

### 3\*\*. Serverless Data Processing Pipeline\*\*

**Objective:** Build a serverless pipeline for processing data (e.g., log processing or ETL jobs).

**Approach:**

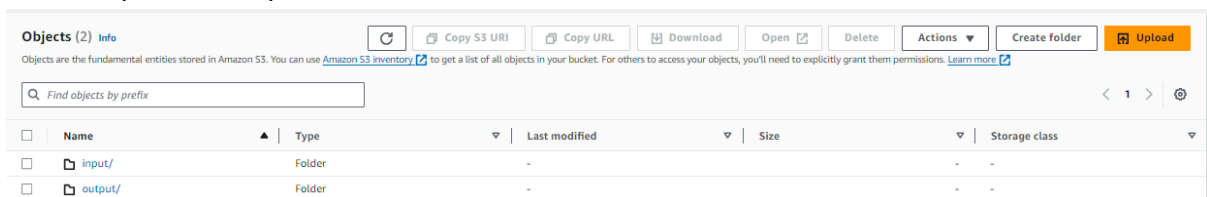
- **Data Ingestion:** Use AWS services like S3 or Kinesis to ingest data.
- **Processing:** Create Lambda functions to process the ingested data.
- **Storage:** Store the processed data in an appropriate AWS service, like S3 or DynamoDB.
- **Monitoring:** Set up CloudWatch to monitor the pipeline's performance and to log any issues.

**Goal:** Learn to build a serverless data processing pipeline, understanding the flow of data through various AWS services.

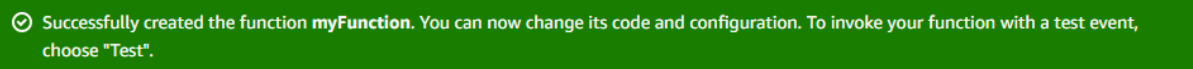
Create an S3 bucket



Create input and output folders



## Create Lambda function



[Lambda](#) > [Functions](#) > myFunction


# myFunction

ThrottleCopy ARNActions

Function overviewInfo

Export to Application ComposerDownload

DiagramTemplate

 myFunction

Layers(0)

+ Add trigger+ Add destination

Description-

Last modified6 seconds ago

Function ARNarn:aws:lambda:us-east-1:171635525037:function:myFunction

Function URLInfo

## Add S3 as trigger

awsServicesSearch[Alt+S]

EC2N. Virginiavoclabs/use

# Add trigger

Trigger configurationInfo

S3awsasynchronousstorage

Bucket

Choose or enter the ARN of an S3 bucket that serves as the event source. The bucket must be in the same region as the function.

s3/mypahiloX

Bucket region: us-east-1

Event types

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 eventsX

Prefix - optional

Enter a single optional prefix to limit the notifications to objects with keys that start with matching characters.

Deploy the following code in the function

The screenshot shows the AWS Lambda console interface. The top navigation bar includes the AWS logo, 'Services', a search bar, and user information. The main content area is titled 'lambda\_function' and shows the 'Environment Variables' tab. The code is a Python function named 'lambda\_handler' that interacts with Amazon S3. The code is as follows:

```
1 import boto3
2 import urllib.parse
3 import os
4
5 def lambda_handler(event, context):
6     s3_client = boto3.client('s3')
7
8     # Get bucket name and object key from the S3 event
9     bucket_name = event['Records'][0]['s3']['bucket']['name']
10    object_key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'], encoding='utf-8')
11
12    # Extract the filename from the object key
13    filename = os.path.basename(object_key)
14
15    # Define destination key (output folder and file name)
16    destination_key = 'output/' + filename
17
18    # Get the file from S3
19    file_obj = s3_client.get_object(Bucket=bucket_name, Key=object_key)
20    file_content = file_obj['Body'].read().decode('utf-8')
21
22    # Convert content to uppercase
23    upper_content = file_content.upper()
24
25    # Upload the modified content back to S3
26    s3_client.put_object(Bucket=bucket_name, Key=destination_key, Body=upper_content)
27
28    return {
29        'statusCode': 200,
30        'body': 'File processed and uploaded successfully'
31    }
```

On the right side, there is a 'Tutorials' panel with a link to 'Create a simple web app'.

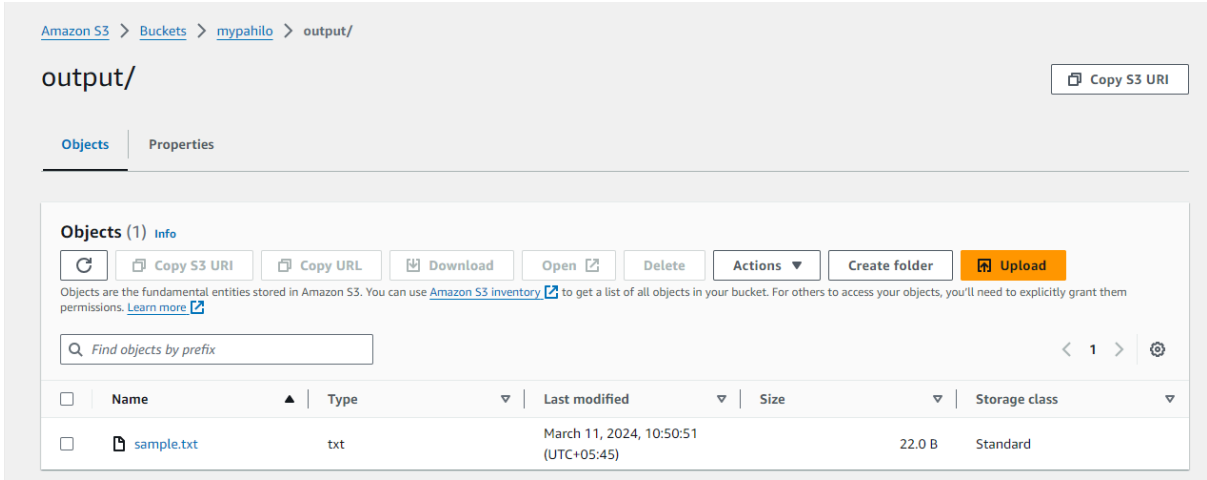
Upload a sample txt file

The screenshot shows the AWS Lambda console interface for uploading a file. The top navigation bar includes the AWS logo, 'Services', a search bar, and user information. The main content area is titled 'lambda\_function' and shows the 'Environment Variables' tab. The code is a Python function named 'lambda\_handler' that interacts with Amazon S3. The code is as follows:

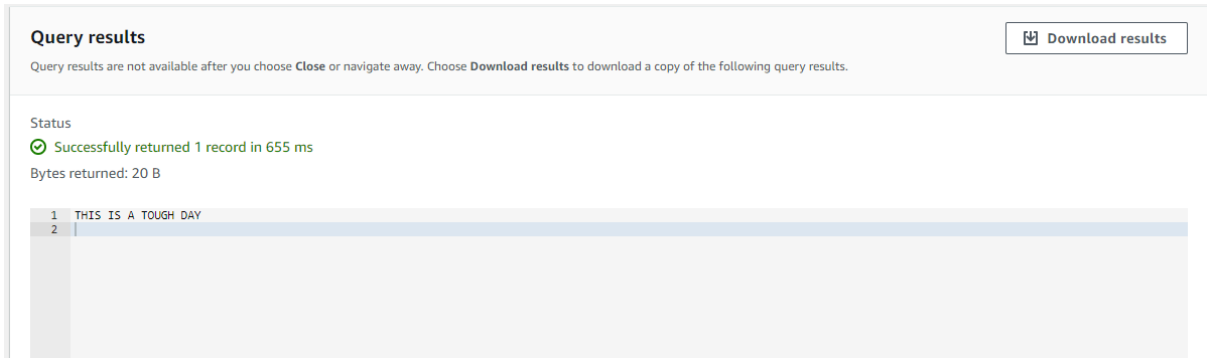
```
1 import boto3
2 import urllib.parse
3 import os
4
5 def lambda_handler(event, context):
6     s3_client = boto3.client('s3')
7
8     # Get bucket name and object key from the S3 event
9     bucket_name = event['Records'][0]['s3']['bucket']['name']
10    object_key = urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'], encoding='utf-8')
11
12    # Extract the filename from the object key
13    filename = os.path.basename(object_key)
14
15    # Define destination key (output folder and file name)
16    destination_key = 'output/' + filename
17
18    # Get the file from S3
19    file_obj = s3_client.get_object(Bucket=bucket_name, Key=object_key)
20    file_content = file_obj['Body'].read().decode('utf-8')
21
22    # Convert content to uppercase
23    upper_content = file_content.upper()
24
25    # Upload the modified content back to S3
26    s3_client.put_object(Bucket=bucket_name, Key=destination_key, Body=upper_content)
27
28    return {
29        'statusCode': 200,
30        'body': 'File processed and uploaded successfully'
31    }
```

On the right side, there is a 'Tutorials' panel with a link to 'Create a simple web app'.

As we can see the file has been replicated in the output folder



We have obtained the query result as follows:



The cloudwatch log events are as follows:

