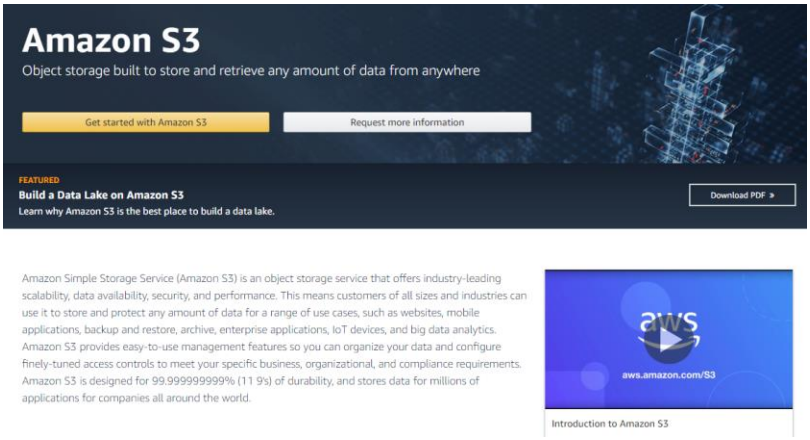
Lab: AWS Lambda, S3 and API Gateway

1. Amazon Simple Storage Service (Amazon S3)

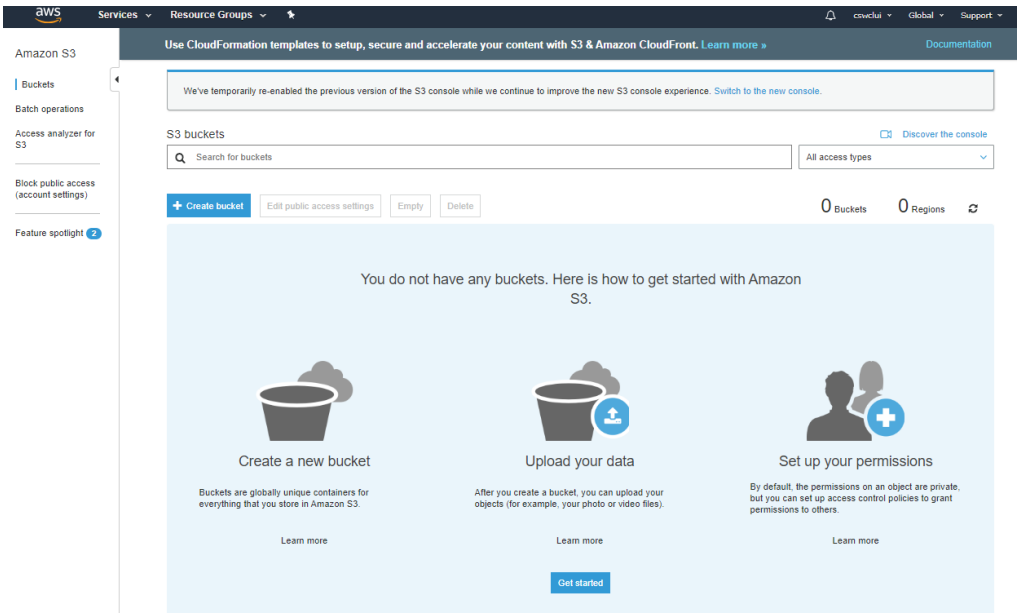
1.1. Overview of AWS S3

Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance.

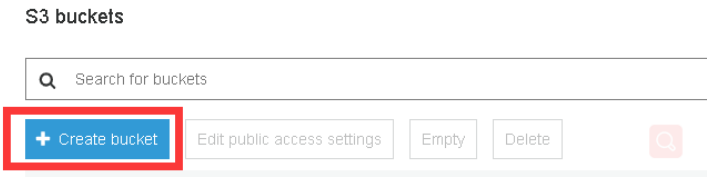
Visit https://aws.amazon.com/s3.



Navigate to S3 console at s3.console.aws.amazon.com.

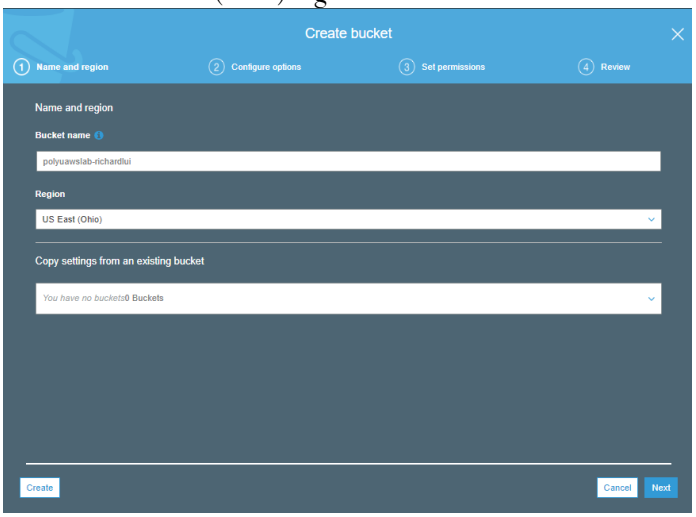


Click the “Create bucket” button.

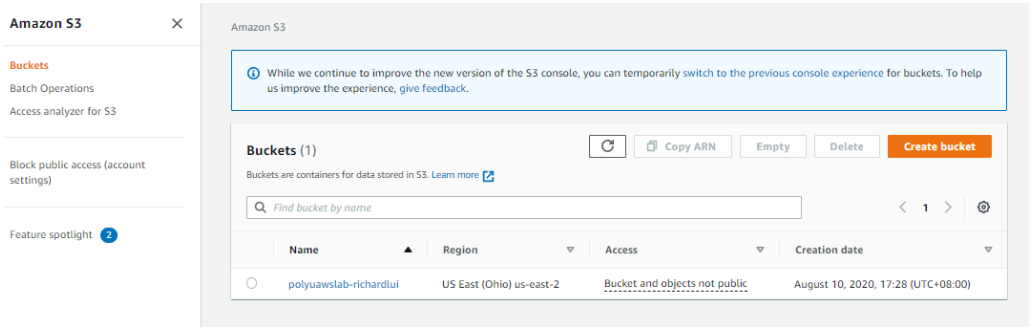


Create a bucket polyuawslab-[your name] (replace [your name] with your name). Note that the name should not contain uppercase characters and you should assign a unique bucket name (over all AWS S3 bucket names)).

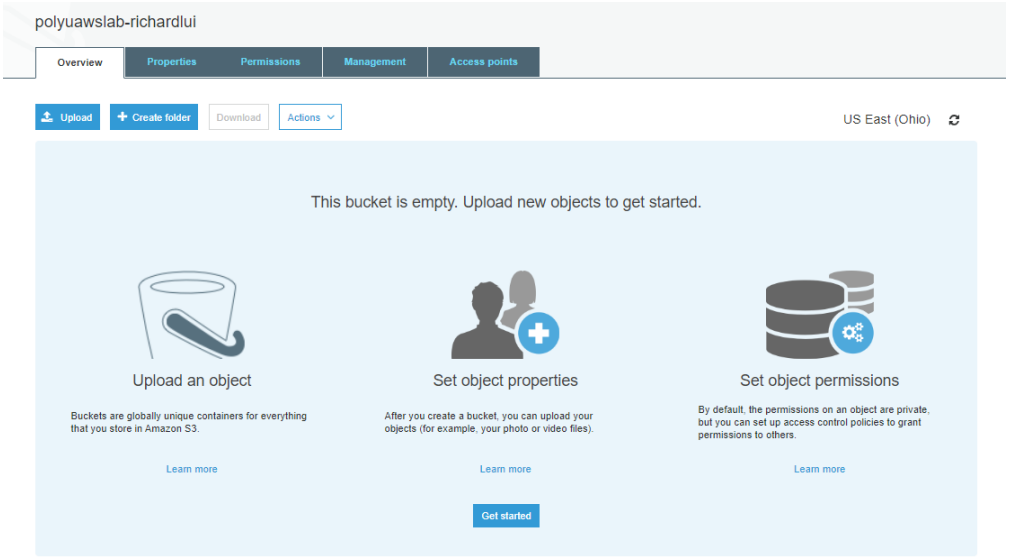
Choose the US East (Ohio) region.



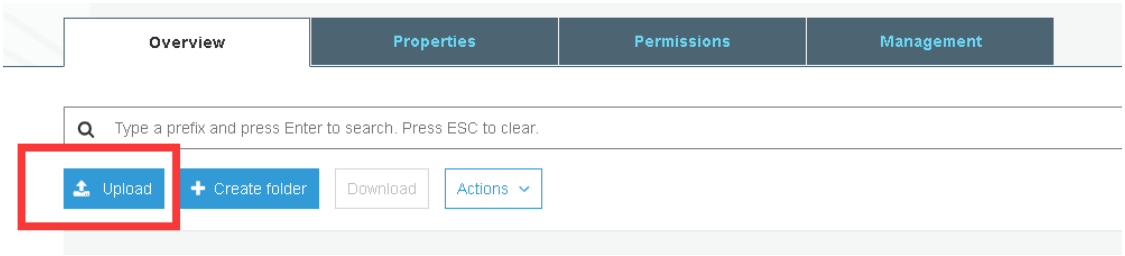
Click **next** a number of times until the bucket is created.

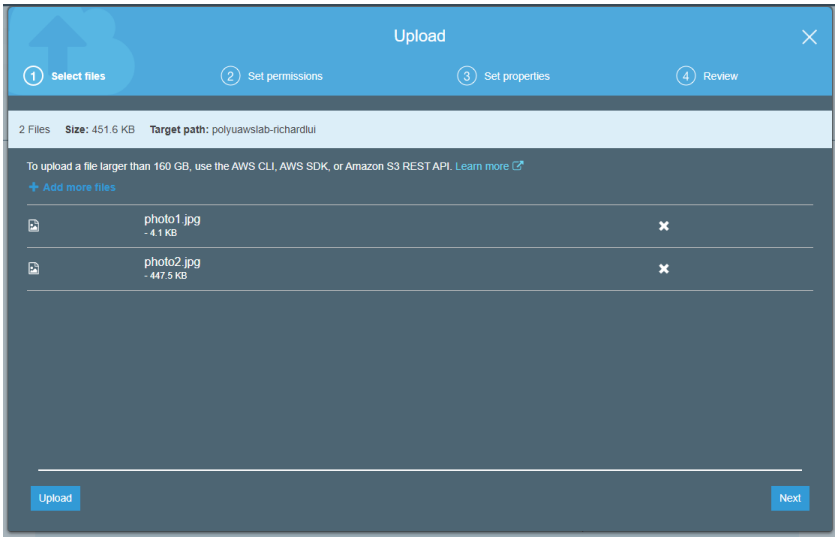


Click the created bucket.

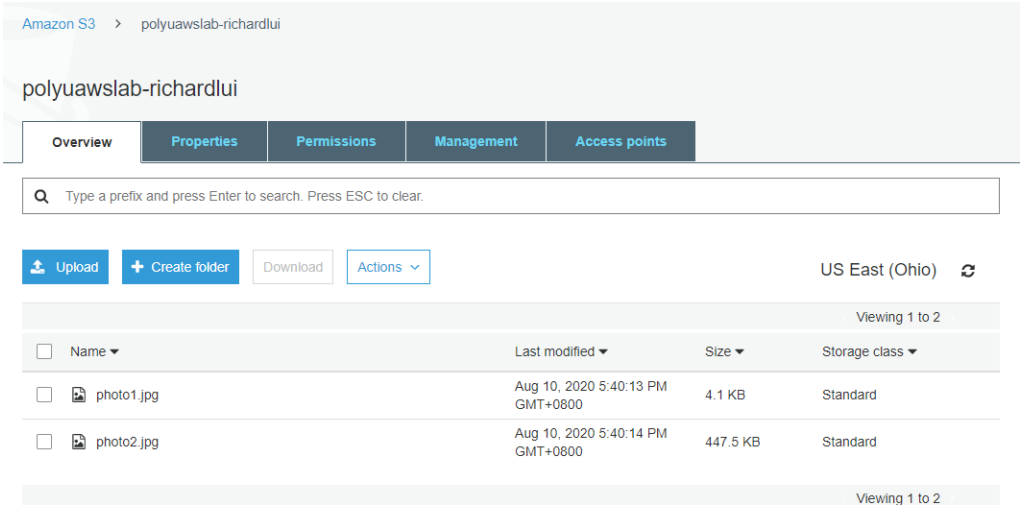


Add some files into your S3 bucket. Click **Next** a number of times until the files are uploaded.





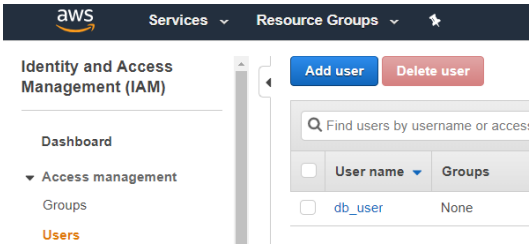
Verify that the files are uploaded.



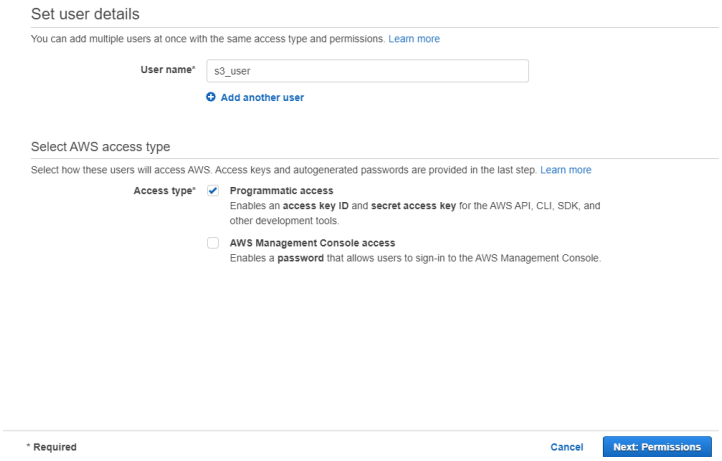
1.2. Create the IAM user with the access permission to S3

Remark: You can only create IAM user if you are using regular AWS accounts. Use the default access key ID and secret key if you are using AWS starter account.

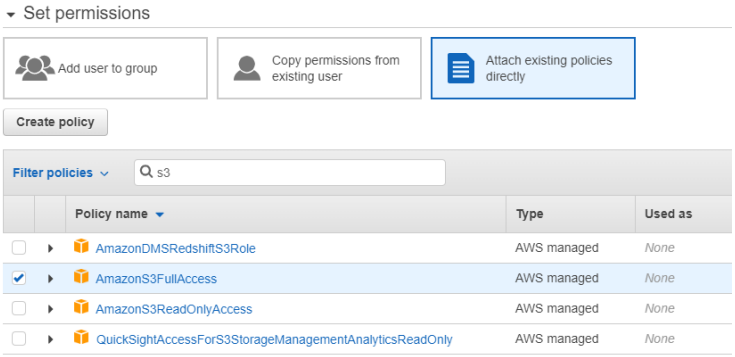
Navigate to **AWS Identity and Access Management (IAM)**. Navigate to the **Users** page.



Click **Add user** to create an IAM user **s3_user**. For access type, choose **Programmatic access**. Click **Next**.



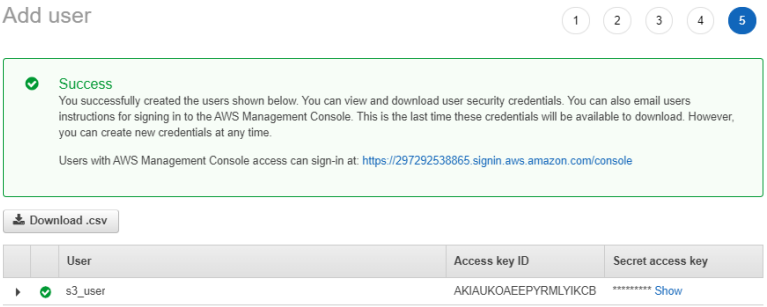
Select **Attach existing policies directly**. Check the box for **AmazonS3FullAccess**.



You may expand the AmazonS3FullAccess to view the policy (JSON tab).

Click **Next** two times. Review the user details and click **Create users**.

Click **Download.csv** to save the Access key ID and the secret access key.



1.3. Accessing S3 with the AWS Command Line Interface

Download and install AWS CLI if you have not done so before.
https://aws.amazon.com/cli.

Open a command prompt, enter:
aws configure

Provide your access Key ID and secret access key. Choose **us-east-2 (**choose **us-east-1** if you are using AWS starter account**)** as the default region and json as the default output.

Try the following commands

- aws s3 ls
- aws s3 ls s3://[your bucket name]

Refer to the following page for command reference for s3.
https://docs.aws.amazon.com/cli/latest/reference/s3/ls.html

1.4. Accessing S3 in python

Try the following code which lists the bucket names in s3.

```
import boto3
from pprint import pprint

s3 = boto3.client('s3',
     aws_access_key_id = '[Your access key ID]',
     aws_secret_access_key='[Your secret key]',
     region_name='us-east-2'
     )

def list_buckets():
  # Output the bucket names
  response = s3.list_buckets()
  print('Existing buckets:')

  for bucket in response['Buckets']:
   print(f'  {bucket["Name"]}')

list_buckets()
```

Refer to the following page for more examples:
https://boto3.amazonaws.com/v1/documentation/api/latest/guide/s3-example-creating-buckets.html

2. AWS Lambda

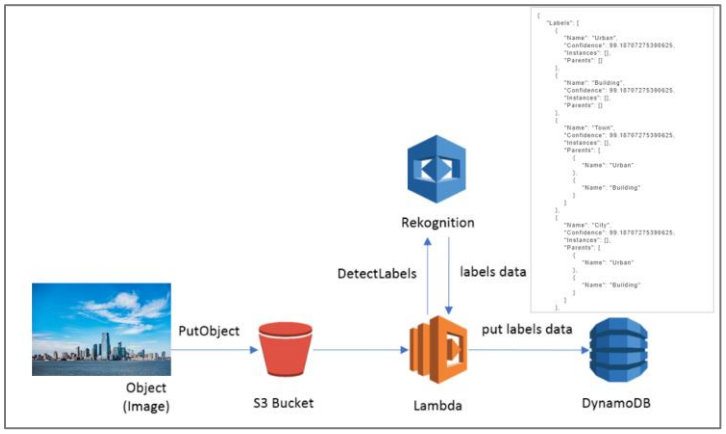
2.1. Overview

AWS Lambda lets you run code without provisioning or managing servers. You pay only for the compute time you consume - there is no charge when your code is not running. With Lambda, you can run code for virtually any type of application or backend service - all with zero administration. Just upload your code and Lambda takes care of everything required to run and scale your code with high availability. You can set up your code to automatically be triggered from other AWS services or call it directly from any web or mobile app. Visit: https://aws.amazon.com/lambda/

Examples of serverless applications with lambda

Example 1: Real-time file processing: https://github.com/aws-samples/lambda-refarch-fileprocessing

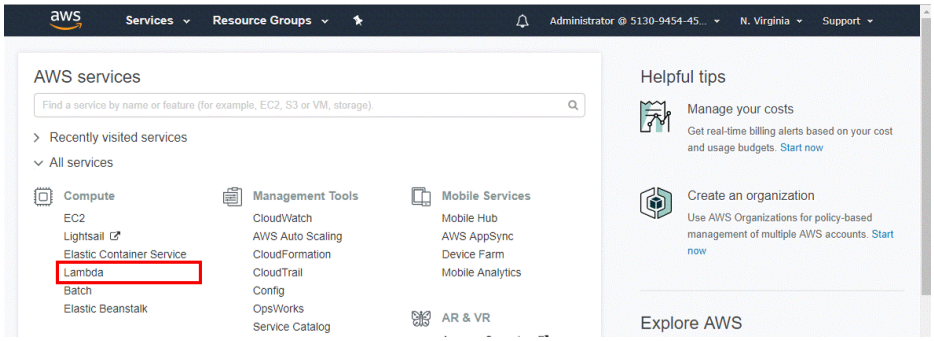
Example 2: Label detection is triggered when images are uploaded



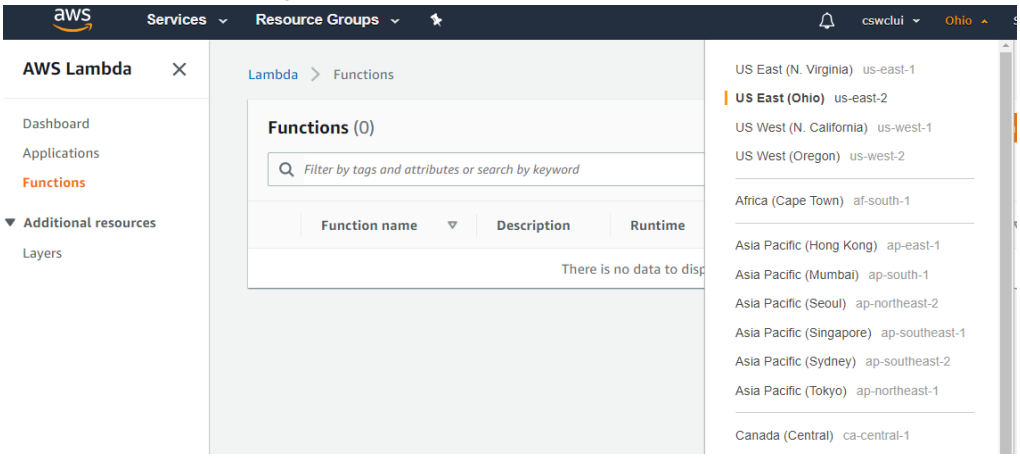
2.2. Create your first AWS Lambda Function.

Step 1: Enter the Lambda Console

Under **Services**, find **Lambda** under Compute and click to open the **AWS Lambda Console**.



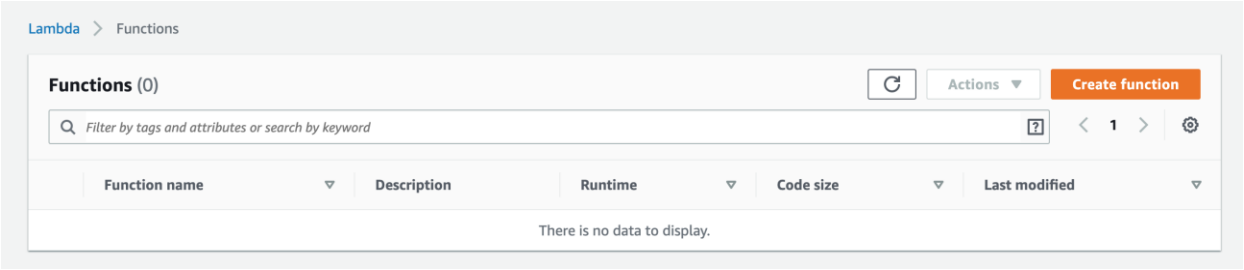
Select **US East (Ohio)** region.



Step 2: Create function

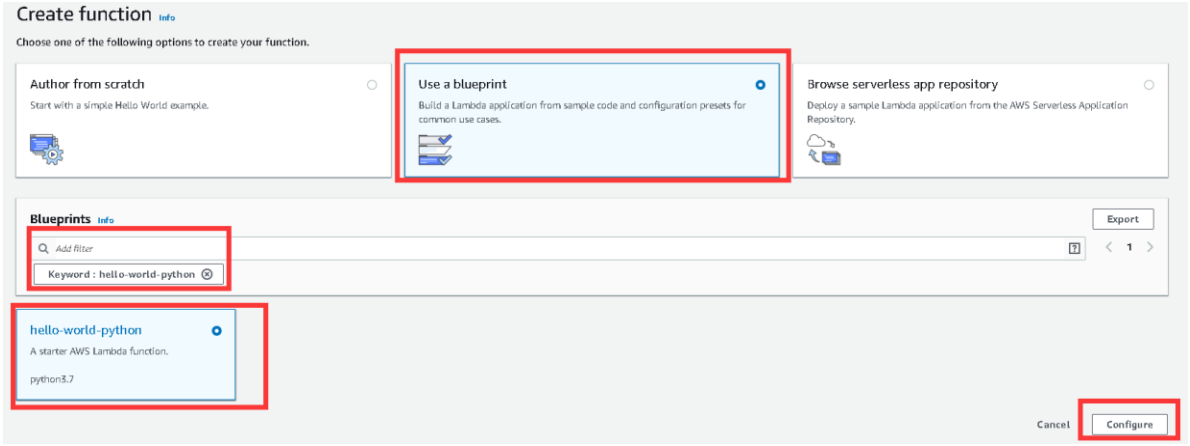
Blueprints provide example code to do some minimal processing. Most blueprints process events from specific event sources, such as Amazon S3, DynamoDB, or a custom application.

a. In the AWS Lambda console, select **Create a Function**.



b. Select **Blueprints**.

c. In the Filter box, type in hello-world-python and select the hello-world-python blueprint. Click **Configure**.

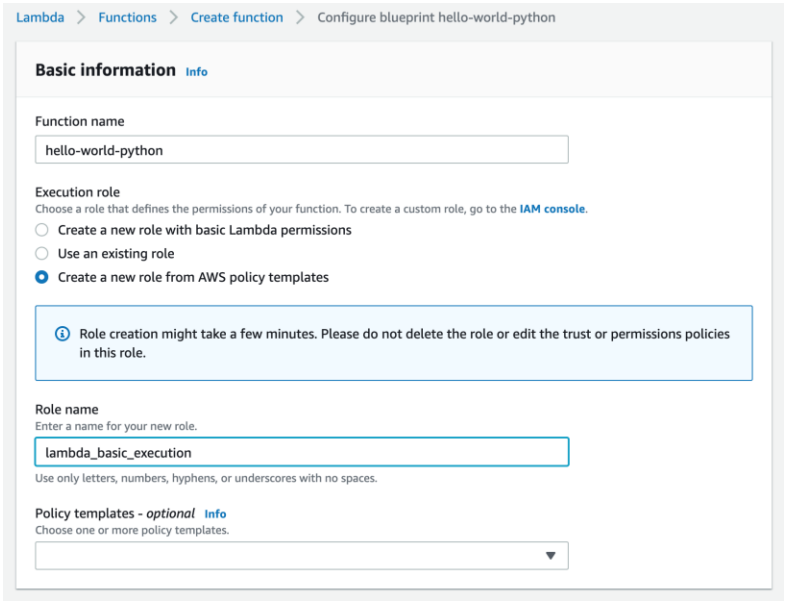


Step 3: Configure and Create Your Lambda Function

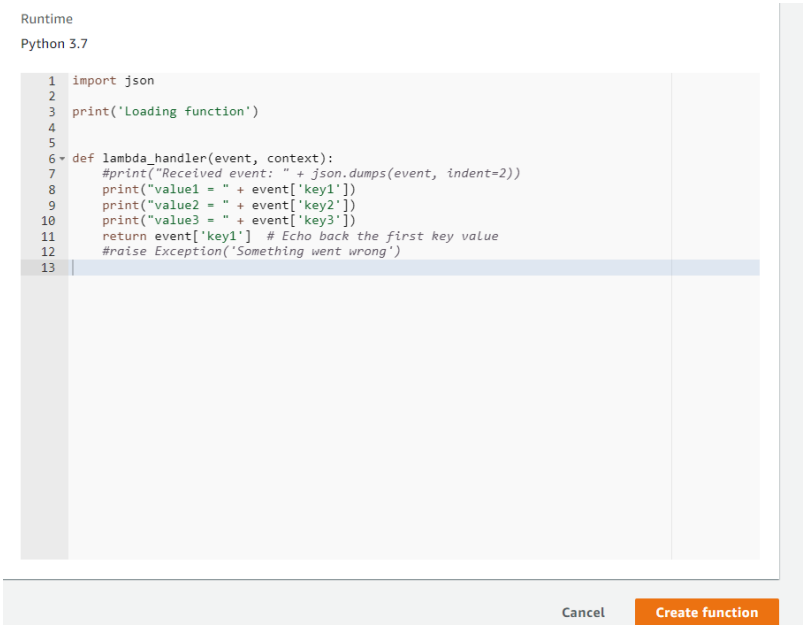
A lambda function consists of code you provide, associated dependencies, and configuration. The configuration information you provide includes the compute resources you want to allocate (for example, memory), execution timeout, and an IAM role that AWS Lambda can assume to execute your Lambda function on your behalf.

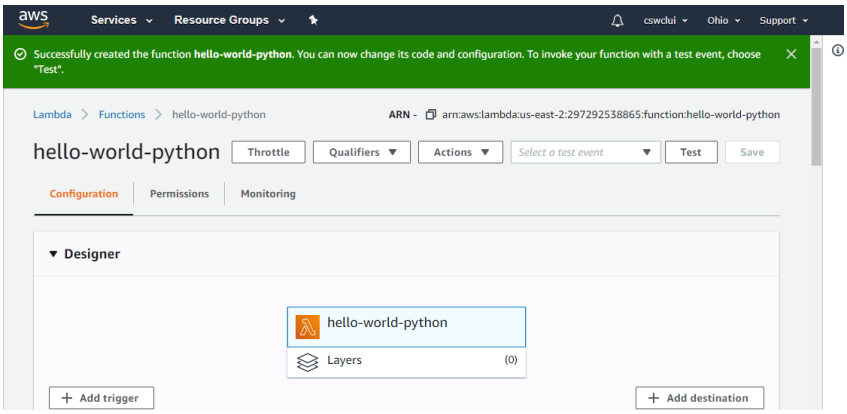
Enter **Basic Information** about your Lambda function.

- Name: You can name your Lambda function here. For this tutorial, enter **hello-world-python**.
- Role: You will create an IAM role (referred as the execution role) with the necessary permissions that AWS Lambda can assume to invoke your Lambda function on your behalf.
  Select Create new role from AWS policy template(s).
- Role name: type **lambda_basic_execution**.

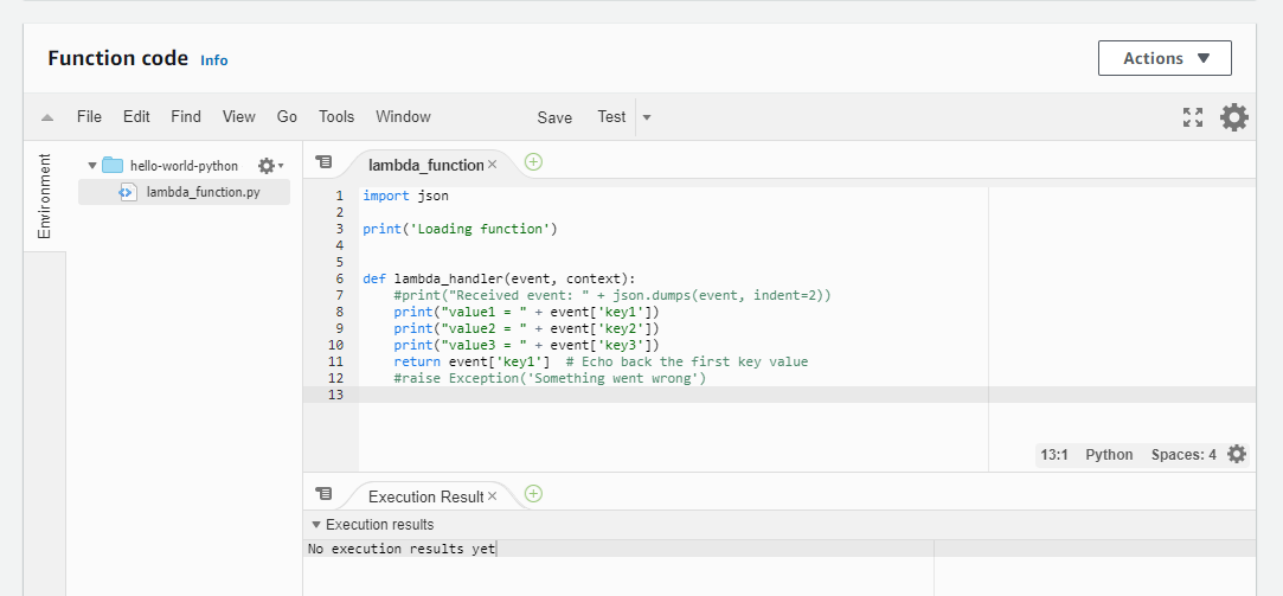


Go to the bottom of the page and select **Create Function**.

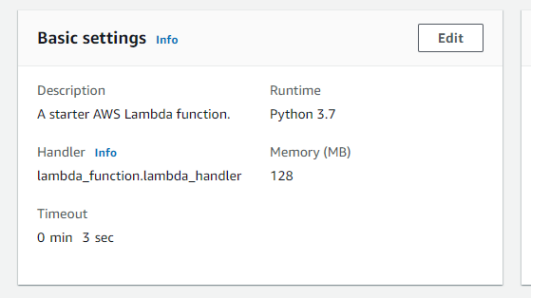




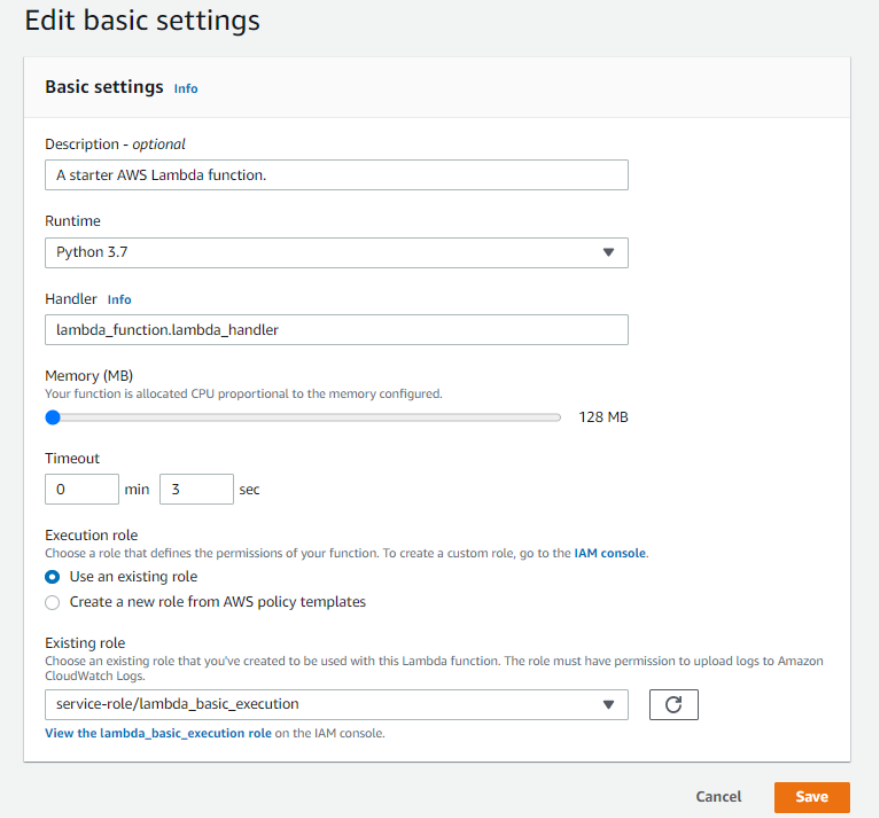
Modify the lambda function in the **Function code** section.



Scroll down the page to view the settings.



Click **Edit** to examine the basic settings.



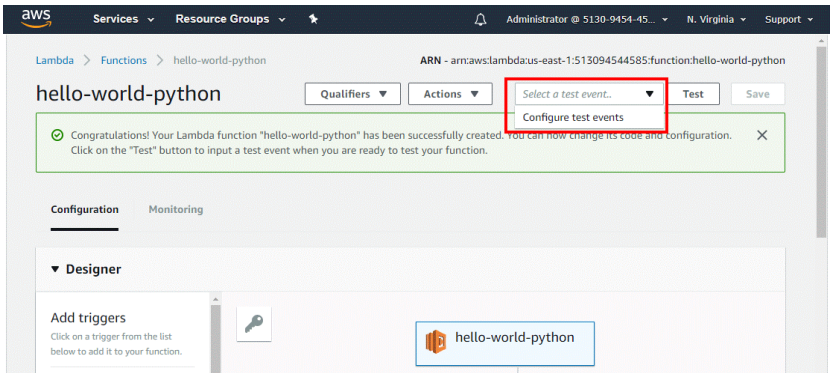
Runtime: Currently, you can author your Lambda function code in Java, Node.js, C#, Go or Python.
For this tutorial, leave this on Python 3.7 as the runtime.

Handler: You can specify a handler (a method/function in your code) where AWS Lambda can begin executing your code. AWS Lambda provides event data as input to this handler, which processes the event. In this example, Lambda identifies this from the code sample and this should be pre-populated with **lambda_function.lambda_handler**.

We will accept the default settings. Click **Cancel**.

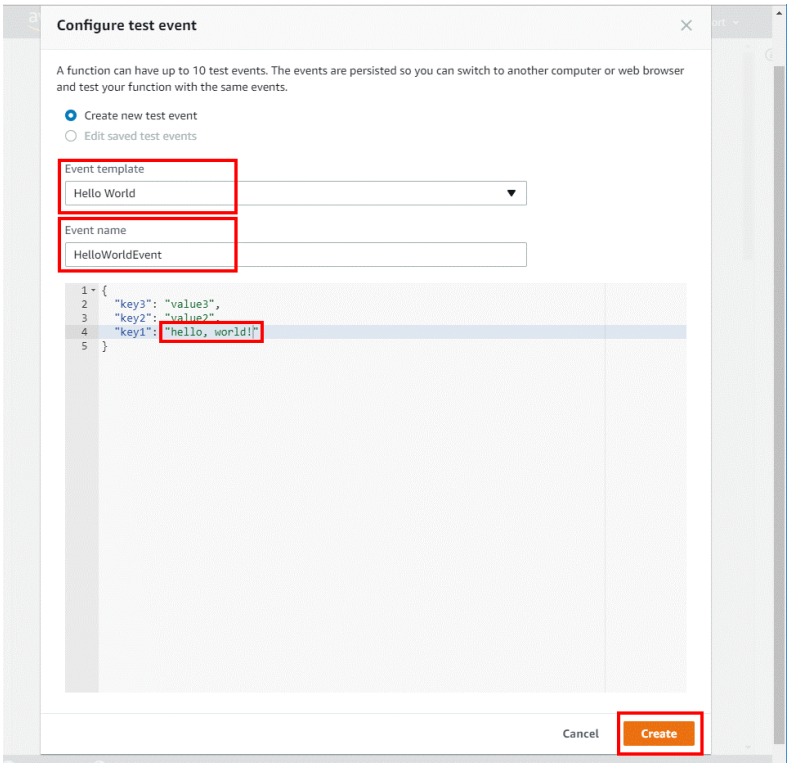
Step 4: Invoke Lambda Function and Verify Results
The console shows the hello-world-python Lambda function - you can now test the function, verify results, and review the logs.

Select **Configure Test Event** from the drop-down menu called "Select a test event…".



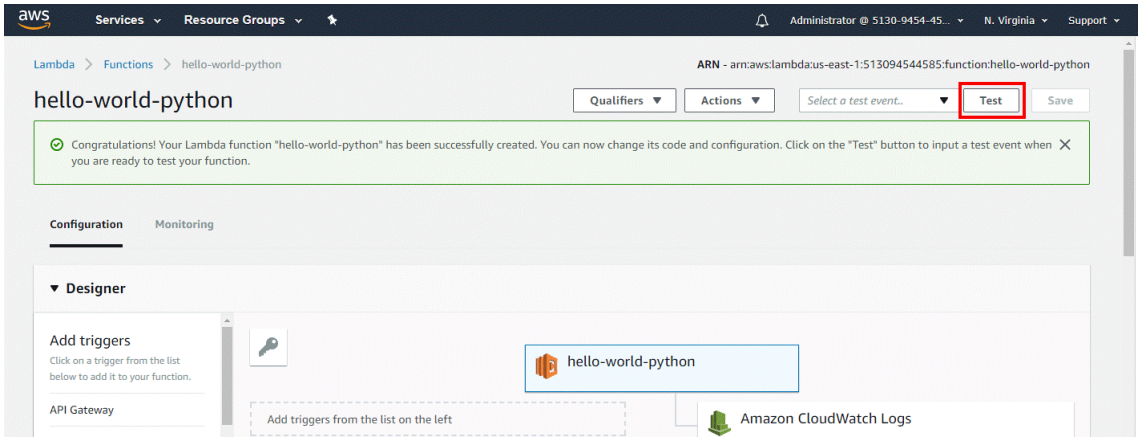
The editor pops up to enter an event to test your function.

- Choose Hello World from the Sample event template list from the Input test event page.
- Type in an event name like *HelloWorldEvent*.
- You can change the values in the sample JSON, but don't change the event structure. For this tutorial, replace value1 with *hello, world!*.



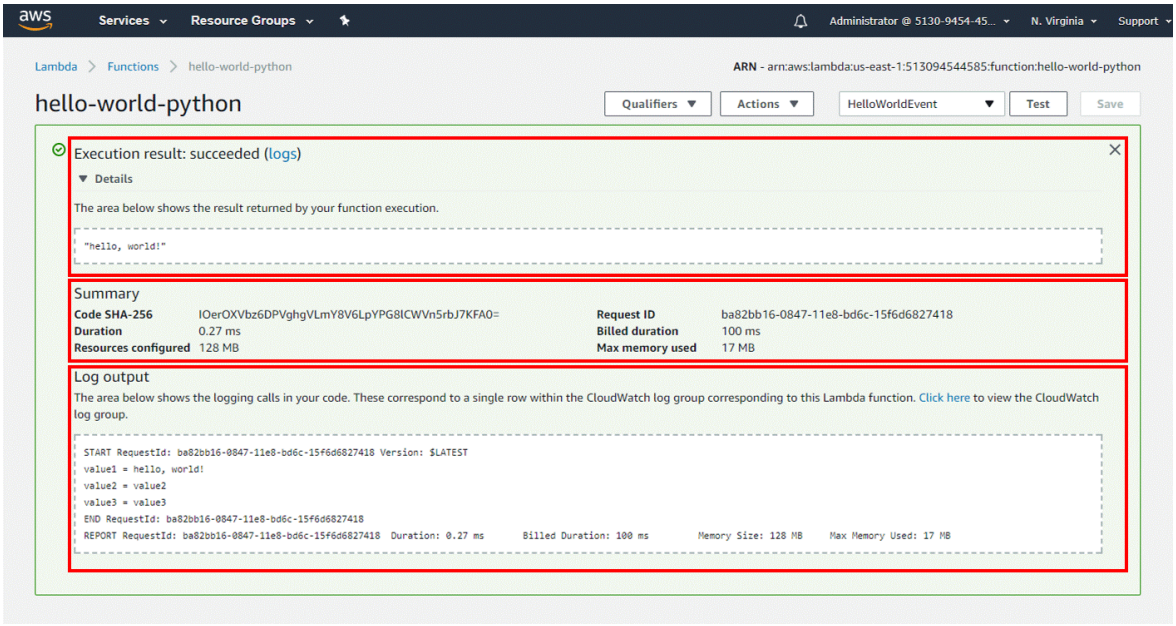
Select **Create**.

Select **Test**.



Upon successful execution, view the results in the console:

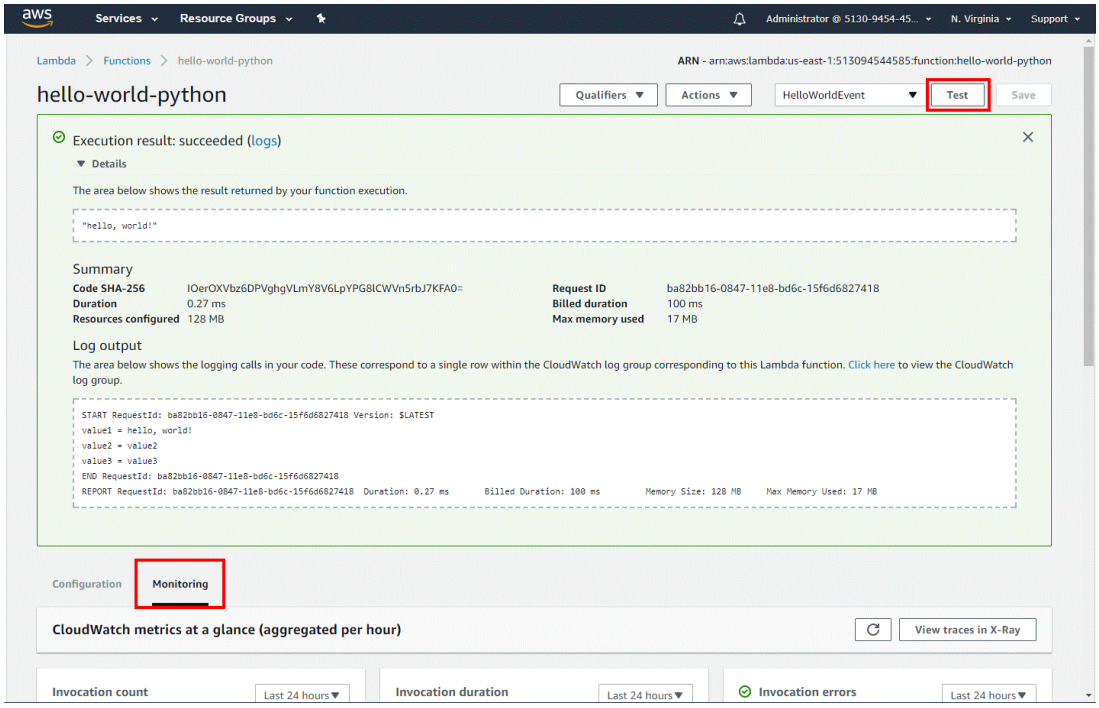
- The **Execution results** section verifies that the execution succeeded.
- The **Summary** section shows the key information reported in the Log output.
- The **Log output** section will show the logs generated by the Lambda function execution.



Step 5: Monitor Your Metrics

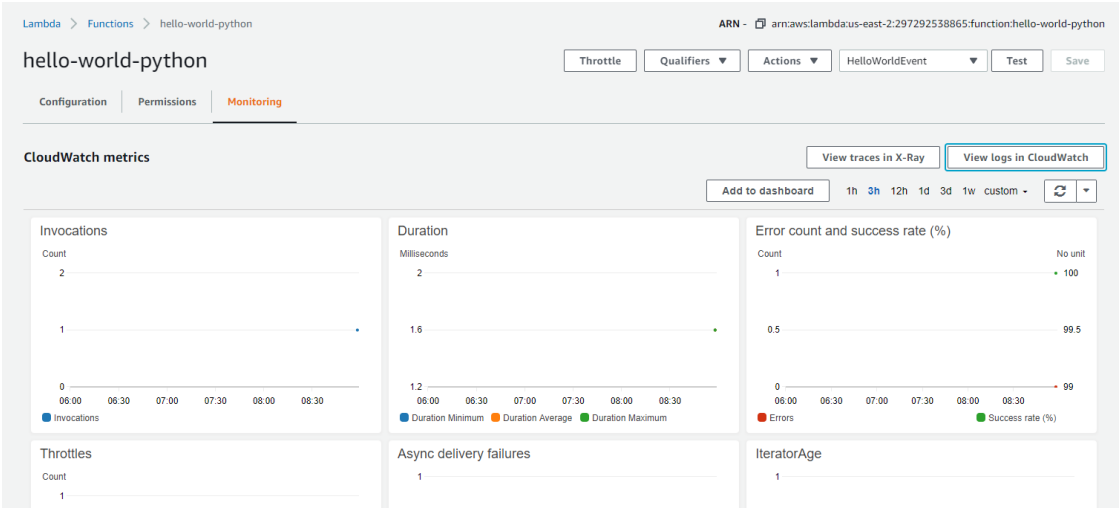
AWS Lambda automatically monitors Lambda functions and reports metrics through Amazon CloudWatch. To help you monitor your code as it executes, Lambda automatically tracks the number of requests, the latency per request, and the number of requests resulting in an error and publishes the associated metrics.

Invoke the Lambda function a few more times by repeatedly clicking the **Test** button. This will generate the metrics that can be viewed in the next step. Select **Monitoring** to view the results.



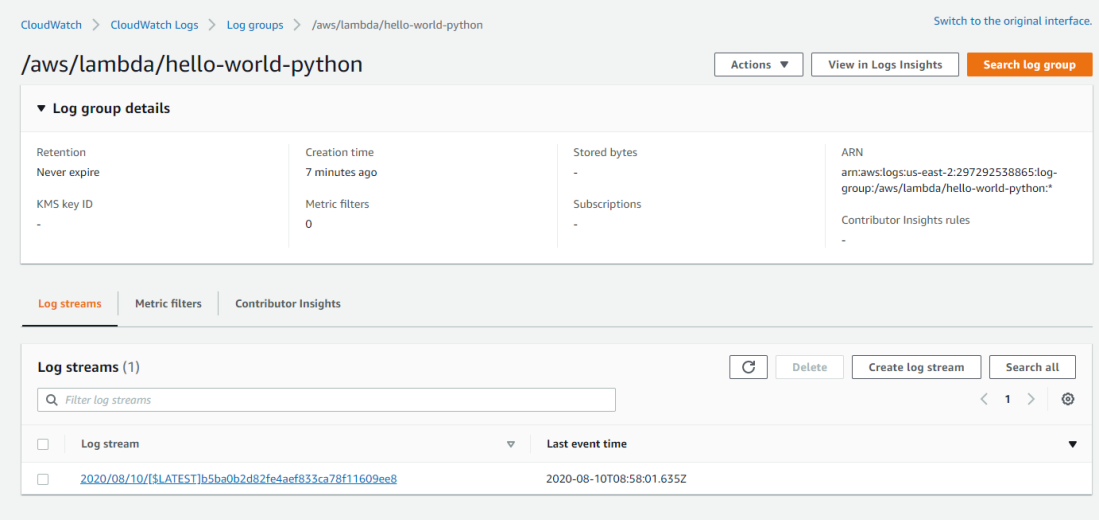
Scroll down to view the metrics for your Lambda function. Lambda metrics are reported through Amazon CloudWatch.

The Monitoring tab will show seven CloudWatch metrics: *Invocation count, duration, Error count and success rate*, *Throttles, Async delivery failures, Iterator age, and Concurrent executions.*

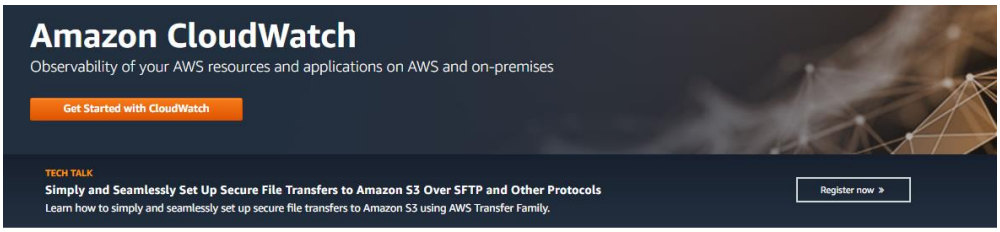


With AWS Lambda, you pay for what you use. After you hit your AWS Lambda free tier limit, you are charged based on the number of requests for your functions (invocation count) and the time your code executes (invocation duration).

Click View logs in **CloudWatch** to examine the log stream.



Visit https://aws.amazon.com/cloudwatch to understand more about AWS CloudWatch.

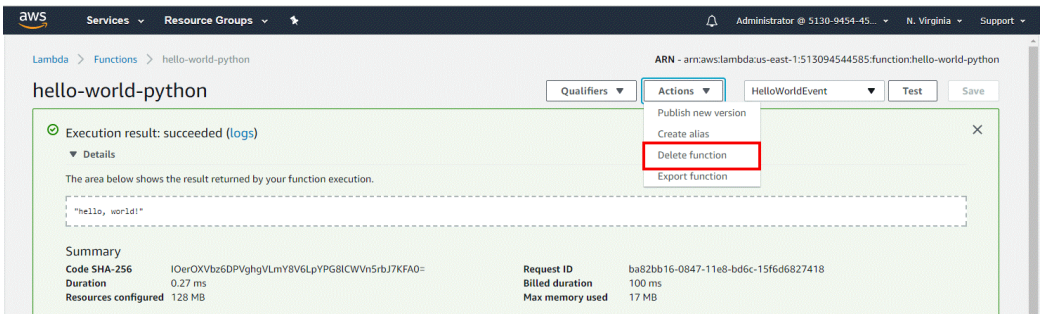


Amazon CloudWatch is a monitoring and observability service built for DevOps engineers, developers, site reliability engineers (SREs), and IT managers. CloudWatch provides you with data and actionable insights to monitor your applications, respond to system-wide performance changes, optimize resource utilization, and get a unified view of operational health. CloudWatch collects monitoring and operational data in the form of logs, metrics, and events, providing you with a unified view of AWS resources, applications, and services that run on AWS and on-premises servers. You can use CloudWatch to detect anomalous behavior in your environments, set alarms, visualize logs and metrics side by side, take automated actions, troubleshoot issues, and discover insights to keep your applications running smoothly.

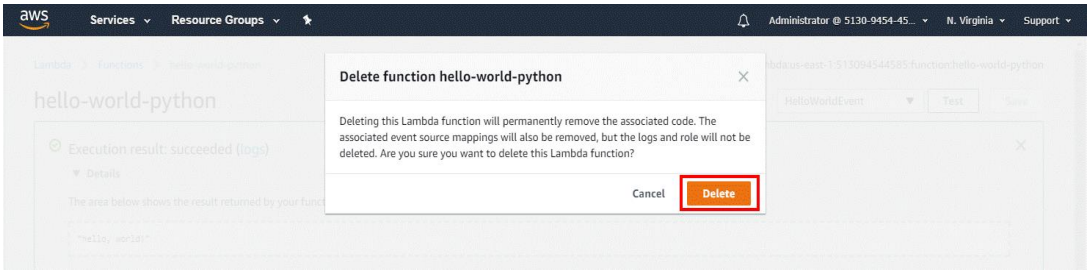
Amazon CloudWatch: Complete visibility of your cloud resources and applications (2:02)

Step 6: Delete the Lambda Function

While you will not get charged for keeping your Lambda function, you can easily delete it from the AWS Lambda console. Select the **Actions** button and click **Delete Function**.



You will be asked to confirm your termination - select **Delete**.



3. AWS API Gateway

3.1. Overview

Amazon API Gateway (https://aws.amazon.com/api-gateway/) is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale.

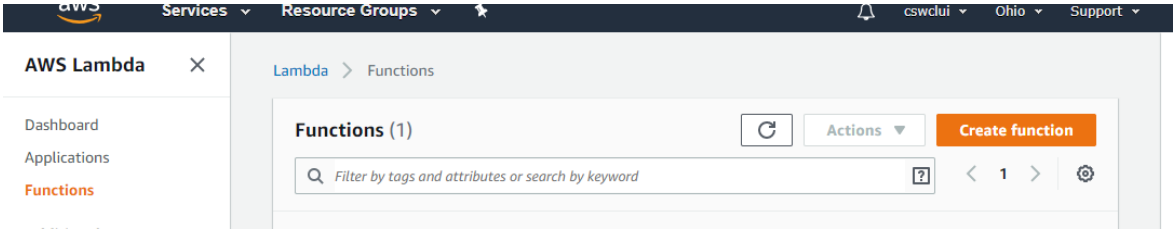
Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. With a few clicks in the AWS Management Console, you can create REST and WebSocket APIs that act as a "front door" for applications to access data, business logic, or functionality from your backend services, such as workloads running on Amazon Elastic Compute Cloud (Amazon EC2), code running on AWS Lambda, any web application, or real-time communication applications.

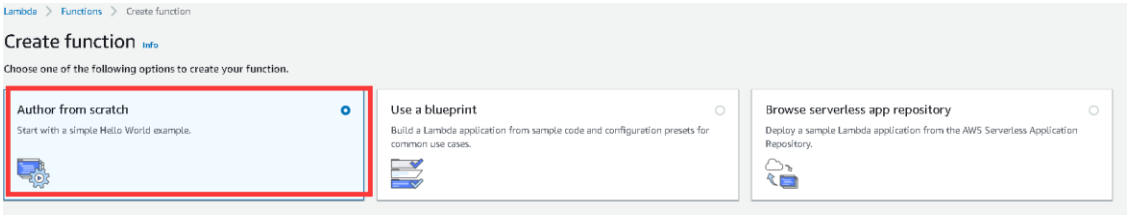
API Gateway handles all the tasks involved in accepting and processing up to hundreds of thousands of concurrent API calls, including traffic management, authorization and access control, monitoring, and API version management. API Gateway has no minimum fees or startup costs. You pay only for the API calls you receive and the amount of data transferred out and, with the API Gateway tiered pricing model, you can reduce your cost as your API usage scales.

3.2. Create a REST API for Lambda Function

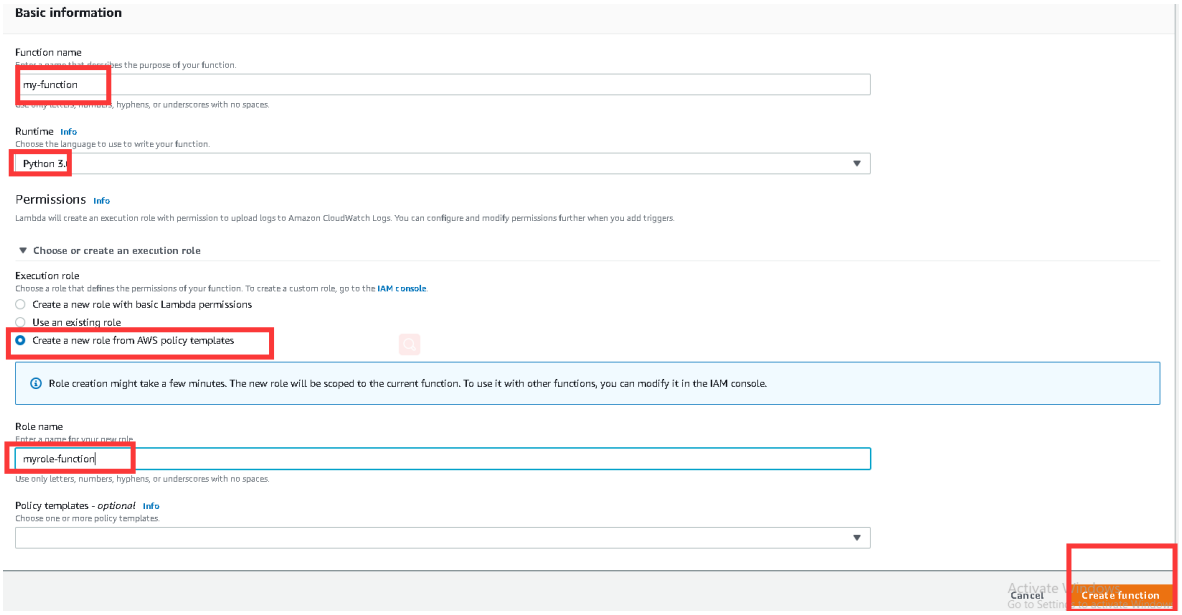
Open the **AWS Lambda** console.

Choose **Create function** and select **Author from scratch**.

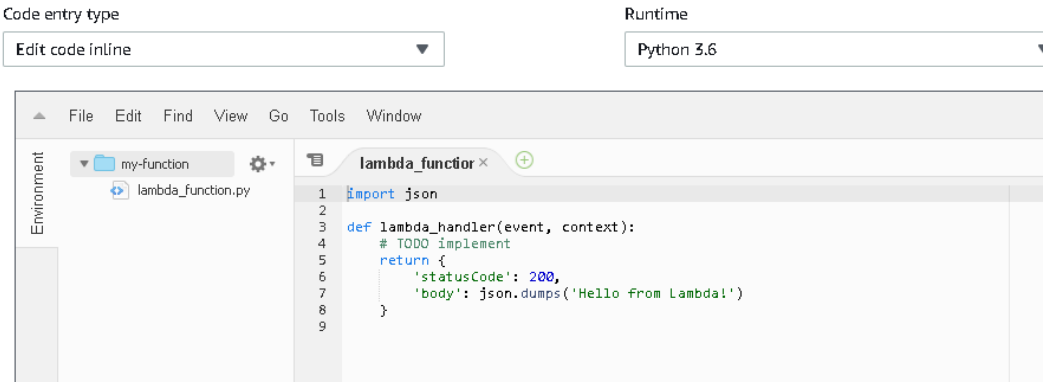




For Function name, enter **my-function**. Choose the options as shown below. Click **Create function**.


Basic information
Function name
my-function
Runtime Info
Python 3.
Permissions Info
Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and modify permissions further when you add triggers.
Choose or create an execution role
Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.
Create a new role with basic Lambda permissions
Use an existing role
Create a new role from AWS policy templates
Role creation might take a few minutes. The new role will be scoped to the current function. To use it with other functions, you can modify it in the IAM console.
Role name
myrole-function
Use only letters, numbers, hyphens, or underscores with no spaces.
Policy templates - optional Info
Choose one or more policy templates.
Cancel
Create function

By default, the lambda handler code is as follows.

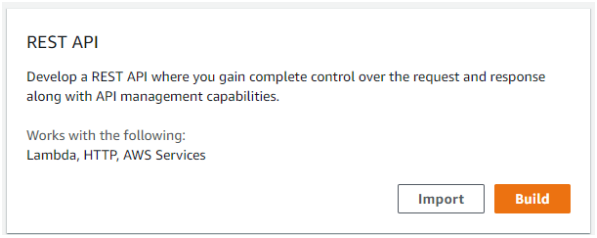


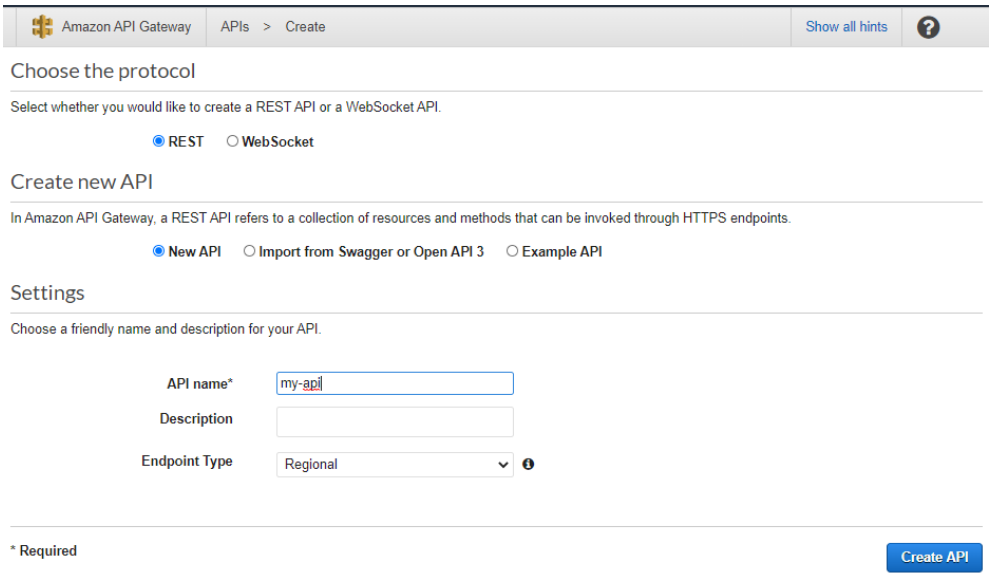
3.3. Create a REST API in the API Gateway Console

In this section, you create a simple REST API in the API Gateway console and attach your Lambda function to it as a backend.

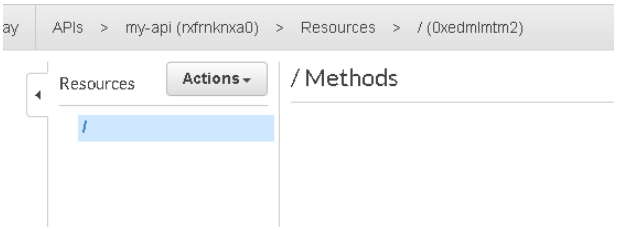
From the **Services** menu, choose **API Gateway** to go to the API Gateway console. Under **Choose an API type**, choose **REST API**, and click **Build**.

1. Under **Create new API**, choose **New API**.
2. Under **Settings**:
    - For **API name**, enter my-api.
    - If desired, enter a description in the **Description** field; otherwise, leave it empty.
    - Leave **Endpoint Type** set to **Regional**.
3. Choose **Create API**.

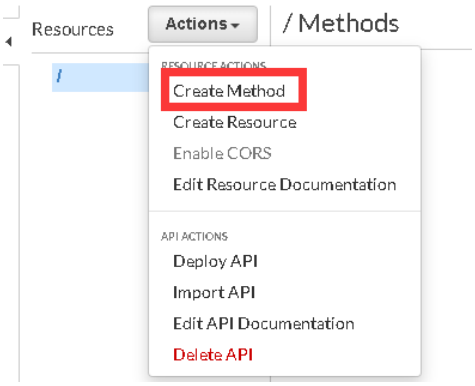




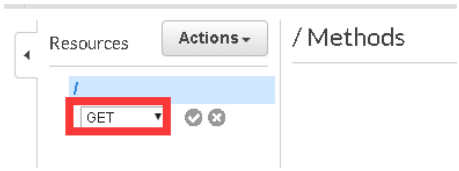
Under **Resources**, you'll see the endpoint `/`. This is the root-level resource, which corresponds to the base path URL for your API



From the **Actions** dropdown menu, choose **Create Method**.



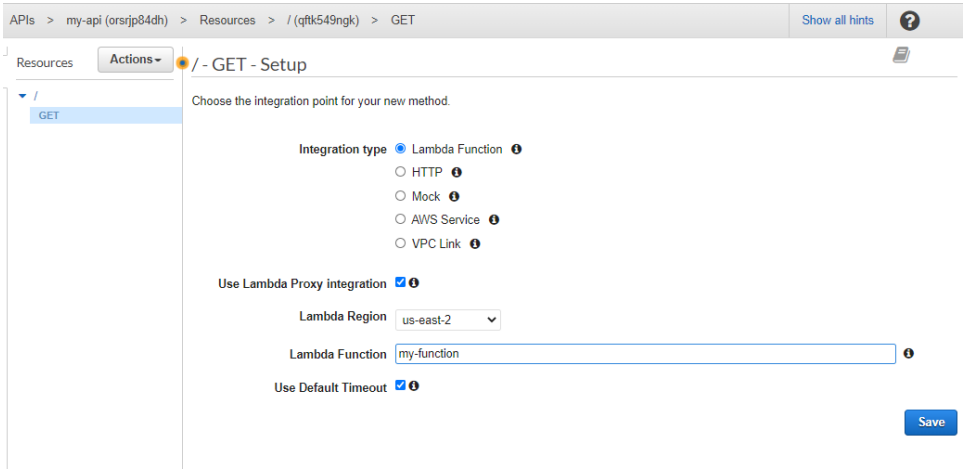
Under the resource name (/), you'll see a dropdown menu. Choose **GET** and then choose the checkmark icon to save your choice.



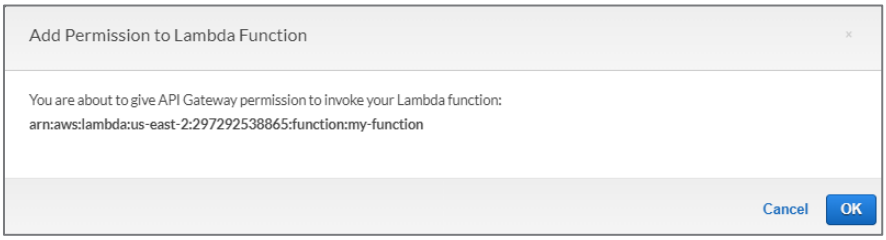
In the **/ – GET – Setup** pane:

- Choose **Lambda Function** as for **Integration type**
- Check **Use Lambda proxy integration**.
- For **Lambda Region**, choose the Region where you created your Lambda function (e.g. us-east-2)
- In the **Lambda Function** field, input **my-function** (or whatever name you gave the function that you created in the previous step) from the dropdown menu.
- Leave Use **Default Timeout** checked.

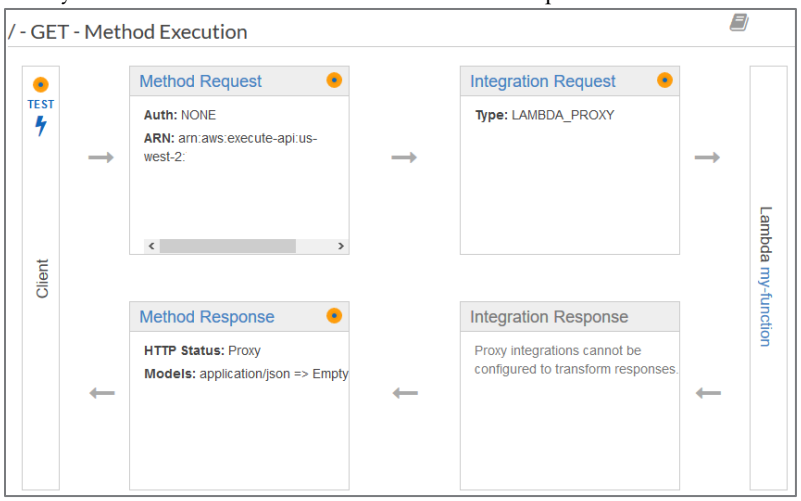
Choose **Save** to save your choice.



When the **Add Permission to Lambda Function** popup appears (saying **"You are about to give API Gateway permission to invoke your Lambda function…"**), choose **OK** to grant API Gateway that permission.



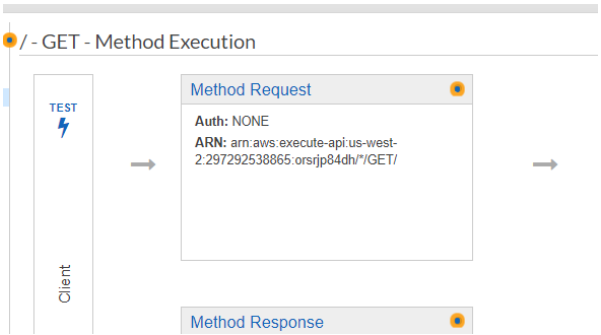
Now you'll see a **/ – GET – Method Execution** pane:



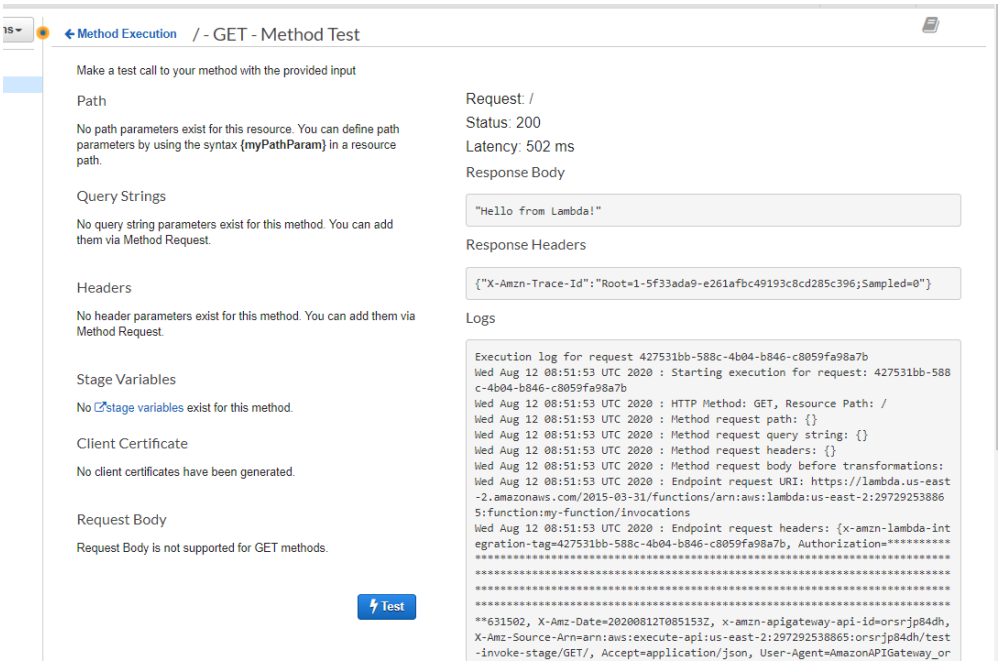
The **Method Execution** pane contains these items, in clockwise order:

- **Client**: This box represents the client (browser or app) that calls your API's GET method.
- **Method Request**: This box represents the client's GET request as it's received by API Gateway.
- **Integration Request**: This box represents the GET request as it's passed to the backend.
- **Lambda my-function**: This box represents the backend Lambda function you created previously.
    - If you choose **my-function**, that opens the my-function Lambda function in the Lambda console.
- **Integration Response**: This box represents the response from the backend, before it's passed to the client as a method response.
    - For Lambda proxy integration, this entire box is grayed out, because a proxy integration returns the Lambda function's output as is.
- **Method Response**: This box represents the method response that's returned to the client as an HTTP status code, an optional response header, and an optional response body.
    - By default, the response body that's returned by your Lambda function is passed through as is as a JSON document, so the response body default setting is **application/json** with an empty model (indicating that the body is passed through as is).

Click **Test**.

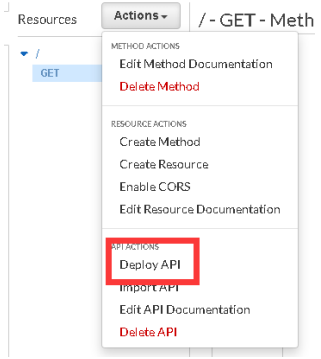


Click **Test** and check that the API responds with the message "Hello from Lambda!".

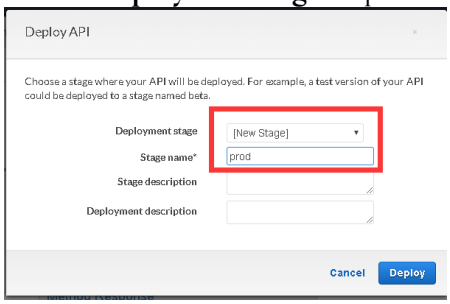


3.4. Deploy Your REST API in the API Gateway Console

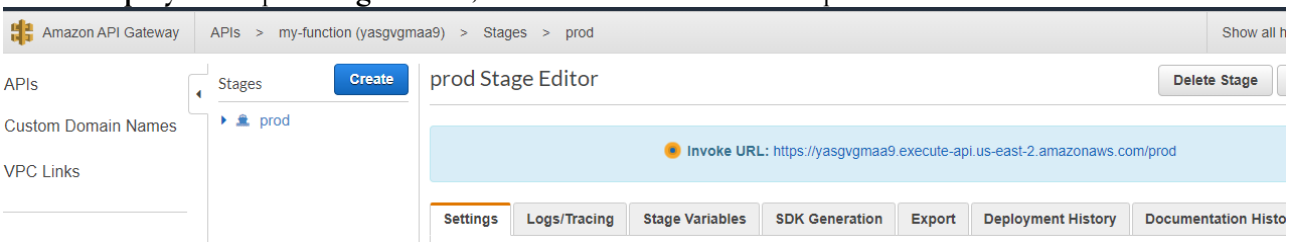
From the **Actions** dropdown menu, choose **Deploy API**.



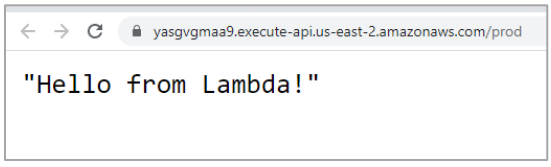
From the **Deployment stage** dropdown menu, choose **[New Stage]**. For **Stage name**, enter prod.



Choose **Deploy**. In the prod **Stage Editor**, note the **Invoke URL** at the top.



If you choose the Invoke URL, it will open a new browser tab with that URL. If you refresh the new browser tab, you'll see the message body ("Hello from Lambda!") returned.



3.5. Passing HTTP Query String to Lambda

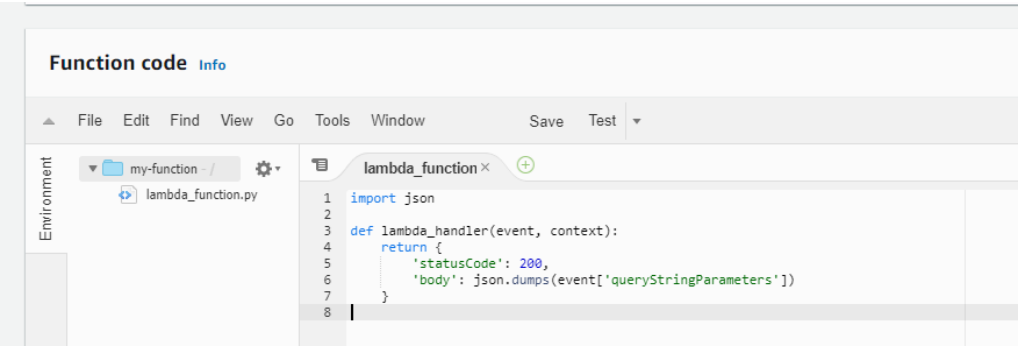
Amazon API Gateway invokes your function synchronously with an event that contains details about the HTTP request that it received. An Example Amazon API Gateway Message Event is shown below.

```
{
  "path": "/test/hello",
  "headers": {
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8",
    "Accept-Encoding": "gzip, deflate, lzma, sdch, br",
    "Accept-Language": "en-US,en;q=0.8",
    "CloudFront-Forwarded-Proto": "https",
    "CloudFront-Is-Desktop-Viewer": "true",
    "CloudFront-Is-Mobile-Viewer": "false",
    "CloudFront-Is-SmartTV-Viewer": "false",
    "CloudFront-Is-Tablet-Viewer": "false",
    "CloudFront-Viewer-Country": "US",
    "Host": "wt6mne2s9k.execute-api.us-west-2.amazonaws.com",
    "Upgrade-Insecure-Requests": "1",
    "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/52.0.2743.82 Safari/537.36 OPR/39.0.2256.48",
    "Via": "1.1 fb7cca60f0ecd82ce07790c9c5eef16c.cloudfront.net (CloudFront)",
    "X-Amz-Cf-Id": "nBsWBOrSHMgnaROZJK1wGCZ9PcRcSpq_oSXZNQwQ10OTZL4cimZo3g==",
    "X-Forwarded-For": "192.168.100.1, 192.168.1.1",
    "X-Forwarded-Port": "443",
    "X-Forwarded-Proto": "https"
  },
  "pathParameters": {
    "proxy": "hello"
  },
  "requestContext": {
    "accountId": "123456789012",
    "resourceId": "us4z18",
    "stage": "test",
    "requestId": "41b45ea3-70b5-11e6-b7bd-69b5aaebc7d9",
    "identity": {
      "cognitoIdentityPoolId": "",
      "accountId": "",
      "cognitoIdentityId": "",
      "caller": "",
      "apiKey": "",
      "sourceIp": "192.168.100.1",
      "cognitoAuthenticationType": "",
      "cognitoAuthenticationProvider": "",
      "userArn": "",
      "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/52.0.2743.82 Safari/537.36 OPR/39.0.2256.48",
      "user": ""
    },
    "resourcePath": "/{proxy+}",
    "httpMethod": "GET",
    "apiId": "wt6mne2s9k"
  },
  "resource": "/{proxy+}",
  "httpMethod": "GET",
  "queryStringParameters": {
    "name": "me"
  },
  "stageVariables": {
    "stageVarName": "stageVarValue"
  }
}
```

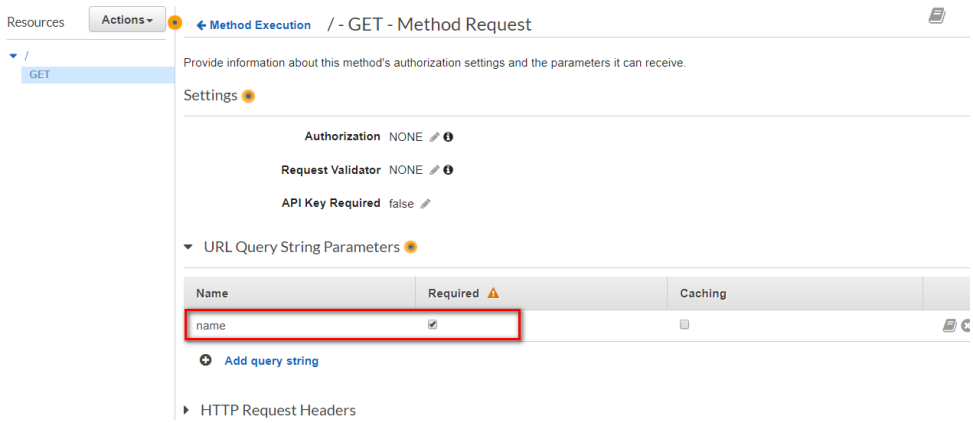
The HTTP query string parameters are passed to the lambda function through the **event** object (with the "queryStringParameters" attribute). Change your **lambda_handler** function as follows.

```
import json

def lambda_handler(event, context):
    return {
        'statusCode': 200,
        'body': json.dumps(event['queryStringParameters'])
    }
```

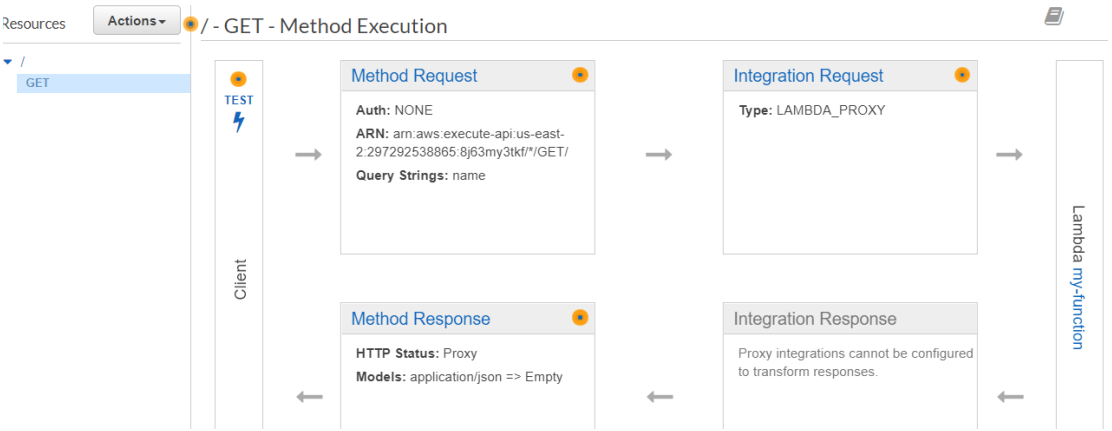
Click **Save**.



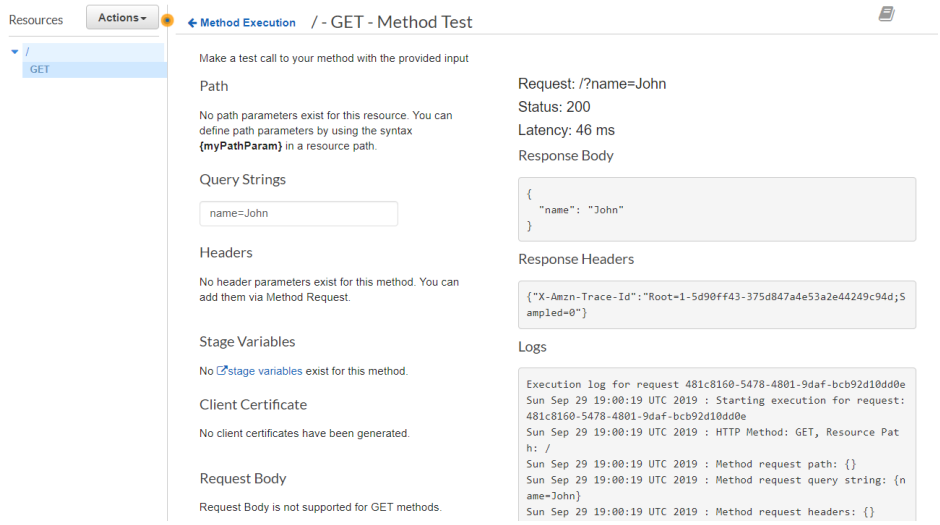
In the API gateway console, select "Method Request" in API Gateway and add query string parameter "name".



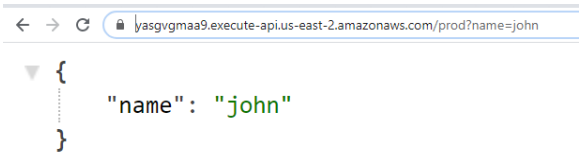
Click ← **Method Execution** to confirm the change.

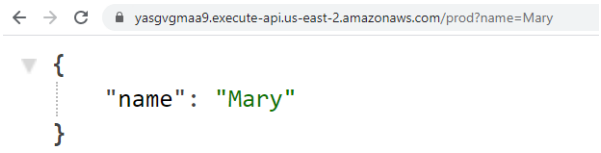


Click **Test** to verify the result. Specify the query string **name=John**.



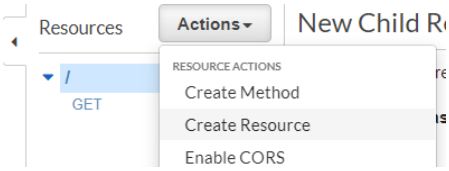
Deploy the API and access the endpoint with different name parameters in query string.



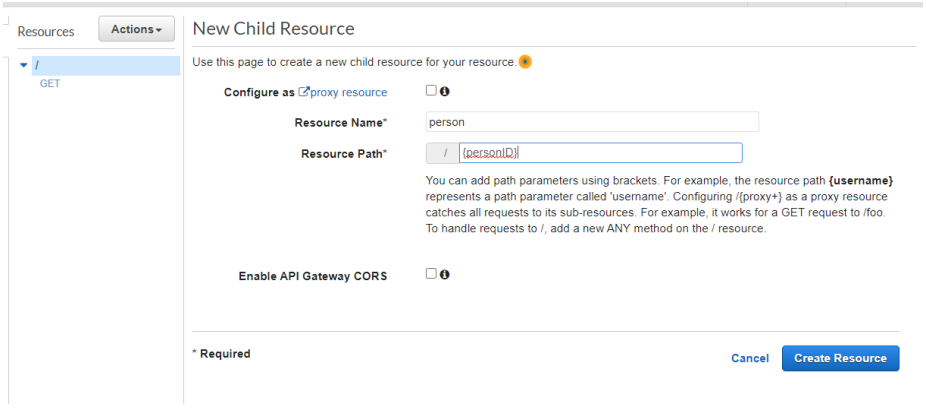


3.6. Passing path parameter to Lambda

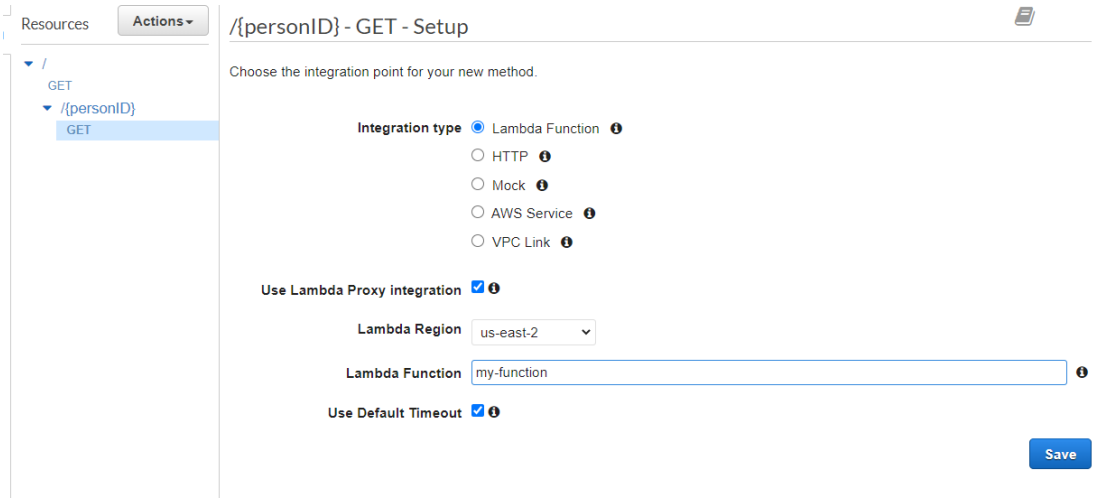
Select the root resource `/`.



Define a resource **person** with {**personID**} as the path parameter.



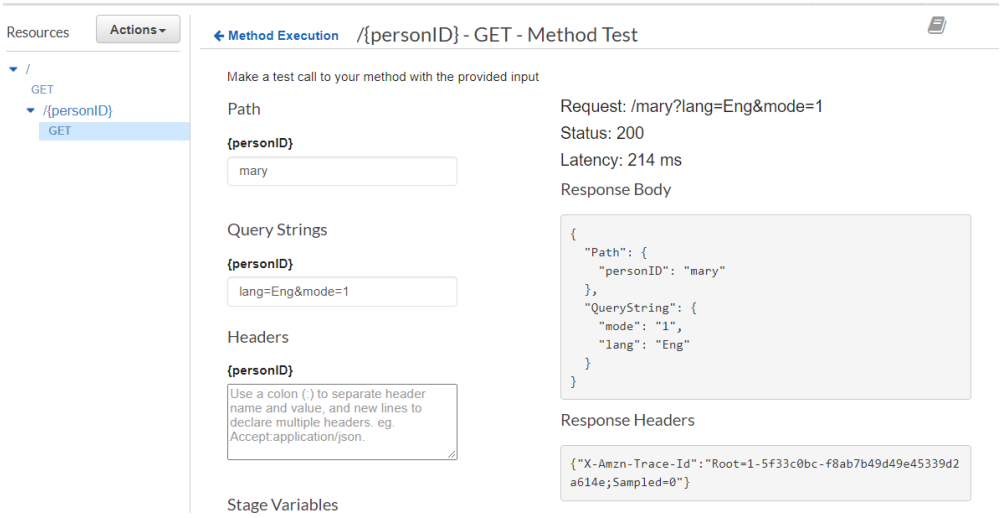
Create a GET method under `/`{**personID**}. Configure the method as follows.



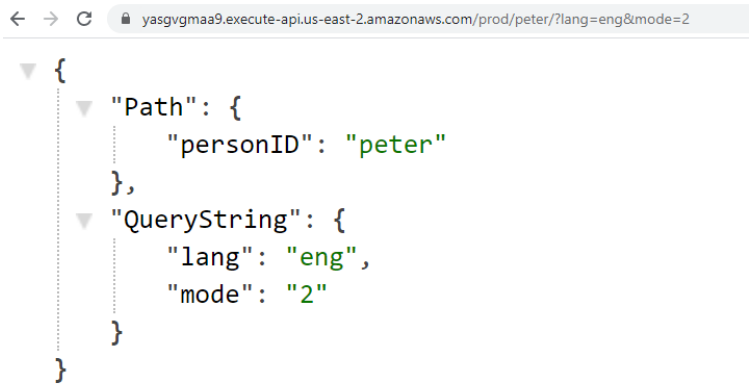
Modify your lambda handler code for **my-function** as follows.

```
import json

def lambda_handler(event, context):
    result = {}
    result["Path"] = event['pathParameters']
    result["QueryString"] = event['queryStringParameters']

    return {
        'statusCode': 200,
        'body': json.dumps(result)
    }
```

Test your API with path parameter **mary** and query string **lang=Eng&mode=1**.



Deploy your API. Access the endpoint with the path parameter and query strings (e.g. /peter/?lang=eng&mode=2).

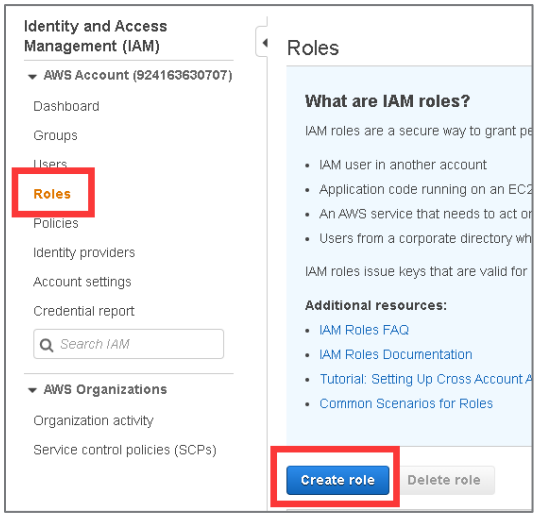


4. Integrating AWS Lambda, S3 and DynamoDB

4.1. Creating a role in AWS Identity and Access Management (IAM)

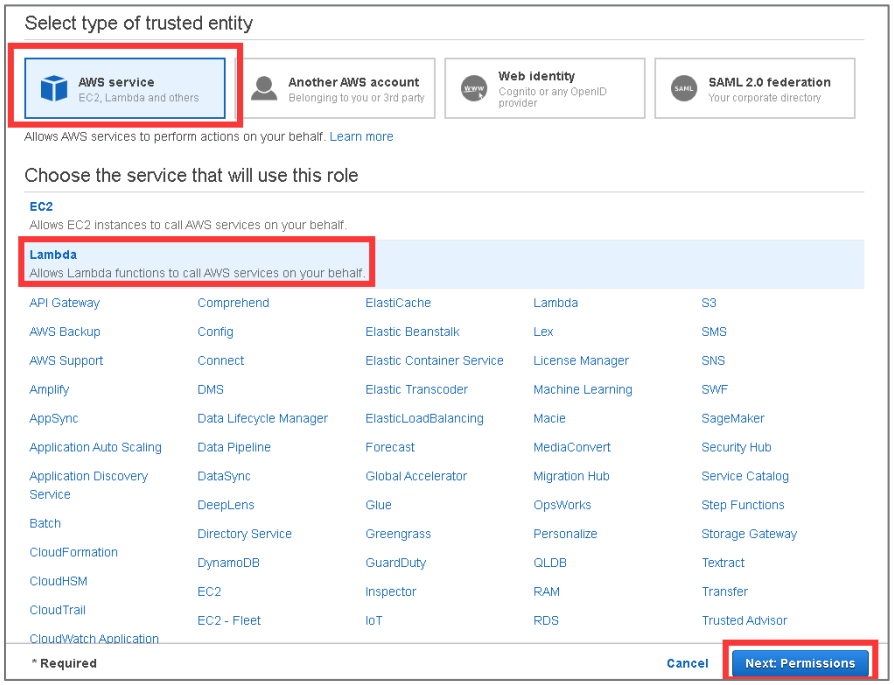
We first create a role inside IAM to allow lambda function to access S3 and Amazon recognition service.

Navigate to IAM console at console.aws.amazon.com/iam. On the left navigation menu click "Roles". Click blue **Create role** button.

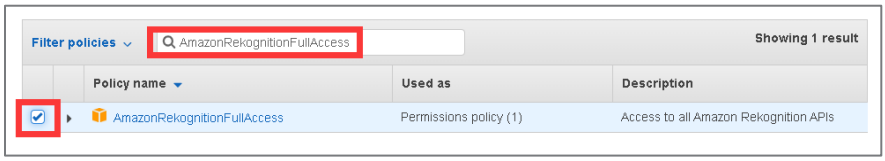


On Select Role type page, select "AWS Service" section and choose "Lambda"

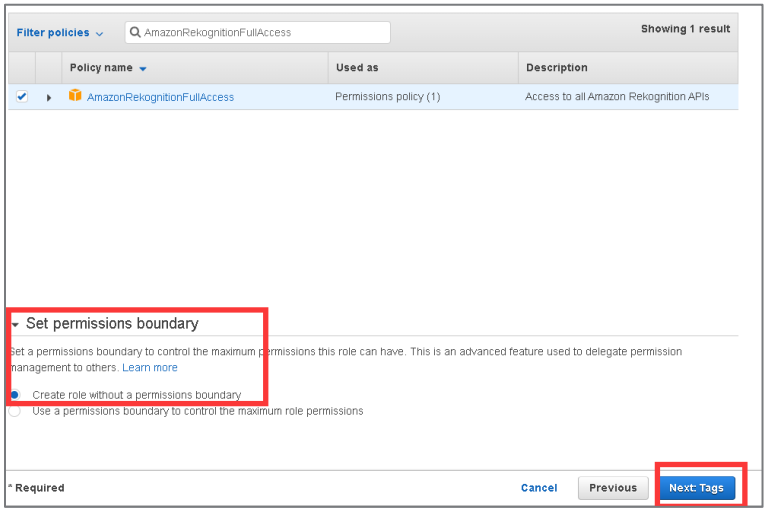
Click blue "Next: Permissions" button.



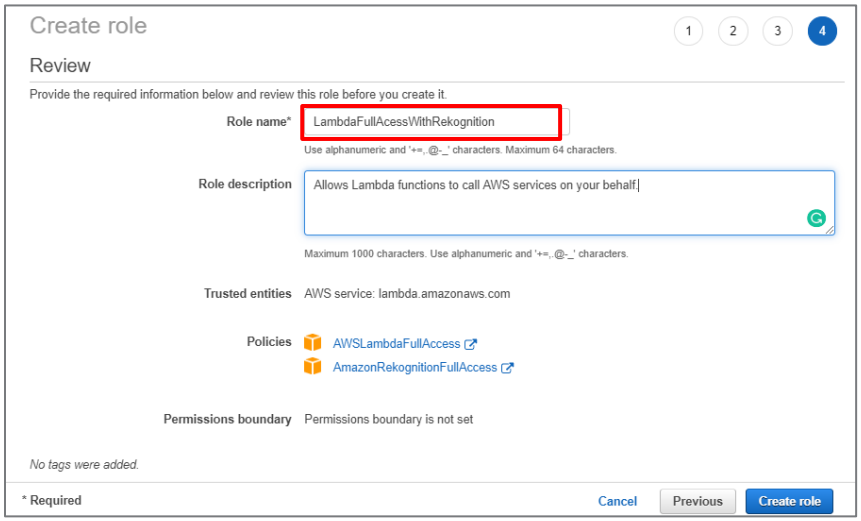
On Attach Permissions page, search for and select the checkboxes next to both ***AWSLambdaFullAccess*** and ***AmazonRekognitionFullAccess***.



Click blue "Next: Review" button

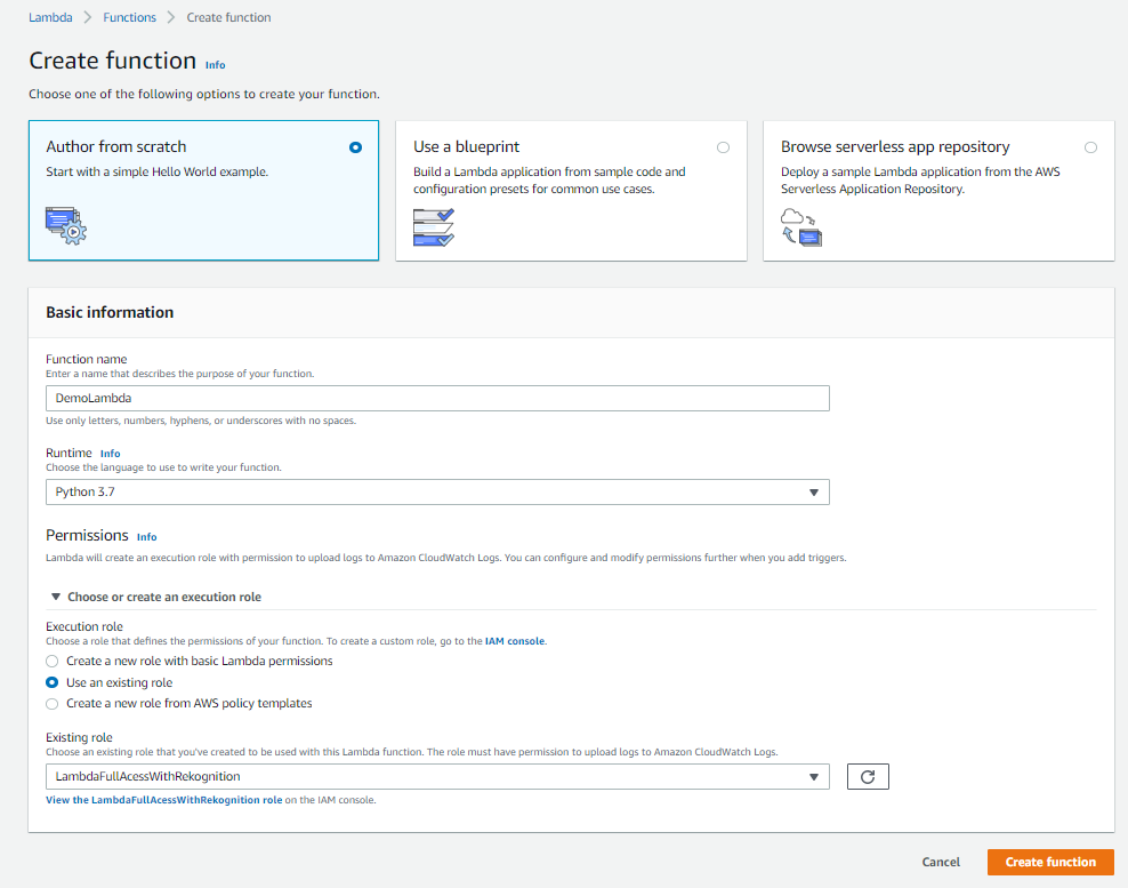


Give your role a name that you like (e.g. LambdaFullAcessWithRekognition). Review the permissions granted by the the policies and Click "Create role".



4.2. Accessing Dynamo DB in AWS Lambda

In AWS Lambda, create a lambda function (e.g. DemoLambda**)** and use the created role (e.g. **LambdaFullAcessWithRekognition**).

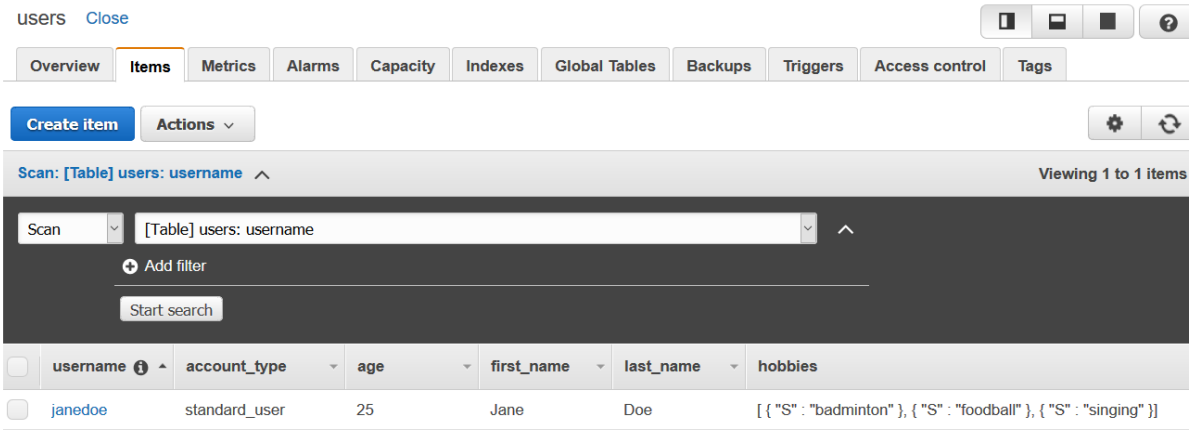


In the function code section, input the following code. Click **Save**.

```
import json
import boto3
import urllib

def insert_item_db():
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('users')
    table.put_item(
        Item={
        'username': 'janedoe',
        'first_name': 'Jane',
        'last_name': 'Doe',
        'age': 25,
        'hobbies':['badminton', 'foodball','singing'],
        'account_type': 'standard_user'
        }
    )

def scan_table(): #scan the entire table
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('users')
    result = table.scan()
    for i in result['Items']:
        print(i)


def get_db_item(): #retrieve an item using primary key
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('users')
    response = table.get_item(
        Key={
            'username': 'janedoe'
        }
    )
    item = response['Item']
    print(item)


def lambda_handler(event, context):
    insert_item_db()
    #scan_table()
    #get_db_item()
```

Click **Test** to test the lambda function.

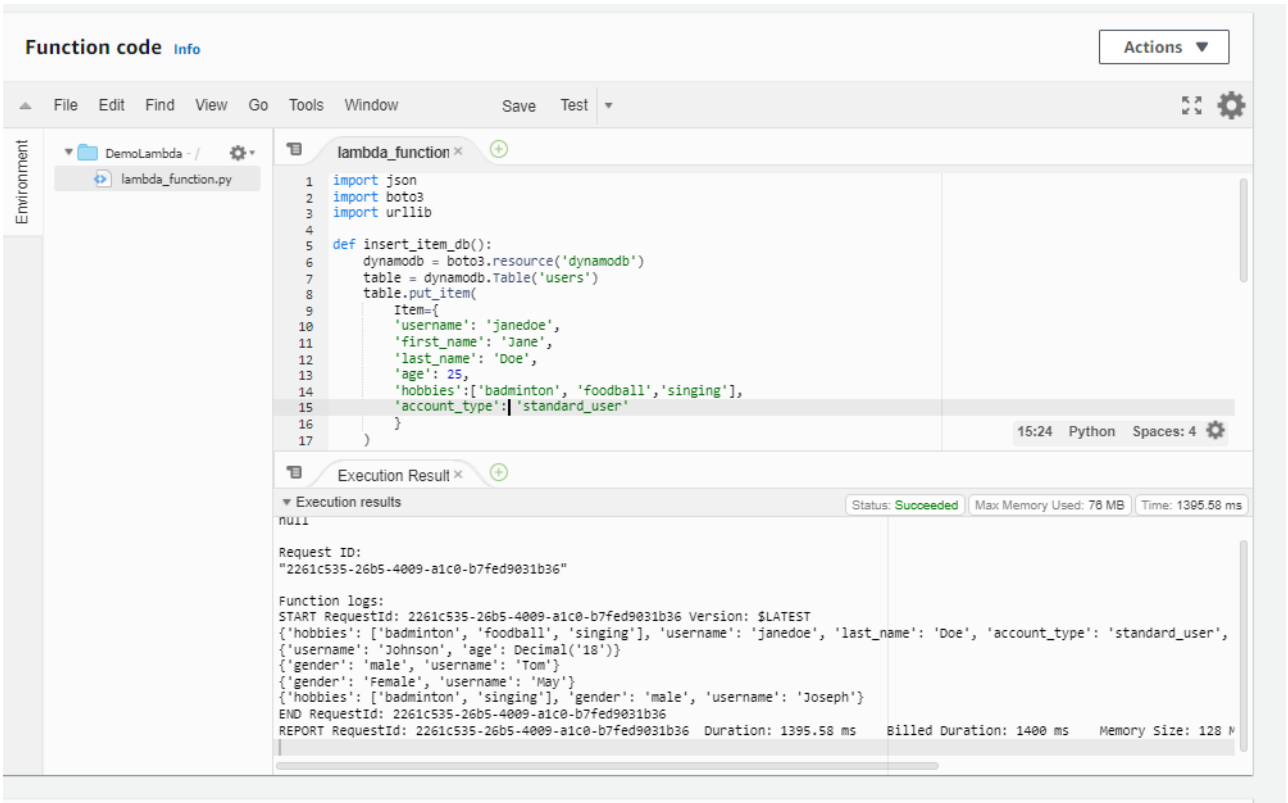
Check that the record is created in Dynamo DB.



Comment out the following statement in the code.
insert_item_db()

Uncomment the following lines. **Test** the lambda function and observe the execution results.
#scan_table()
#get_db_item()

Sample output for scan_table()

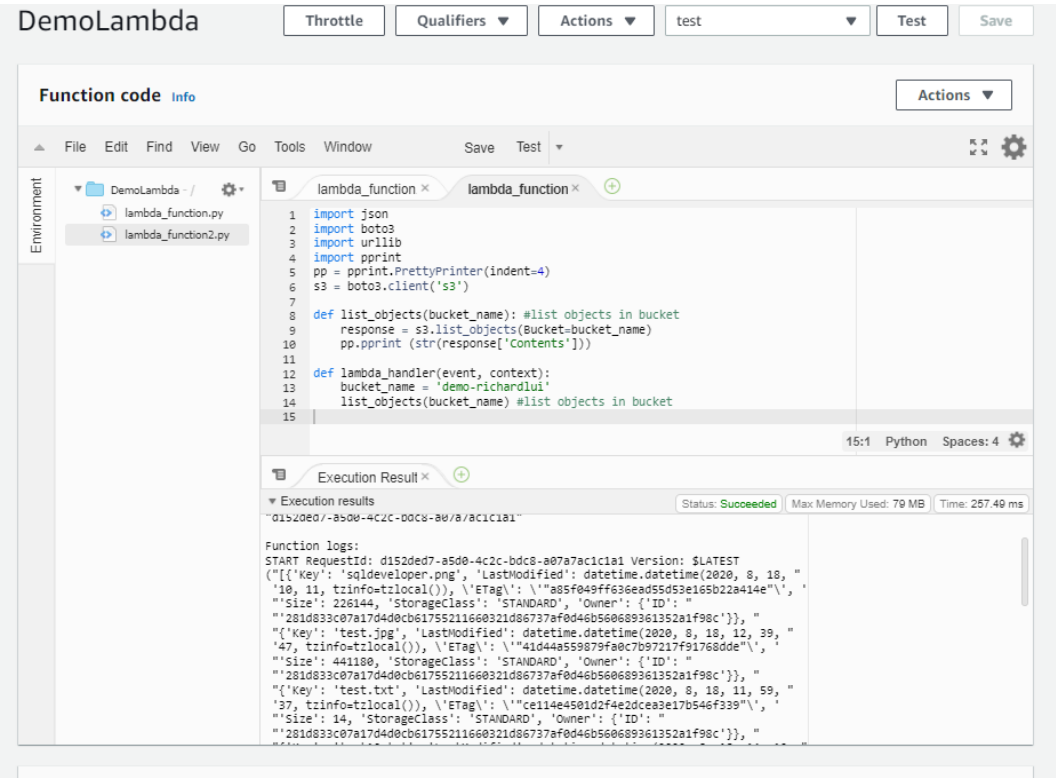


4.3. S3 integration in AWS Lambda

Modify your lambda code as follows. Add your S3 bucket name to the code. **Save** the function

```
import json
import boto3
import urllib
import pprint
pp = pprint.PrettyPrinter(indent=4)
s3 = boto3.client('s3')

def list_objects(bucket_name): #list objects in bucket
    response = s3.list_objects(Bucket=bucket_name)
    pp.pprint (str(response['Contents']))

def lambda_handler(event, context):
    bucket_name = '<your bucket name>'
    list_objects(bucket_name) #list objects in bucket
```

Click **Test** to test the lambda function. Check the execution results.



The objects in the bucket should be printed in the log.

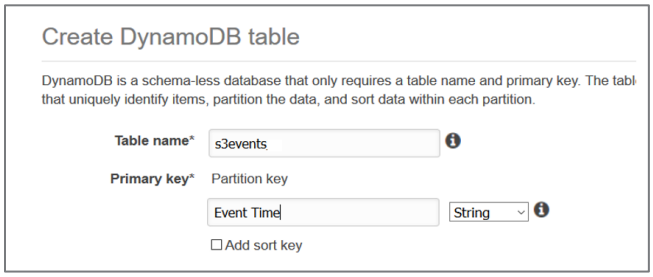
4.4. Handling S3 events in AWS Lambda

The Amazon S3 notification feature enables you to receive notifications when certain events happen in your bucket. Visit the following page to know more about the S3 event types https://docs.aws.amazon.com/AmazonS3/latest/dev/NotificationHowTo.html#supported-notification-event-types.

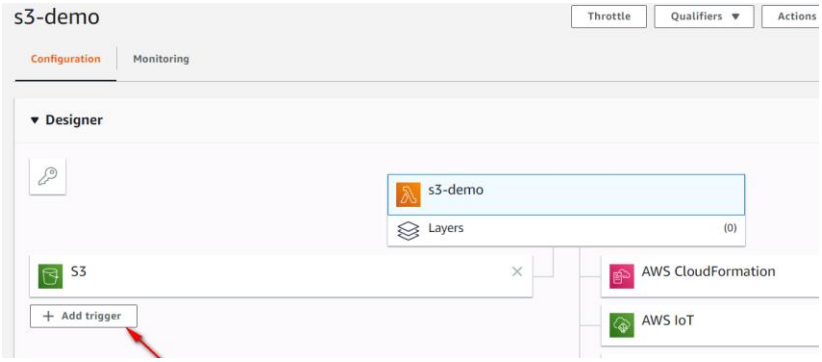
Sample S3 Event Structure:

```
{
 "Records": [
  {
   "eventVersion": "2.1",
   "eventSource": "aws:s3",
   "awsRegion": "us-east-2",
   "eventTime": "2019-09-03T19:37:27.192Z",
   "eventName": "ObjectCreated:Put",
   "userIdentity": {
    "principalId": "AWS:AIDAINPONIXQXHT3IKHL2"
   },
   "requestParameters": {
    "sourceIPAddress": "205.255.255.255"
   },
   "responseElements": {
    "x-amz-request-id": "D82B88E5F771F645",
    "x-amz-id-2": "vlR7PnpV2Ce81l0PRw6jlUpck7Jo5ZsQjryTjKlc5aLWGVHPZLj5NeC6qMa0emYBDXOo6QBU0Wo="
   },
   "s3": {
    "s3SchemaVersion": "1.0",
    "configurationId": "828aa6fc-f7b5-4305-8584-487c791949c1",
    "bucket": {
     "name": "lambda-artifacts-deafc19498e3f2df",
     "ownerIdentity": {
      "principalId": "A3I5XTEXAMAI3E"
     },
     "arn": "arn:aws:s3:::lambda-artifacts-deafc19498e3f2df"
    },
    "object": {
     "key": "b21b84d653bb07b05b1e6b33684dc11b",
     "size": 1305107,
     "eTag": "b21b84d653bb07b05b1e6b33684dc11b",
     "sequencer": "0C0F6F405D6ED209E1"
    }
   }
  }
 ]
}
```

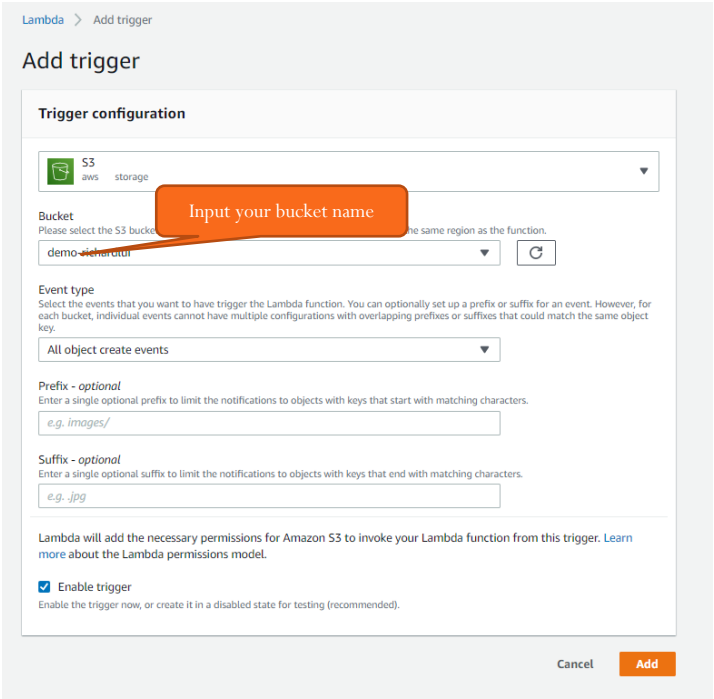
In DynamoDB, create an **s3events** table with partition key "Event Time".



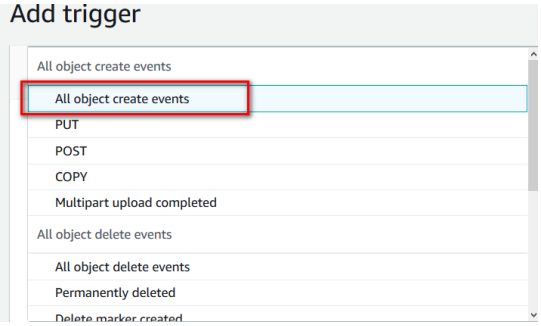
In AWS Lambda, add a trigger from **S3** and select the bucket that you have created.



Select **S3** service and select the bucket that you have created.



For **Event type**, choose "**All object create events**". Click **Add**.

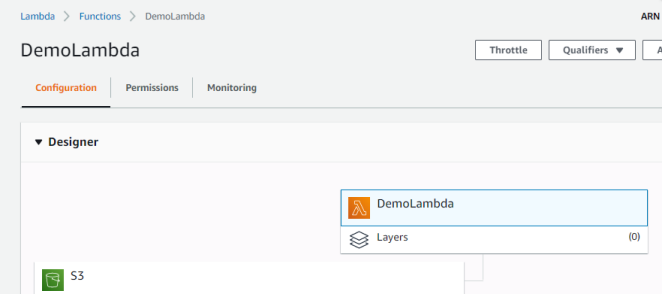


Modify the AWS Lambda Code as follows. **Save** the function.

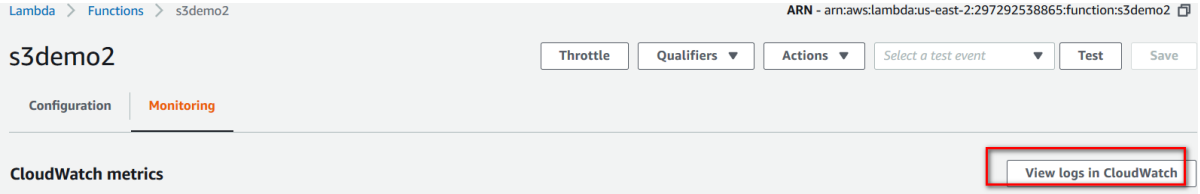
```
import json
import boto3
import urllib

def lambda_handler(event, context):
    r = event['Records'][0]
    print("*** Received Event from S3 ***")
    event_name = r['eventName']
    event_time = r['eventTime']
    bucket = r['s3']['bucket']['name']
    key = urllib.parse.unquote_plus(r['s3']['object']['key'], encoding='utf-8')

    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('s3events')
    table.put_item(
        Item={
            'Event Time':event_time,
            'Event Name':event_name,
            'Bucket':bucket,
            'Key':key
        }
    )
```

Add some celebrities images to your S3 bucket.

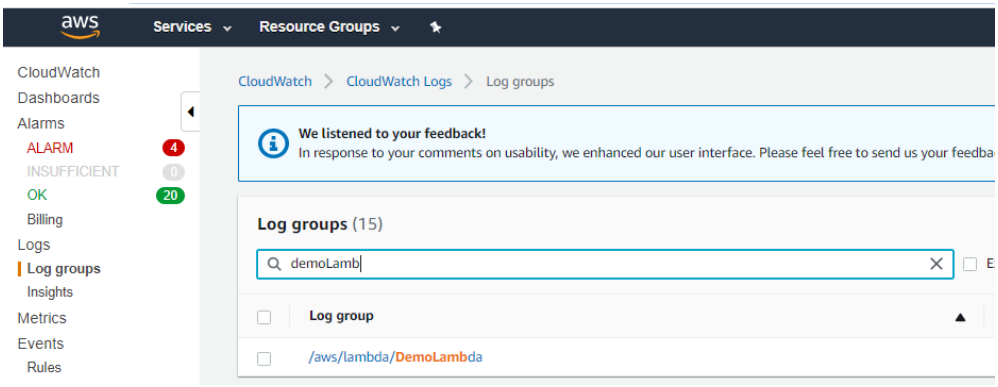
Select the **Monitoring** tab.



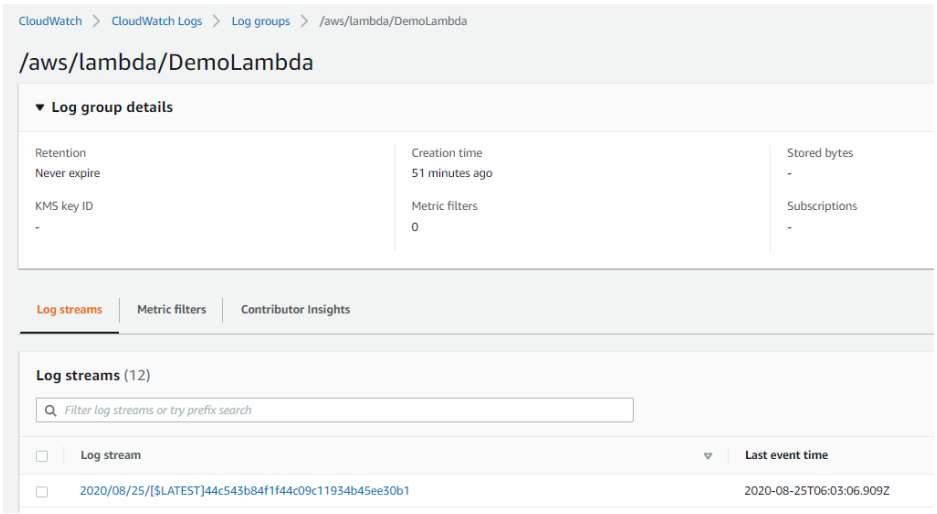
View the latest log in **CloudWatch**.



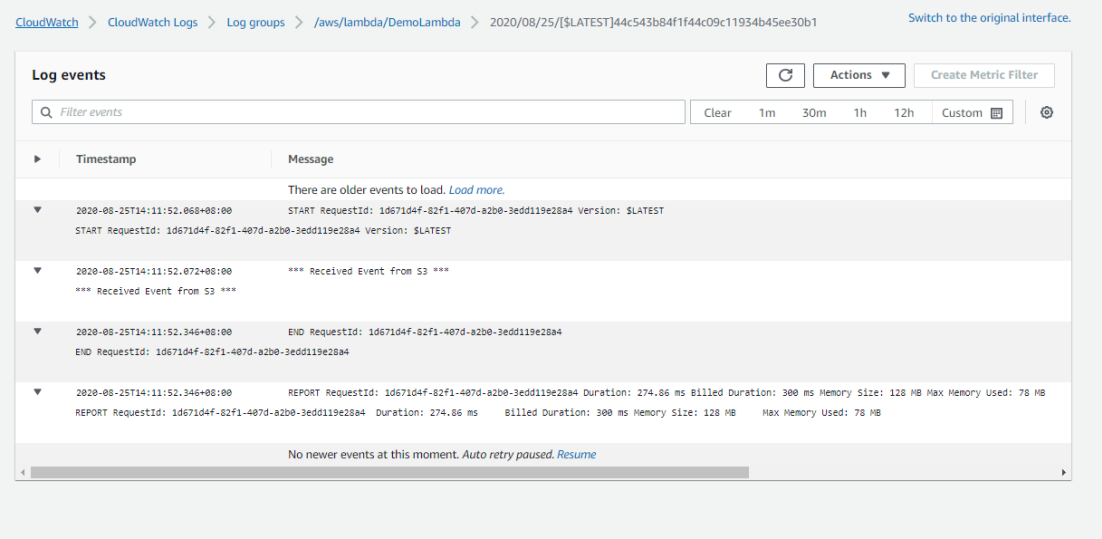
Navigate to the **Logs group** page. Search for the lambda function you have created.



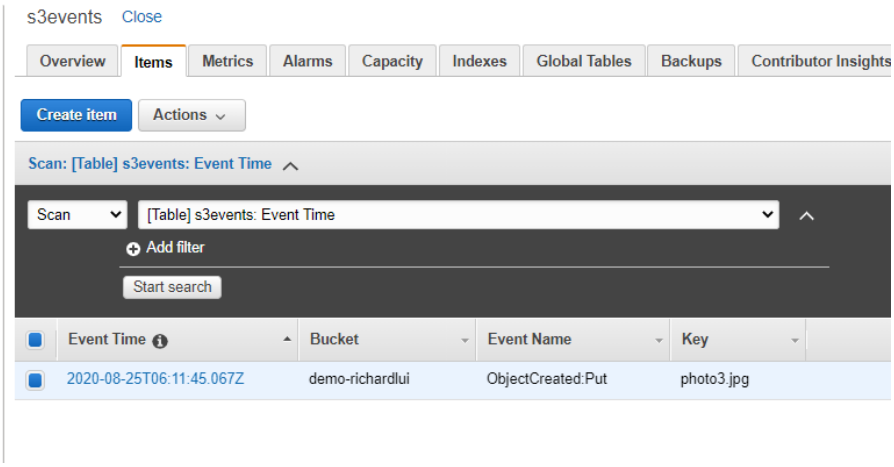
Select the log group and view the latest log streams.



Verify that the lambda function is triggered by s3 events.



View the **s3events** table in DynamoDB.



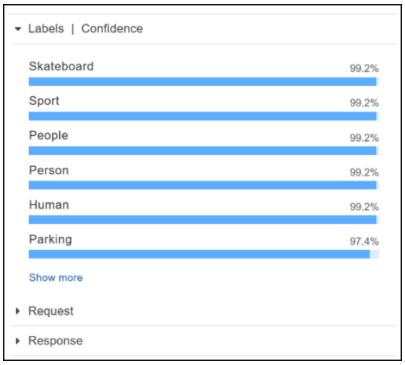
5. AWS Rekognition

5.1. Overview

Amazon Rekognition makes it easy to add image and video analysis to your applications. You just provide an image or video to the Rekognition API, and the service can identify the objects, people, text, scenes, and activities, as well as detect any inappropriate content. Amazon Rekognition also provides highly accurate facial analysis and facial recognition on images and video that you provide. You can detect, analyze, and compare faces for a wide variety of user verification, people counting, and public safety use cases.

https://aws.amazon.com/rekognition

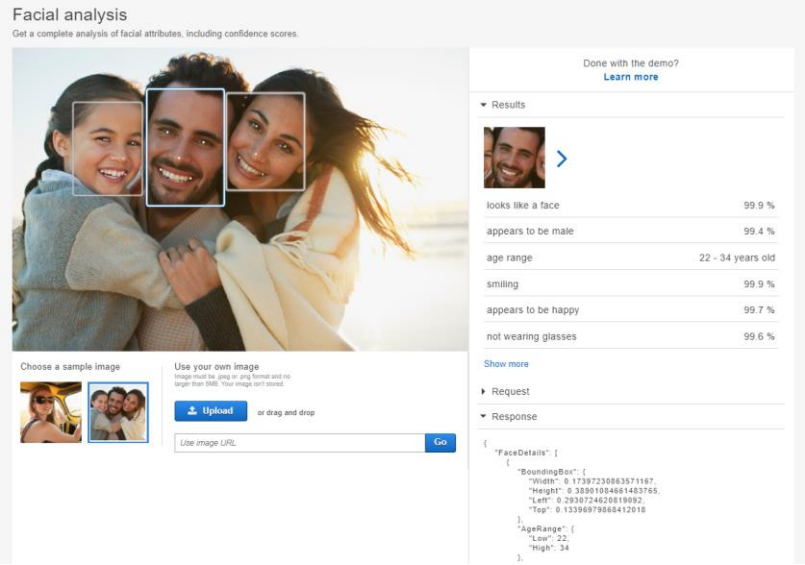
Detect Objects and Scenes in an Image



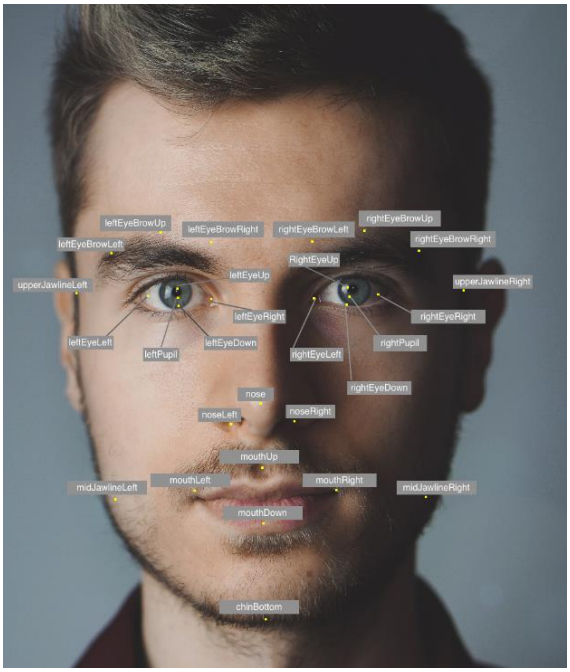
HTTP Request	HTTP Response

<pre>{
"contentString":{
"Attributes":[
"ALL"
],
"Image":{
"S3Object":{
"Bucket":"console-sample-images",
"Name":"skateboard.jpg"
}
}
}
}</pre>
	<pre>{
	"Labels":[
	{
	"Confidence":99.25359344482422,
	"Name":"Skateboard"
	},
	{
	"Confidence":99.25359344482422,
	"Name":"Sport"
	},
	{
	"Confidence":99.24723052978516,
	"Name":"People"
	},
	…
	]}</pre>

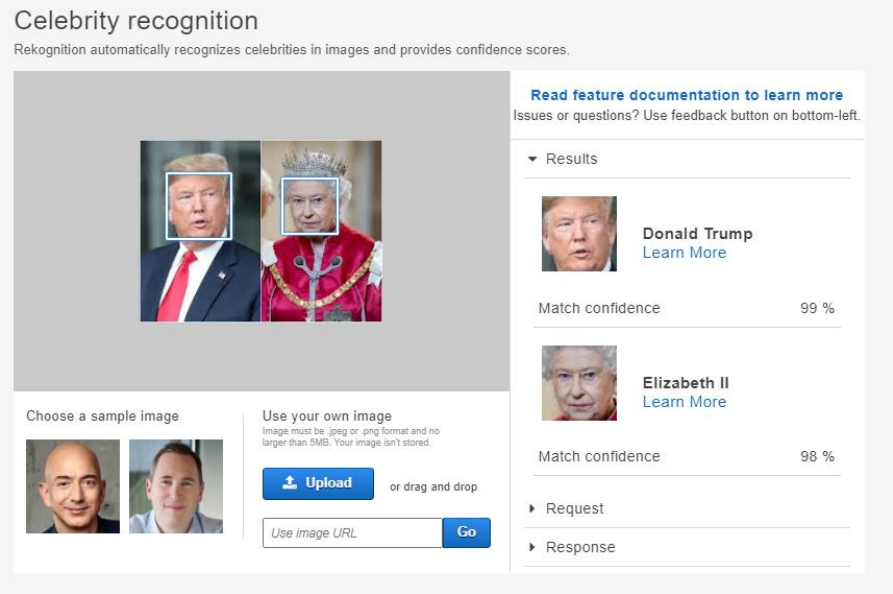
Analyze Faces in an Image



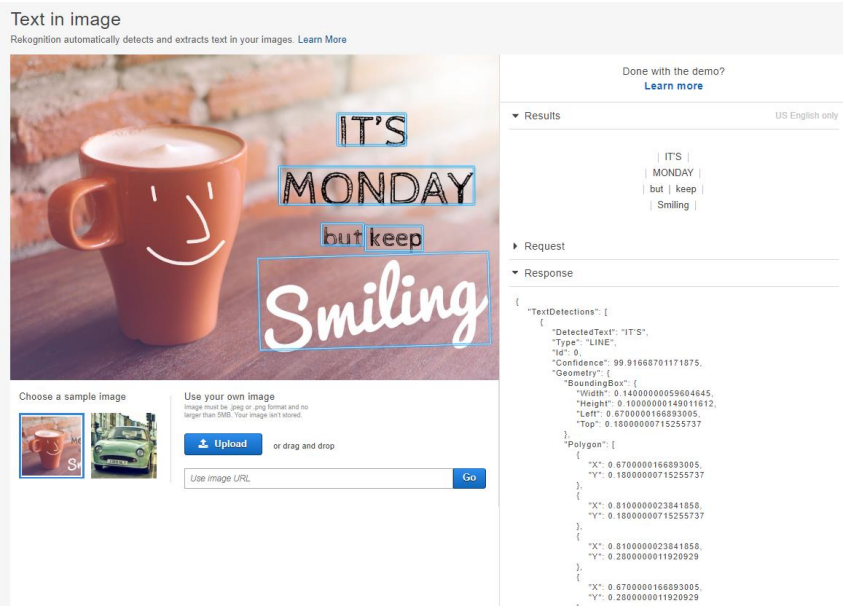
The API can detect up to 100 faces in the image and detect the facial landmarks/attributes in the identified faces.



Celebrity recognition



Text Recognition



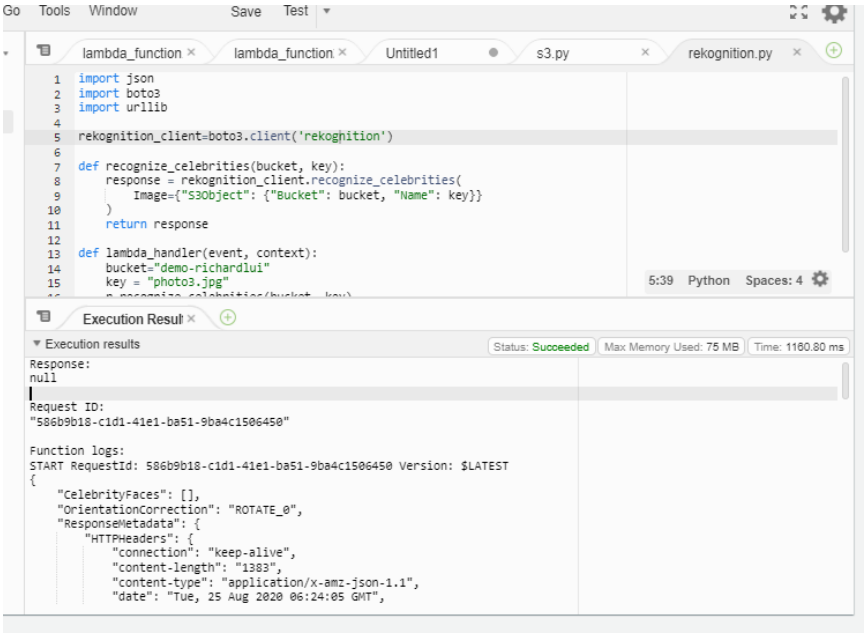
For more details, please refer to https://docs.aws.amazon.com/rekognition/latest/dg/what-is.html.

5.2. Image recognition in Lambda

Modify the AWS Lambda Code as follows. **Save** the function.

```
import json
import boto3
import urllib

rekognition_client=boto3.client('rekognition')

def recognize_celebrities(bucket, key):
    response = rekognition_client.recognize_celebrities(
        Image={"S3Object": {"Bucket": bucket, "Name": key}}
    )
    return response

def lambda_handler(event, context):
    bucket="<your bucket name>"
    key = "<name of your photo with celebrities>"
    r=recognize_celebrities(bucket, key)
    print(json.dumps(r, indent=4, sort_keys=True))
```

Click **Test** to test the lambda function. Check the execution results.



Example JSON returned from **recognize_celebrities** in Amazon Rekognition.


```
{'CelebrityFaces': [{'Face': {'BoundingBox': {'Height': 0.3711340129375458,
                                              'Left': 0.10424710065126419,
                                              'Top': 0.17525772750377655,
                                              'Width': 0.27799227833747864},
                             'Confidence': 100.0,
                             'Landmarks': [{'Type': 'eyeLeft',
                                            'X': 0.18664862215518951,
                                            'Y': 0.31319668889045715},
                                           {'Type': 'eyeRight',
                                            'X': 0.2869987487792969,
                                            'Y': 0.29720112681388855},
                                           {'Type': 'nose',
                                            'X': 0.24276573956012726,
                                            'Y': 0.3709307014942169},
                                           {'Type': 'mouthLeft',
                                            'X': 0.21337087452411652,
                                            'Y': 0.4577915668487549},
                                           {'Type': 'mouthRight',
                                            'X': 0.28056931495666504,
                                            'Y': 0.4482462704181671}],
                             'Pose': {'Pitch': 9.145691871643066,
                                      'Roll': -7.346296310424805,
                                      'Yaw': 0.1183946281671524},
                             'Quality': {'Brightness': 80.88742065429688,
                                         'Sharpness': 78.74752044677734}},
                  'Id': 'I4ma5e',
                  'MatchConfidence': 99.0,
                  'Name': 'Donald Trump',
                  'Urls': ['www.imdb.com/name/nm0874339']},
                 {'Face': {'BoundingBox': {'Height': 0.3144329786300659,
                                           'Left': 0.5830115675926208,
                                           'Top': 0.20618556439876556,
                                           'Width': 0.23552124202251434},
                           'Confidence': 99.98589324951172,
                           'Landmarks': [{'Type': 'eyeLeft',
                                          'X': 0.6541863083839417,
                                          'Y': 0.33158624172210693},
                                         {'Type': 'eyeRight',
                                          'X': 0.7313788533210754,
                                          'Y': 0.32824045419692993},
                                         {'Type': 'nose',
                                          'X': 0.6891570091247559,
                                          'Y': 0.4082064926624298},
                                         {'Type': 'mouthLeft',
                                          'X': 0.6679861545562744,
                                          'Y': 0.4509185254573822},
                                         {'Type': 'mouthRight',
                                          'X': 0.7279360294342041,
                                          'Y': 0.44970467686653137}],
                           'Pose': {'Pitch': -9.11340045928955,
                                    'Roll': -1.725683331489563,
                                    'Yaw': -6.982924938201904},
                           'Quality': {'Brightness': 90.80744171142578,
                                       'Sharpness': 78.74752044677734}},
                  'Id': 'Wo7i18',
                  'MatchConfidence': 98.0,
                  'Name': 'Elizabeth II',
                  'Urls': ['www.imdb.com/name/nm0703070']}],
 'OrientationCorrection': 'ROTATE_0',
 'ResponseMetadata': {'HTTPHeaders': {'connection': 'keep-alive',
                                      'content-length': '1606',
                                      'content-type': 'application/x-amz-json-1.1',
                                      'date': 'Tue, 25 Aug 2020 07:35:49 GMT',
                                      'x-amzn-requestid': '11daaa8b-2d8d-4706-803c-d000e2b9cd37'},
                      'HTTPStatusCode': 200,
                      'RequestId': '11daaa8b-2d8d-4706-803c-d000e2b9cd37',
                      'RetryAttempts': 0},
 'UnrecognizedFaces': []}
```


You may experiment with other functions in AWS rekognition (e.g. detect_faces(), detect_labels).

https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/rekognition.html

6. References

- Using Amazon API Gateway as a proxy for DynamoDB
  - https://aws.amazon.com/blogs/compute/using-amazon-api-gateway-as-a-proxy-for-dynamodb/
- Tutorial: Create a REST API as an Amazon S3 proxy in API Gateway
  - https://docs.aws.amazon.com/apigateway/latest/developerguide/integrating-api-with-aws-services-s3.html
- Tutorial: Viewing Photos in an Amazon S3 Bucket from a Browser
  - https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/s3-example-photos-view.html
- AWS Lambda
  - https://aws.amazon.com/getting-started/tutorials/run-serverless-code/?trk=gs_card
- Create a REST API with Lambda Integrations in Amazon API Gateway
  - https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-getting-started-with-rest-apis.html
- BOTO 3
  - https://boto3.amazonaws.com/v1/documentation/api/latest/index.html
- Creating an AWS IAM Role allowing API Gateway to access DynamoDB
  - https://medium.com/@JCDubs/creating-an-aws-iam-role-allowing-api-gateway-to-access-dynamodb-e32c9286b835