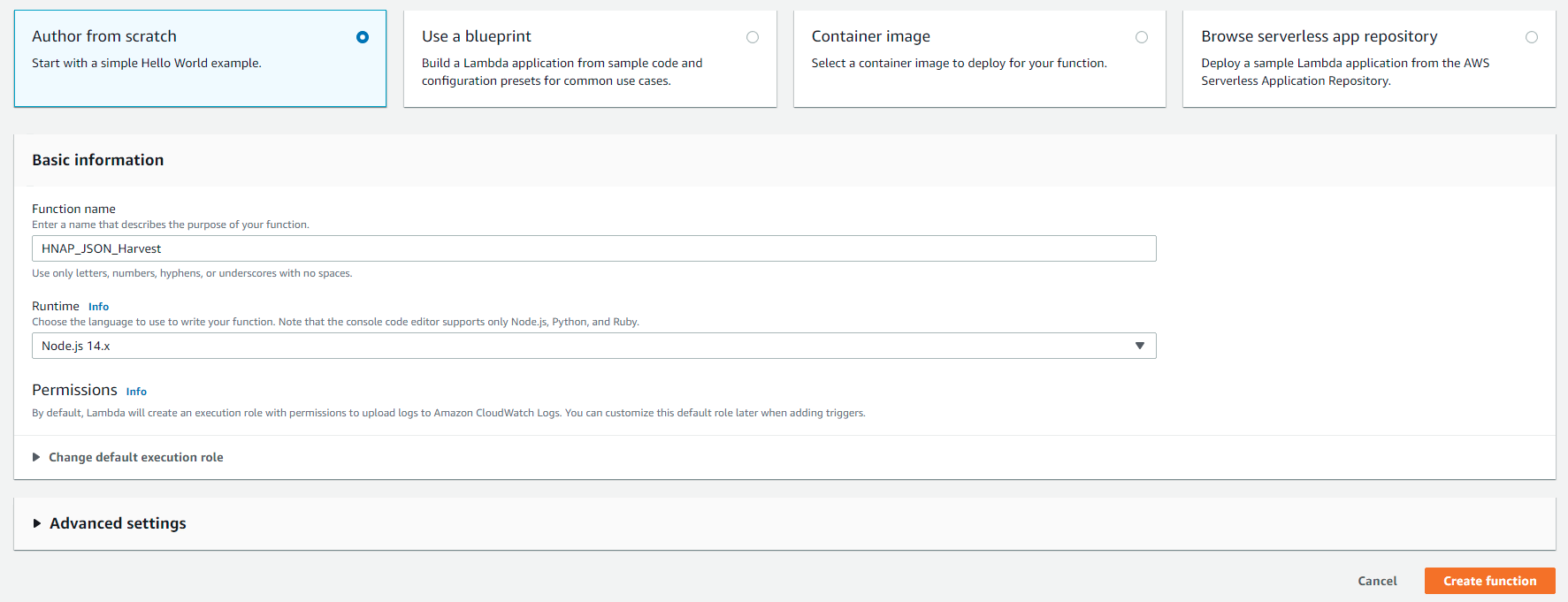
1. Click on data sources in the left panel
2. Click Connect data source
3. Click on DynamoDB
4. Enter analytics as the data source name
5. Provide a short description for the analytics system.
6. Select the Lambda function from the connection CloudFormation that is named as the same name as the CloudFormation stack.
7. Click Connect Dataset.

### Creating Analytics API endpoints

There are several lambda functions that will need to be created to support the Analytics system of the web presence at GEO.CA. The two main functions are analytics and get\_analytics.   
Navigate to the Canadian Geospatial Platform’s github organisation. Download the production level zip file found under the geoCore\_analytics repository for analytics and get\_analytics. Once this is available, navigate to the AWS Lambda Management Console.



1. Click Create Function
2. Make sure Author from scratch is selected
3. Name the function with analytics in the name to easily identify the function for later stages.
4. Select the latest Nodejs runtime
5. Click Create Function



1. When the function is created, the configuration window will open.
2. Click upload, and upload the zip that you received from the GitHub repository.
3. Confirm that the DynamoDB database in the function is the same created as part of this document with the analytics table.
4. Click deploy to deploy the function.
5. Configure the Athena database as mentioned in the code to point to the Athena database created in steps 187-205.
6. Configure a test event with the following code

***ADD THIS CODE BLOCK TO THE TEST EVENT***

{

"search": "salmon 2",

"theme": "environment",

"filter": "",

"geo": "[[75,75],[75,75]]",

"uuid": "",

"resource": "",

"resource\_type": "",

"loc": "search",

"lang": "en",

"type": "search",

"org": "DFO",

"foundational": "yes",

"type\_filter": "API"

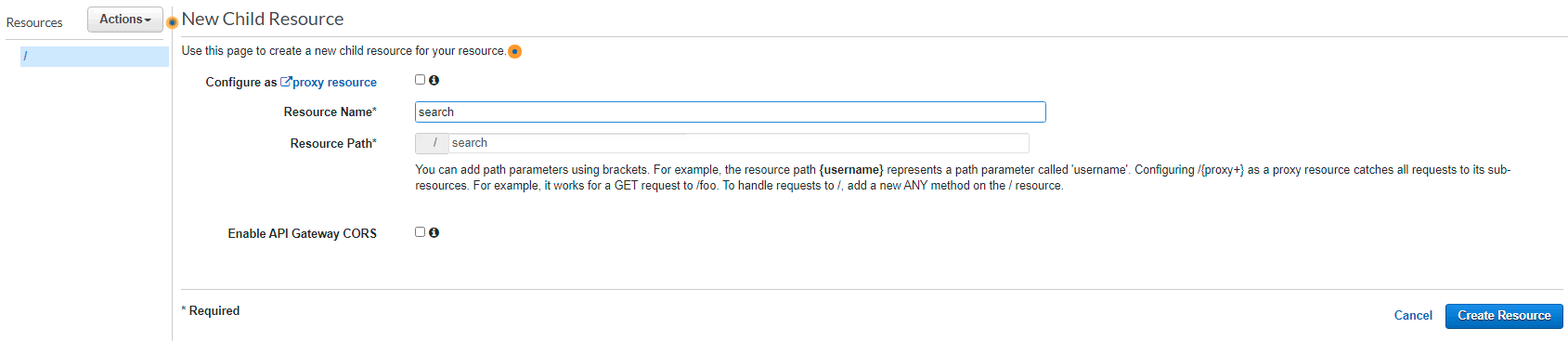
}

***END OF CODE BLOCK***

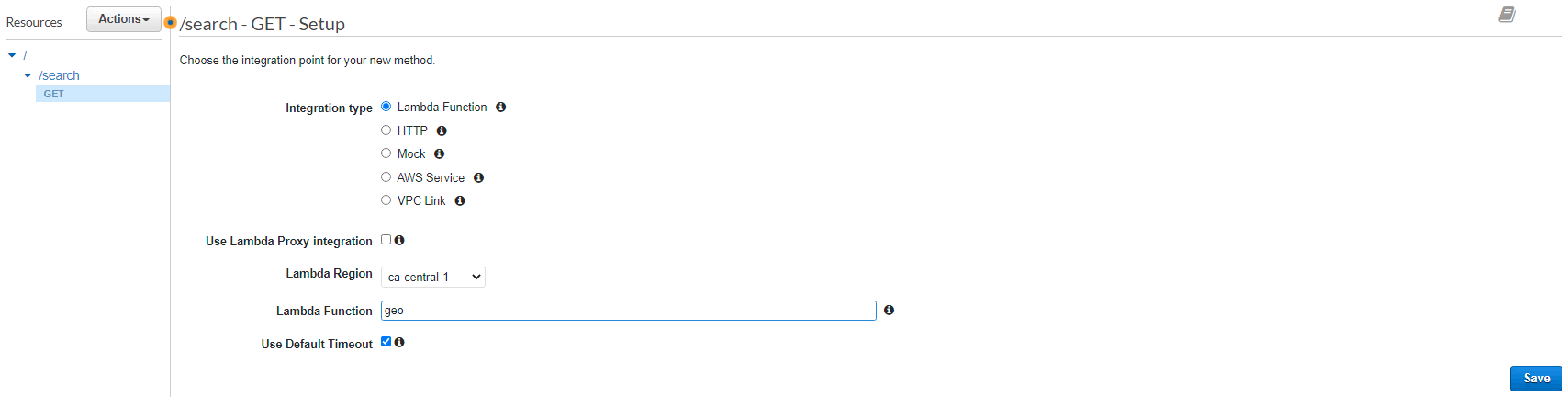
1. Click test, execution results should appear. If this is successful, move to the next step.

Now we will create an API access point to trigger the Lambda function. Navigate to the API Gateway management console.

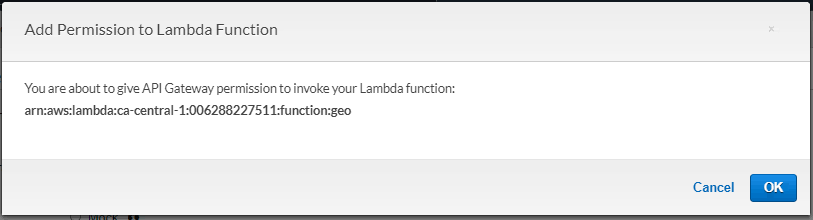
1. Click on the geocore API in the list.
2. Once at the API console, click Action and Create Resource.
3. Add the resource name as analytics, and click Create Resource



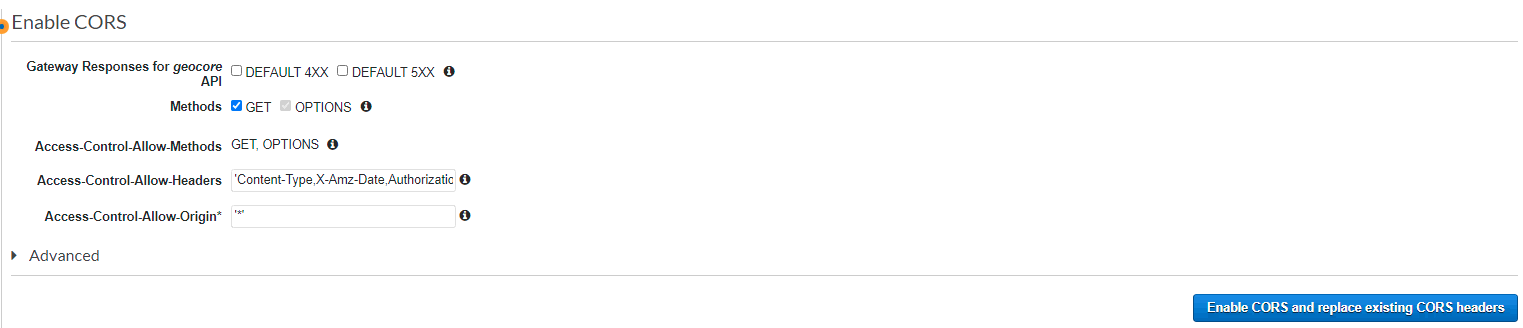
1. You will see analytics show up in the resources list on the left, click on analytics
2. Then click the action drop down and select Create Method, a small drop down will be presented below the search resource
3. Click POST in this drop down and select the checkmark next to the method
4. From here the configuration console will be displayed for this method



1. Select Lambda Function
2. Select the name of the analytics function created in an earlier step.
3. Click save.
4. A popup window will display, click OK



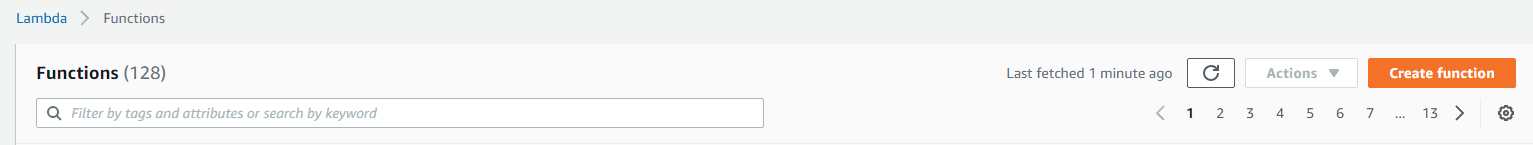
1. From here, from the resources left hand panel, click Actions
2. Click Enable CORS
3. Click Enable CORS and replace existing CORS headers



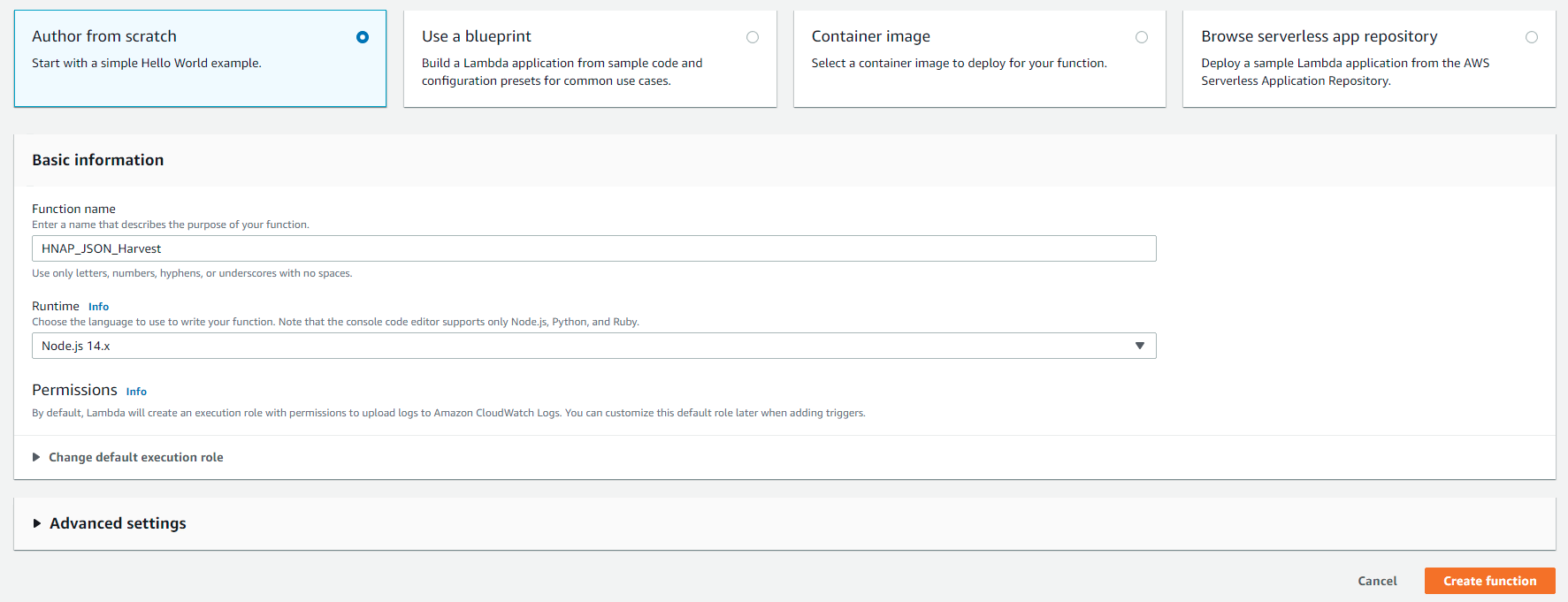
1. A popup window will be displayed, click yes, replace existing values
2. Now navigate to the Actions drop down, and click Deploy API
3. Select the stage created in an earlier step.
4. Click deploy
5. From the API Gateway management console, navigate to the API Keys menu.
6. Click Actions
7. Click Create API Key
8. Enter a name, and description, and allow the system to auto generate a key.
9. Click Save
10. Click on Usage plans in the navigation menu
11. Click Create
12. Name the usage plan
13. Uncheck Enable throttling and quotas
14. Click Next
15. Click Add API stage
16. Select the API and stage
17. Click the checkmark
18. Click Next
19. Click API Key to Usage Plan
20. Type in the name of the API Key, Select it from the list
21. Click the checkmark
22. Click Done
23. Navigate back to the analytics POST resource
24. Click on Method Request
25. Click Edit on API Key Required and make it true
26. Redeploy the API

Now that we have the analytics API available to record analytical events, we can start to use those analytics with the creation of the next function.

Navigate to the AWS Lambda console



1. Click Create Function
2. Make sure Author from scratch is selected
3. Name the function with get\_analytics in the name to easily identify the function for later stages.
4. Select the latest Nodejs runtime
5. Click Create Function



1. When the function is created, the configuration window will open.
2. Click upload, and upload the zip that you received from the GitHub repository.
3. Confirm that the DynamoDB database in the function is the same created as part of this document with the analytics table.
4. Click deploy to deploy the function.
5. Configure a test event with the following code

***ADD THIS CODE BLOCK TO THE TEST EVENT***

{

"id": "1"

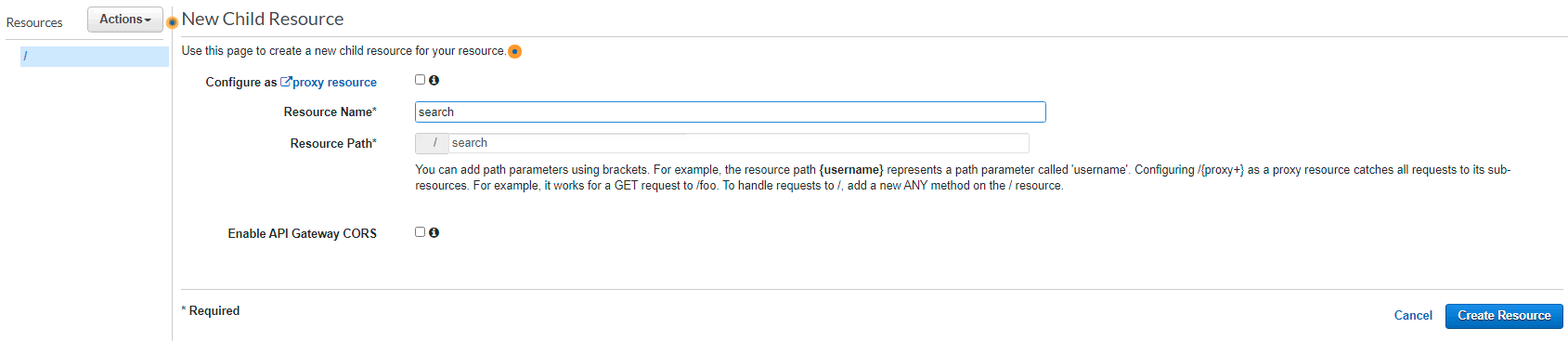
}

***END OF CODE BLOCK***

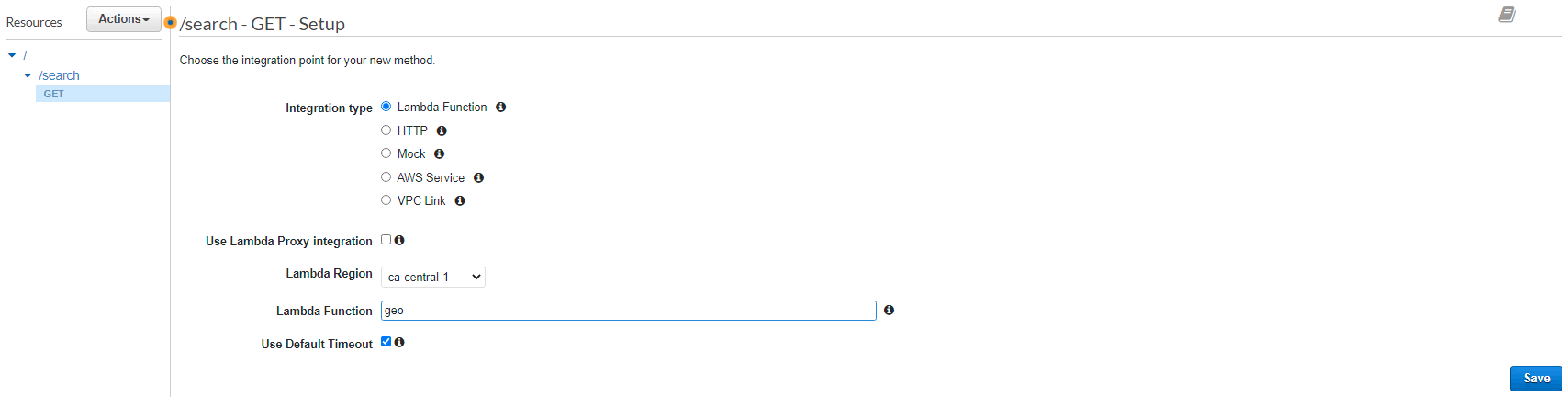
1. Click test, execution results should appear. If this is successful, move to the next step.

Now we will create an API access point to trigger the Lambda function. Navigate to the API Gateway management console. Unlike other API endpoints, we will be creating sub-API endpoints for the different functions of the API.

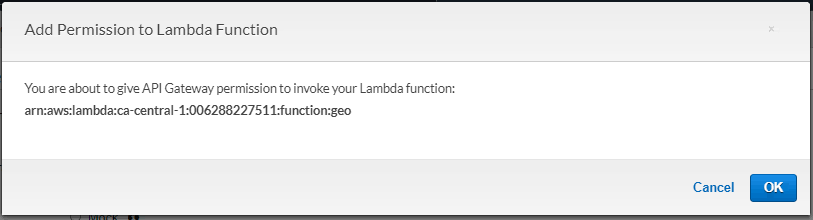
1. Click on the geocore API in the list.
2. Once at the API console, click analytics
3. Click Action and Create Resource.
4. Add the resource name as 1, and click Create Resource



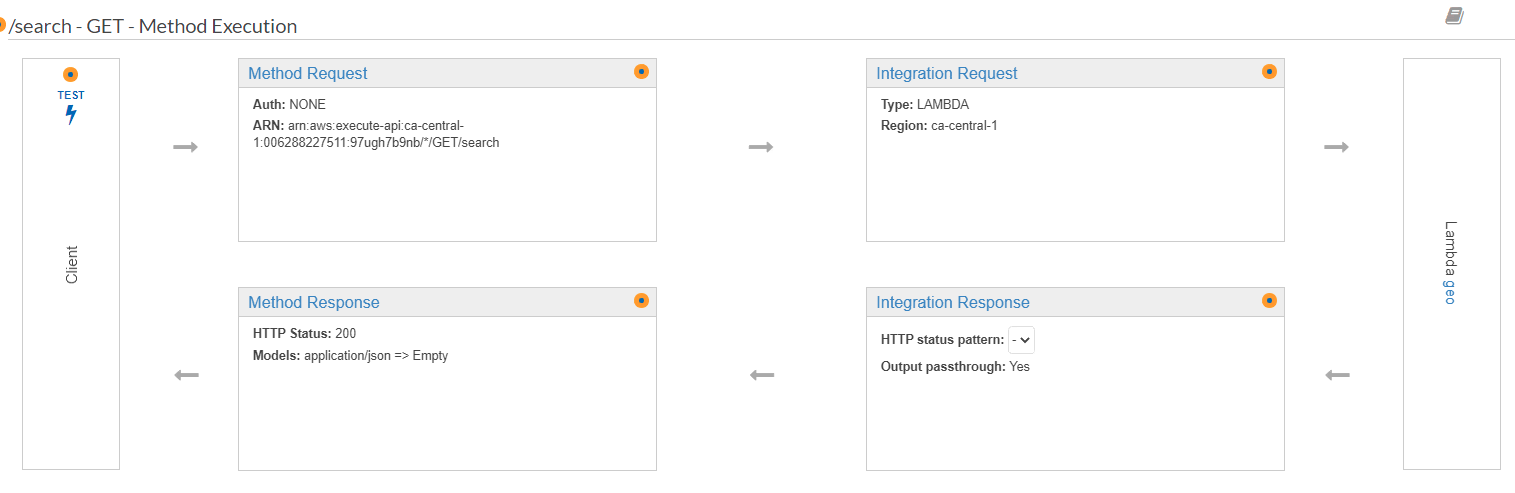
1. You will see the resource 1 show up in the resources list on the left, click on 1
2. Then click the action drop down and select Create Method, a small drop down will be presented below the search resource
3. Click GET in this drop down and select the checkmark next to the method
4. From here the configuration console will be displayed for this method



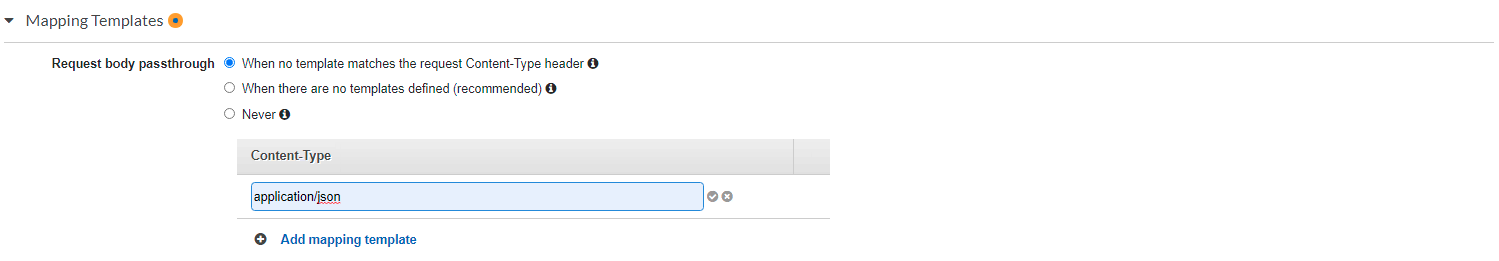
1. Select Lambda Function
2. Select the name of the get\_analytics function created in an earlier step.
3. Click save.
4. A popup window will display, click OK



1. The method execution window will display, click Integration Request



1. Once the configuration is displayed, scroll down and click Mapping Template
2. Select When there are no templates defined (recommended)
3. Click add mapping template
4. Add application/json as the name and click the checkmark next to it



1. A popup will be displayed, click Yes, secure this integration
2. In the template text box add the following code

***ADD THIS CODE BLOCK TO THE TEMPLATE***

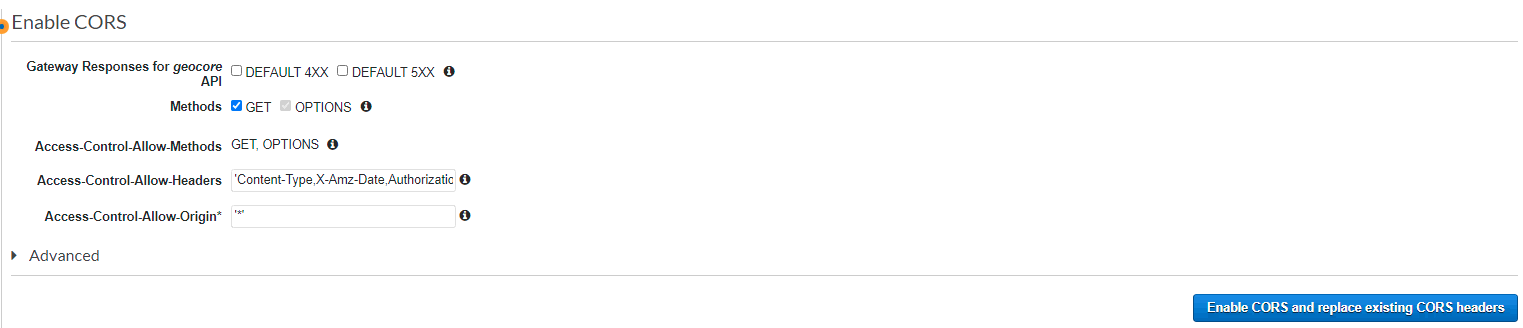
{

"id" : "1"

}

***END OF CODE BLOCK***

1. Click Save
2. From here, from the resources left hand panel, click Actions
3. Click Enable CORS
4. Click Enable CORS and replace existing CORS headers



1. A popup window will be displayed, click yes, replace existing values
2. Now navigate to the Actions drop down, and click Deploy API
3. Select the stage created in an earlier step.
4. Click deploy
5. Repeat steps 250-274 with each of the id’s from the get\_analytics functions. From id’s 1-10 using the following template for step 267.

| **Analytics Resource Name** | **Mapping Template** |
| --- | --- |
| 1 | { “id”: “1”  } |
| 2 | {  “id”: “2”  } |
| 3 | {  “id”: “3”  } |
| 4 | {  “id”: “4”  } |
| 5 | {  “id”: “5”  } |
| 6 | {  “id”: “6”  } |
| 7 | {  “id”: “7”  } |
| 8 | {  "id" : "8",  "lang" : "$input.params('lang')",  "theme" : "$input.params('theme')"  } |
| 9 | {  "id" : "9",  “lang" : "$input.params('lang')",  "theme" : "$input.params('theme')"  } |
| 10 | {  "id" : "10",  "uuid" : "$input.params('uuid')"  } |

### Creating DynamoDB Data Update Mechanism

To populate the DynamoDB with data for use with the system we can use a simple CloudFormation to ingest a csv stored in an s3 bucket which will update the DynamoDB table. Each table will have to be updated individually. Complete this process for each table you will have to update.

1. Navigate to the CloudFormation management console.
2. Click Create Stack with new resources
3. Upload the CloudFormation template from <https://github.com/Canadian-Geospatial-Platform/deployment_accessories/blob/main/dynamodb-update.yml>
4. Enter the stack name including the table name you will be updating from step 107
5. Enter a bucket name including the table name where you will place a csv file to upload to DynamoDB
6. Enter the DynamoDB table name you will be updating.
7. Enter the filename that you will use to update the data, an example of this is theme-update.csv This filename will be used to update the dynamodb table, any files uploaded to the s3 bucket without this filename will not update the table.
8. Click Next
9. Accept the defaults and Click Next.
10. Acknowledge the creation of resources, and click Create Stack
11. You can now navigate to the s3 bucket and upload the file to update the DynamoDB table.

The build of the geocore geospatial library and supporting query function is now complete. You will need to confirm endpoint URLs to the GEO.CA Web Presence and Search and Discovery Application for the environment that you will be deploying in.