

# Lab: AWS Lambda, S3 and API Gateway

---

1.	Amazon Simple Storage Service (Amazon S3) .....	2
1.1.	Overview of AWS S3.....	2
1.2.	Create the IAM user with the access permission to S3 .....	6
1.3.	Accessing S3 with the AWS Command Line Interface .....	8
1.4.	Accessing S3 in python .....	8
2.	AWS Lambda .....	9
2.1.	Overview .....	9
2.2.	Create your first AWS Lambda Function.....	10
3.	AWS API Gateway .....	20
3.1.	Overview .....	20
3.2.	Create a REST API for Lambda Function .....	20
3.3.	Create a REST API in the API Gateway Console.....	23
3.4.	Deploy Your REST API in the API Gateway Console.....	28
3.5.	Passing HTTP Query String to Lambda .....	29
3.6.	Passing path parameter to Lambda .....	32
4.	Integrating AWS Lambda, S3 and DynamoDB.....	34
4.1.	Creating a role in AWS Identity and Access Management (IAM) .....	34
4.2.	Accessing Dynamo DB in AWS Lambda.....	36
4.3.	S3 integration in AWS Lambda .....	39
4.4.	Handling S3 events in AWS Lambda .....	40
5.	AWS Rekognition.....	45
5.1.	Overview .....	45
5.2.	Image recognition in Lambda.....	48
6.	References .....	50

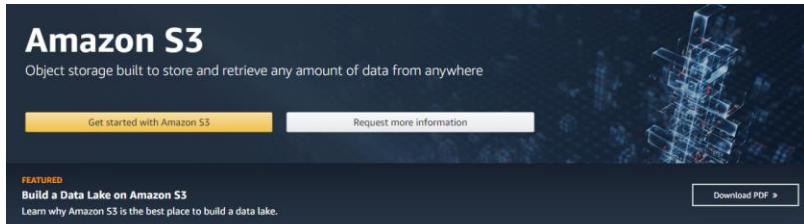
# Lab: AWS Lambda, S3 and API Gateway

## 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>.

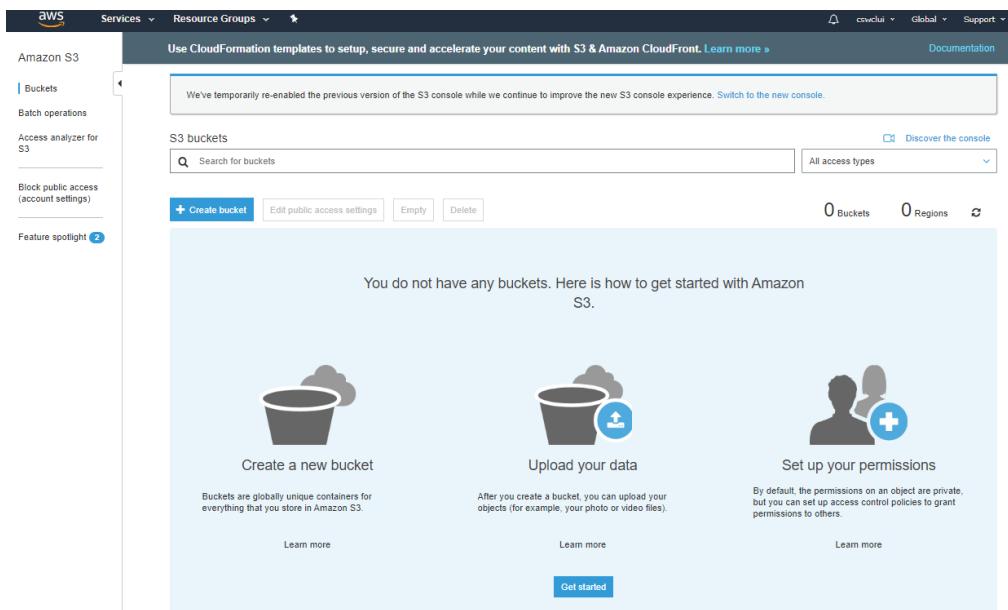


The screenshot shows the official Amazon S3 website. At the top, there's a dark header with the "Amazon S3" logo and a sub-header: "Object storage built to store and retrieve any amount of data from anywhere". Below the header are two buttons: "Get started with Amazon S3" and "Request more information". To the right is a large, abstract blue and white graphic of a 3D grid or network structure. Further down, there's a "FEATURED" section titled "Build a Data Lake on Amazon S3" with a link to "Learn why Amazon S3 is the best place to build a data lake." On the far right, there's a "Download PDF >" button. The main content area contains a paragraph about the service's benefits and a video thumbnail titled "Introduction to Amazon S3".

Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance. This means customers of all sizes and industries can use it to store and protect any amount of data for a range of use cases, such as websites, mobile applications, backup and restore, archive, enterprise applications, IoT devices, and big data analytics. Amazon S3 provides easy-to-use management features so you can organize your data and configure finely-tuned access controls to meet your specific business, organizational, and compliance requirements. Amazon S3 is designed for 99.999999999% (11 9%) of durability, and stores data for millions of applications for companies all around the world.



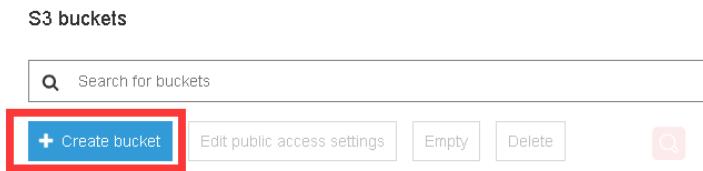
Navigate to S3 console at [s3.console.aws.amazon.com](https://s3.console.aws.amazon.com).



The screenshot shows the AWS S3 console. The left sidebar has a "Buckets" section with a "Create bucket" button and other options like "Edit public access settings", "Empty", and "Delete". It also includes sections for "Batch operations", "Access analyzer for S3", and "Block public access (account settings)". The main content area has a message: "We've temporarily re-enabled the previous version of the S3 console while we continue to improve the new S3 console experience. Switch to the new console." Below this, there's a "S3 buckets" section with a search bar and a "Discover the console" link. The central part of the screen says "You do not have any buckets. Here is how to get started with Amazon S3." It features three cards: "Create a new bucket" (with a bucket icon), "Upload your data" (with a bucket and upload icon), and "Set up your permissions" (with a user icon). Each card has a "Learn more" link and a "Get started" button.

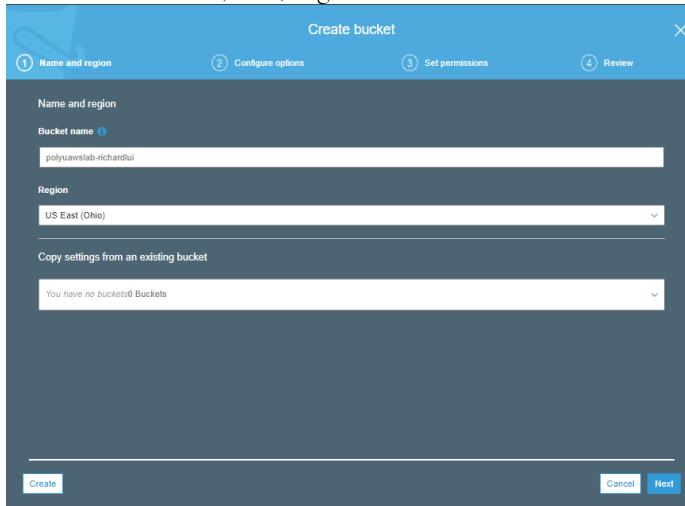
# Lab: AWS Lambda, S3 and API Gateway

Click the “Create bucket” button.

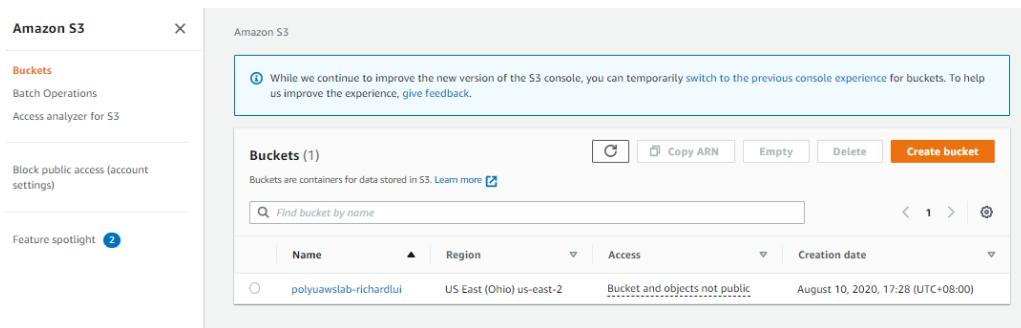


Create a bucket polyuawslab-[your name] (replace [your name] with your name). Note that the name should not contain uppercase characters and you should assign a unique bucket name (over all AWS S3 bucket names)).

Choose the US East (Ohio) region.



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



# Lab: AWS Lambda, S3 and API Gateway

Click the created bucket.

The screenshot shows the AWS S3 Bucket Overview page for the bucket 'polyuawslab-richardlui'. The top navigation bar includes tabs for Overview, Properties, Permissions, Management, and Access points. Below the tabs are buttons for Upload, Create folder, Download, and Actions. The status bar indicates 'US East (Ohio)' and a refresh icon. The main content area displays a message: 'This bucket is empty. Upload new objects to get started.' It features three large icons: 'Upload an object' (a bucket with a file), 'Set object properties' (two user profiles with a plus sign), and 'Set object permissions' (three database cylinders with a gear). Below each icon are descriptive text snippets and 'Learn more' links. A prominent blue 'Get started' button is at the bottom.

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

The screenshot shows the AWS S3 Bucket Overview page for the bucket 'polyuawslab-richardlui'. The top navigation bar includes tabs for Overview, Properties, Permissions, and Management. Below the tabs is a search bar with placeholder text 'Type a prefix and press Enter to search. Press ESC to clear.' A red box highlights the 'Upload' button. Other buttons include '+ Create folder', 'Download', and 'Actions'. The main content area shows a modal window titled 'Upload' with four steps: 1. Select files, 2. Set permissions, 3. Set properties, 4. Review. The 'Select files' step shows two files selected: 'photo1.jpg' (4.1 KB) and 'photo2.jpg' (447.5 KB). The 'Review' step shows the same files with 'Next' and 'Upload' buttons at the bottom.

# Lab: AWS Lambda, S3 and API Gateway

---

Verify that the files are uploaded.

The screenshot shows the Amazon S3 console interface. The top navigation bar includes 'Amazon S3' and the path 'polyuawslab-richardlui'. Below the navigation is a header with tabs: 'Overview' (selected), 'Properties', 'Permissions', 'Management', and 'Access points'. A search bar at the top right contains the placeholder 'Type a prefix and press Enter to search. Press ESC to clear.' Below the header are buttons for 'Upload', '+ Create folder', 'Download', and 'Actions ▾'. To the right of these buttons is the location 'US East (Ohio)' and a refresh icon. The main area displays a table of file contents. The columns are 'Name ▾', 'Last modified ▾', 'Size ▾', and 'Storage class ▾'. There are two files listed:

Name	Last modified	Size	Storage class
photo1.jpg	Aug 10, 2020 5:40:13 PM GMT+0800	4.1 KB	Standard
photo2.jpg	Aug 10, 2020 5:40:14 PM GMT+0800	447.5 KB	Standard

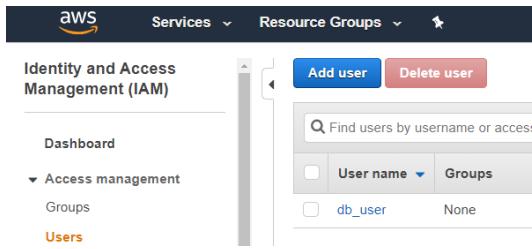
At the bottom of the table area, it says 'Viewing 1 to 2'.

# Lab: AWS Lambda, S3 and API Gateway

## 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**.

The screenshot shows the 'Set user details' step of the IAM 'Add user' wizard. It includes fields for 'User name' (set to 's3\_user'), a link to 'Add another user', and sections for 'Select AWS access type' and 'Access type' (with 'Programmatic access' selected). At the bottom, there are 'Cancel' and 'Next: Permissions' buttons.

Set user details

You can add multiple users at once with the same access type and permissions. [Learn more](#)

User name\*  [Add another user](#)

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. [Learn more](#)

Access type\*  **Programmatic access**  
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.

**AWS Management Console access**  
Enables a **password** that allows users to sign-in to the AWS Management Console.

\* Required [Cancel](#) [Next: Permissions](#)

# Lab: AWS Lambda, S3 and API Gateway

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

The screenshot shows the 'Set permissions' section of the AWS IAM console. There are three main options: 'Add user to group', 'Copy permissions from existing user', and 'Attach existing policies directly'. The 'Attach existing policies directly' button is highlighted with a blue border. Below these buttons is a 'Create policy' button. Underneath is a search bar with 'Filter policies' and a dropdown menu set to 's3'. A table lists available policies:

	Policy name	Type	Used as
<input type="checkbox"/>	AmazonDMSRedshiftS3Role	AWS managed	None
<input checked="" type="checkbox"/>	AmazonS3FullAccess	AWS managed	None
<input type="checkbox"/>	AmazonS3ReadOnlyAccess	AWS managed	None
<input type="checkbox"/>	QuickSightAccessForS3StorageManagementAnalyticsReadOnly	AWS managed	None

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

The screenshot shows the 'AmazonS3FullAccess' policy summary. It provides a brief description: 'Provides full access to all buckets via the AWS Management Console.' Below is a 'Policy summary' tab and a 'JSON' tab. The JSON tab is currently active, displaying the policy's JSON structure:

```
1 * {
2     "Version": "2012-10-17",
3     "Statement": [
4         {
5             "Effect": "Allow",
6             "Action": "s3:*",
7             "Resource": "*"
8         }
]
```

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.

The screenshot shows the 'Add user' success message: 'Success' with a checkmark. It states: 'You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.' Below is a 'Download .csv' button. A table lists the user details:

User	Access key ID	Secret access key
s3_user	AKIAUKOAEPPYRMLYIKCB	***** Show

# Lab: AWS Lambda, S3 and API Gateway

---

## 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>

# Lab: AWS Lambda, S3 and API Gateway

## 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/>

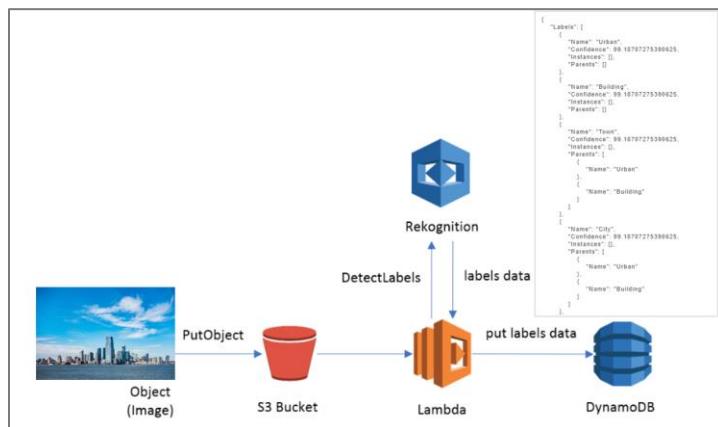


Examples of serverless applications with lambda

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



Example 2: Label detection is triggered when images are uploaded



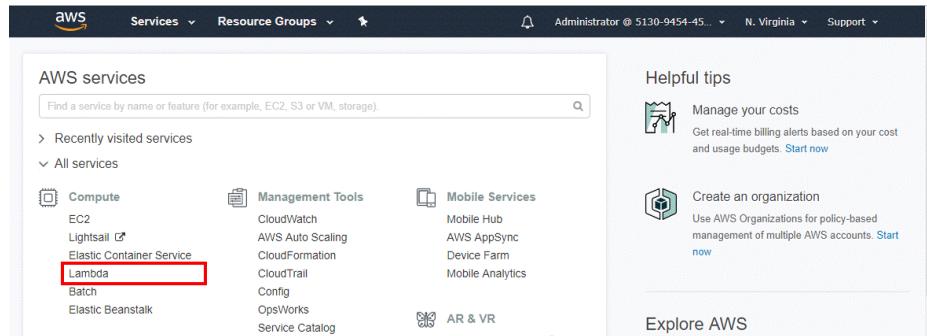
```
[{"Label": [{"Name": "Urban", "Confidence": 99.18707275390025, "Basename": "", "Parent": ""}, {"Name": "Building", "Confidence": 99.18707275390025, "Basename": "", "Parent": ""}, {"Name": "Tree", "Confidence": 99.18707275390025, "Basename": "", "Parent": ""}, {"Name": "Urban", "Confidence": 99.18707275390025, "Basename": "", "Parent": ""}, {"Name": "Building", "Confidence": 99.18707275390025, "Basename": "", "Parent": ""}, {"Name": "Car", "Confidence": 99.18707275390025, "Basename": "", "Parent": ""}, {"Name": "Urban", "Confidence": 99.18707275390025, "Basename": "", "Parent": ""}, {"Name": "Building", "Confidence": 99.18707275390025, "Basename": "", "Parent": ""}], "Timestamp": 1583111111111}
```

# Lab: AWS Lambda, S3 and API Gateway

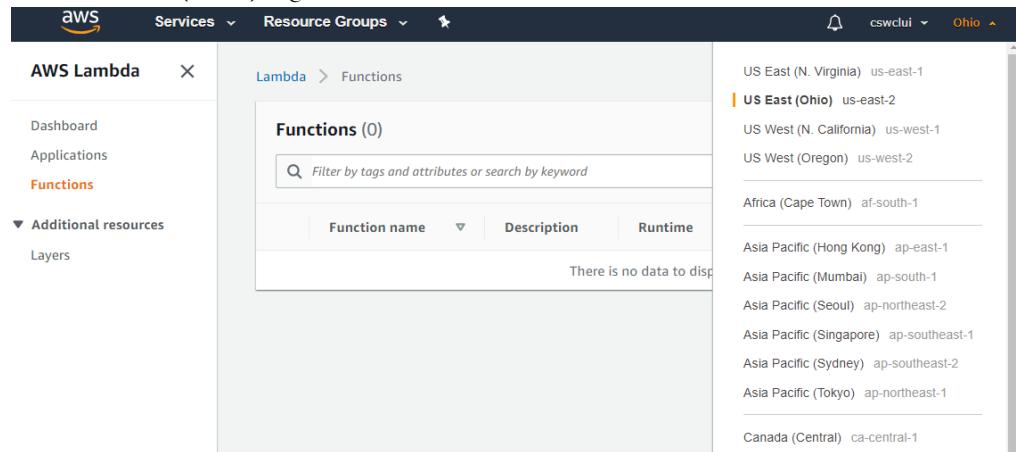
## 2.2. Create your first AWS Lambda Function.

Step 1: Enter the Lambda Console

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



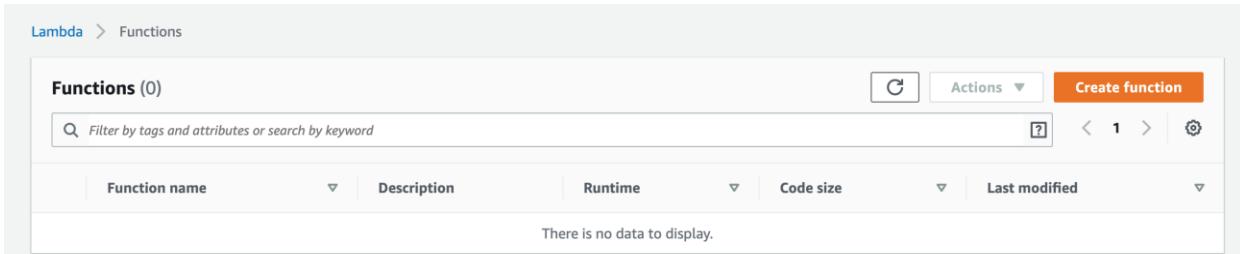
Select **US East (Ohio)** region.



Step 2: Create function

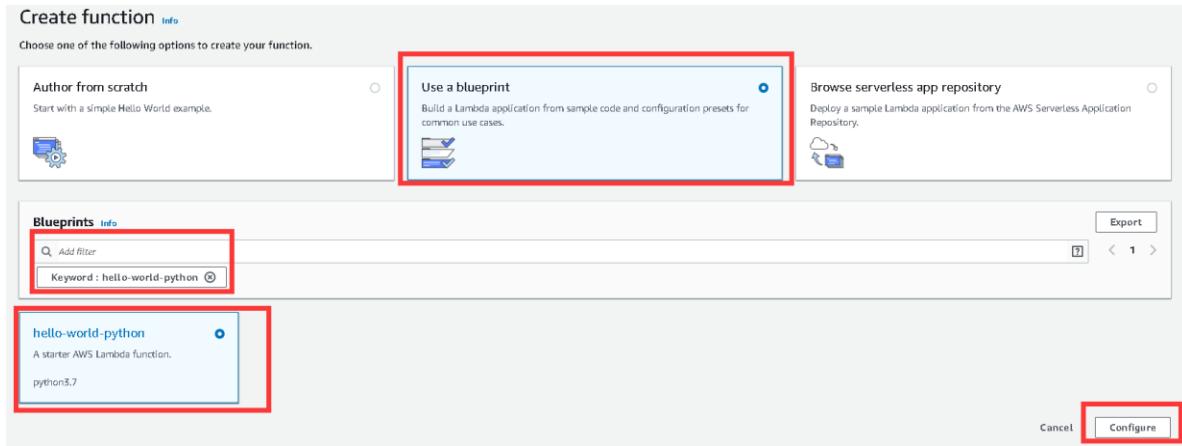
Blueprints provide example code to do some minimal processing. Most blueprints process events from specific event sources, such as Amazon S3, DynamoDB, or a custom application.

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



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

# Lab: AWS Lambda, S3 and API Gateway



## 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**.

The screenshot shows the 'Basic information' configuration page for the Lambda function 'hello-world-python'. The 'Function name' field contains 'hello-world-python'. Under 'Execution role', the radio button for 'Create a new role from AWS policy templates' is selected. A note below states: 'Role creation might take a few minutes. Please do not delete the role or edit the trust or permissions policies in this role.' The 'Role name' field is filled with 'lambda\_basic\_execution'. The 'Policy templates - optional' field is empty. At the bottom right of the page, there is a 'Next Step' button.

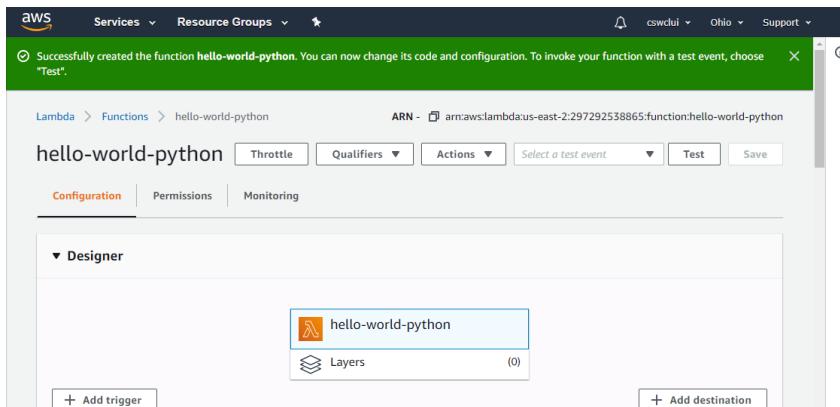
# Lab: AWS Lambda, S3 and API Gateway

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

The screenshot shows the AWS Lambda function creation interface. At the top, it says "Runtime Python 3.7". Below that is a code editor containing the following Python code:

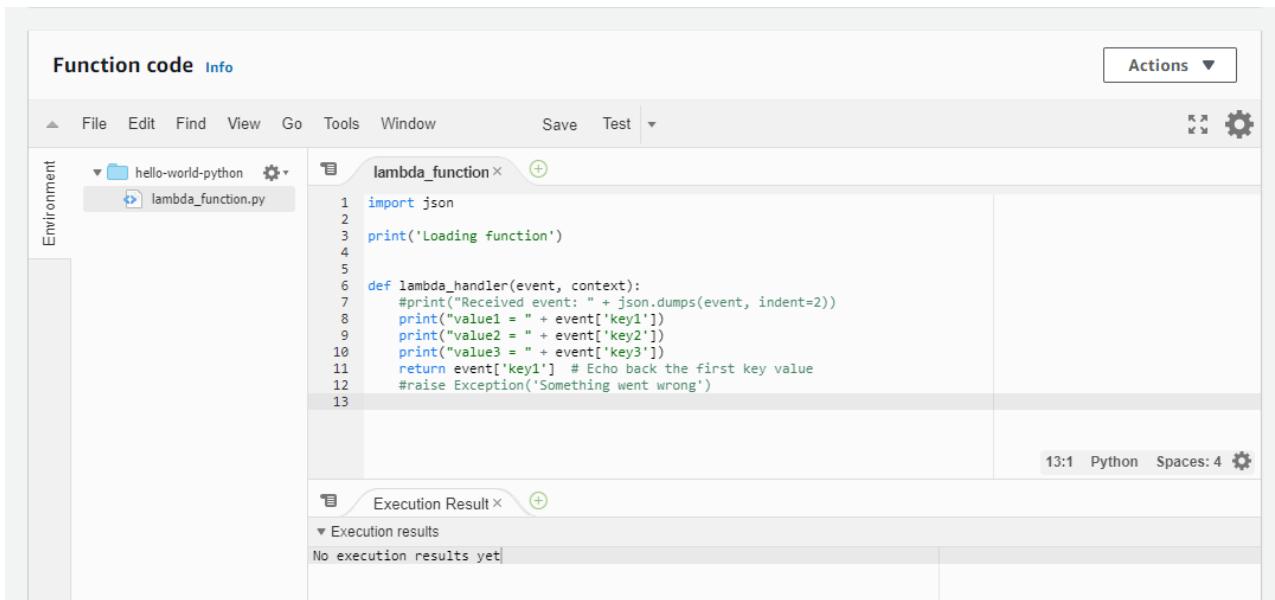
```
1 import json
2
3 print('Loading function')
4
5
6 def lambda_handler(event, context):
7     #print("Received event: " + json.dumps(event, indent=2))
8     print("value1 = " + event['key1'])
9     print("value2 = " + event['key2'])
10    print("value3 = " + event['key3'])
11    return event['key1'] # Echo back the first key value
12    #raise Exception('Something went wrong')
13
```

At the bottom right of the code editor is a "Create function" button.



# Lab: AWS Lambda, S3 and API Gateway

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

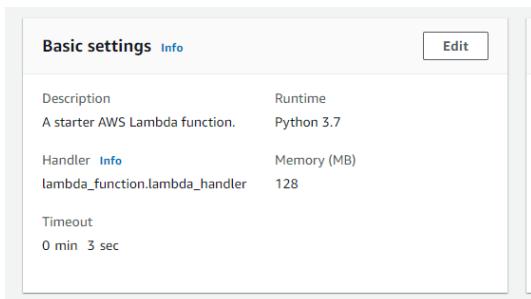


The screenshot shows the AWS Lambda Function code editor interface. At the top, there's a toolbar with File, Edit, Find, View, Go, Tools, Window, Save, and Test buttons. To the right of the toolbar is an "Actions" dropdown and a gear icon. On the left, there's a sidebar labeled "Environment" with a "hello-world-python" folder containing a "lambda\_function.py" file. The main area displays the Python code for the lambda function:

```
1 import json
2
3 print('Loading function')
4
5
6 def lambda_handler(event, context):
7     #print("Received event: " + json.dumps(event, indent=2))
8     print("value1 = " + event['key1'])
9     print("value2 = " + event['key2'])
10    print("value3 = " + event['key3'])
11    return event['key1'] # Echo back the first key value
12    #raise Exception('Something went wrong')
13
```

Below the code editor is a "Execution Result" panel with a "Execution results" section that says "No execution results yet". In the bottom right corner of the editor area, it shows "13:1 Python Spaces: 4".

Scroll down the page to view the settings.



# Lab: AWS Lambda, S3 and API Gateway

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

**Edit basic settings**

**Basic settings** [Info](#)

Description - optional  
A starter AWS Lambda function.

Runtime  
Python 3.7

Handler [Info](#)  
lambda\_function.lambda\_handler

Memory (MB)  
Your function is allocated CPU proportional to the memory configured.  
128 MB

Timeout  
0 min 3 sec

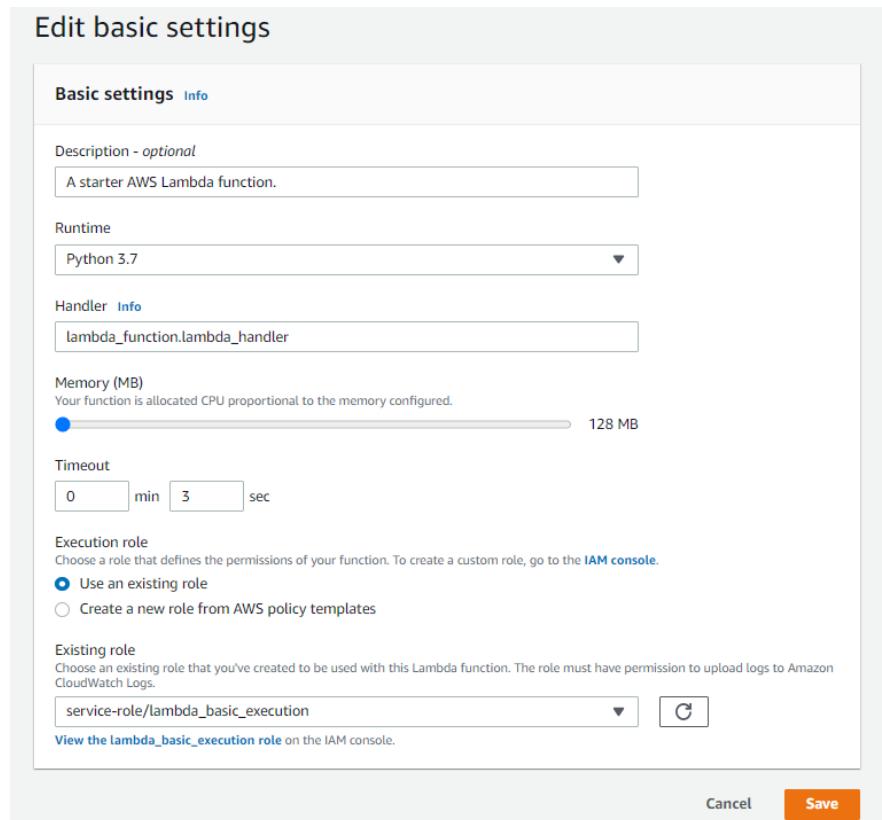
Execution role  
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

Use an existing role  
 Create a new role from AWS policy templates

Existing role  
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

service-role/lambda\_basic\_execution [View the lambda\\_basic\\_execution role](#) on the IAM console.

[Cancel](#) [Save](#)



**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**.

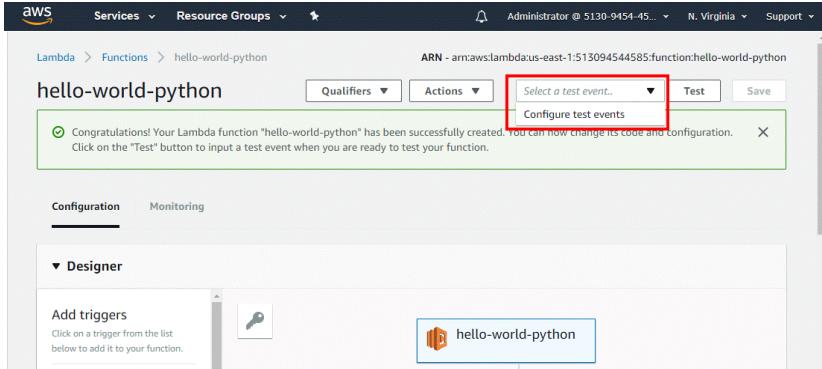
We will accept the default settings. Click **Cancel**.

## 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.

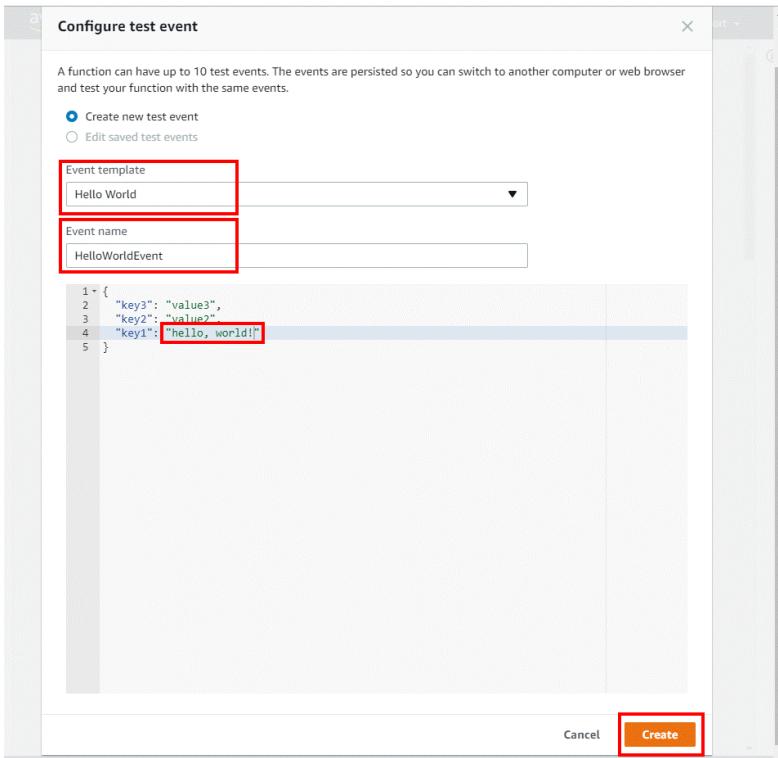
Select **Configure Test Event** from the drop-down menu called "Select a test event...".

# Lab: AWS Lambda, S3 and API Gateway



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

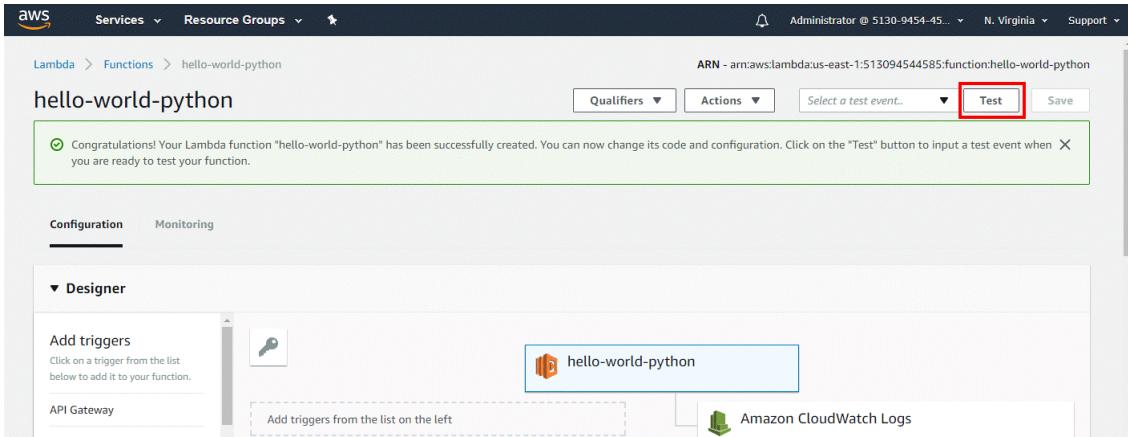
- Choose Hello World from the Sample event template list from the Input test event page.
- Type in an event name like *HelloWorldEvent*.
- You can change the values in the sample JSON, but don't change the event structure. For this tutorial, replace value1 with *hello, world!*.



Select **Create**.

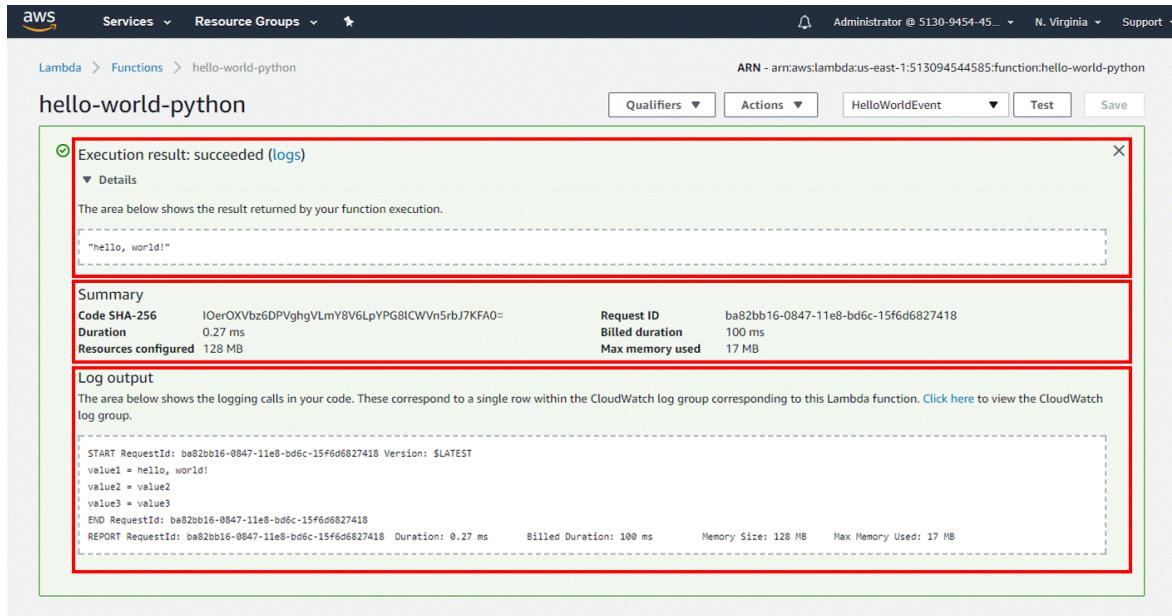
# Lab: AWS Lambda, S3 and API Gateway

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.



# Lab: AWS Lambda, S3 and API Gateway

## Step 5: Monitor Your Metrics

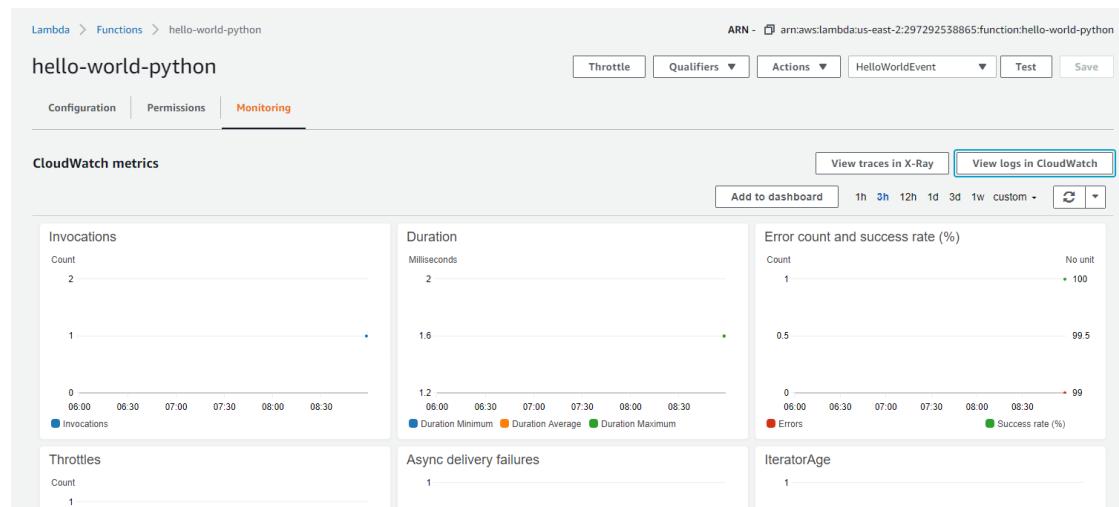
AWS Lambda automatically monitors Lambda functions and reports metrics through Amazon CloudWatch. To help you monitor your code as it executes, Lambda automatically tracks the number of requests, the latency per request, and the number of requests resulting in an error and publishes the associated metrics.

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.

The screenshot shows the AWS Lambda Test interface for the 'hello-world-python' function. The 'Test' button is highlighted with a red box. The execution result is shown as succeeded, with a log entry: '["hello, world!"]'. The summary section includes details like Code SHA-256, Duration (0.27 ms), and Resources configured (128 MB). The log output section shows CloudWatch logs with entries for REQUEST, REPORT, and END Request. Below the test interface, the 'Monitoring' tab is selected in the main Lambda function page, also highlighted with a red box.

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*.



# Lab: AWS Lambda, S3 and API Gateway

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.

The screenshot shows the AWS CloudWatch Log Groups interface. At the top, there's a breadcrumb navigation: CloudWatch > CloudWatch Logs > Log groups > /aws/lambda/hello-world-python. To the right of the breadcrumb is a link to "Switch to the original interface". Below the breadcrumb is the log group name: /aws/lambda/hello-world-python. On the right side of the header are three buttons: "Actions", "View in Logs Insights", and "Search log group". Underneath the header, there's a section titled "Log group details" with a collapse arrow. This section contains several key metrics: Retention (Never expire), Creation time (7 minutes ago), Stored bytes (-), KMS key ID (-), Metric filters (0), Subscriptions (-), ARN (arn:aws:logs:us-east-2:297292538865:log-group:/aws/lambda/hello-world-python:\*), and Contributor Insights rules (-). Below the "Log group details" section is a horizontal navigation bar with three tabs: "Log streams" (which is selected and highlighted in orange), "Metric filters", and "Contributor Insights". The main content area below the navigation bar is titled "Log streams (1)". It includes a search bar with a placeholder "Filter log streams" and a timestamp "Last event time" set to "2020-08-10T08:58:01.635Z". A single log stream is listed: "2020/08/10/[\$LATEST]b5ba0b2d82fe4aef833ca78f11609ee8".

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

The screenshot shows the Amazon CloudWatch landing page. The top features the "Amazon CloudWatch" logo and a sub-headline: "Observability of your AWS resources and applications on AWS and on-premises". Below the headline is a prominent orange "Get Started with CloudWatch" button. Further down, there's a "TECH TALK" section with a video thumbnail titled "Simply and Seamlessly Set Up Secure File Transfers to Amazon S3 Over SFTP and Other Protocols". The thumbnail includes a "Register now" button. The main content area describes Amazon CloudWatch as a monitoring and observability service for DevOps engineers, developers, site reliability engineers (SREs), and IT managers. It highlights its ability to collect monitoring and operational data from various sources like logs, metrics, and events, providing a unified view of operational health. A video player on the right shows a thumbnail for a video titled "Amazon CloudWatch: Complete visibility of your cloud resources and applications (2:02)".

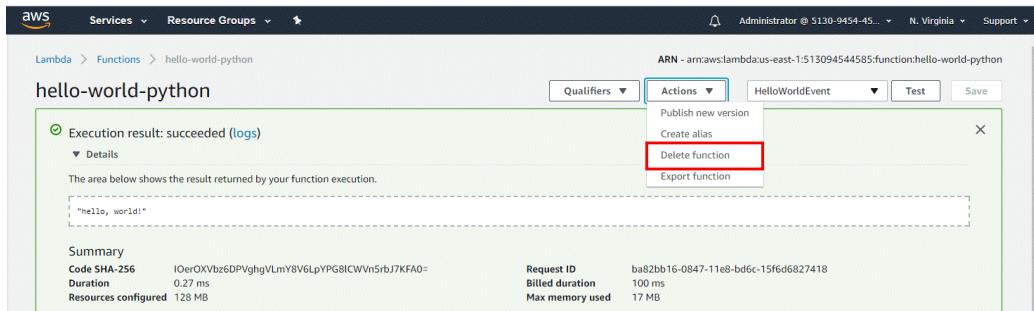
Amazon CloudWatch is a monitoring and observability service built for DevOps engineers, developers, site reliability engineers (SREs), and IT managers. CloudWatch provides you with data and actionable insights to monitor your applications, respond to system-wide performance changes, optimize resource 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.



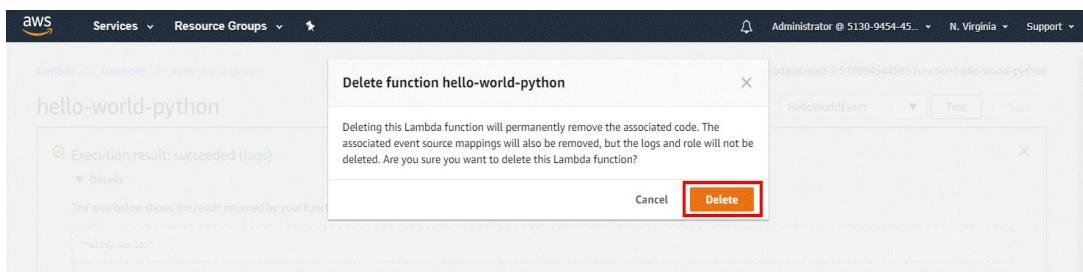
# Lab: AWS Lambda, S3 and API Gateway

## 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**.



# Lab: AWS Lambda, S3 and API Gateway

## 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.



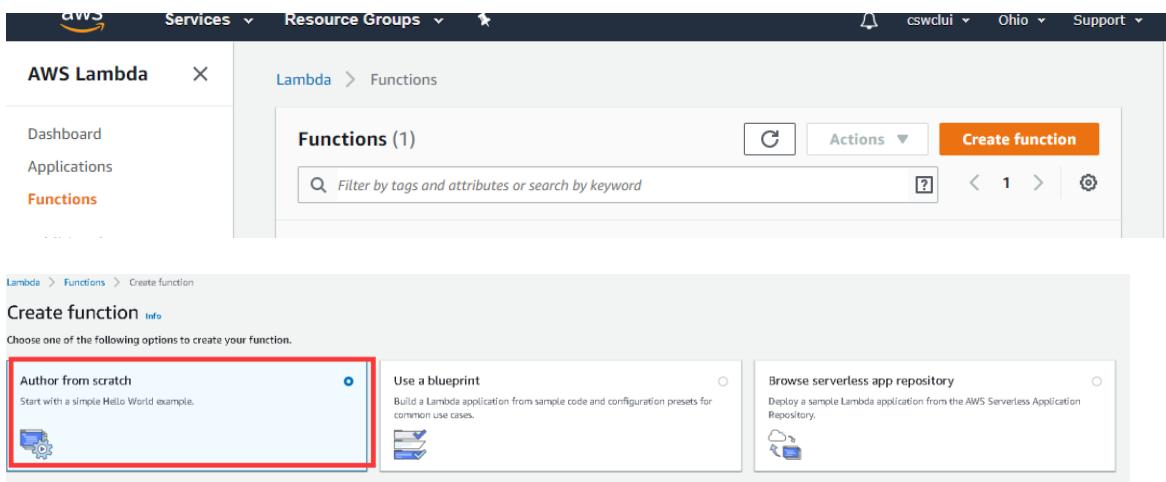
Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. With a few clicks in the Management Console, you can create REST and WebSocket APIs that act as a “front door” for applications to access data, business 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**.

# Lab: AWS Lambda, S3 and API Gateway

**Basic information**

Function name Enter a short name that describes the purpose of your function.  
my-function  

Runtime Info  
Choose the language to use to write your function.  
Python 3.6  

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 Enter a name for your new role.  
myrole-function  

Policy templates - optional Info  
Choose one or more policy templates.

Activate | Cancel | Go to Settings Create function

## Lab: AWS Lambda, S3 and API Gateway

---

By default, the lambda handler code is as follows.

Code entry type: Edit code inline      Runtime: Python 3.6

```
import json
def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

# Lab: AWS Lambda, S3 and API Gateway

## 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.

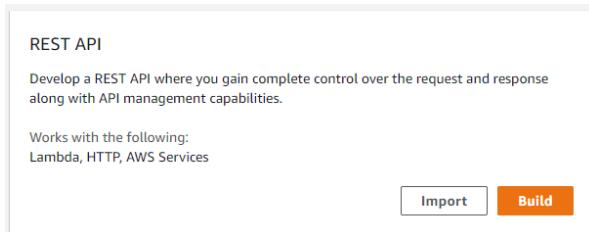
From the **Services** menu, choose **API Gateway** to go to the API Gateway console. Under **Choose an API type**, choose **REST API**, and click **Build**.

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**.

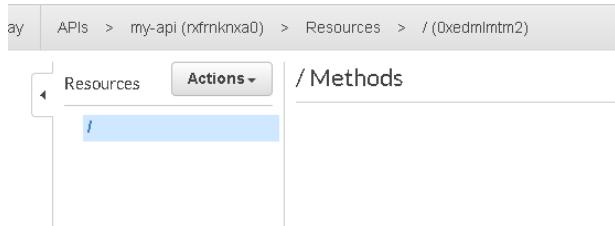
3. Choose **Create API**.



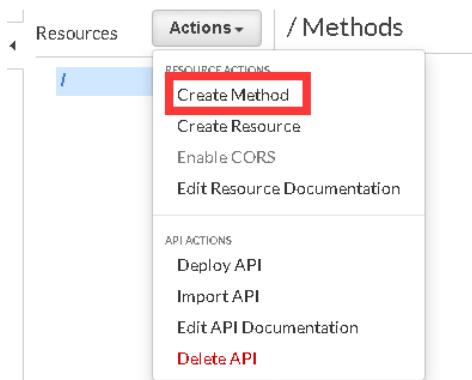
The screenshot shows the 'Create new API' settings page. At the top, it says 'Choose the protocol' with options for 'REST' (selected) and 'WebSocket'. Below that, it says 'Create new API' and provides a note about what a REST API is. It has three radio button options: 'New API' (selected), 'Import from Swagger or Open API 3', and 'Example API'. The 'Settings' section allows entering an 'API name\*' (set to 'my-api'), a 'Description', and selecting an 'Endpoint Type' (set to 'Regional'). A note at the bottom left says '\* Required'. A 'Create API' button is at the bottom right.

## Lab: AWS Lambda, S3 and API Gateway

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**.



## Lab: AWS Lambda, S3 and API Gateway

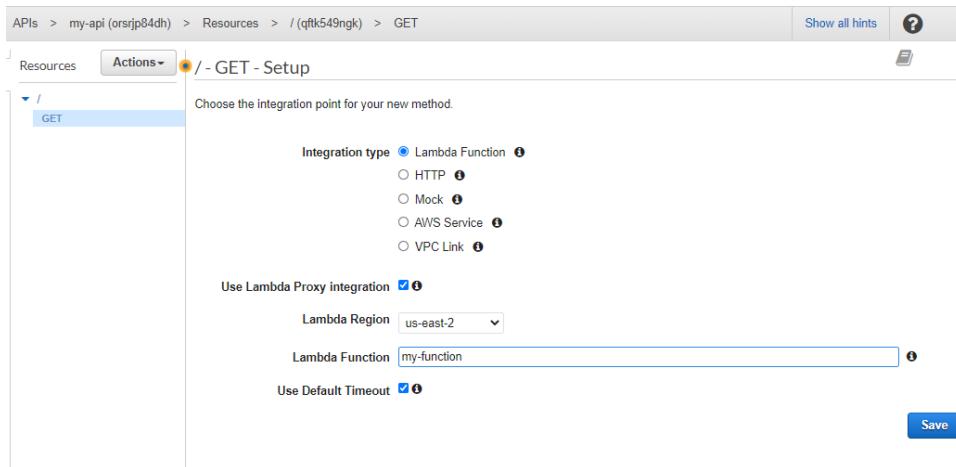
Under the resource name (/), you'll see a dropdown menu. Choose **GET** and then choose the checkmark icon to save your choice.



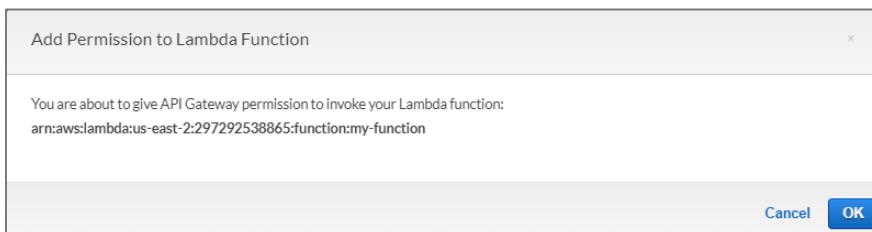
In the / – **GET** – Setup pane:

- 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.

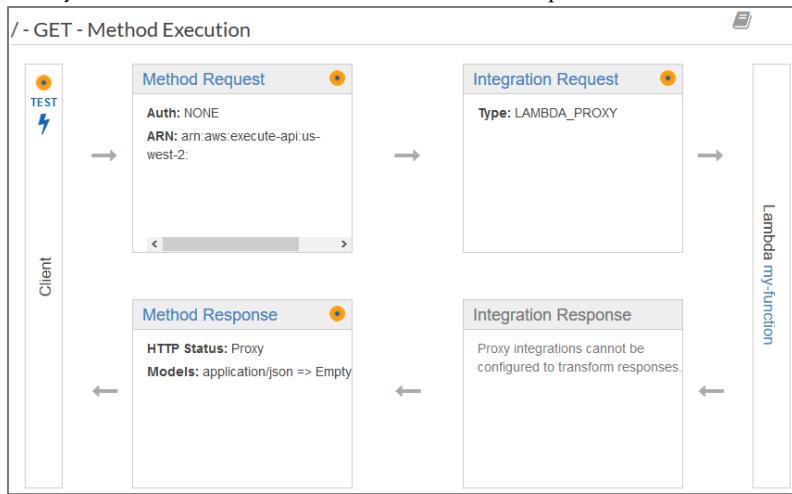


When the **Add Permission to Lambda Function** popup appears (saying "**You are about to give API Gateway permission to invoke your Lambda function...**"), choose **OK** to grant API Gateway that permission.



# Lab: AWS Lambda, S3 and API Gateway

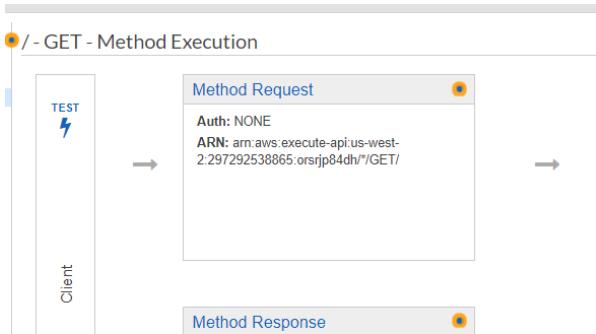
Now you'll see a **/ – GET – Method Execution** pane:



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**.



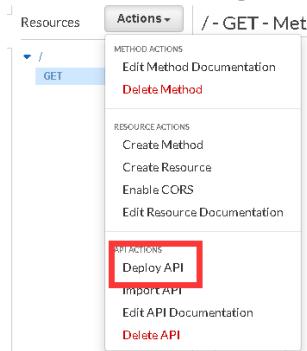
# Lab: AWS Lambda, S3 and API Gateway

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

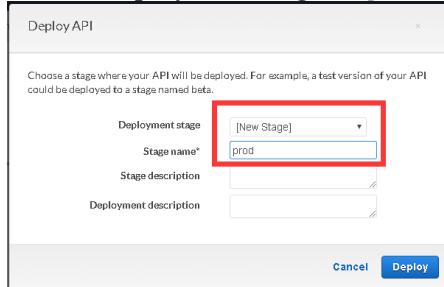
# Lab: AWS Lambda, S3 and API Gateway

## 3.4. Deploy Your REST API in the API Gateway Console

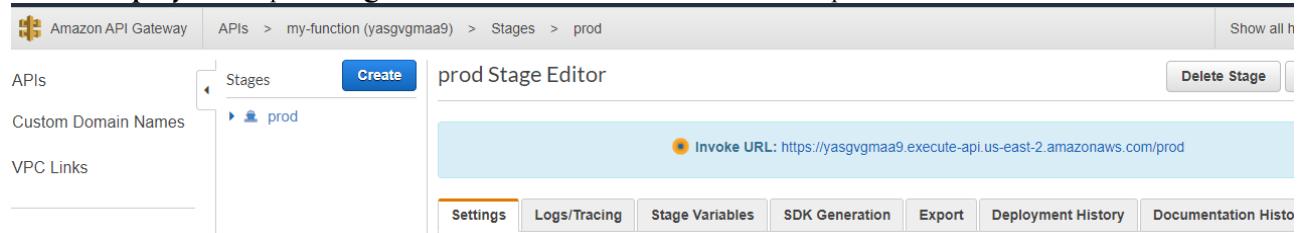
From the **Actions** dropdown menu, choose **Deploy API**.



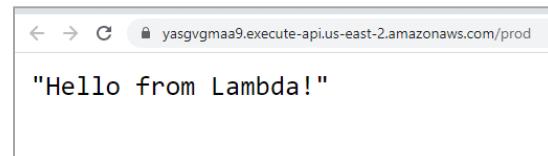
From the **Deployment stage** dropdown menu, choose [**New Stage**]. For **Stage name**, enter prod.



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.



# Lab: AWS Lambda, S3 and API Gateway

## 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_oSXZNQwQ100TzL4cimZo3g==",  
        "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'])  
    }
```

# Lab: AWS Lambda, S3 and API Gateway

Click **Save**.

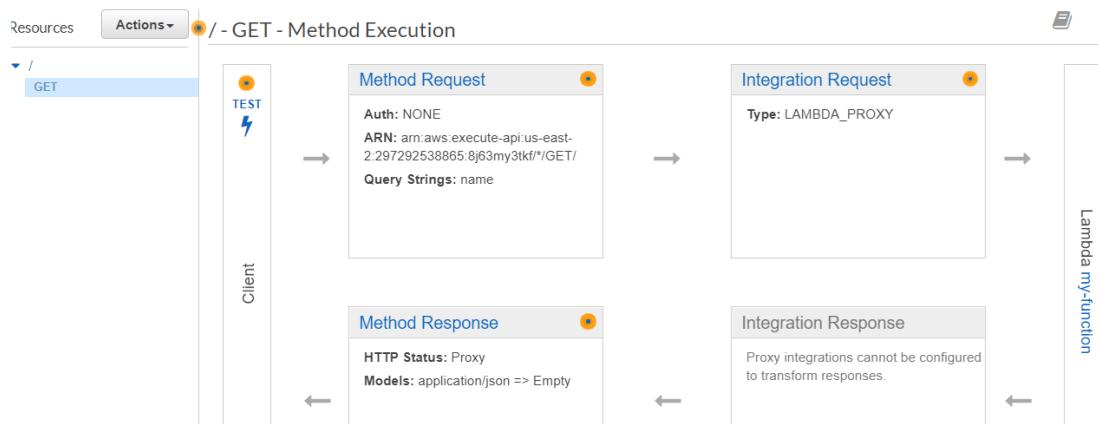
The screenshot shows the AWS Lambda function code editor. The left sidebar shows the environment with a folder named "my-function" containing a file named "lambda\_function.py". The main area displays the Python code for the lambda function:

```
1 import json
2
3 def lambda_handler(event, context):
4     return {
5         'statusCode': 200,
6         'body': json.dumps(event['queryStringParameters'])
7     }
8
```

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

The screenshot shows the AWS API Gateway Method Execution configuration for a GET method. Under "URL Query String Parameters", a parameter named "name" is listed with the "Required" checkbox checked. This field is highlighted with a red box.

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



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

# Lab: AWS Lambda, S3 and API Gateway

The screenshot shows the AWS Lambda Function Test interface. At the top, it says "Method Execution / - GET - Method Test". Below that, it says "Make a test call to your method with the provided input". Under "Path", it says "No path parameters exist for this resource. You can define path parameters by using the syntax {myPathParam} in a resource path.". Under "Query Strings", there is a text input field containing "name=John". Under "Headers", it says "No header parameters exist for this method. You can add them via Method Request.". Under "Stage Variables", it says "No stage variables exist for this method.". Under "Client Certificate", it says "No client certificates have been generated.". Under "Request Body", it says "Request Body is not supported for GET methods.". On the right side, there are sections for "Request", "Status", "Latency", "Response Body", "Response Headers", and "Logs". The "Request" section shows "Request: /?name=John", "Status: 200", and "Latency: 46 ms". The "Response Body" section shows a JSON object: { "name": "John" }. The "Response Headers" section shows "X-Amzn-Trace-Id: Root=1-5d90ff43-375d847a4e53a2e44249c94d; Sampled=0". The "Logs" section shows execution logs for the request.

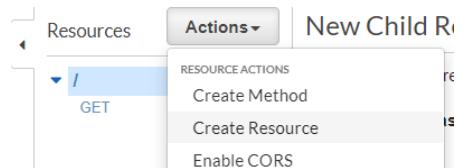
Deploy the API and access the endpoint with different name parameters in query string.

The first screenshot shows a browser developer tools Network tab with a request to "yasgvgmaa9.execute-api.us-east-2.amazonaws.com/prod?name=john". The request body is a JSON object: { "name": "john" }. The second screenshot shows another request to "yasgvgmaa9.execute-api.us-east-2.amazonaws.com/prod?name=Mary" with the request body: { "name": "Mary" }.

# Lab: AWS Lambda, S3 and API Gateway

## 3.6. Passing path parameter to Lambda

Select the root resource `/`.



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

This screenshot shows the 'New Child Resource' dialog. In the 'Resource Name' field, 'person' is entered. In the 'Resource Path' field, '/{personID}' is entered. A note below the path field explains that you can add path parameters using brackets. There are checkboxes for 'Configure as proxy resource' and 'Enable API Gateway CORS'. At the bottom are 'Create Resource' and 'Cancel' buttons.

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

This screenshot shows the ' /{personID} - GET - Setup' configuration dialog. Under 'Integration type', 'Lambda Function' is selected. 'Use Lambda Proxy integration' is checked. 'Lambda Region' is set to 'us-east-2' and 'Lambda Function' is set to 'my-function'. A 'Save' button is at the bottom right.

# Lab: AWS Lambda, S3 and API Gateway

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**.

The screenshot shows the AWS Lambda Test API interface. On the left, the resources list shows a single GET method for the path '/{personID}'. The 'Actions' dropdown is set to 'Method Execution'. The test input fields show a path parameter '{personID}' set to 'mary' and a query string parameter 'lang=Eng&mode=1'. The response details on the right indicate a successful 200 status with a latency of 214 ms. The response body is displayed as JSON:

```
{
  "Path": {
    "personID": "mary"
  },
  "QueryString": {
    "mode": "1",
    "lang": "Eng"
  }
}
```

The response headers section shows an X-Amzn-Trace-Id header with the value 'Root=1-5f33c0bc-f8ab7b49d49e45339d2a614e;Sampled=0'.

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

The screenshot shows a browser developer tools Network tab. A request is listed with the URL `yasgvgmaa9.execute-api.us-east-2.amazonaws.com/prod/peter/?lang=eng&mode=2`. The request body is shown as JSON:

```
{
  "Path": {
    "personID": "peter"
  },
  "QueryString": {
    "lang": "eng",
    "mode": "2"
  }
}
```

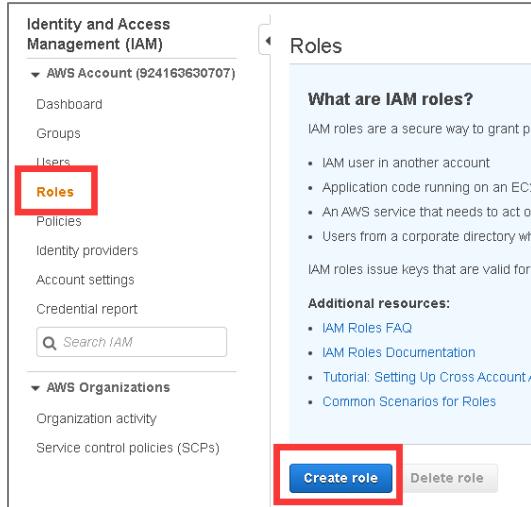
# Lab: AWS Lambda, S3 and API Gateway

## 4. Integrating AWS Lambda, S3 and DynamoDB

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

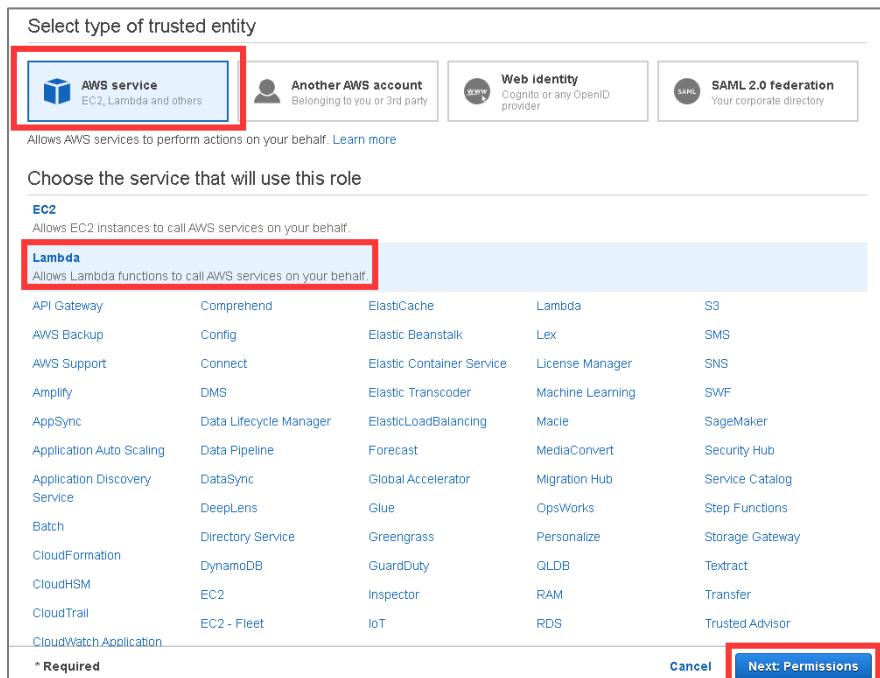
We first create a role inside IAM to allow lambda function to access S3 and Amazon recognition service.

Navigate to IAM console at [console.aws.amazon.com/iam](https://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**.

# Lab: AWS Lambda, S3 and API Gateway

Filter policies ▾  Showing 1 result

Policy name	Used as	Description
<input checked="" type="checkbox"/> AWSLambdaFullAccess	Permissions policy (1)	Provides full access to Lambda, S3, Dyn...

Filter policies ▾  Showing 1 result

Policy name	Used as	Description
<input checked="" type="checkbox"/> AmazonRekognitionFullAccess	Permissions policy (1)	Access to all Amazon Rekognition APIs

Click blue “Next: Review” button

Filter policies ▾  Showing 1 result

Policy name	Used as	Description
<input checked="" type="checkbox"/> AmazonRekognitionFullAccess	Permissions policy (1)	Access to all Amazon Rekognition APIs

▼ Set permissions boundary

Set a permissions boundary to control the maximum permissions this role can have. This is an advanced feature used to delegate permission management to others. [Learn more](#)

Create role without a permissions boundary

Use a permissions boundary to control the maximum role permissions

\* Required      Cancel      Previous      **Next: Tags**

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

Create role

Review

Provide the required information below and review this role before you create it.

Role name\*  Use alphanumeric and '+', '@', '-' characters. Maximum 64 characters.

Role description  Maximum 1000 characters. Use alphanumeric and '+', '@', '-' characters.

Trusted entities AWS service: lambda.amazonaws.com

Policies  AWSLambdaFullAccess  AmazonRekognitionFullAccess

Permissions boundary Permissions boundary is not set

No tags were added.

\* Required      Cancel      Previous      **Create role**

# Lab: AWS Lambda, S3 and API Gateway

## 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. **LambdaFullAccessWithRekognition**).

Lambda > Functions > Create function

### Create function Info

Choose one of the following options to create your function.

- Author from scratch   
Start with a simple Hello World example.
- Use a blueprint   
Build a Lambda application from sample code and configuration presets for common use cases.
- Browse serverless app repository   
Deploy a sample Lambda application from the AWS Serverless Application Repository.

#### Basic information

Function name  
Enter a name that describes the purpose of your function.  
  
Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info  
Choose the language to use to write your function.

#### 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

Existing role  
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.  
  

[View the LambdaFullAccessWithRekognition role on the IAM console](#).

[Cancel](#) [Create function](#)

## Lab: AWS Lambda, S3 and API Gateway

---

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.

# Lab: AWS Lambda, S3 and API Gateway

Check that the record is created in Dynamo DB.

The screenshot shows the AWS DynamoDB console with the 'users' table selected. The 'Items' tab is active. A search bar at the top says 'Scan [Table] users: username'. Below it, there's a 'Start search' button and a 'Scan' dropdown. The results table has columns: username, account\_type, age, first\_name, last\_name, hobbies. One item is listed: janedoe, standard\_user, 25, Jane, Doe, [ { "S": "badminton" }, { "S": "foodball" }, { "S": "singing" } ].

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()

The screenshot shows the AWS Lambda function editor for a function named 'lambda\_function'. The code in 'lambda\_function.py' is:

```
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'  
        }  
    )
```

The 'Execution Result' tab shows the output of a test run:

```
Execution results  
null  
  
Request ID:  
"2261c535-26b5-4009-a1c0-b7fed9031b36"  
  
Function logs:  
START RequestId: 2261c535-26b5-4009-a1c0-b7fed9031b36 Version: $LATEST  
{"hobbies": ["badminton", "foodball", "singing"], "username": "janedoe", "last_name": "Doe", "account_type": "standard_user", "username": "Johnson", "age": Decimal('18')}, {"gender": "male", "username": "Tom"}, {"gender": "female", "username": "May"}, {"hobbies": ["badminton", "singing"]}, {"gender": "male", "username": "Joseph"}  
END RequestId: 2261c535-26b5-4009-a1c0-b7fed9031b36 Duration: 1395.58 ms Billed Duration: 1400 ms Memory Size: 128 MB
```

# Lab: AWS Lambda, S3 and API Gateway

## 4.3. S3 integration in AWS Lambda

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

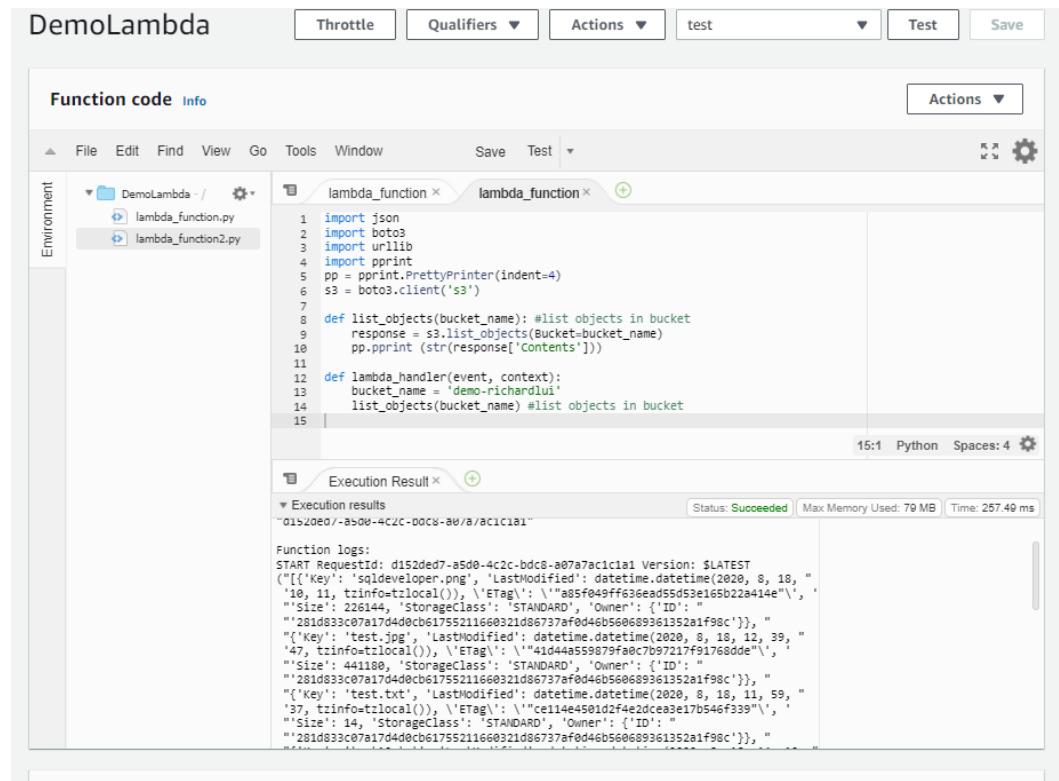
```
import json
import boto3
import urllib
import pprint
pp = pprint.PrettyPrinter(indent=4)
s3 = boto3.client('s3')

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.

# Lab: AWS Lambda, S3 and API Gateway

---

## 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": "A3I5XTEXAMA13E"  
          },  
          "arn": "arn:aws:s3:::lambda-artifacts-deafc19498e3f2df"  
        },  
        "object": {  
          "key": "b21b84d653bb07b05b1e6b33684dc11b",  
          "size": 1305107,  
          "eTag": "b21b84d653bb07b05b1e6b33684dc11b",  
          "sequencer": "0C0F6F405D6ED209E1"  
        }  
      }  
    }  
  ]  
}
```

# Lab: AWS Lambda, S3 and API Gateway

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

Create DynamoDB table

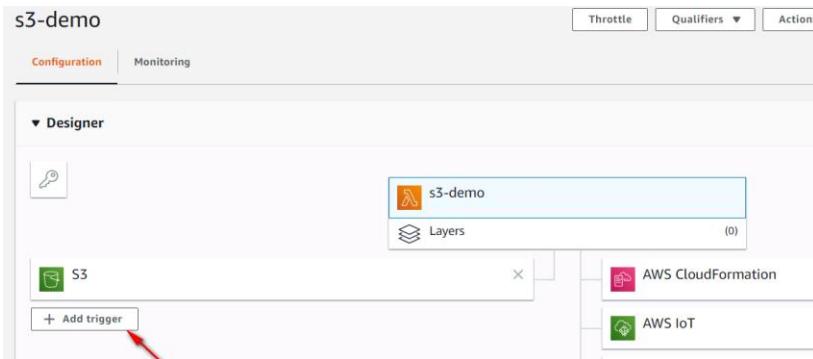
DynamoDB is a schema-less database that only requires a table name and primary key. The table uniquely identifies items, partitions the data, and sorts data within each partition.

Table name\*  (i)

Primary key\* Partition key  
 String (i)

Add sort key

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.

Lambda > Add trigger

Add trigger

Trigger configuration

S3 aws storage

Bucket Please select the S3 bucket to receive notifications. The bucket must be in the same region as the function.  
demo-vihardul (i)

Event type Select the events that you want to have trigger the Lambda function. You can optionally set up a prefix or suffix for an event. However, for each bucket, individual events cannot have multiple configurations with overlapping prefixes or suffixes that could match the same object key.

All object create events

Prefix - optional Enter a single optional prefix to limit the notifications to objects with keys that start with matching characters.  
e.g. images/

Suffix - optional Enter a single optional suffix to limit the notifications to objects with keys that end with matching characters.  
e.g. jpg

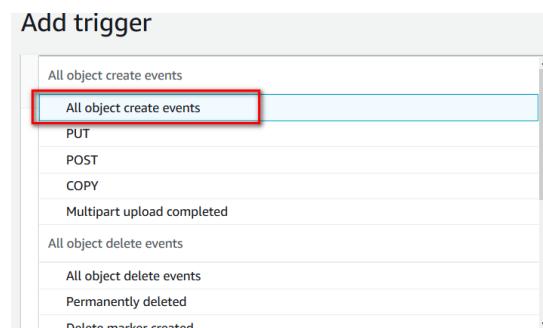
Lambda will add the necessary permissions for Amazon S3 to invoke your Lambda function from this trigger. Learn more about the Lambda permissions model.

Enable trigger  
Enable the trigger now, or create it in a disabled state for testing (recommended).

Cancel Add

# Lab: AWS Lambda, S3 and API Gateway

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

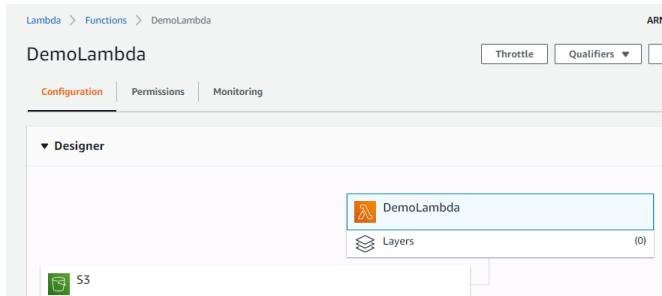
def lambda_handler(event, context):
    r = event['Records'][0]
    print("*** Received Event from S3 ***")
    event_name = r['eventName']
    event_time = r['eventTime']
    bucket = r['s3']['bucket']['name']
    key = urllib.parse.unquote_plus(r['s3']['object']['key'], encoding='utf-8')

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

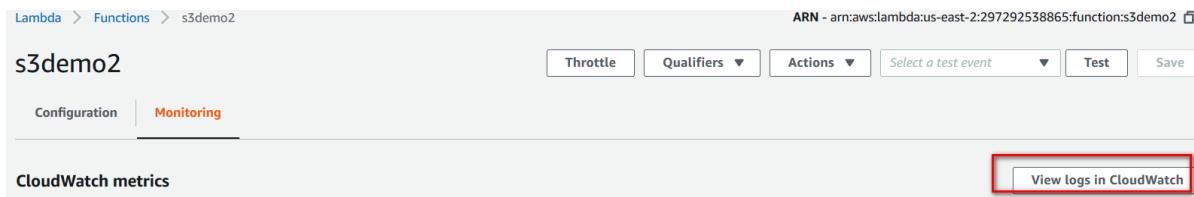
Add some celebrities images to your S3 bucket.

# Lab: AWS Lambda, S3 and API Gateway

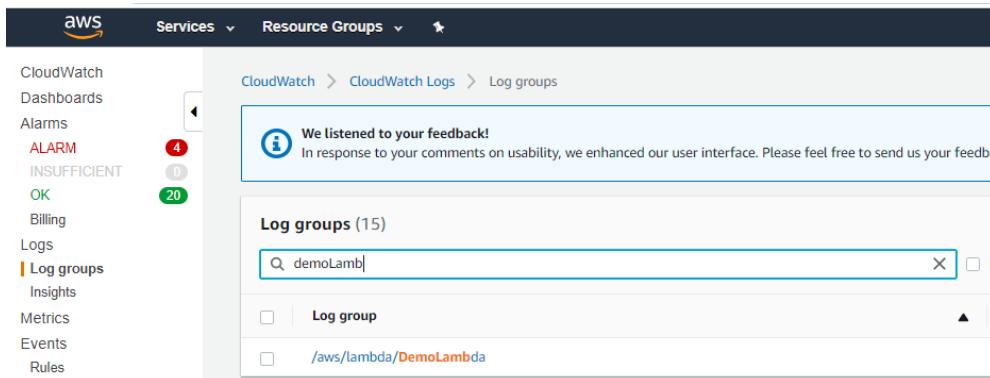
Select the **Monitoring** tab.



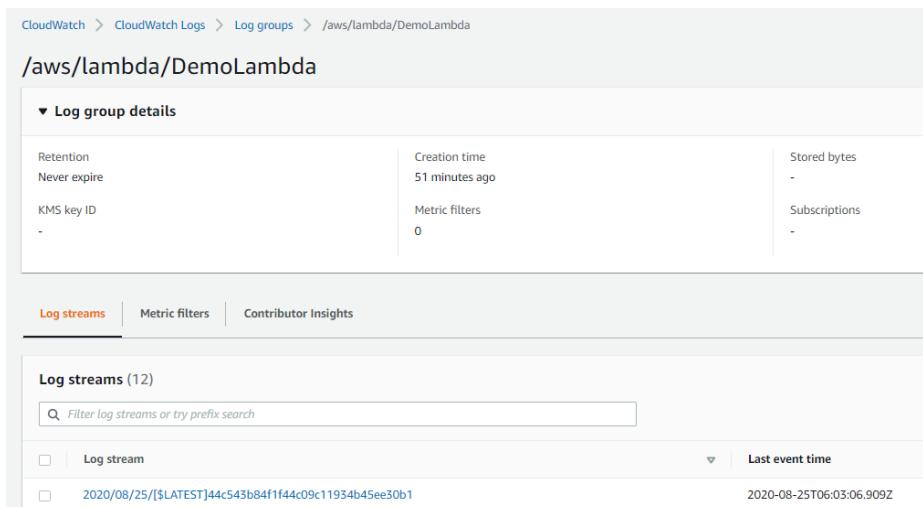
View the latest log in **CloudWatch**.



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



Select the log group and view the latest log streams.



# Lab: AWS Lambda, S3 and API Gateway

Verify that the lambda function is triggered by s3 events.

The screenshot shows the CloudWatch Logs interface for the log group /aws/lambda/DemoLambda. It displays log events for a specific request. The first event is a START RequestId, followed by an S3 event received from S3, and finally an END RequestId with detailed metrics like Duration, Billed Duration, and Memory Usage. The log concludes with a message indicating no newer events at the moment.

Timestamp	Message
2020-08-25T14:11:52.068+08:00	START RequestId: 1d671d4f-82f1-407d-a2b0-3edd119e28a4 Version: \$LATEST
2020-08-25T14:11:52.072+08:00	*** Received Event from S3 *** *** Received Event from S3 ***
2020-08-25T14:11:52.346+08:00	END RequestId: 1d671d4f-82f1-407d-a2b0-3edd119e28a4
2020-08-25T14:11:52.346+08:00	REPORT RequestId: 1d671d4f-82f1-407d-a2b0-3edd119e28a4 Duration: 274.86 ms Billed Duration: 300 ms Memory Size: 128 MB Max Memory Used: 78 MB
	REPORT RequestId: 1d671d4f-82f1-407d-a2b0-3edd119e28a4 Duration: 274.86 ms Billed Duration: 300 ms Memory Size: 128 MB Max Memory Used: 78 MB

View the **s3events** table in DynamoDB.

The screenshot shows the AWS DynamoDB console for the s3events table. The 'Items' tab is selected. A search bar at the top is set to 'Scan: [Table] s3events: Event Time'. The results table shows one item with the following details:

Event Time	Bucket	Event Name	Key
2020-08-25T06:11:45.067Z	demo-richardlui	ObjectCreated:Put	photo3.jpg

# Lab: AWS Lambda, S3 and API Gateway

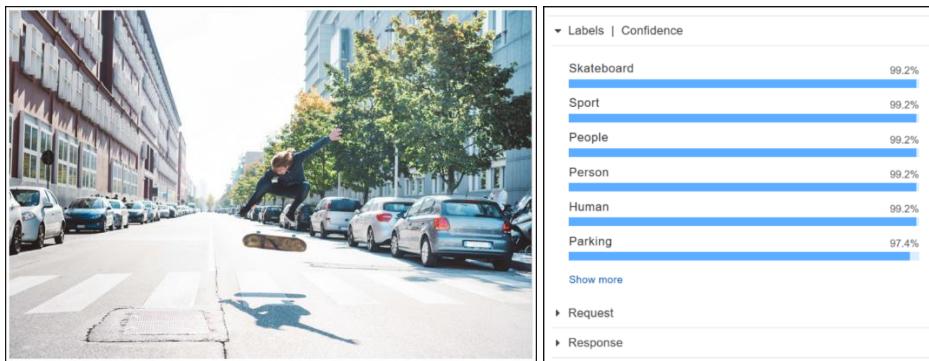
## 5. AWS Rekognition

### 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>

# Lab: AWS Lambda, S3 and API Gateway

## Analyze Faces in an Image

Facial analysis

Get a complete analysis of facial attributes, including confidence scores.



Done with the demo? [Learn more](#)

Results

	Confidence Score
looks like a face	99.9 %
appears to be male	99.4 %
age range	22 - 34 years old
smiling	99.9 %
appears to be happy	99.7 %
not wearing glasses	99.6 %

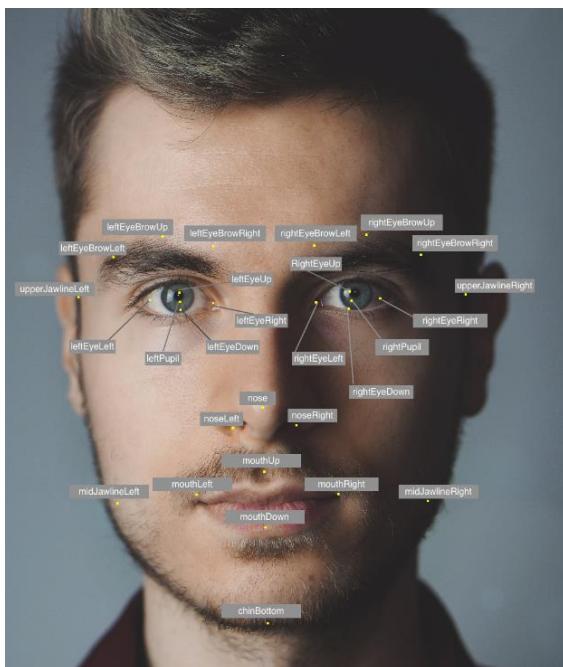
Show more

Request

Response

```
[{"FaceDetails": { "BoundingBox": { "Width": 0.1739723088537148, "Height": 0.1098810881483785, "Left": 0.239307234238119092, "Top": 0.13396979886412018 }, "AgeRange": { "Low": 22, "High": 34 } } ]
```

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



# Lab: AWS Lambda, S3 and API Gateway

## Celebrity recognition

Celebrity recognition

Rekognition automatically recognizes celebrities in images and provides confidence scores.

Choose a sample image

Use your own image

Upload or drag and drop

Use image URL Go

Read feature documentation to learn more

Issues or questions? Use feedback button on bottom-left.

▼ Results

Donald Trump  
Learn More

Match confidence 99 %

Elizabeth II  
Learn More

Match confidence 98 %

► Request

► Response

## Text Recognition

Text in image

Rekognition automatically detects and extracts text in your images. [Learn More](#)

Choose a sample image

Use your own image

Upload or drag and drop

Use image URL Go

Done with the demo?

[Learn more](#)

▼ Results

US English only

| IT'S |  
| MONDAY |  
| but | keep |  
| Smiling |

► Request

► Response

```
{ "TextDetections": [ { "DetectedText": "IT'S", "Type": "LINE", "Id": 0, "Confidence": 99.916687017171875, "Geometry": { "BoundingBox": { "X": 0.1400000059604845, "Height": 0.10000000149011612, "Left": 0.6700000166893005, "Top": 0.18000000715255737 }, "Polygon": [ { "X": 0.6700000166893005, "Y": 0.18000000715255737 }, { "X": 0.8100000023841856, "Y": 0.18000000715255737 }, { "X": 0.8100000023841856, "Y": 0.2800000011920929 }, { "X": 0.6700000166893005, "Y": 0.2800000011920929 } ] } ] }
```

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

# Lab: AWS Lambda, S3 and API Gateway

## 5.2. Image recognition in Lambda

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

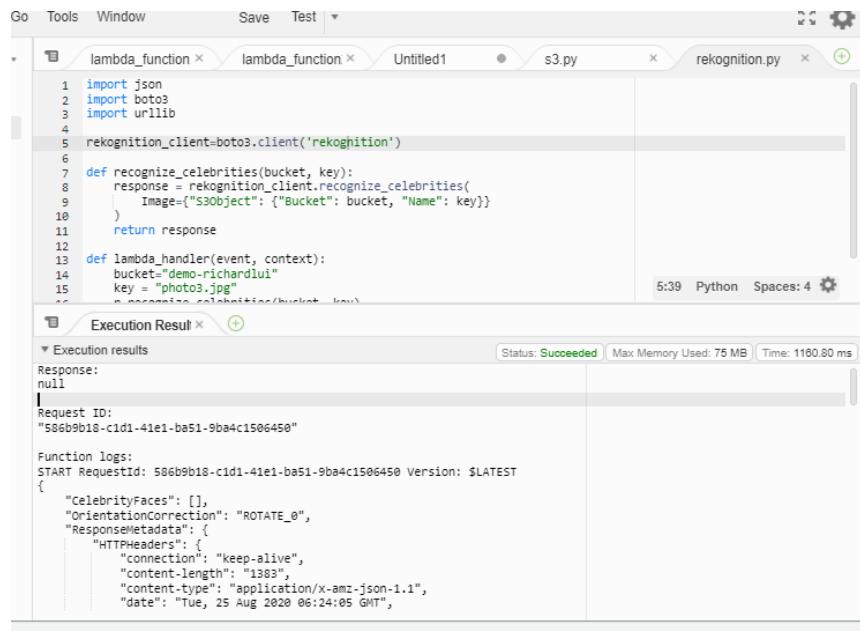
```
import json
import boto3
import urllib

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.



The screenshot shows the AWS Lambda Test interface. At the top, there are tabs for 'lambda\_function' (selected), 'lambda\_function', 'Untitled1', 's3.py', and 'rekognition.py'. Below the tabs is the code editor window containing the provided Python script. To the right of the code editor is the execution result window. The execution result shows a successful run with the following details:

- Status: Succeeded
- Max Memory Used: 75 MB
- Time: 1160.80 ms

The execution result output is as follows:

```
Response:
null

Request ID:
"586b9b18-c1d1-41e1-ba51-9ba4c1506450"

Function logs:
START RequestId: 586b9b18-c1d1-41e1-ba51-9ba4c1506450 Version: $LATEST
{
  "CelebrityFaces": [],
  "OrientationCorrection": "ROTATE_0",
  "ResponseMetadata": {
    "HTTPHeaders": {
      "Connection": "keep-alive",
      "Content-Length": "1383",
      "Content-Type": "application/x-amz-json-1.1",
      "Date": "Tue, 25 Aug 2020 06:24:05 GMT"
    }
  }
}
```

# Lab: AWS Lambda, S3 and API Gateway

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": "Wc7118", "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>

# Lab: AWS Lambda, S3 and API Gateway

---

## 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](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>