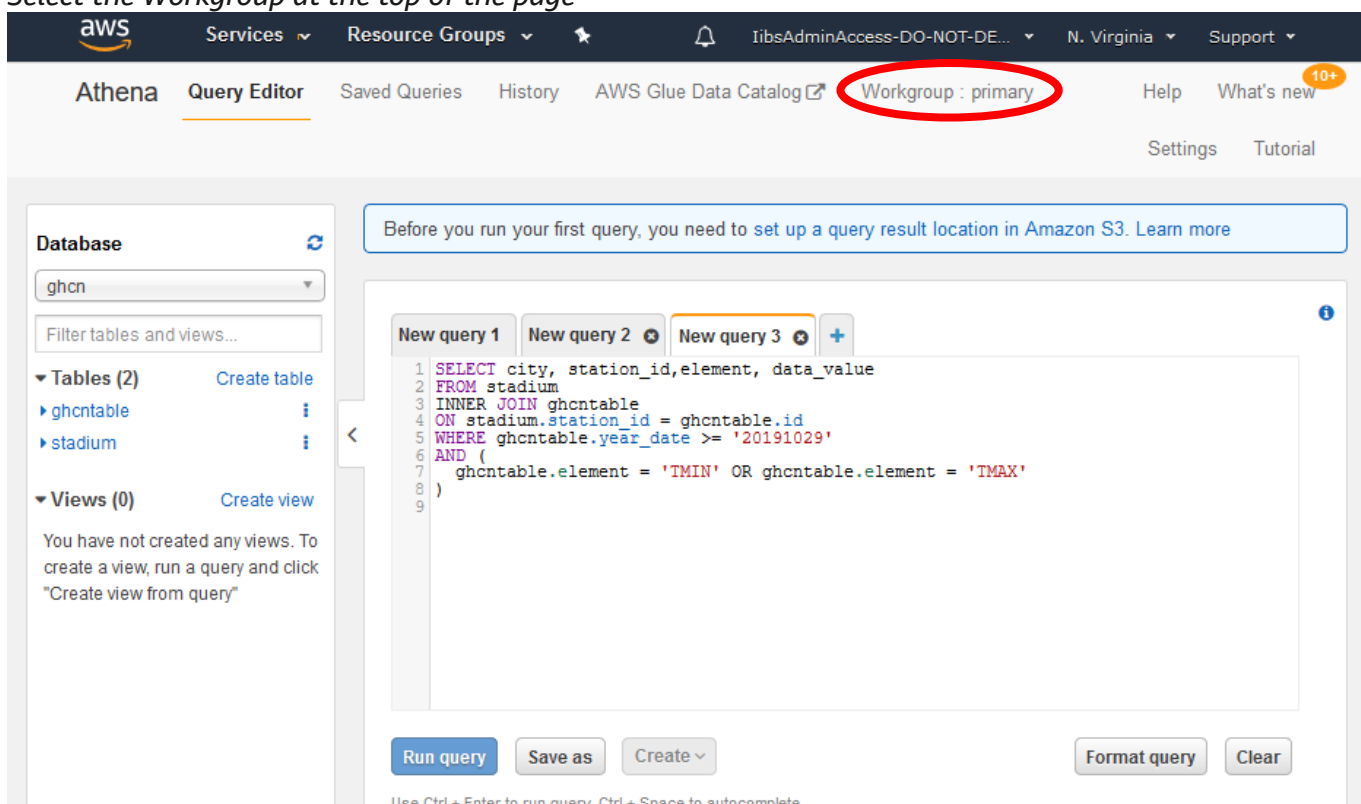


Section 3: Creating the Endpoint and Query Athena

In this section, you will create a RESTful endpoint to issue queries against your Athena database. Amazon API Gateway adds a layer between your application users and your app logic that enables the ability to throttle individual users or requests, protect against Distributed Denial of Service attacks, and provide a caching layer to cache response from your Lambda function. In this example, we will use Amazon API Gateway to trigger an AWS Lambda function, which will issue a query against the Athena database we created in the previous example.

- 1) In this example, Lambda will only be allowed to query the default workgroup that was created in the previous example. From the main console, search Athena and go into the Athena console. Select the Workgroup at the top of the page



The screenshot shows the AWS Athena console interface. At the top, the navigation bar includes 'Services', 'Resource Groups', and a dropdown menu showing 'Workgroup : primary', which is circled in red. Below the navigation bar, the 'Query Editor' is active. On the left, the 'Database' dropdown is set to 'ghcn', and a list of tables ('ghcntable', 'stadium') and views is shown. The main area contains a SQL query editor with the following query:

```
1 SELECT city, station_id, element, data_value
2 FROM stadium
3 INNER JOIN ghcntable
4 ON stadium.station_id = ghcntable.id
5 WHERE ghcntable.year_date >= '20191029'
6 AND (
7     ghcntable.element = 'TMIN' OR ghcntable.element = 'TMAX'
8 )
9
```

At the bottom of the query editor, there are buttons for 'Run query', 'Save as', 'Create', 'Format query', and 'Clear'. A note at the bottom states: 'Use Ctrl + Enter to run query. Ctrl + Space to autocomplete.'

- 2) Select the primary workgroup and click View Details
- 3) Copy the Workgroup ARN, by clicking on the copy icon. You will need this later to specify which workgroup you want to give Lambda access to. Copy this ARN to a text editor




Workgroup: primary

[Edit workgroup](#) [Delete workgroup](#) [Disable workgroup](#) [Enable workgroup](#)

[Overview](#) [Data usage controls](#) [Tags](#)

To grant access to the workgroup, [create an IAM policy](#) and attach it to a user, group or role. [Learn more](#)

Description	Not defined
Query result location	Not defined
CloudWatch metrics	Disabled
Encrypt query results	Not defined
Workgroup status	Enabled
Workgroup ARN	arn:aws:athena:us-east-1:[redacted]:primary 
Bytes scanned cut off per query	Not defined
Override client-side settings	Disabled
Queries with requester pays buckets	Disabled

- 4) You will need Permissions for your Lambda function. Regardless of what invokes a Lambda function, AWS Lambda executes the function by assuming the IAM role (execution role) that you specify at the time you create the Lambda function. Using the permissions policy associated with this role, you grant your Lambda function the permissions that it needs. In this example, the Lambda function needs to issue a query to Athena, you grant permissions for the relevant Amazon Athena actions in the permissions policy. For more information, see [AWS Lambda Execution Role](#). From the main console, search IAM and open the Identify and Access Management console and select Policies on the left hand side



Identity and Access Management (IAM)

▼ AWS Account

Dashboard

Groups

Users

Roles

Policies

Identity providers

Account settings

Credential report

Q Search IAM

▼ AWS Organizations

Organization activity

Welcome to Identity and Access Management

IAM users sign-in link:

[Redacted link]

| [Customize](#)

IAM Resources

Users: 1

Roles: 7

Groups: 1

Identity Providers: 0

Customer Managed Policies: 3

Security Status

5 out of 5 complete.

- ✓ Delete your root access keys
- ✓ Activate MFA on your root account
- ✓ Create individual IAM users
- ✓ Use groups to assign permissions
- ✓ Apply an IAM password policy

- 5) Click on **Create Policy**, then click on the **JSON** tab. Remove the template text in the JSON editor. You will find a file: `iam.json` that has the IAM policy document you will be using for this lab. Replace `YOUR_BUCKET_NAME` with the name of the S3 bucket you created earlier in this lab. Paste in the Athena ARN you copied earlier in this lab. Both of these areas are highlighted below.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::YOUR_BUCKET_NAME/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetBucketLocation",
        "s3:GetObject",
        "s3:ListBucket",
        "s3:ListBucketMultipartUploads",
```

```
        "s3:ListMultipartUploadParts",
        "s3:AbortMultipartUpload",
        "s3:PutObject"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "glue:GetTable"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "athena:StartQueryExecution",
        "athena:GetQueryResults",
        "athena:GetQueryResultsStream",
        "athena:GetQueryExecution",
        "athena:StopQueryExecution"
    ],
    "Resource": [
        "YOUR_ATHENA_ARN"
    ]
}
]
```

- 6) *Click Review policy*
- 7) *Give it the name: Lambda-Access-to-Athena-ghcn*
- 8) *Click Create Policy*
- 9) *From the IAM console, select Roles on the left hand side*



Identity and Access Management (IAM)

▼ AWS Account

Dashboard

Groups

Users

Roles

Policies

Identity providers

Account settings

Credential report

Q Search IAM

▼ AWS Organizations

Organization activity

Welcome to Identity and Access Management

IAM users sign-in link:

[Redacted]

| [Customize](#)

IAM Resources

Users: 1

Roles: 7

Groups: 1

Identity Providers: 0

Customer Managed Policies: 3

Security Status

5 out of 5 complete.


- ✓ Delete your root access keys
- ✓ Activate MFA on your root account
- ✓ Create individual IAM users
- ✓ Use groups to assign permissions
- ✓ Apply an IAM password policy


10) Click on *Create role* and select *AWS Service* under *type of trusted entity*, then click on *Lambda* and click *Next: Permissions*





Create role 1 2 3 4

Select type of trusted entity

 **AWS service**
EC2, Lambda and others

 **Another AWS account**
Belonging to you or 3rd party

 **Web identity**
Cognito or any OpenID provider

 **SAML 2.0 federation**
Your corporate directory

Allows AWS services to perform actions on your behalf. [Learn more](#)

Choose the service that will use this role

EC2
Allows EC2 instances to call AWS services on your behalf.

Lambda
Allows Lambda functions to call AWS services on your behalf.

API Gateway	CodeBuild	EMR	Lambda	S3
AWS Backup	CodeDeploy	ElastiCache	Lex	SMS
AWS Chatbot	Comprehend	Elastic Beanstalk	License Manager	SNS
AWS Support	Config	Elastic Container Service	Machine Learning	SWF
Amplify	Connect	Elastic Transcoder	Macie	SageMaker
AppStream 2.0	DMS	ElasticLoadBalancing	MediaConvert	Security Hub
AppSync	Data Lifecycle Manager	Forecast	Migration Hub	Service Catalog
Application Auto Scaling	Data Pipeline	Global Accelerator	OpsWorks	Step Functions
Application Discovery Service	DataSync	Glue	Personalize	Storage Gateway
Batch	DeepLens	Greengrass	QLDB	Textract
Chime	Directory Service	GuardDuty	RAM	Transfer
CloudFormation	DynamoDB	Inspector	RDS	Trusted Advisor
CloudHSM	EC2	IoT	Redshift	VPC
CloudTrail	EC2 - Fleet	IoT Things Graph	Rekognition	WorkLink
CloudWatch Application	EC2 Auto Scaling	KMS	RoboMaker	WorkMail
	EKS	Kinesis		

* Required

Cancel **Next: Permissions**

11) In the search bar, search for the policy you just created: *Lambda-Access-to-Athena-ghcn*

12) Check the box next to this policy and click **Next: Tags**

13) Add a tag of: *Key=project* and *value: AWS-CodeGreen-Hackathon*

14) Click **Next: Review**

15) In the role name box enter: *Lambda-ghcn* and click **Create role**

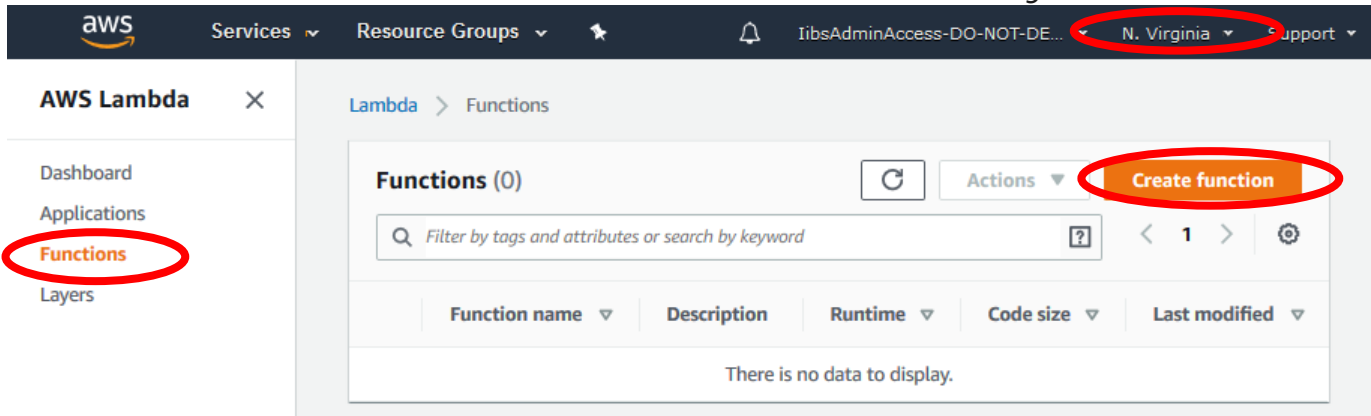
16) The second permissions required is for Amazon API Gateway to invoke your Lambda function.

Amazon API Gateway cannot invoke your Lambda function without your permission. You grant



this permission via the permission policy associated with the Lambda function when you create the API gateway.

- 17) Click on the AWS logo in the upper left hand corner to return to the main AWS console window
- 18) Search Lambda and select Lambda
- 19) Select the N. Virginia Region in the upper right hand corner
- 20) Choose Functions from the left hand side and click on Create function on the right hand side



- 21) Select Author from scratch
- 22) Under Basic Information, select a function name of query-ghcn
- 23) Under runtime, click on the down arrow to see what runtime environments are available. In this example, we will be using "Python 3.8" select Python 3.8
- 24) Click on the triangle next to Choose or create an execution role, select Use an existing role. In the existing role dropdown, you should see the role we created earlier in this session: Lambda-ghcn and select that. Click Create Function
- 25) In the function code section, delete the sample code and replace it with the following. Remember, Python is spacing sensitive, please use the Lambda-code.py and a text editor to copy in the lambda code. It has been included here for information purposes and to highlight the sections that need to be edited to the names you used in this lab. (BE SURE TO EDIT THE S3 QUERY AND S3 BUCKET NAMES WITH THE NAMES YOU CREATED IN THIS LAB HIGHLIGHTED IN YELLOW):

```
import time
import boto3
import json
import collections
import operator

# athena constant
DATABASE = 'ghcn'

# S3 constant
S3_QUERY='query-result'
S3_BUCKET ='YOUR_BUCKET_HERE'
# set defaults
```



```
S3_OUTPUT = 's3://' + S3_BUCKET + '/' + S3_QUERY
DEFAULT_CITIES = "best" # choices are 'list' (returns all cities) or
                        'best' (returns city with closest temp to target temp)
DEFAULT_TARGET = 230 # Any int that is represented in tenths of celcius

# number of retries
RETRY_COUNT = 15

def lambda_handler(event, context):

    try:
        city = event['queryStringParameters']['cities']

        if ((city != "list") and (city != "best")):
            city = DEFAULT_CITIES

    except:
        city = DEFAULT_CITIES

    try:
        target = int(event['queryStringParameters']['target'])
    except:
        target = DEFAULT_TARGET

    # query has hardcoded elements for simplicity of this workshop
    query = """SELECT city, avg(CAST(data_value as INTEGER)) as temp
FROM stadium
    INNER JOIN ghcntable ON stadium.station_id = ghcntable.id
    WHERE ghcntable.year_date >= '20191029'
    AND ghcntable.element = 'TAVG'
    GROUP BY city"""

    # athena client
    client = boto3.client('athena')

    # Execution
    response = client.start_query_execution(
        QueryString=query,
        QueryExecutionContext={
            'Database': DATABASE
        },
        ResultConfiguration={
            'OutputLocation': S3_OUTPUT,
        }
    )

    # get query execution id
    query_execution_id = response['QueryExecutionId']
    print(query_execution_id)
```




```
# get execution status
for i in range(1, 1 + RETRY_COUNT):

    # get query execution
    query_status =
client.get_query_execution(QueryExecutionId=query_execution_id)
    query_execution_status =
query_status['QueryExecution']['Status']['State']

    if query_execution_status == 'SUCCEEDED':
        print("STATUS:" + query_execution_status)
        break

    if query_execution_status == 'FAILED':
        raise Exception("STATUS:" + query_execution_status)
    else:
        print("STATUS:" + query_execution_status)
        time.sleep(i)
else:

client.stop_query_execution(QueryExecutionId=query_execution_id)
    raise Exception('TIME OVER')

# get query results
result =
client.get_query_results(QueryExecutionId=query_execution_id)

# Convert the result set into something a bit easier to manage
i=1
stations= {}

num_cities = len(result['ResultSet']['Rows'])

while i < num_cities:
    # Pull out the station city and station avg temp from the json
    returned from query
    station_city =
result['ResultSet']['Rows'][i]['Data'][0]['VarCharValue']
    station_temp =
int(float(result['ResultSet']['Rows'][i]['Data'][1]['VarCharValue']))

    # the delta from target shows how far (in tenths of a degree)
    we are from the target temp
    delta_from_target = abs(station_temp - target)

    # save it in a new dict. Station[<City Name>] = [ degree delta
    from target, avg temp of city]
    stations[station_city] = [ delta_from_target, station_temp ]
```

```
i = i+1

sorted_stations = sorted(stations.items(),
key=operator.itemgetter(1))
stations_dict = collections.OrderedDict(sorted_stations)

best_city = list(stations_dict)[0]

if (city == "list"):
    return {
        'statusCode': 200,
        'headers': { 'Content-Type': 'application/json' },
        'body': json.dumps(stations_dict)
    }
elif (city == "best"):
    return_val = { }
    return_val[best_city] = stations[best_city]
    return {
        'statusCode': 200,
        'headers': { 'Content-Type': 'application/json' },
        'body': json.dumps(return_val)
    }
else:
    return {
        'statusCode': 200,
        'headers': { 'Content-Type': 'application/json' },
        'body': json.dumps(stations_dict)
    }
```

26) Scroll down to Basic Settings and change the timeout 29 seconds (the query should complete in about 15 seconds). This will set the Lambda timeout to the same for API GW

27) Click Save

28) Now you can test your Lambda function and make sure it can connect to Athena and read the results from S3. When you call this endpoint from a web browser, you can pass in parameters in the URL string such as:

[https://\[redacted\]/test/query?cities=best&target=100](https://[redacted]/test/query?cities=best&target=100)

These parameters are passed to the handler in the event variable. You can simulate this by creating a test event. Start by clicking on the Test button. In the Event name box, enter: testList. You will find a file: test-lambda.json in the git repo. This file contains the syntax for your test event. Delete the template data and paste this data into the Configure test event

```
{
  "queryStringParameters": {
    "cities": "list",
    "days": "10",
    "target": "180"
```



```
}  
}
```

29) Click Create

30) Now, click the Test button. You will see the button grey out while the function is running.

Remember, it takes about 15 seconds to query Athena. If successful, in the Execution Results window, you will see something like:

Response:

```
{  
  "statusCode": 200,  
  "headers": {  
    "Content-Type": "application/json"  
  },  
  "body": "{\"Mexico City\": [2, 178], \"San Diego\": [3, 177],  
  \"Sydney\": [20, 200], \"Shanghai\": [21, 159], \"Rio de Janeiro\":  
  [23, 203], \"Tokyo\": [35, 145], \"Kiev\": [56, 124], \"Seoul\": [69,  
  111], \"Berlin\": [101, 79], \"Boston\": [109, 71], \"Ireland\": [120,  
  60], \"Toronto\": [160, 20]}\""  
}
```

31) Navigate back to the main console by clicking on the AWS logo in the upper left

32) Search API and select API Gateway

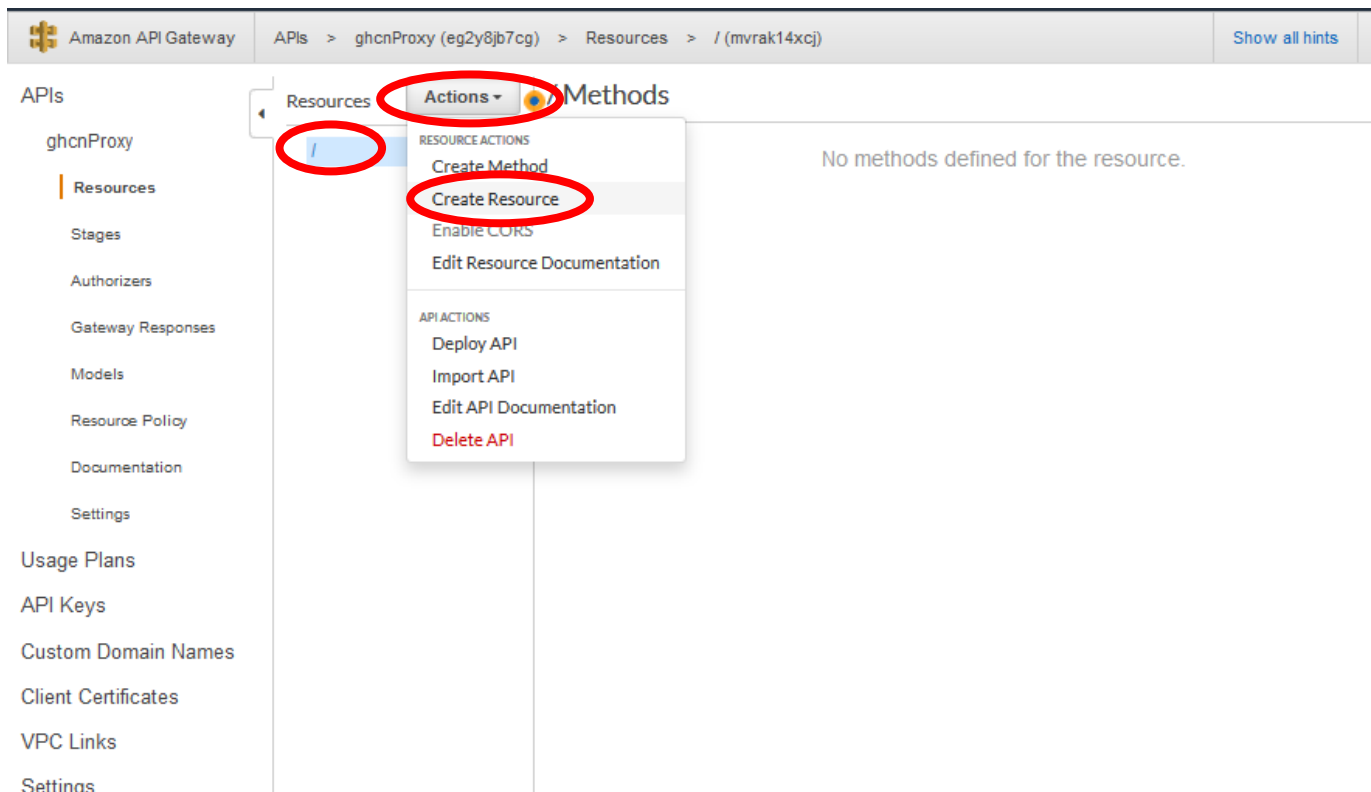
33) If the API Gateway screen gives you a "getting started" page, click Get Started or Click Create API

34) Create an empty API by:

- a. Choosing protocol as REST
- b. Under Create new API select NEW API
- c. Under Settings give it the API Name of ghcnProxy
- d. Leave the Endpoint Type as Regional
- e. Click Create API

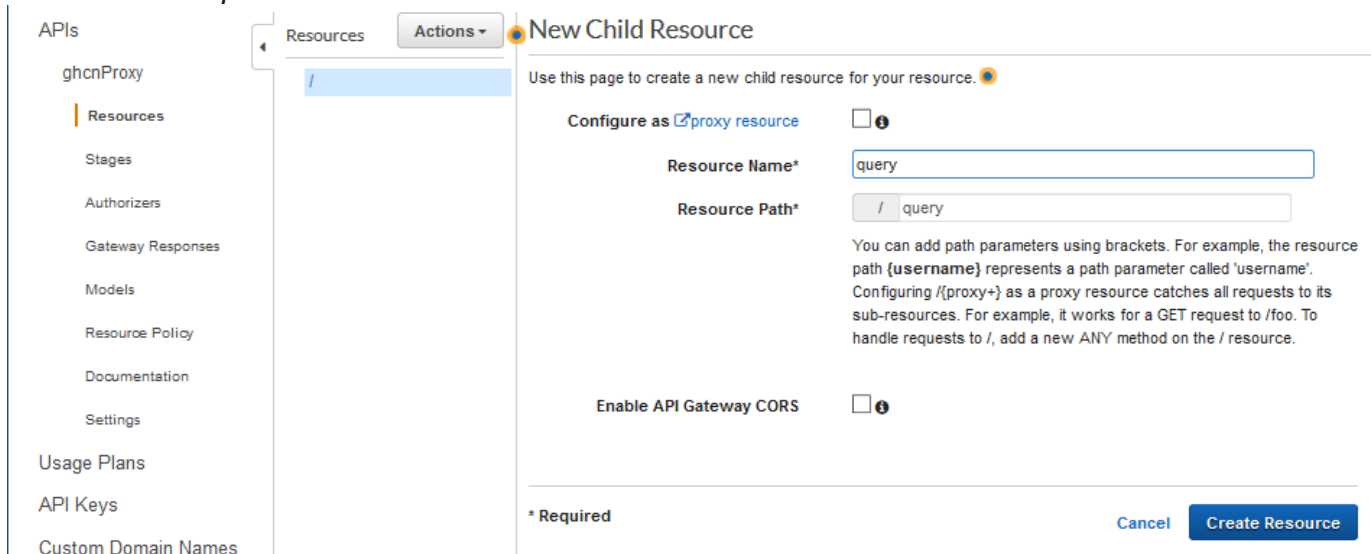
35) Choose the root resource (/) in the Resources tree

36) From the Actions button, choose Create Resource



37) For Resource Name, give it a name of: query

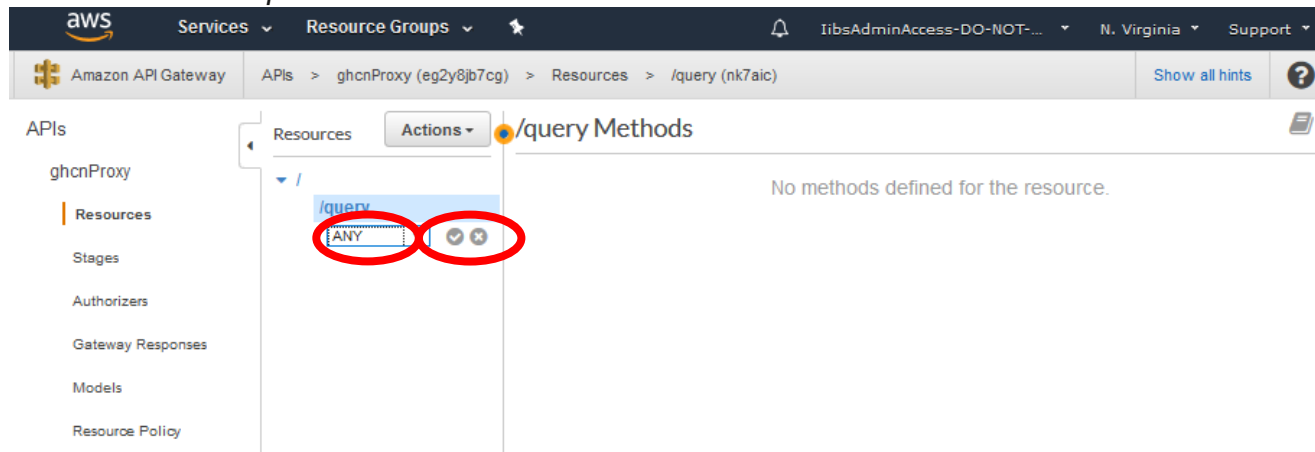
38) Leave all other options at the default and click Create Resource



39) In a proxy integration, the entire request is sent to the backend Lambda function as-is, via a catch-all ANY method that represents any HTTP method. The actual HTTP method is specified by the client at run time. The ANY method allows you to use a single API method setup for all of the

supported HTTP methods: DELETE, GET, HEAD, OPTIONS, PATCH, POST, and PUT. To setup an ANY method, do the following:

- In the Resources list, choose /query
- In the Actions dropdown, choose Create method
- You will see a black drop down under the /query Resource, choose ANY from the dropdown menu and choose the checkmark icon that appears next to it. This will then show you a new window to setup the ANY method



- Leave the Integration Type to Lambda Function
 - Check Lambda Proxy integration
 - Select the US-east-1 region
 - In the Lambda Function, start typing the name of the function (query-ghcn) we created earlier and the function should pop-up as a selection.
 - Click Save
 - Select OK when prompted with Add Permission to Lambda Function
- 40) Choose Deploy API in from the Actions dropdown
- In the Deployment Stage select [New Stage]
 - For Stage name enter test
 - Click Deploy
- 41) Test the API using curl (replacing https:// with the Invoke URL listed on your screen). Don't forget to add the /query to the end of the URL. Remember, it takes about 17 seconds for the Athena query to run. On Linux/Mac a single quote (') may be required instead of the double quote (") listed below
- ```
curl -v -X POST "https://[redacted].amazonaws.com/test/query"
```
- 42) Save this endpoint URL to your text editor as you will need it for the next section (including the /query)
- 43) This API Endpoint can pass parameters to the Lambda function. You can specify the type of output you want (List cities or Select the "best" city) and you can also specify what temperature you consider "ideal". To use these features, at the end of your URL you can specify, cities=best or cities=list. The default is "best" For example,
- [https://\[redacted\].amazonaws.com/test/query?cities=best](https://[redacted].amazonaws.com/test/query?cities=best)

To change the default ideal temperature, specify target=<number>. For example,



---

[https://\[REDACTED\]/test/query?cities=best&target=100](https://[REDACTED]/test/query?cities=best&target=100)

*Like the temperature in the dataset, this is measured in tenths of Celsius. So, in the example above, 100 represents 10.0 degrees Celsius. The default is 230.*

*The final input variable is the number of days back to query to get the average temperature. The default is 14 days. To change the default behavior, you can specify `days=<number>` to query the database that many days in the past. Because not all stations are updated immediately, the minimum number of days you can go back is 5, anything below that value will result in a query of `days=5`*

[https://\[REDACTED\]/test/query?cities=best&target=100&days=30](https://[REDACTED]/test/query?cities=best&target=100&days=30)