

Advanced DevOps Lab

Experiment 12

Aim: To create a Lambda function which will log “An Image has been added” once you add an object to a specific bucket in S3

Theory:

AWS Lambda

AWS Lambda is a serverless computing service provided by Amazon Web Services (AWS). Users of AWS Lambda create functions, self-contained applications written in one of the supported languages and runtimes, and upload them to AWS Lambda, which executes those functions in an efficient and flexible manner. The Lambda functions can perform any kind of computing task, from serving web pages and processing streams of data to calling APIs and integrating with other AWS services.

The concept of “serverless” computing refers to not needing to maintain your own servers to run these functions. AWS Lambda is a fully managed service that takes care of all the infrastructure for you. And so “serverless” doesn’t mean that there are no servers involved: it just means that the servers, the operating systems, the network layer and the rest of the infrastructure have already been taken care of so that you can focus on writing application code.

Features of AWS Lambda

- AWS Lambda easily scales the infrastructure without any additional configuration. It reduces the operational work involved.
- It offers multiple options like AWS S3, CloudWatch, DynamoDB, API Gateway, Kinesis, CodeCommit, and many more to trigger an event.
- You don’t need to invest upfront. You pay only for the memory used by the lambda function and minimal cost on the number of requests hence cost-efficient.
- AWS Lambda is secure. It uses AWS IAM to define all the roles and security policies.
- It offers fault tolerance for both services running the code and the function. You do not have to worry about the application down.

Packaging Functions

Lambda functions need to be packaged and sent to AWS. This is usually a process of compressing the function and all its dependencies and uploading it to an S3 bucket. And letting AWS know that you want to use this package when a specific event takes place. To help us with

this process we use the Serverless Stack Framework (SST). We'll go over this in detail later on in this guide.

Execution Model

The container (and the resources used by it) that runs our function is managed completely by AWS. It is brought up when an event takes place and is turned off if it is not being used. If additional requests are made while the original event is being served, a new container is brought up to serve a request. This means that if we are undergoing a usage spike, the cloud provider simply creates multiple instances of the container with our function to serve those requests.

This has some interesting implications. Firstly, our functions are effectively stateless. Secondly, each request (or event) is served by a single instance of a Lambda function. This means that you are not going to be handling concurrent requests in your code. AWS brings up a container whenever there is a new request. It does make some optimizations here. It will hang on to the container for a few minutes (5 - 15mins depending on the load) so it can respond to subsequent requests without a cold start.

Stateless Functions

The above execution model makes Lambda functions effectively stateless. This means that every time your Lambda function is triggered by an event it is invoked in a completely new environment. You don't have access to the execution context of the previous event.

However, due to the optimization noted above, the actual Lambda function is invoked only once per container instantiation. Recall that our functions are run inside containers. So when a function is first invoked, all the code in our handler function gets executed and the handler function gets invoked. If the container is still available for subsequent requests, your function will get invoked and not the code around it.

For example, the `createNewDbConnection` method below is called once per container instantiation and not every time the Lambda function is invoked. The `myHandler` function on the other hand is called on every invocation.

Common Use Cases for Lambda

Due to Lambda's architecture, it can deliver great benefits over traditional cloud computing setups for applications where:

1. Individual tasks run for a short time;
2. Each task is generally self-contained;
3. There is a large difference between the lowest and highest levels in the workload of the application.

Some of the most common use cases for AWS Lambda that fit these criteria are: Scalable APIs. When building APIs using AWS Lambda, one execution of a Lambda function can serve a single HTTP request. Different parts of the API can be routed to different Lambda functions via Amazon API Gateway. AWS Lambda automatically scales individual functions according to the demand for them, so different parts of your API can scale differently according to current usage levels. This allows for cost-effective and flexible API setups.

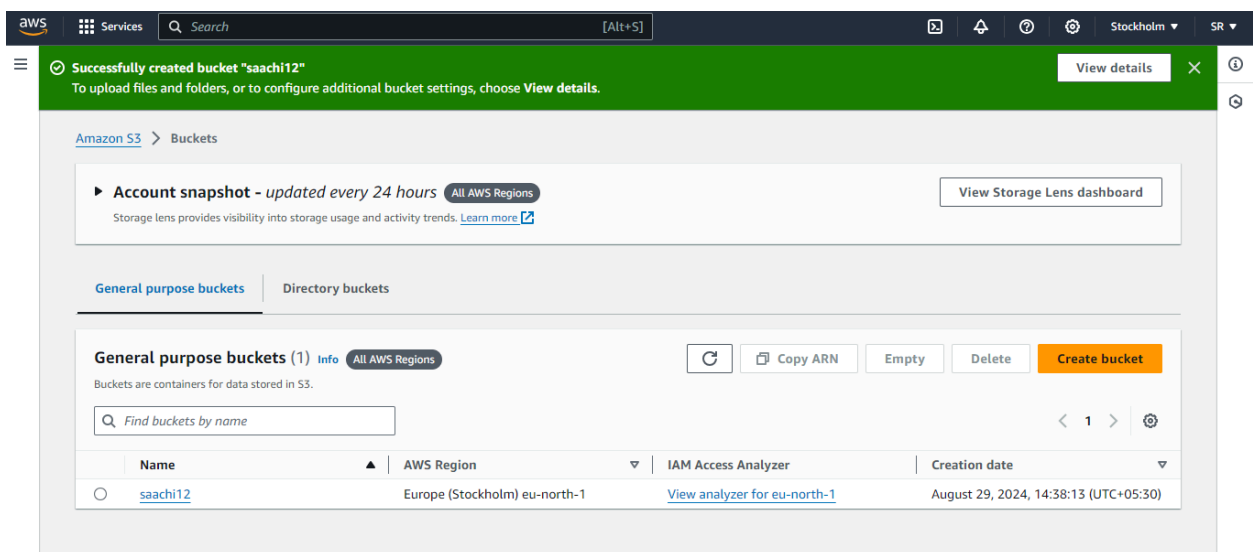
Data processing. Lambda functions are optimized for event-based data processing. It is easy to integrate AWS Lambda with data sources like Amazon DynamoDB and trigger a Lambda

function for specific kinds of data events. For example, you could employ Lambda to do some work every time an item in DynamoDB is created or updated, thus making it a good fit for things like notifications, counters and analytics.

STEPS:

1. Create an S3 Bucket

- Go to the AWS Management Console.
- Navigate to the S3 service.
- Click on "Create bucket."
- Enter a unique bucket name and choose a region.
- Configure other settings as needed and click "Create bucket."



2. Create a Lambda Function

- Go to the AWS Management Console.
- Navigate to the Lambda service.
- Click on "Create function."
- Choose "Author from scratch."
- Enter a name for your function, e.g., S3ImageLogger.
- Select a runtime (e.g., Python 3.x or Node.js).
- Click "Create function."

3. Write the Lambda Function Code

- In the Lambda function console, scroll down to the code editor.

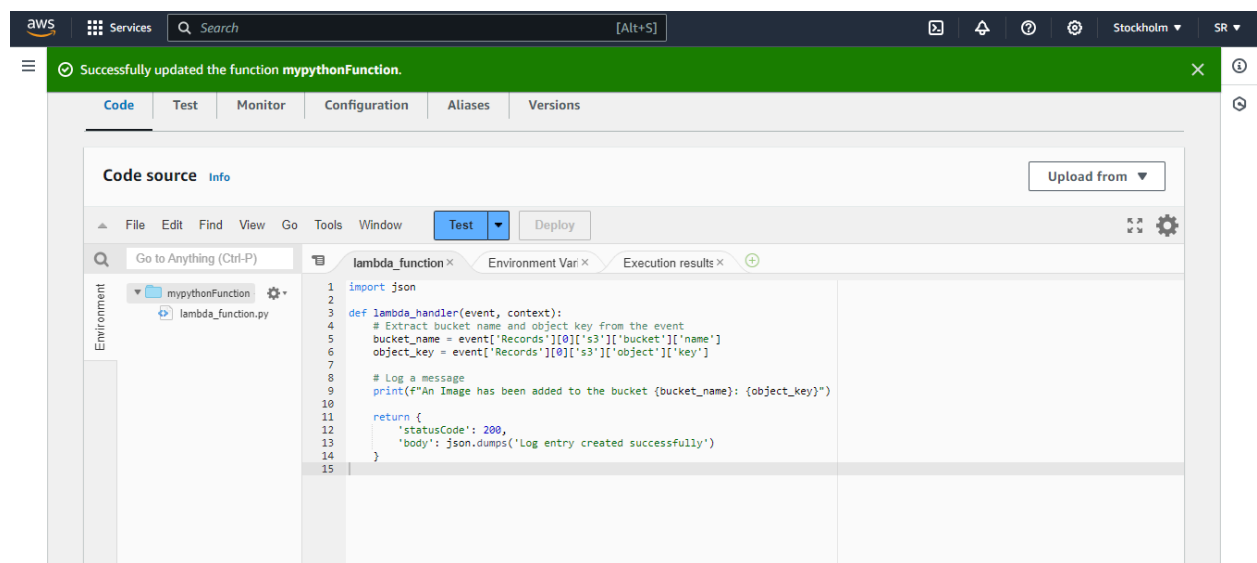
Replace the default code with the following code snippet (assuming you're using Python):
python

Copy code

```
import json
```

```
def lambda_handler(event, context):  
    # Extract bucket name and object key from the event  
    bucket_name = event['Records'][0]['s3']['bucket']['name']  
    object_key = event['Records'][0]['s3']['object']['key']  
  
    # Log a message  
    print(f"An Image has been added to the bucket {bucket_name}:  
{object_key}")  
  
    return {  
        'statusCode': 200,  
        'body': json.dumps('Log entry created successfully')  
    }
```

-
- Click "Deploy" to save your changes.

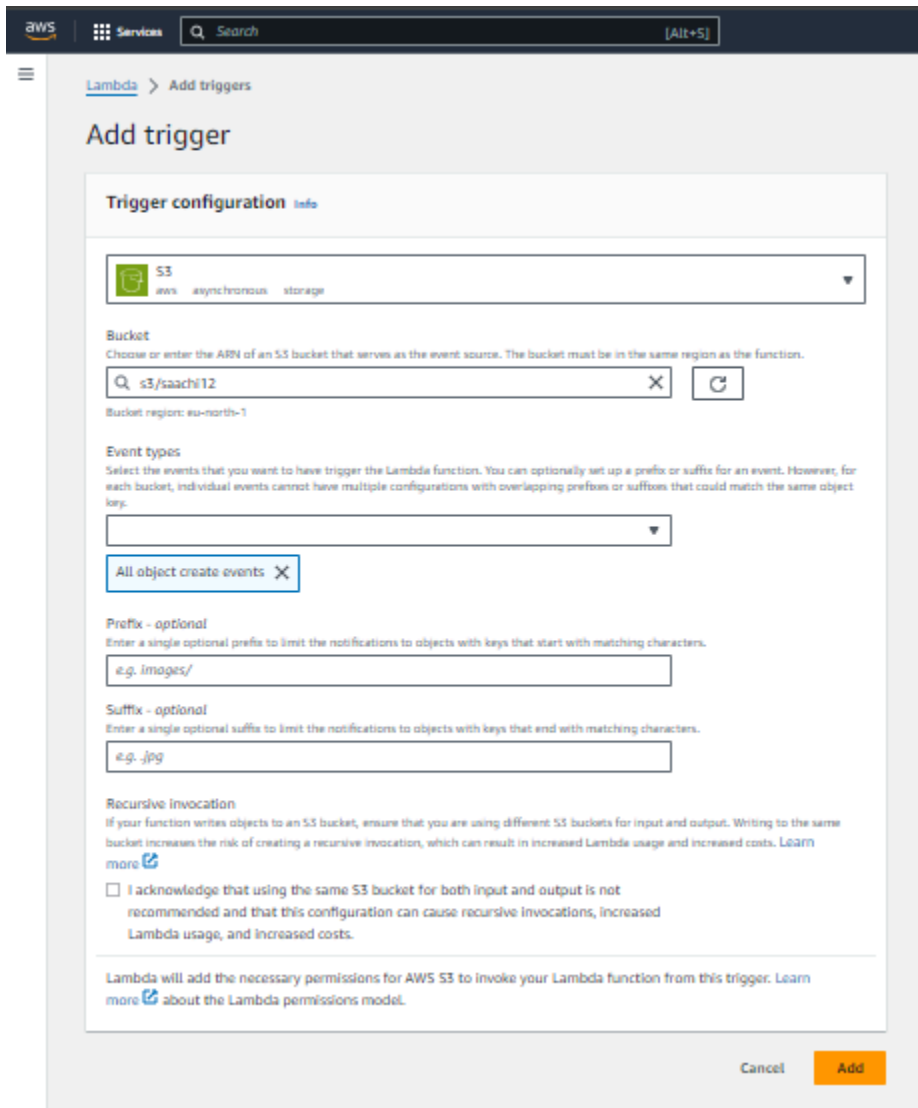


4. Set Up S3 Trigger for the Lambda Function

- Scroll down to the "Function overview" section in the Lambda console.
- Click on "Add trigger."
- Select "S3" from the list of triggers.
- Choose the S3 bucket you created earlier.
- In the "Event type" dropdown, select "All object create events."
- Optionally, specify a prefix or suffix to filter the events (e.g., for images only, you can use suffix .jpg, .png).
- Click "Add."

5. Grant Permissions to Lambda

- Navigate to the "Permissions" tab of your Lambda function.
- Ensure the Lambda function's execution role has the necessary permissions to access the S3 bucket.
- If needed, attach the AmazonS3ReadOnlyAccess policy or create a custom policy with the necessary permissions.



The screenshot shows the AWS Lambda console's 'Add trigger' page. At the top, there's a navigation bar with the AWS logo, 'Services', a search bar, and a '[Alt+S]' shortcut. Below the navigation bar, the page title is 'Add trigger'. The main content area is titled 'Trigger configuration' with an 'Info' link. It features a dropdown menu for the trigger type, currently set to 'S3'. Below this, there's a 'Bucket' section with a text input field containing 's3/saachi12' and a 'Bucket region' dropdown set to 'eu-north-1'. The 'Event types' section has a dropdown menu and a button labeled 'All object create events'. There are also optional fields for 'Prefix' (containing 'e.g. images/') and 'Suffix' (containing 'e.g. .jpg'). A 'Recursive invocation' section includes a checkbox and a warning message. At the bottom, there are 'Cancel' and 'Add' buttons.

aws Services Search [Alt+S]

Lambda > Add triggers

Add trigger

Trigger configuration Info

S3 aws asynchronous storage

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/saachi12 X

Bucket region: eu-north-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 events X

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

Recursive invocation
If your function writes objects to an S3 bucket, ensure that you are using different S3 buckets for input and output. Writing to the same bucket increases the risk of creating a recursive invocation, which can result in increased Lambda usage and increased costs. [Learn more](#)

☐ I acknowledge that using the same S3 bucket for both input and output is not recommended and that this configuration can cause recursive invocations, increased Lambda usage, and increased costs.

Lambda will add the necessary permissions for AWS S3 to invoke your Lambda function from this trigger. [Learn more](#) about the Lambda permissions model.

Cancel Add

6. Test the Setup

- Upload an image file to your S3 bucket.
- Go to the "Monitoring" tab in your Lambda function to check the logs.
- Alternatively, use CloudWatch Logs to view the output and confirm that the message "An Image has been added" has been logged.

This setup should ensure that each time an image is uploaded to the specified S3 bucket, the Lambda function will log the appropriate message.

aws

Services

Search

[Alt+S]

Stockholm

SR

Amazon S3

Buckets

saachi12

Upload

Upload

Info

Add the files and folders you want to upload to S3. To upload a file larger than 160GB, use the AWS CLI, AWS SDK or Amazon S3 REST API. [Learn more](#)

Drag and drop files and folders you want to upload here, or choose **Add files** or **Add folder**.

Files and folders (1 Total, 17.4 KB)

All files and folders in this table will be uploaded.

Find by name

< 1 >

<input type="checkbox"/>	Name	Folder	Type
<input type="checkbox"/>	butterfly.jpg	-	image/jpeg

Destination [Info](#)

Destination
s3://saachi12

aws

Services

Search

[Alt+S]

Stockholm

SR

Upload succeeded

View details below.

The information below will no longer be available after you navigate away from this page.

Summary

Destination
s3://saachi12

Succeeded

1 file, 17.4 KB (100.00%)

Failed

0 files, 0 B (0%)

Files and folders

Configuration

Files and folders (1 Total, 17.4 KB)

Find by name

< 1 >

Name	Folder	Type	Size	Status	Error
butterfly.jpg	-	image/jpeg	17.4 KB	Succeeded	-

aws

Services

Search

[Alt+S]

Stockholm

SR

CloudWatch

Log groups

/aws/lambda/mypythonFunction

2024/08/29/[LATEST]e5aba80bf7644ec2bf57477b2ebc3734

Log events

Actions

Start tailing

Create metric filter

Filter events - press enter to search

Clear

1m

30m

1h

12h

Custom

UTC timezone

Display

Timestamp	Message
No older events at this moment. Retry	
2024-08-29T09:14:54.054Z	INIT_START Runtime Version: python:3.12.v30 Runtime Version ARN: arn:aws:lambda:eu-north-1::runtime:acd65000e3f6a085fb07933...
2024-08-29T09:14:54.136Z	START RequestId: 36e50767-fac0-4ed3-83ff-87872352f94b Version: \$LATEST
2024-08-29T09:14:54.136Z	An Image has been added to the bucket saachi12: butterfly.jpg
2024-08-29T09:14:54.138Z	END RequestId: 36e50767-fac0-4ed3-83ff-87872352f94b
2024-08-29T09:14:54.138Z	REPORT RequestId: 36e50767-fac0-4ed3-83ff-87872352f94b Duration: 1.08 ms Billed Duration: 2 ms Memory Size: 128 MB Max Memor...
No newer events at this moment. Auto retry paused . Resume	