

Section 3

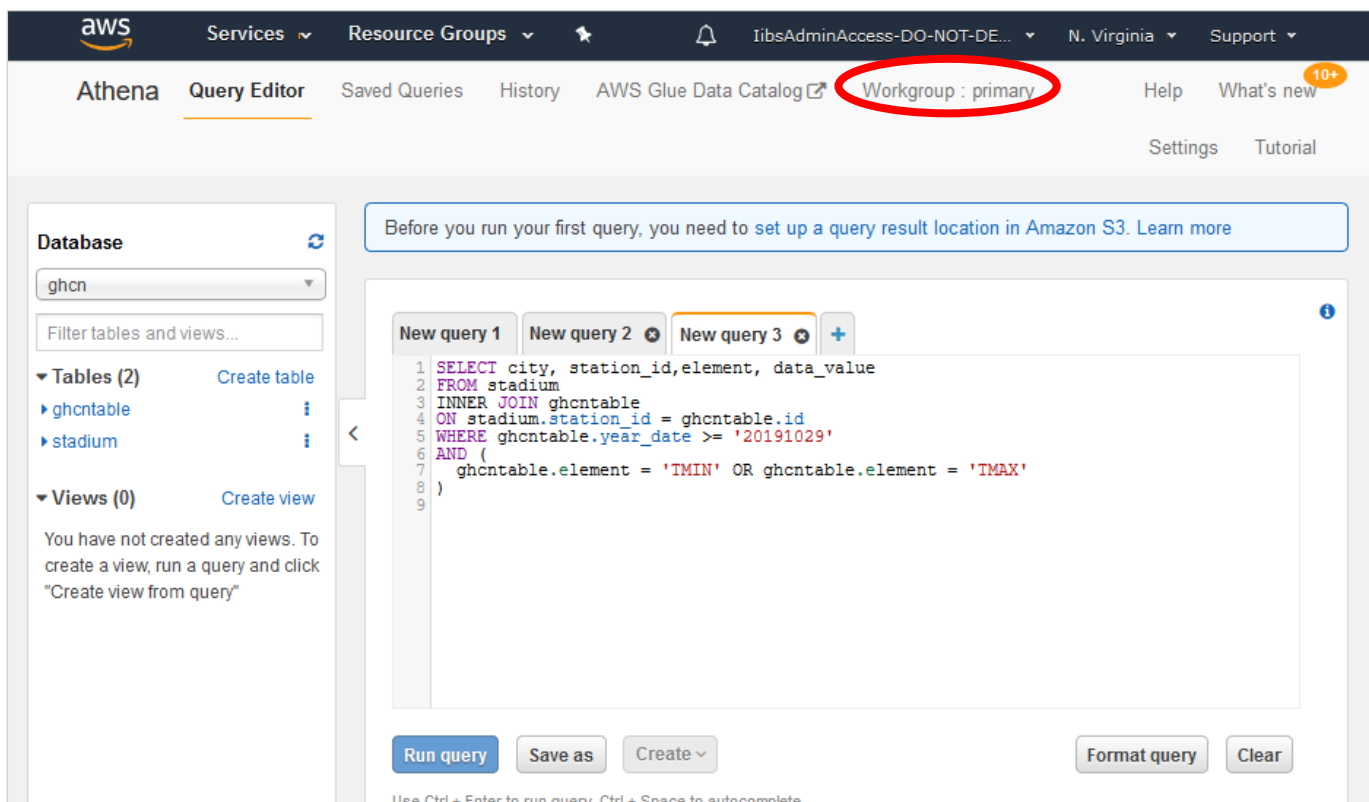
Creating the endpoint and querying Athena

In this section you will create a RESTful endpoint to call your Athena database. You'll do this with [Amazon API Gateway](#), an AWS service for building, yes, APIs. API Gateway adds a layer between your application users and your app logic. It allows you to do things like throttle users, protect against Distributed Denial of Service attacks, and cache responses.

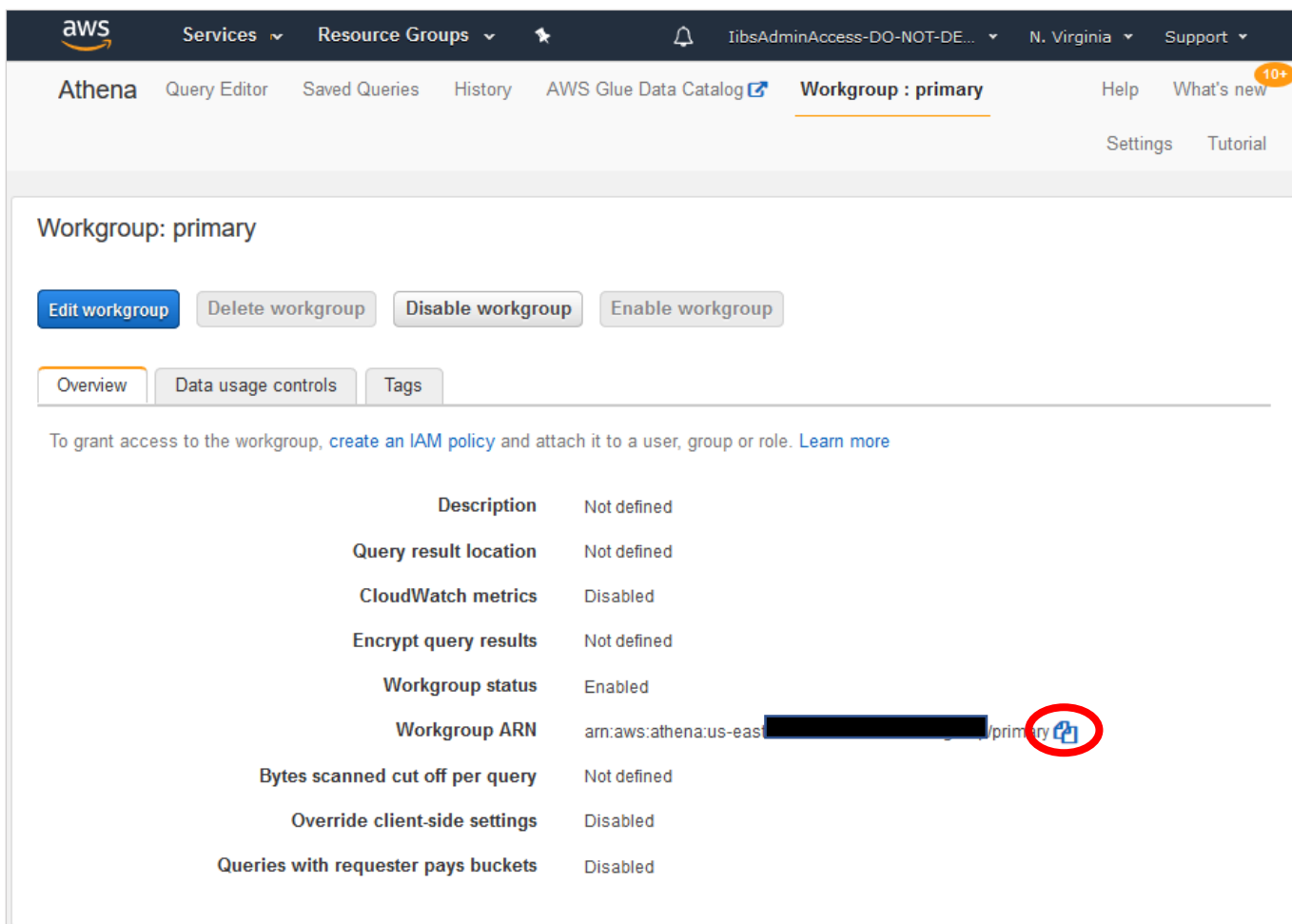
Our endpoint will be powered by [AWS Lambda](#). Lambda lets you upload code to AWS and configure it to be run in response to any number of events, including calls to API Gateway endpoints. AWS itself manages the servers on which your Lambdas run, all you do is upload the code and connect it to the events triggering it.

Our Lambda will issue a query against the Athena database created in the previous example, and return the results via our API Gateway.


- 1) We only want our Lambda to be able to query the default workgroup that was created in the previous example. To do this, from the main console search Athena and go into the Athena console. Select the Workgroup at the top of the page:



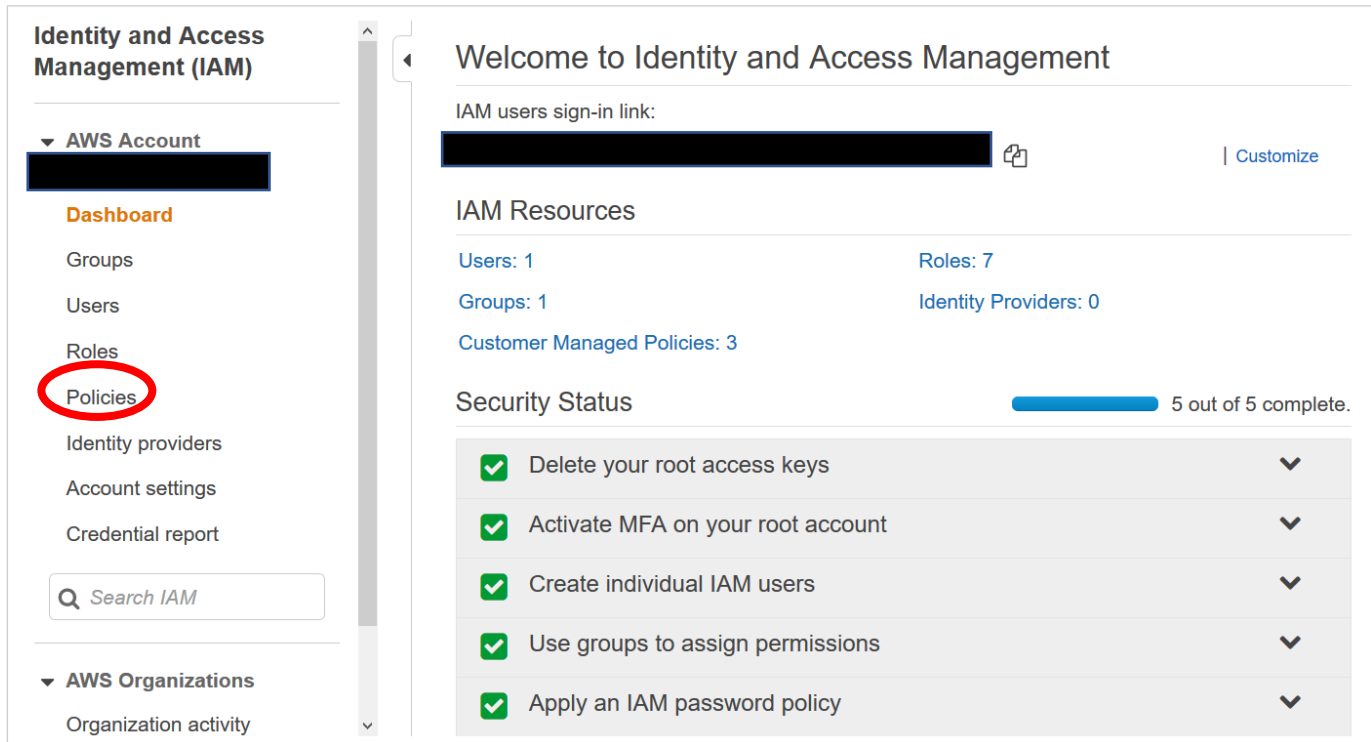
- 2) Select the primary workgroup and click "View details".
- 3) Copy the Workgroup ARN by clicking on the copy icon. You will need this later to specify which workgroup you want to give Lambda access to. Copy this ARN to a text editor.



The screenshot shows the AWS Athena console interface. At the top, there's a navigation bar with the AWS logo, 'Services', 'Resource Groups', and a search bar. Below this, the 'Athena' section is active, showing options like 'Query Editor', 'Saved Queries', 'History', 'AWS Glue Data Catalog', and 'Workgroup : primary'. The 'Workgroup : primary' page has tabs for 'Overview', 'Data usage controls', and 'Tags'. Under 'Overview', there are buttons for 'Edit workgroup', 'Delete workgroup', 'Disable workgroup', and 'Enable workgroup'. A message states: 'To grant access to the workgroup, create an IAM policy and attach it to a user, group or role. [Learn more](#)'. Below this is a table of settings:

Setting	Value
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-1:123456789012:workgroup/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 (ie, 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, so 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, then select "Policies" on the left-hand side.



- 5) Click on "Create policy", then click on the JSON tab. Remove the template text in the JSON editor. The code directory contains 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 in Section 1, then paste in the Athena workgroup ARN you set aside above in step 3. 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.

The screenshot shows the AWS Identity and Access Management (IAM) console. On the left sidebar, under the 'AWS Account' section, the 'Roles' menu item is highlighted with a red circle. The main content area displays a 'Welcome to Identity and Access Management' message, followed by 'IAM Resources' statistics (Users: 1, Roles: 7, Groups: 1, Identity Providers: 0, Customer Managed Policies: 3) and a 'Security Status' section with five checklist items, all of which are marked as complete with green checkmarks.

Identity and Access Management (IAM)

- ▼ AWS Account
 - Dashboard
 - Groups
 - Users
 - Roles**
 - Policies
 - Identity providers
 - Account settings
 - Credential report
 - 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. Click "Lambda", and then click "Next: Permissions".

Create role

1 2 3 4

Select type of trusted entity

AWS service
EC2, Lambda and others

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

Another AWS account
Belonging to you or 3rd party

Web identity
Cognito or any OpenID provider

SAML 2.0 federation
Your corporate directory

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.

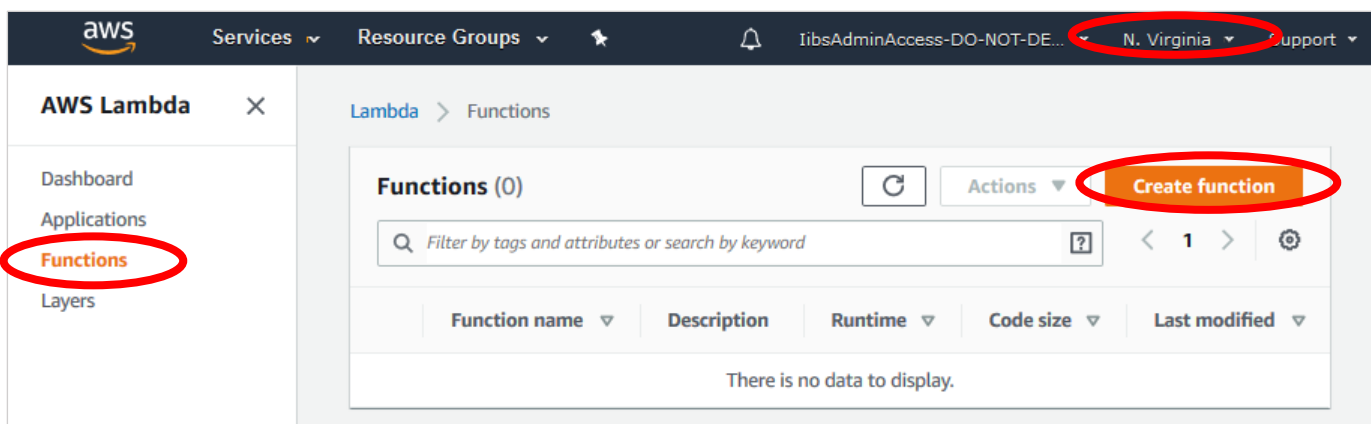
API Gateway	CodeBuild	EMR	Lambda	S3
AWS Backup	CodeDeploy	ElastiCache	Lex	SMS
AWS Chatbot	Comprehend	Elastic Beanstalk	License Manager	SNS
AWS Support	Config	Elastic Container Service	Machine Learning	SWF
Amplify	Connect	Elastic Transcoder	Macie	SageMaker
AppStream 2.0	DMS	ElasticLoadBalancing	MediaConvert	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 with the key "project" and value "AWS-CodeGreen-Hackathon", then click "Next".

- 14) Click "Next: Review".
- 15) In the role name box enter "Lambda-ghcn" and click "Create role".
- 16) The second permission required is for API Gateway to invoke your Lambda function. 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 then select it.
- 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" enter the function name "query-ghcn".
- 23) Under "Runtime" click on the down arrow to see what runtime environments are available. In this example we'll be using "Python 3.8", so select that.
- 24) Click on the triangle next to "Choose or create an execution role", then select "Use an existing role". In the existing role dropdown, you should see the role we created earlier in this session, "Lambda-ghcn". Select that and click "Create function".
- 25) In the function code section, delete the sample code and replace it with the following. Remember, Python is white space sensitive, so please open the [lambda-code.py file](#) in a text editor, copy the contents, and paste it into the function code section. The code is included below in this section for information purposes and to highlight the sections that need to be edited to reflect the names you used in this lab. *Be sure to edit the S3 query and bucket names in the Lambda with the names you created earlier in the lab!* Below, these areas are highlighted in yellow.



```
import time
import boto3
import json
import collections
import operator
import datetime
import os

# athena database name
athenaDatabase = 'ghcn'

# S3 constant
S3_QUERY='query-result'
S3_BUCKET = 'YOUR_BUCKET_HERE'

# set defaults
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
DEFAULT_DATE_HISTORY = 14 # defaults to 14 days from current day in SQL query
DEFAULT_MIN_LOOKBACK = 5
# number of retries
RETRY_COUNT = 15

## override defaults with Environment variables if available
if 'GLUE_DATABASE' in os.environ:
    athenaDatabase = os.environ['GLUE_DATABASE']

if 'S3_QUERY_OUTPUT_LOCATION' in os.environ:
    S3_OUTPUT = os.environ['S3_QUERY_OUTPUT_LOCATION']
else:
    S3_OUTPUT = 's3://' + S3_BUCKET + '/' + S3_QUERY

if 'GHCN_TABLE_NAME' in os.environ:
    GHCN_TABLE_NAME = os.environ['GHCN_TABLE_NAME']
else:
    GHCN_TABLE_NAME = 'ghcntable'

if 'STADIUM_TABLE_NAME' in os.environ:
    STADIUM_TABLE_NAME = os.environ['STADIUM_TABLE_NAME']
else:
    STADIUM_TABLE_NAME = 'stadium'

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

    try:
        lookbackDays = int(event['queryStringParameters']['days'])
    except:
        lookbackDays = DEFAULT_DATE_HISTORY

    if (lookbackDays < DEFAULT_MIN_LOOKBACK):
        lookbackDays = DEFAULT_MIN_LOOKBACK

    dateObj = datetime.date.today() - datetime.timedelta(days=lookbackDays)
    queryDate = int(dateObj.strftime('%Y%m%d'))

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

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

    # Execution
    response = client.start_query_execution(
        QueryString=query,
        QueryExecutionContext={
            'Database': athenaDatabase
        },
        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', 'Access-Control-
Allow-Origin': '*' },
            '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', 'Access-Control-
```



```
Allow-Origin': '*' },
    'body': json.dumps(return_val)
}
else:
    return {
        'statusCode': 200,
        'headers': { 'Content-Type': 'application/json', 'Access-Control-
Allow-Origin': '*' },
        'body': json.dumps(stations_dict)
    }
```

26) Scroll down to “Basic Settings” and change the timeout to 29 seconds (the query should complete in about 15 seconds*). This will set the Lambda timeout to the same value as the API Gateway timeout.

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”.

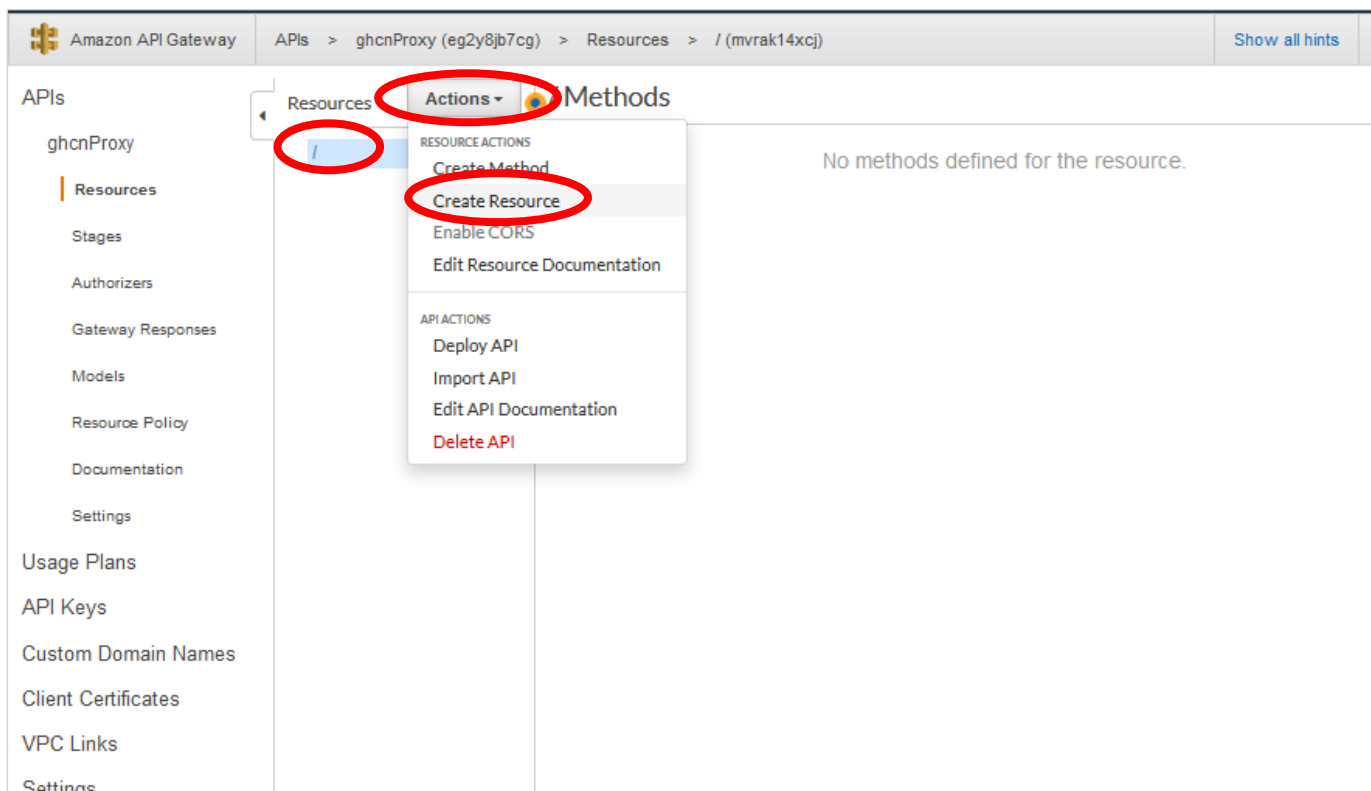
30) Click the “Test” button. You will see the button grey out while the function is running. Remember, it takes about 15 seconds for this Athena query. If successful, in the Execution Results window, you will see something like:

Response:

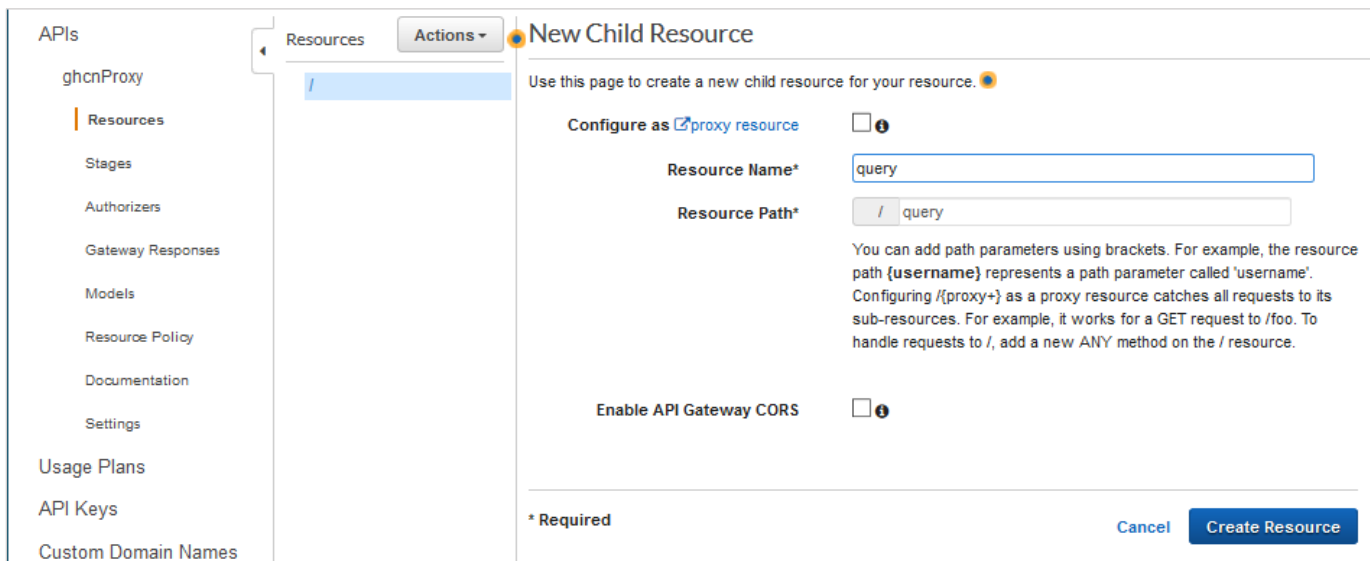
```
{
  "statusCode": 200,
  "headers": {
    "Content-Type": "application/json"
  },
  "body": "{\"Mexico City\": [2, 178], \"San Diego\": [3, 177], \"Sydney\": [20, 200], \"Shanghai\": [21, 159], \"Rio de Janeiro\": [23, 203], \"Tokyo\": [35, 145], \"Kiev\": [56, 124], \"Seoul\": [69, 111], \"Berlin\": [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) Enter "API" in the search box and then select API Gateway.
- 33) If the API Gateway screen gives you a "Getting started" page, click "Get Started", otherwise click "Create API".
- 34) Create an empty API by:
 - a. Choosing REST as the protocol.
 - b. Under "Create new API" select "New API".
 - c. Under Settings give it the API Name "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":

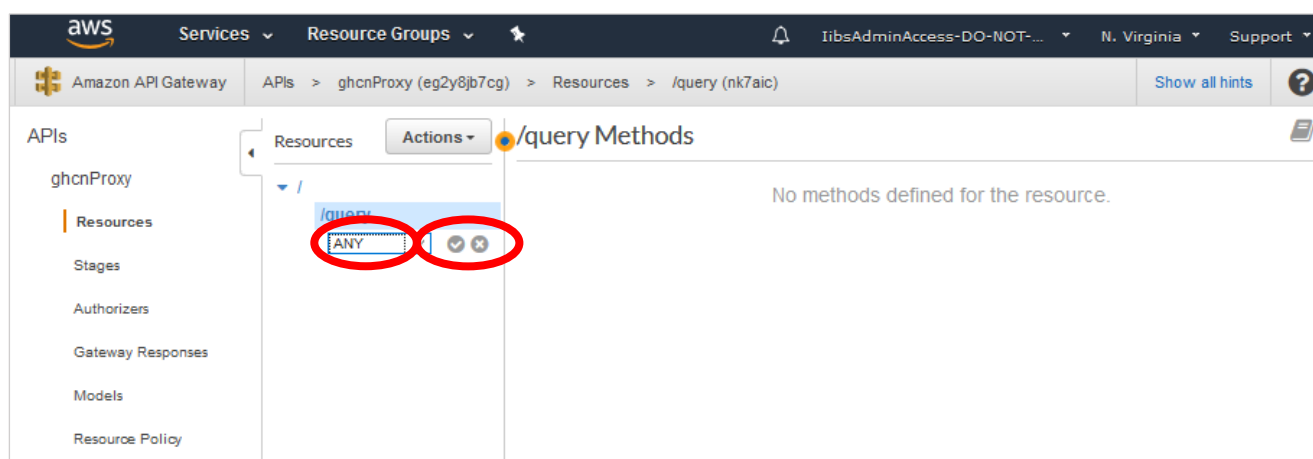


- 37) In the "Resource Name" field enter "query".
- 38) Use the default settings for all other options, and click "Create Resource".



39) In an API Gateway 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 create a single API method and use it 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 then click 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 set to "Lambda Function".
- Check "Use Lambda Proxy integration".
- Select the "us-east-1" as the Lambda region.

- g. 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.
 - h. Click "Save".
 - i. Select "OK" when prompted with Add Permission to Lambda Function
- 40) Choose "Deploy API" in from the Actions dropdown.
- a. For "Deployment stage" select "[New Stage]".
 - b. For "Stage name" enter "test".
 - c. 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 15 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
```

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

Like the temperature in the dataset, this is measured in tenths of a degree Celsius. So, in the example above, 100 represents 10.0 degrees Celsius. The default is 230, or 23 degrees Celsius.

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; anything below that value will result in a query of days=5

[https://\[redacted\].amazonaws.com/test/query?cities=best&target=100&days=30](https://[redacted].amazonaws.com/test/query?cities=best&target=100&days=30)



In this section you created an API Gateway with an ANY method, and created a Lambda to be triggered by that method. That Lambda queries the Athena table you created in Section 2 and returns the results to the user.

You've now completed section 3 of the workshop and can move on to the next section, "[Create S3 bucket and subdirectories as a webserver.](#)"

*** A note on performance**

This workshop is designed to illustrate how to query ASDI data directly using Athena. Due to the size and format of the dataset being used, queries take around 15 seconds, which might not meet requirements for a real user-facing application. Athena is a powerful tool for ad hoc data analysis, but not always the right choice in a runtime environment.

In a production application, you might evaluate techniques like converting data into a more efficiently queried format like Apache Parquet or ORC, exporting a subset of the data to reduce the volume being scanned by the query, or perhaps most likely, you might export the data you need from S3 into a more performant data store such as DynamoDB. Those steps are beyond the scope of this workshop, but ones you might want to consider in the real world.