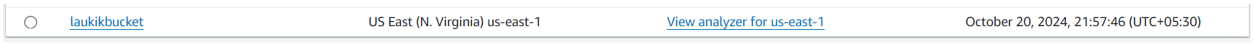
Name-Laukik Padgaonkar D15C Roll no-37

Case Study: AWS Lambda, S3, and DynamoDB Integration

Step 1: Create S3 Bucket

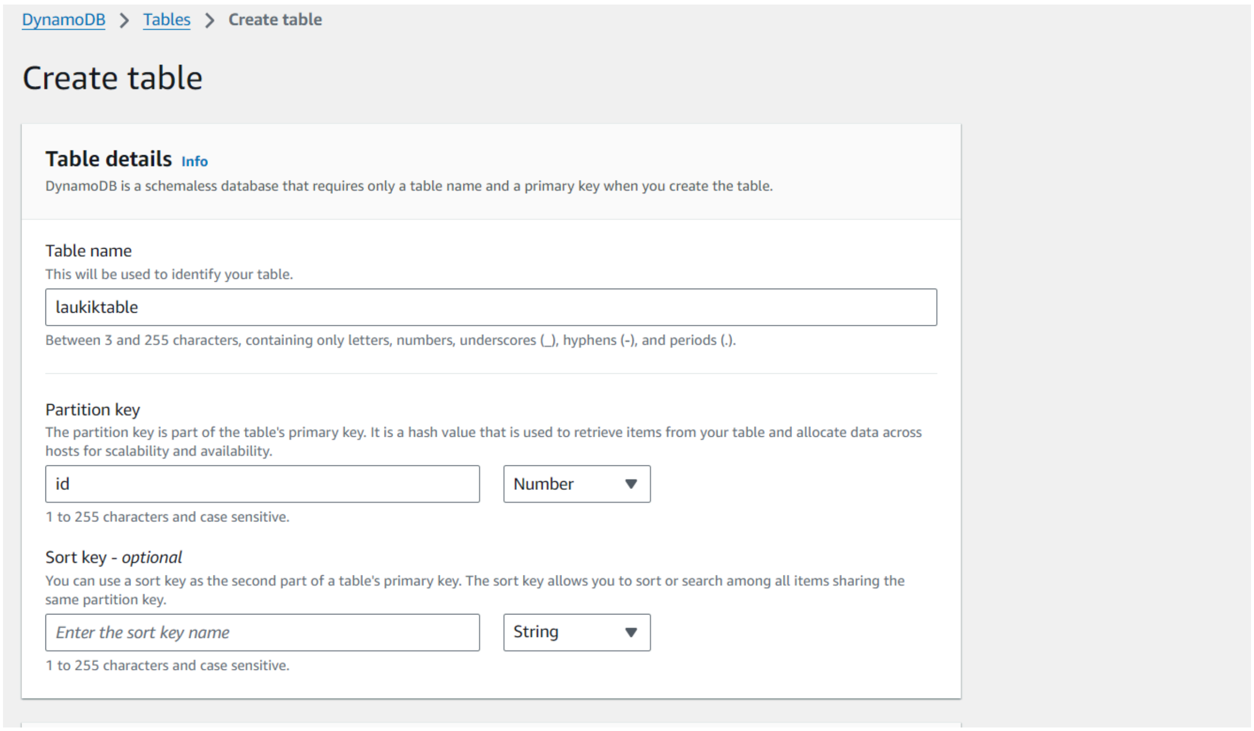
- Begin by creating a new S3 bucket to store the JSON file that will be processed.



laukikbucket	US East (N. Virginia) us-east-1	View analyzer for us-east-1	October 20, 2024, 21:57:46 (UTC+05:30)

Step 2: Set Up DynamoDB Table

- Create a DynamoDB table that will store the JSON data. Ensure the table has the appropriate schema to match the data structure in the JSON file.



DynamoDB > Tables > Create table

Create table

Table details Info

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name

This will be used to identify your table.

laukiktable

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).

Partition key

The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

id | Number

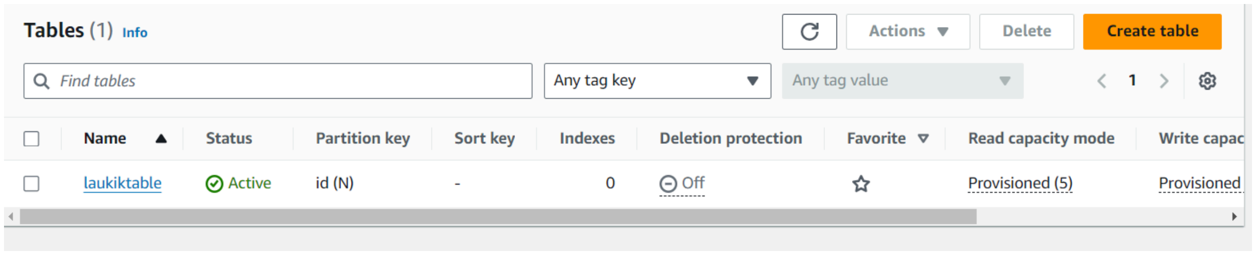
1 to 255 characters and case sensitive.

Sort key - *optional*

You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

Enter the sort key name | String

1 to 255 characters and case sensitive.

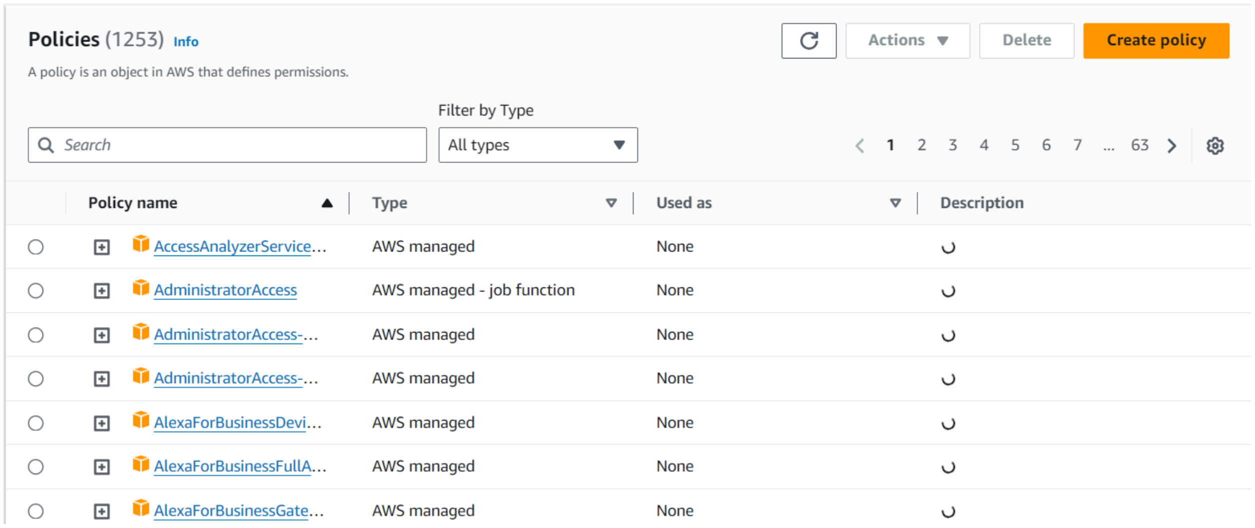


Name	Status	Partition key	Sort key	Indexes	Deletion protection	Favorite	Read capacity mode	Write capac
laukiktable	Active	id (N)	-	0	Off	☆	Provisioned (5)	Provisioned

Name-Laukik Padgaonkar D15C Roll no-37

Step 3: Configure IAM Roles and Policies

- Go to IAM roles and create the necessary policies to provide your Lambda function with access to both S3 and DynamoDB.
- Create a role and attach the policies to it, ensuring the Lambda function can read from S3 and write to DynamoDB.



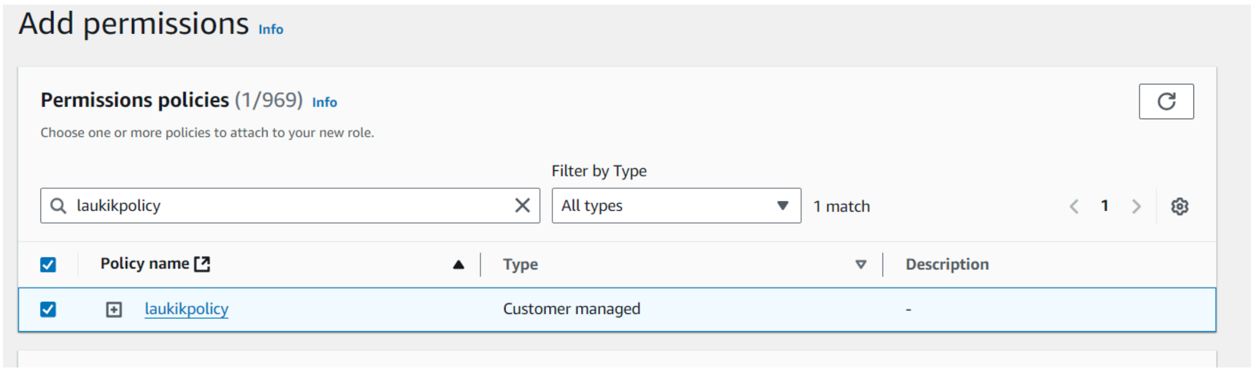
Policies (1253) Info

A policy is an object in AWS that defines permissions.

Policy name	Type	Used as	Description
AccessAnalyzerService…	AWS managed	None	
AdministratorAccess	AWS managed - job function	None	
AdministratorAccess-…	AWS managed	None	
AdministratorAccess-…	AWS managed	None	
AlexaForBusinessDevi…	AWS managed	None	
AlexaForBusinessFullA…	AWS managed	None	
AlexaForBusinessGate…	AWS managed	None	

Create role

Select created policy while creating roles:



Add permissions Info

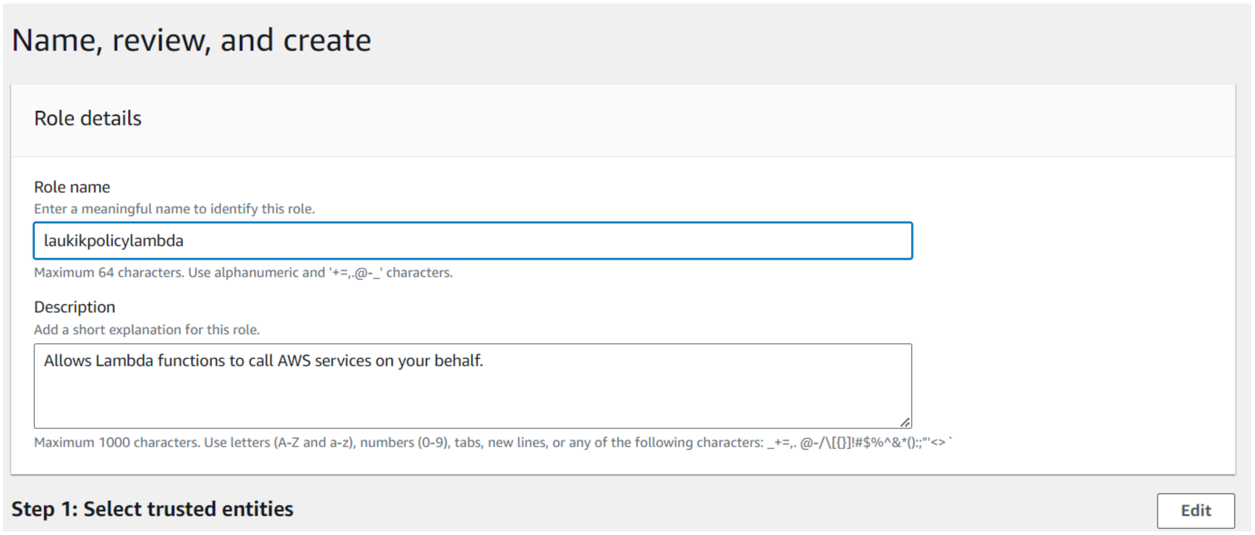
Permissions policies (1/969) Info

Choose one or more policies to attach to your new role.

Search: laukikpolicy

Policy name	Type	Description
✓ laukikpolicy	Customer managed	-

Name-Laukik Padgaonkar D15C Roll no-37



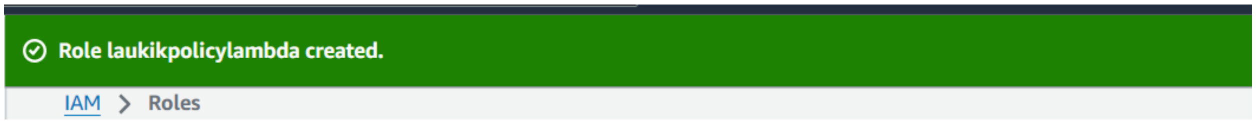
Name, review, and create

Role details

Role name
Enter a meaningful name to identify this role.
laukikpolicylambda
Maximum 64 characters. Use alphanumeric and '+=,.@-_' characters.

Description
Add a short explanation for this role.
Allows Lambda functions to call AWS services on your behalf.
Maximum 1000 characters. Use letters (A-Z and a-z), numbers (0-9), tabs, new lines, or any of the following characters: _+=,. @-/\[{}]!#$%^&*();:"'<>`

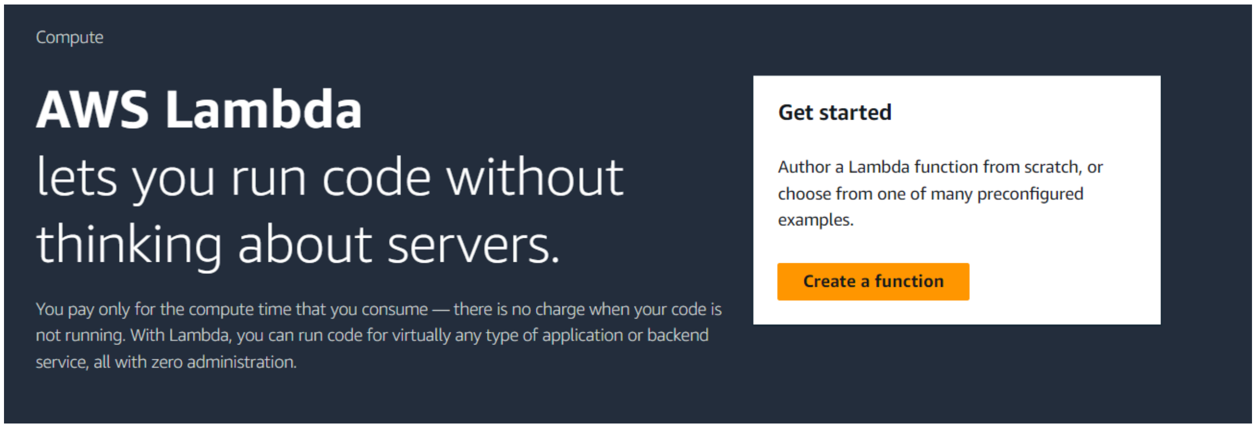
Step 1: Select trusted entities



Role laukikpolicylambda created.

Step 4: Create Lambda Function

- In the AWS Lambda console, create a new Lambda function.
- While creating the function, select "Use an existing role" and choose the IAM role you created earlier.
- Define the Lambda function to read a JSON file from the S3 bucket and store its data in DynamoDB.



Compute

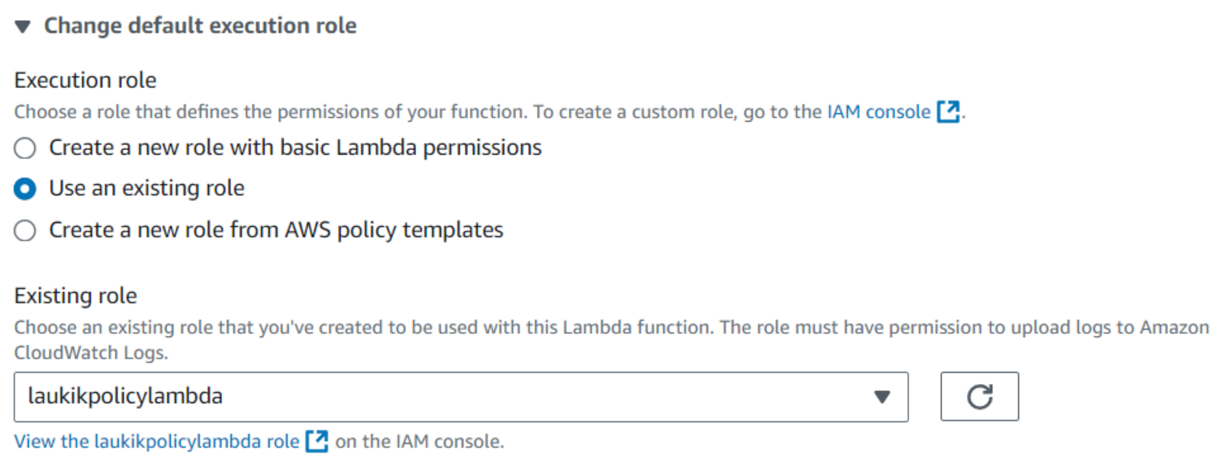
AWS Lambda lets you run code without thinking about servers.

You pay only for the compute time that 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.

Get started

Author a Lambda function from scratch, or choose from one of many preconfigured examples.

Name-Laukik Padgaonkar D15C Roll no-37



▼ Change default 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

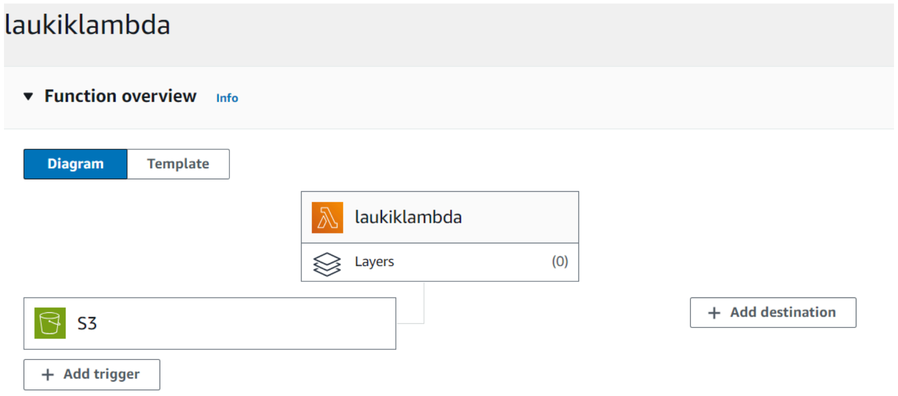
Existing role

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.

laukikpolicylambda

View the laukikpolicylambda role on the IAM console.

Create function

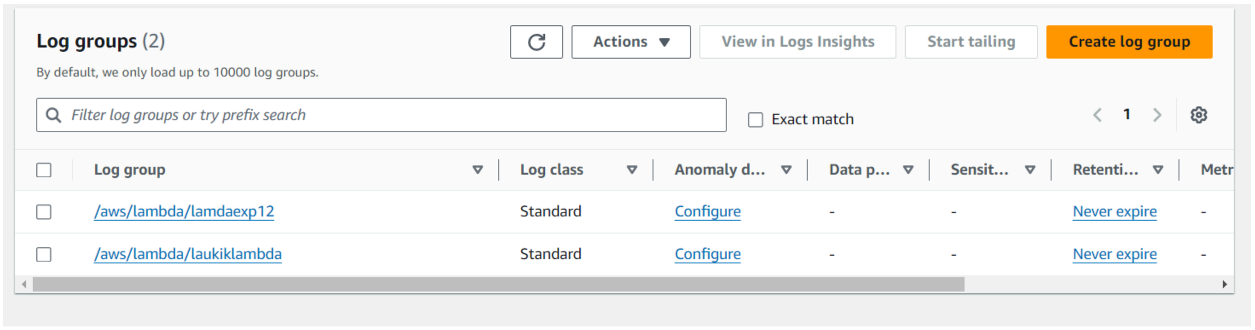


laukiklambda

▼ Function overview Info

Step 5: CloudWatch Monitoring and Testing

- After creating the Lambda function, check CloudWatch logs to ensure it's properly set up.
- Run a test in the Lambda console and verify the logs again to ensure the function behaves as expected.



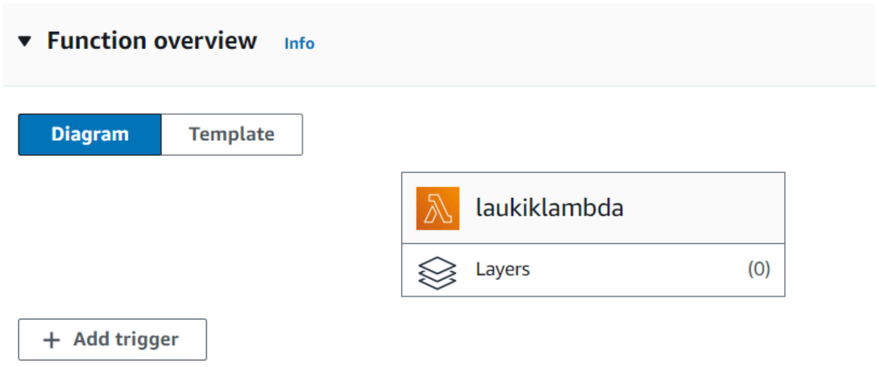
Log groups (2)

By default, we only load up to 10000 log groups.

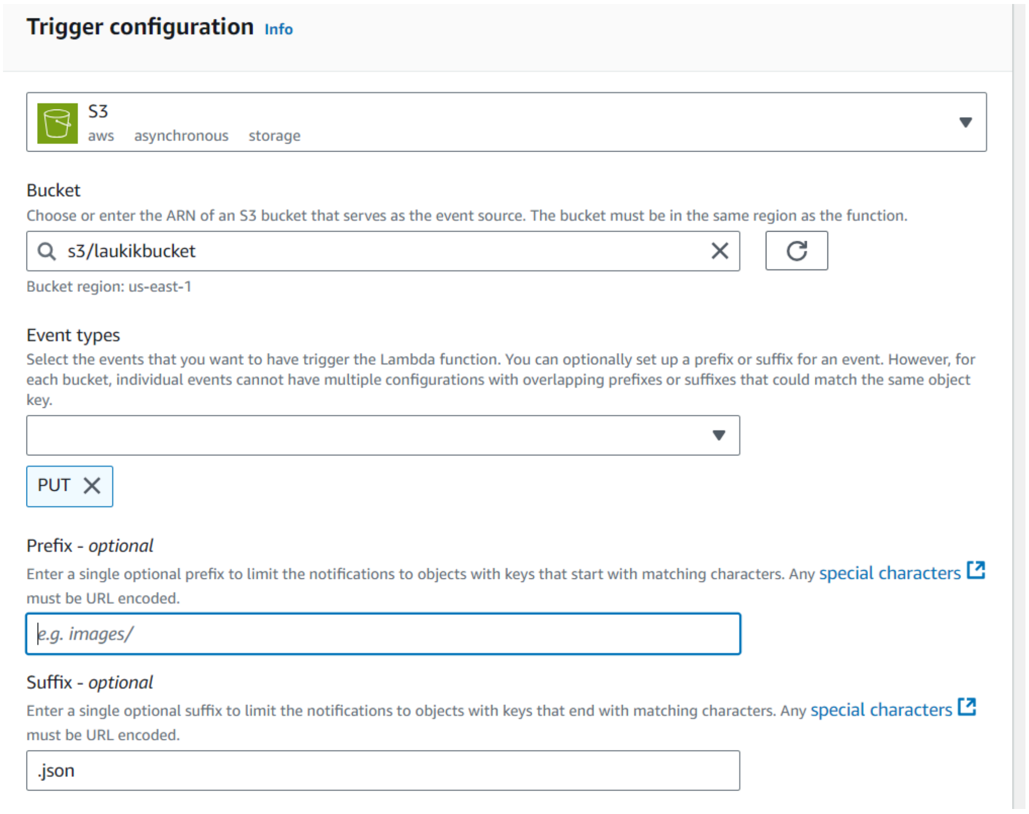
	Log group	Log class	Anomaly d...	Data p...	Sensit...	Retenti...	Metr
☐	/aws/lambda/lamdaexp12	Standard	Configure	-	-	Never expire	-
☐	/aws/lambda/laukiklambda	Standard	Configure	-	-	Never expire	-

Step 6: Add S3 Trigger to Lambda

- Once the Lambda function passes tests, add a trigger that runs the function whenever a file is uploaded to the S3 bucket.
- After adding the trigger, check the S3 bucket's properties to verify the event has been added.



Add the following configurations:


Trigger configuration Info

S3
aws asynchronous storage

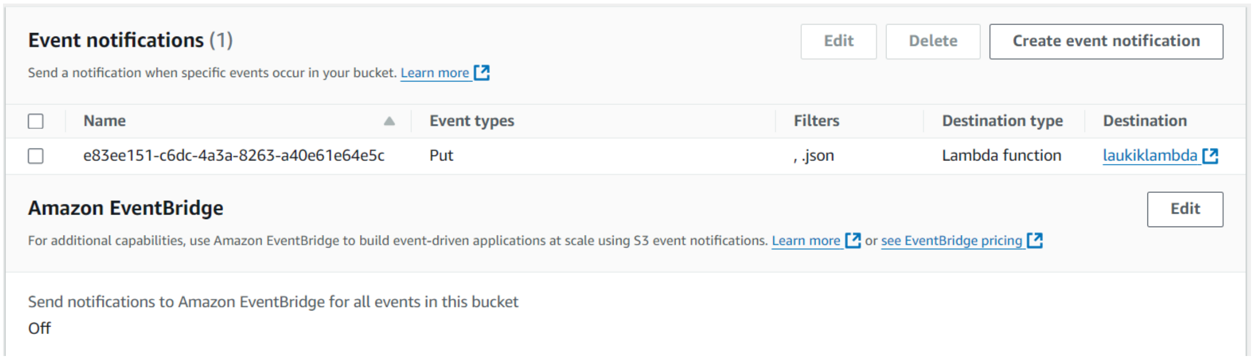
Bucket
Choose or enter the ARN of an S3 bucket that serves as the event source. The bucket must be in the same region as the function.
s3/laukikbucket
Bucket region: us-east-1

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.
PUT

Prefix - *optional*
Enter a single optional prefix to limit the notifications to objects with keys that start with matching characters. Any special characters must be URL encoded.
e.g. images/

Suffix - *optional*
Enter a single optional suffix to limit the notifications to objects with keys that end with matching characters. Any special characters must be URL encoded.
.json

Now after adding the trigger we can see in lambda bucket properties to see if event is added:



Event notifications (1)

Send a notification when specific events occur in your bucket. Learn more

Name	Event types	Filters	Destination type	Destination
e83ee151-c6dc-4a3a-8263-a40e61e64e5c	Put	, .json	Lambda function	laukiklambda

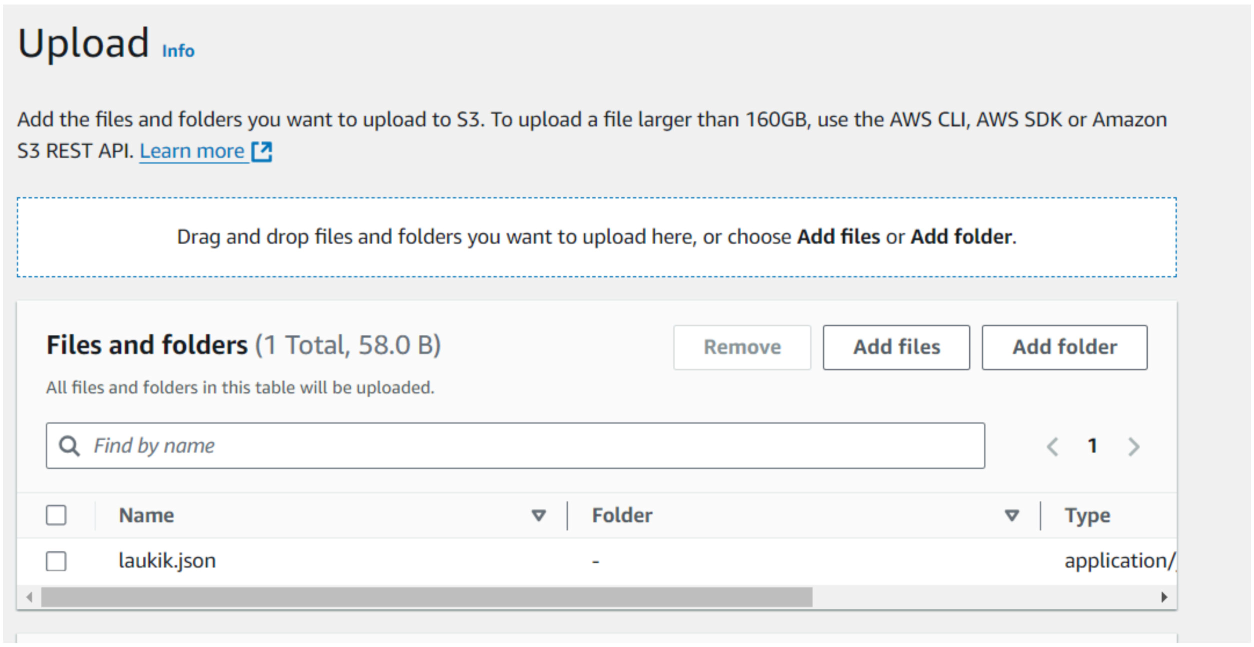
Amazon EventBridge

For additional capabilities, use Amazon EventBridge to build event-driven applications at scale using S3 event notifications. Learn more or see EventBridge pricing

Send notifications to Amazon EventBridge for all events in this bucket
Off

Step 7: Upload JSON File to S3

- Upload a test JSON file (`laukik.json`) to the S3 bucket.
- In CloudWatch logs, you should see a new log stream with details about the file uploaded.



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

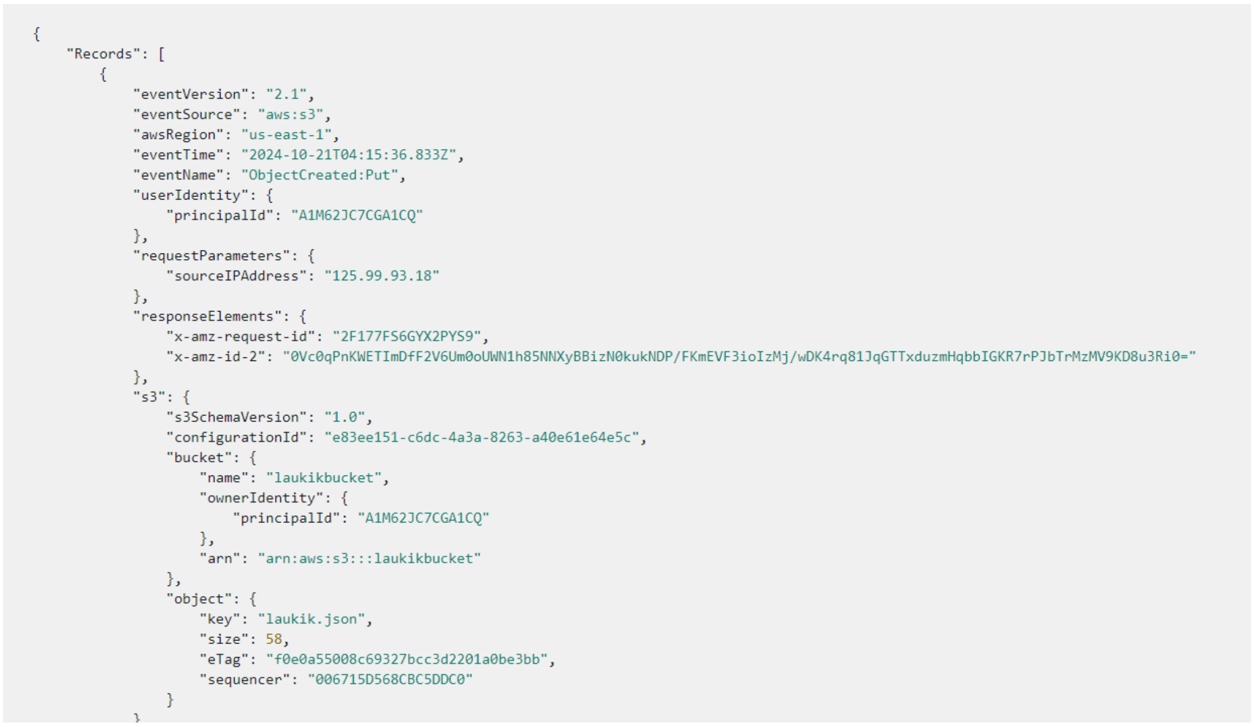
Drag and drop files and folders you want to upload here, or choose **Add files** or **Add folder**.

Files and folders (1 Total, 58.0 B)

All files and folders in this table will be uploaded.

Name	Folder	Type
laukik.json	-	application/

In logs a new log stream must have been created where the log of the file which was uploaded can be seen:



```
{
    "Records": [
        {
            "eventVersion": "2.1",
            "eventSource": "aws:s3",
            "awsRegion": "us-east-1",
            "eventTime": "2024-10-21T04:15:36.833Z",
            "eventName": "ObjectCreated:Put",
            "userIdentity": {
                "principalId": "A1M62JC7CGA1CQ"
            },
            "requestParameters": {
                "sourceIPAddress": "125.99.93.18"
            },
            "responseElements": {
                "x-amz-request-id": "2F177FS6GYX2PYS9",
                "x-amz-id-2": "0Vc0qPnKWETImDfF2V6Um0oUWN1h85NNXyBBizN0kukNDP/FKmEVF3ioIzMj/wDK4rq81JqGTTxduzmHqbbIGKR7rPJbTrMzMV9KD8u3Ri0="
            },
            "s3": {
                "s3SchemaVersion": "1.0",
                "configurationId": "e83ee151-c6dc-4a3a-8263-a40e61e64e5c",
                "bucket": {
                    "name": "laukikbucket",
                    "ownerIdentity": {
                        "principalId": "A1M62JC7CGA1CQ"
                    },
                    "arn": "arn:aws:s3:::laukikbucket"
                },
                "object": {
                    "key": "laukik.json",
                    "size": 58,
                    "eTag": "f0e0a55008c69327bcc3d2201a0be3bb",
                    "sequencer": "006715D568CBC5DDC0"
                }
```

Step 8: Update Lambda Code

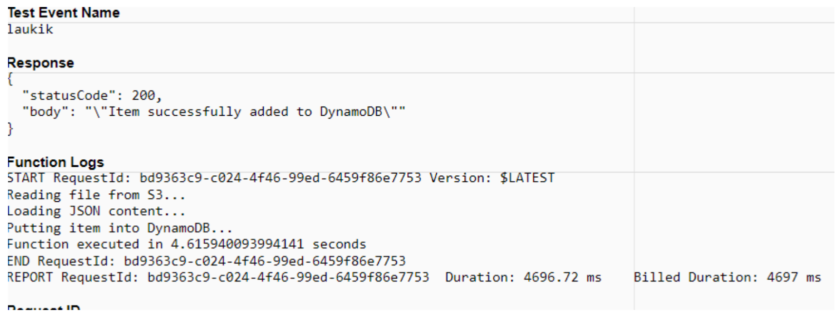
- Use the following updated code in the Lambda function to process the JSON file:

Now update the following code:

```
import json
import boto3
import time

def lambda_handler(event, context):
    start_time = time.time()  # Start timing

    bucket = 'laukikbucket'
    file_name = 'laukik.json'

    s3 = boto3.client('s3')
    table = boto3.resource('dynamodb').Table('laukiktable')

    try:
        # Read the file from S3
        print("Reading file from S3...")
        file_obj = s3.get_object(Bucket=bucket, Key=file_name)
        file_reader = file_obj['Body'].read().decode('utf-8')

        # Load the content of the file
        print("Loading JSON content...")
        file_content = json.loads(file_reader)

        # Convert id to integer if necessary
        if 'id' in file_content:
            file_content['id'] = int(file_content['id'])  # Convert to integer

        # Put the item into the DynamoDB table
        print("Putting item into DynamoDB...")
        table.put_item(Item=file_content)

    except Exception as e:
        print(f"Error: {e}")
        return {
            'statusCode': 500,
            'body': json.dumps('Error processing the request')
        }

    end_time = time.time()
    print(f"Function executed in {end_time - start_time} seconds")

    return {
        'statusCode': 200,
        'body': json.dumps('Item successfully added to DynamoDB')
    }
```

tep 9: Deployment and Testing

- Deploy the updated Lambda function and run a test case.
- If successful, the logs will show confirmation that the data was added to DynamoDB.



Test Event Name

```
laukik
```

Response

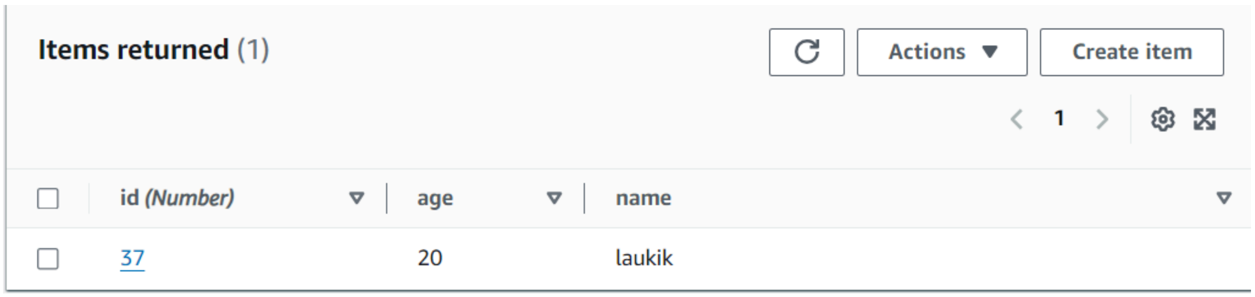
```
{
  "statusCode": 200,
  "body": "\"Item successfully added to DynamoDB\""
}
```

Function Logs

```
START RequestId: bd9363c9-c024-4f46-99ed-6459f86e7753 Version: $LATEST
Reading file from S3...
Loading JSON content...
Putting item into DynamoDB...
Function executed in 4.615940093994141 seconds
END RequestId: bd9363c9-c024-4f46-99ed-6459f86e7753
REPORT RequestId: bd9363c9-c024-4f46-99ed-6459f86e7753  Duration: 4696.72 ms    Billed Duration: 4697 ms
```

Step 10: Verify Data in DynamoDB

- Navigate to DynamoDB > Tables > Select your table > Explore items.
- Here, you will see the items that have been successfully uploaded.



id (Number)	age	name
37	20	laukik

Conclusion: To conclude, this experiment aimed to create an AWS Lambda function that processes a JSON file uploaded to an S3 bucket and stores the data in a DynamoDB table. Several issues were encountered, including JSON parsing errors, type mismatches, and timeout errors, but these were resolved by correcting the JSON format, ensuring proper type alignment, and improving execution time. In the end, the Lambda function successfully processed the JSON file and stored its contents in DynamoDB, demonstrating effective integration of AWS services in a serverless environment.