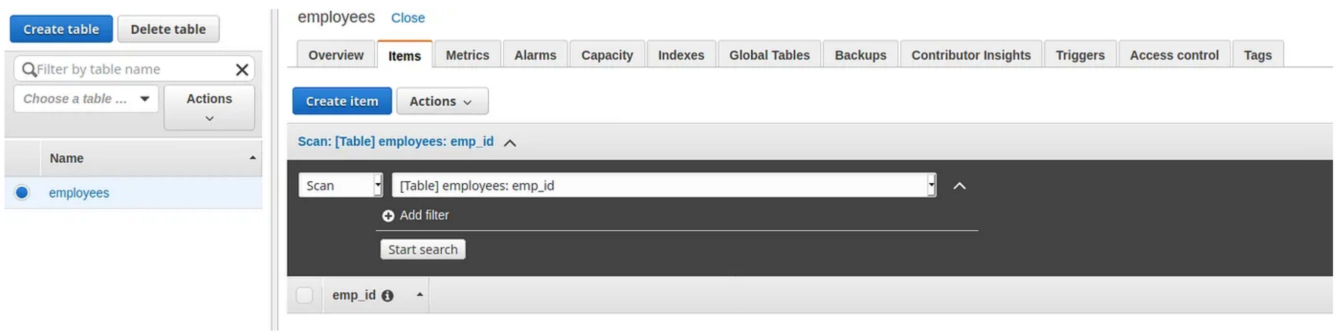
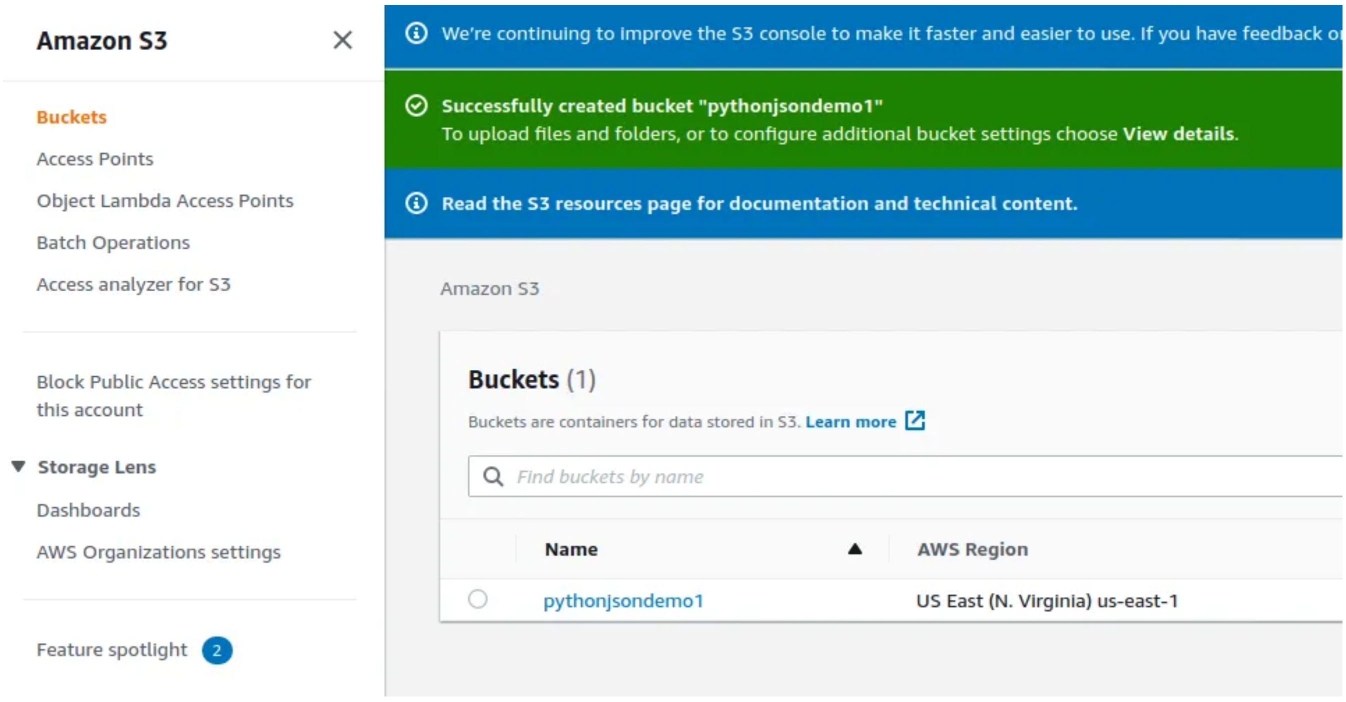
AWS Lambda to a DynamoDB Table from S3

In this scenario we are going to be creating an AWS Lambda in Python to automatically process any JSON files uploaded to an S3 bucket into a DynamoDB table.

In DynamoDB I've gone ahead and created a table called "employees" and the the primary key is employee ID. It can be anything you like.



Next let's create the S3 bucket, where we will be placing the JSON files to be processed by a Lambda function we configure.



We're continuing to improve the S3 console to make it faster and easier to use. If you have feedback o

Successfully created bucket "pythonjsondemo1"
To upload files and folders, or to configure additional bucket settings choose **View details**.

Read the S3 resources page for documentation and technical content.

Amazon S3

Buckets (1)

Buckets are containers for data stored in S3. Learn more

Find buckets by name

Name	AWS Region
pythonjsondemo1	US East (N. Virginia) us-east-1

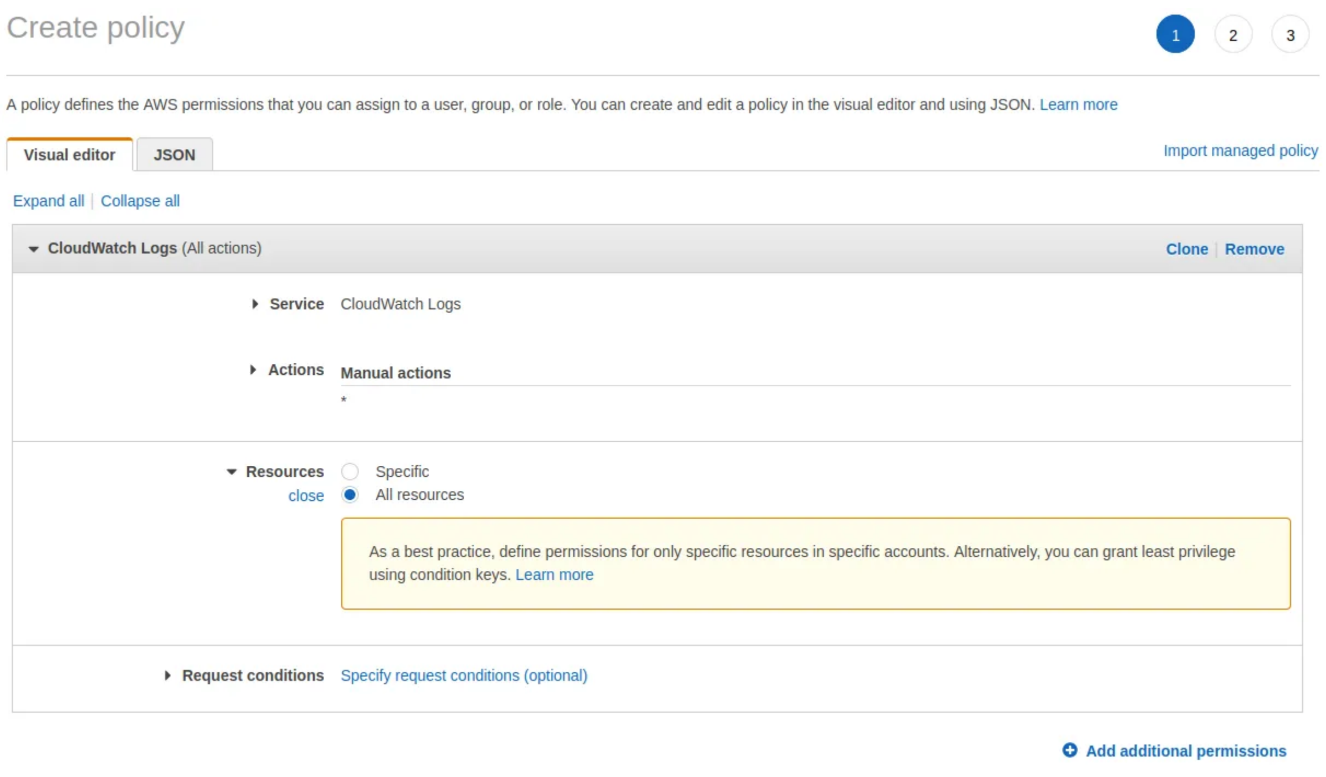
Before we can set up our Lambda function, we need to set up an IAM role for it first.

A few things we need it to have permissions for:

- access to S3
- access to DynamoDB
- access to CloudWatch logs

Go into IAM

We will create the policy first. Select Actions, then "All CloudWatch Logs, then under resources select "All Resources". Then add additional permissions. S3 do the same actions and resources. In a professional environment, never give more permissions than is needed. We would normally select specific resources, etc. As you can see you are warned by AWS.



Create policy

1 2 3

A policy defines the AWS permissions that you can assign to a user, group, or role. You can create and edit a policy in the visual editor and using JSON. Learn more

Visual editor | JSON | Import managed policy

Expand all | Collapse all

CloudWatch Logs (All actions) — Clone | Remove

Service: CloudWatch Logs

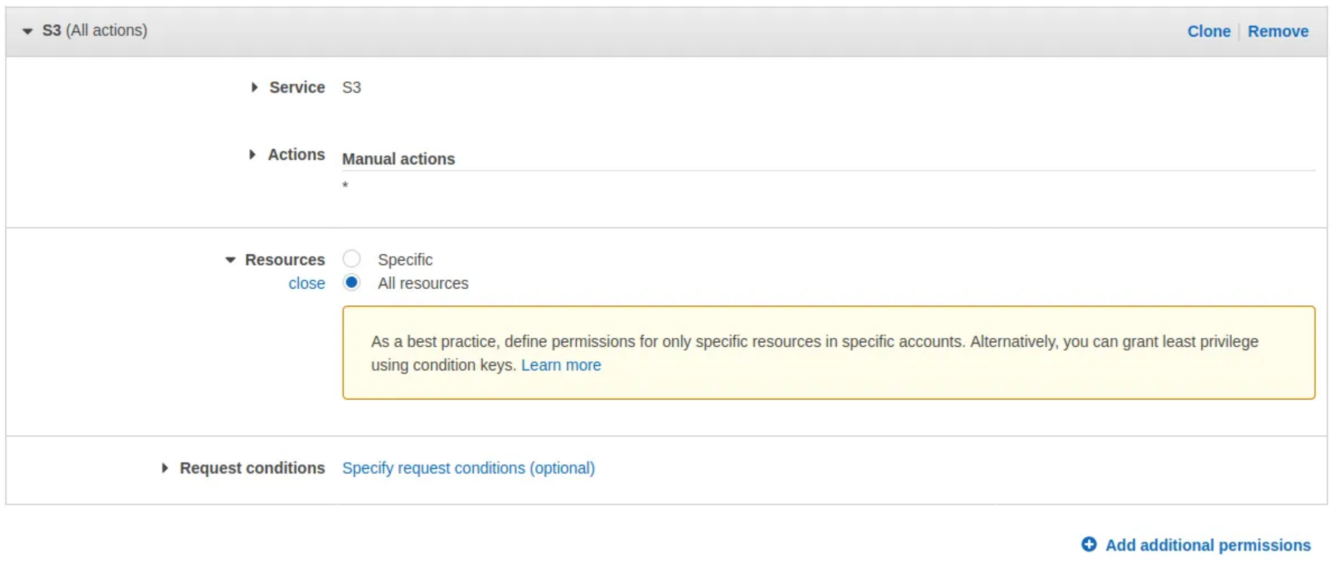
Actions: Manual actions
*

Resources (close): ( ) Specific [✓ All resources]

As a best practice, define permissions for only specific resources in specific accounts. Alternatively, you can grant least privilege using condition keys. Learn more

Request conditions: Specify request conditions (optional)

Add additional permissions



S3 (All actions) — Clone | Remove

Service: S3

Actions: Manual actions
*

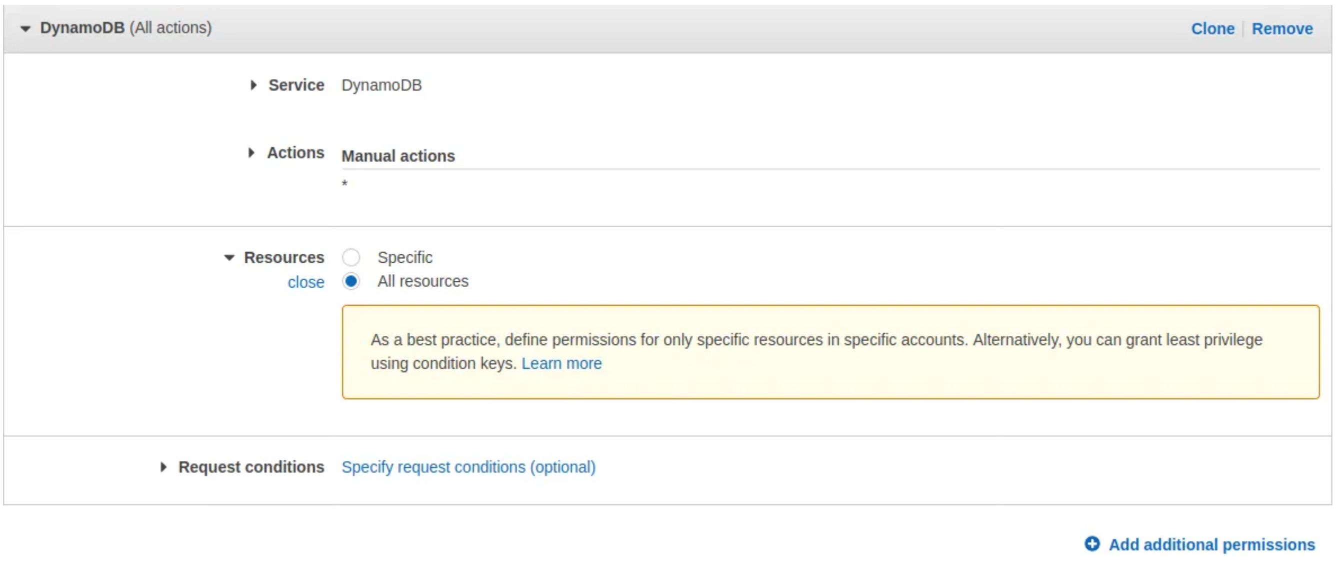
Resources (close): ( ) Specific [✓ All resources]

As a best practice, define permissions for only specific resources in specific accounts. Alternatively, you can grant least privilege using condition keys. Learn more

Request conditions: Specify request conditions (optional)

Add additional permissions

follow with similar selections for S3



DynamoDB (All actions) — Clone | Remove

Service DynamoDB

Actions **Manual actions**
*

Resources ○ Specific
close ● All resources

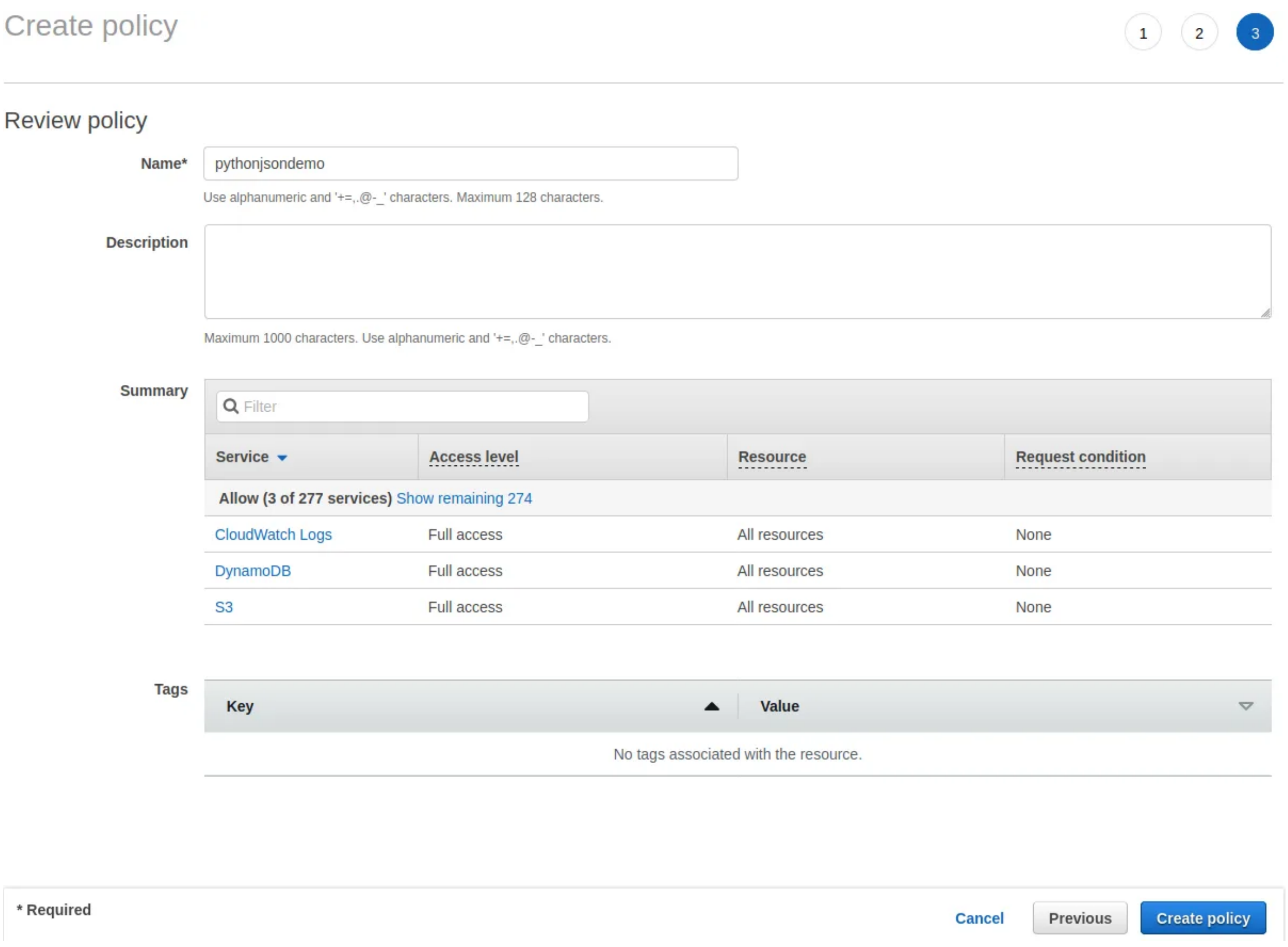
As a best practice, define permissions for only specific resources in specific accounts. Alternatively, you can grant least privilege using condition keys. Learn more

Request conditions Specify request conditions (optional)

Add additional permissions

Last we do the same for DynamoDB.

Name and create the policy.



Create policy — 1 2 3

Review policy

Name* pythonjsondemo
Use alphanumeric and '+=,.@-_' characters. Maximum 128 characters.

Description
Maximum 1000 characters. Use alphanumeric and '+=,.@-_' characters.

Summary

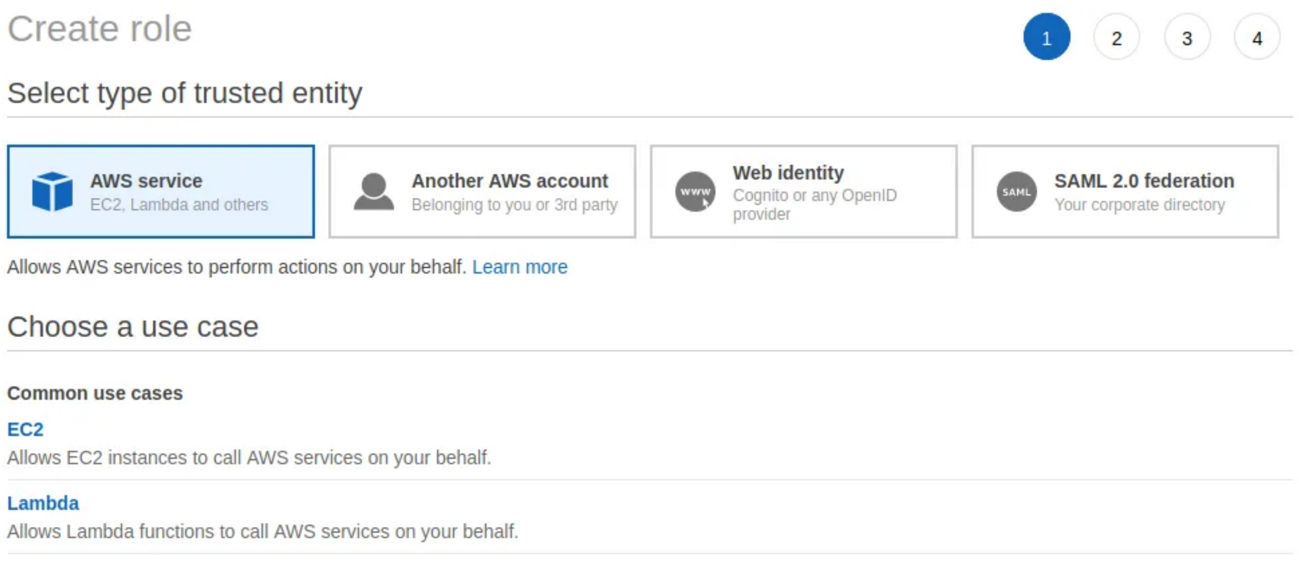
Service	Access level	Resource	Request condition
Allow (3 of 277 services) Show remaining 274			
CloudWatch Logs	Full access	All resources	None
DynamoDB	Full access	All resources	None
S3	Full access	All resources	None

Tags

Key	Value
No tags associated with the resource.	

*** Required** — Cancel | Previous | Create policy

Now we move on to creating the Role.



Select type of trusted entity

Allows AWS services to perform actions on your behalf. Learn more

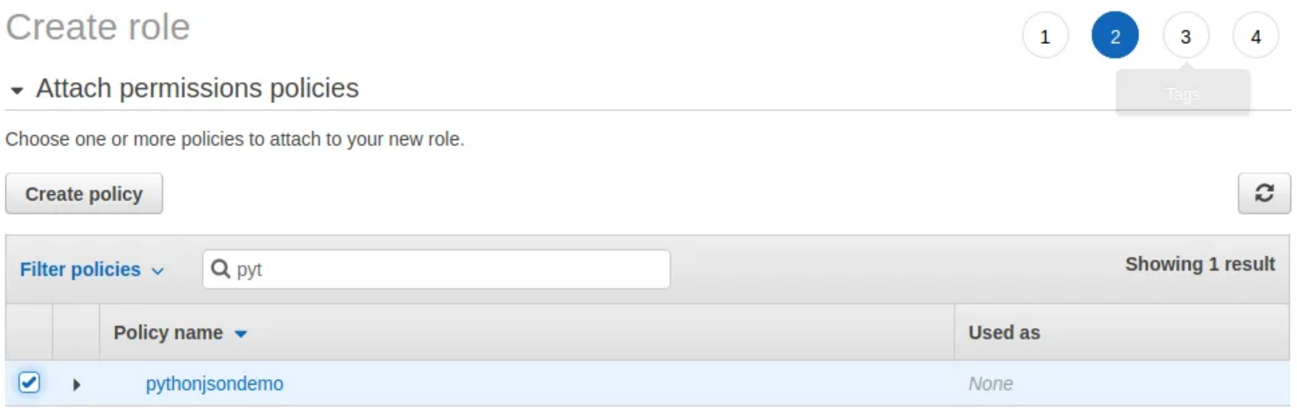
Choose a use case

Common use cases

EC2
Allows EC2 instances to call AWS services on your behalf.

Lambda
Allows Lambda functions to call AWS services on your behalf.

Select Lambda and then move to permissions and find our freshly created policy.



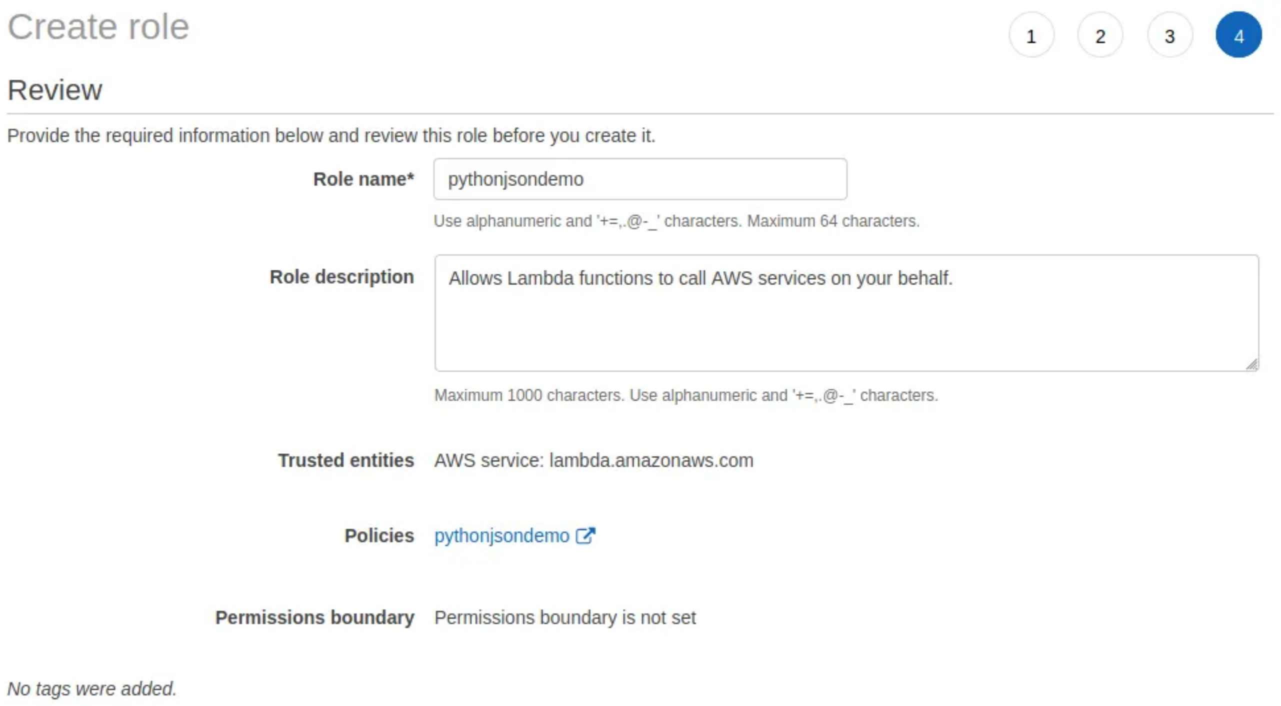
Attach permissions policies

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

pyt

Showing 1 result

		Policy name	Used as
✓	▸	pythonjsondemo	*None*



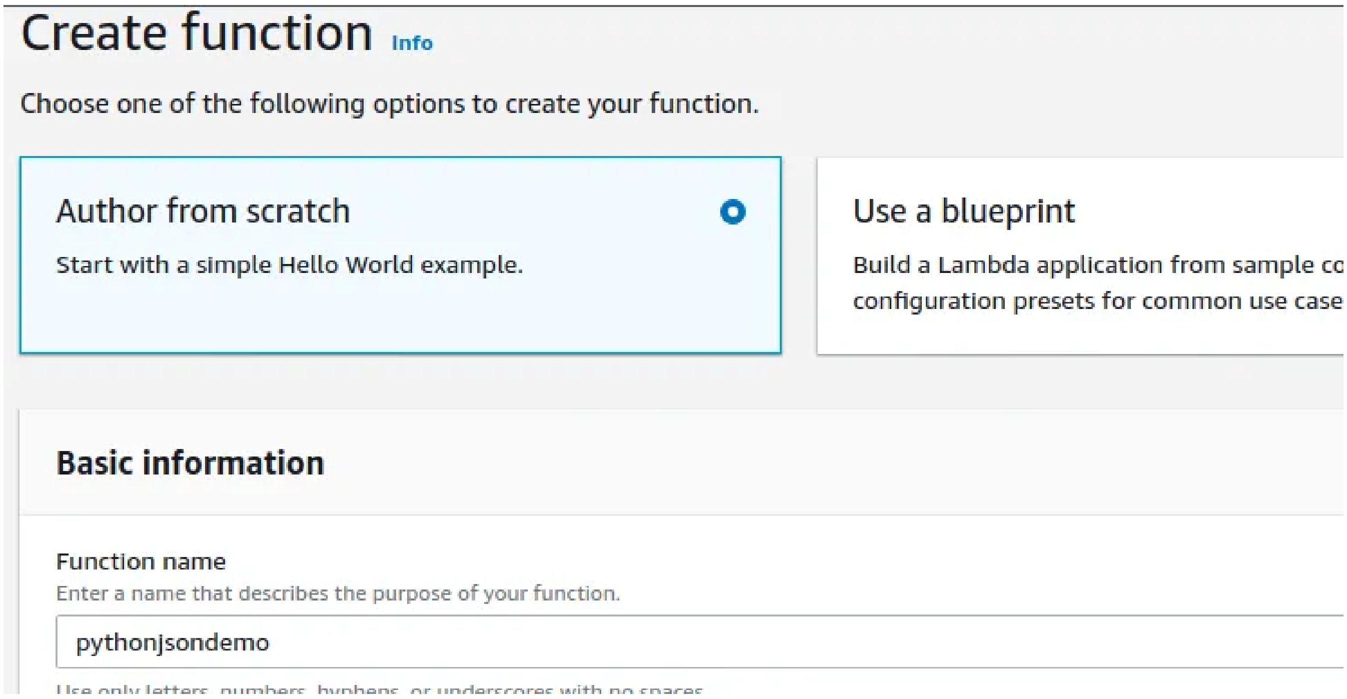
Review

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

Role name*	pythonjsondemo Use alphanumeric and '+=,.@-_' characters. Maximum 64 characters.
Role description	Allows Lambda functions to call AWS services on your behalf. Maximum 1000 characters. Use alphanumeric and '+=,.@-_' characters.
Trusted entities	AWS service: lambda.amazonaws.com
Policies	pythonjsondemo
Permissions boundary	Permissions boundary is not set

No tags were added.

Now it's time to create our Lambda function. We will go to create function, name it, choose python as your Runtime. Under execution role you're going to select the one we just configured.



Create function Info

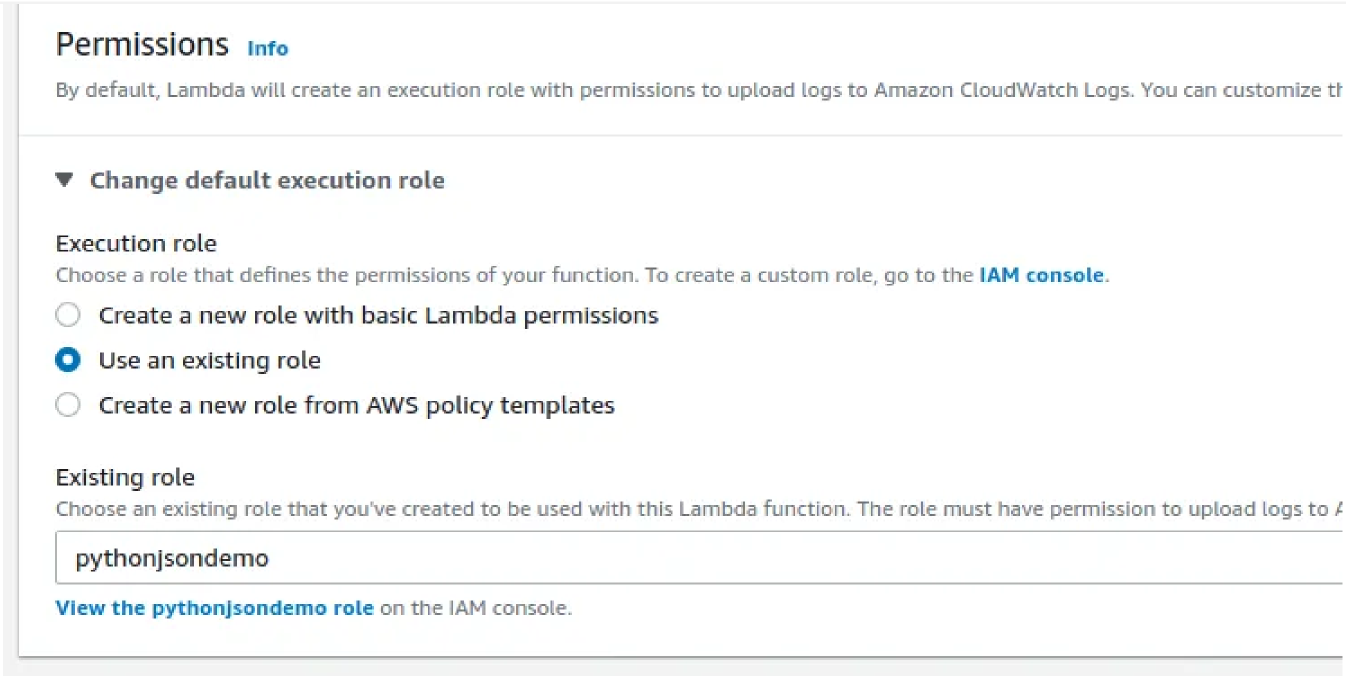
Choose one of the following options to create your function.

[✓ Author from scratch] Start with a simple Hello World example.

Use a blueprint Build a Lambda application from sample co configuration presets for common use case

Basic information

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



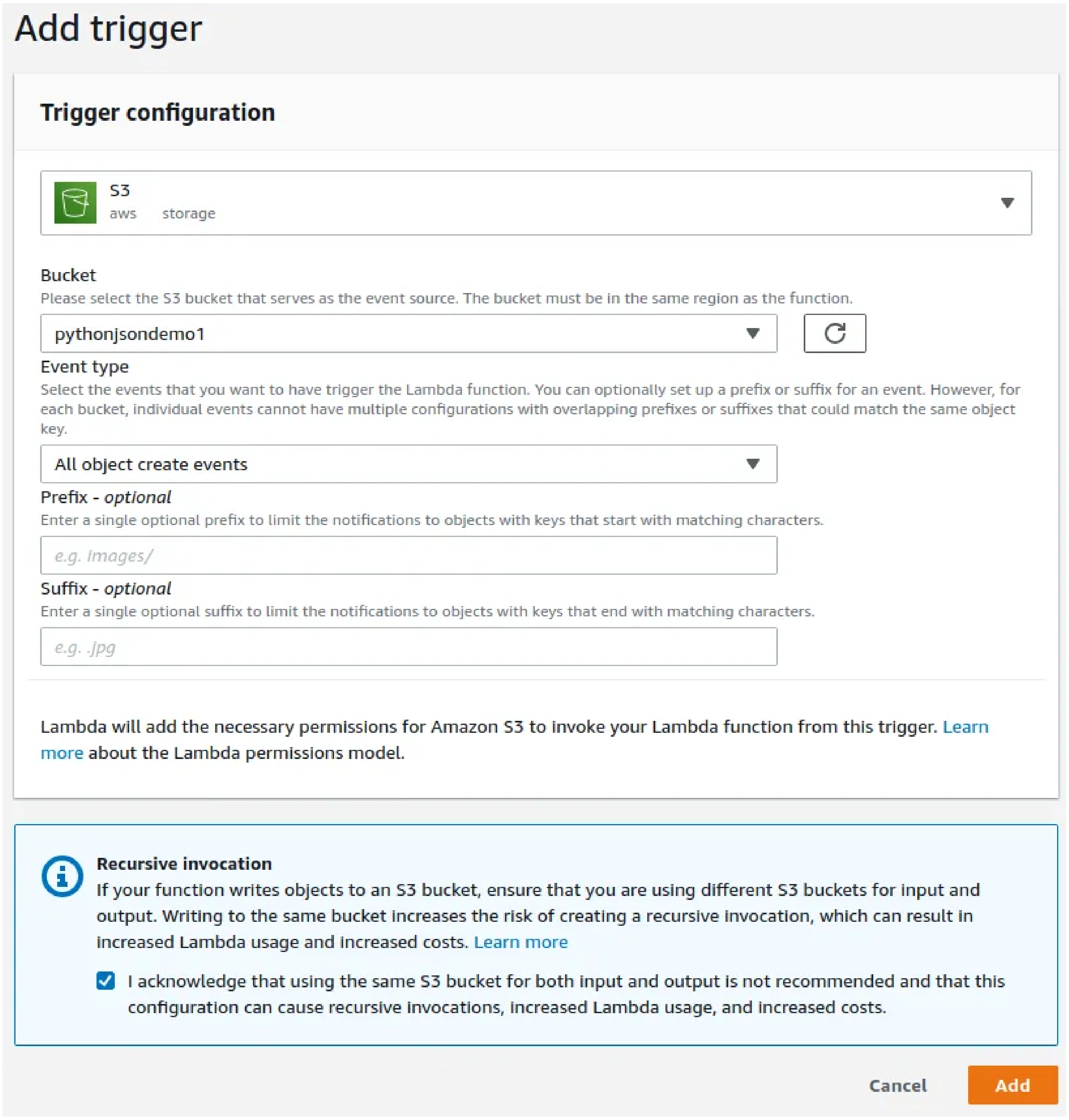
Permissions Info
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize th

▼ 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 A
pythonjsondemo
View the pythonjsondemo role on the IAM console.

Go to add trigger, choose our bucket. When an object is created in the bucket, it will trigger the event for us. So Event type will be All Object Created. Suffix should be .json. So whenever a file with that extenstion is uploaded, it triggers our function.



Add trigger

Trigger configuration

S3
aws storage

Bucket
Please select the S3 bucket that serves as the event source. The bucket must be in the same region as the function.

pythonjsondemo1

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.

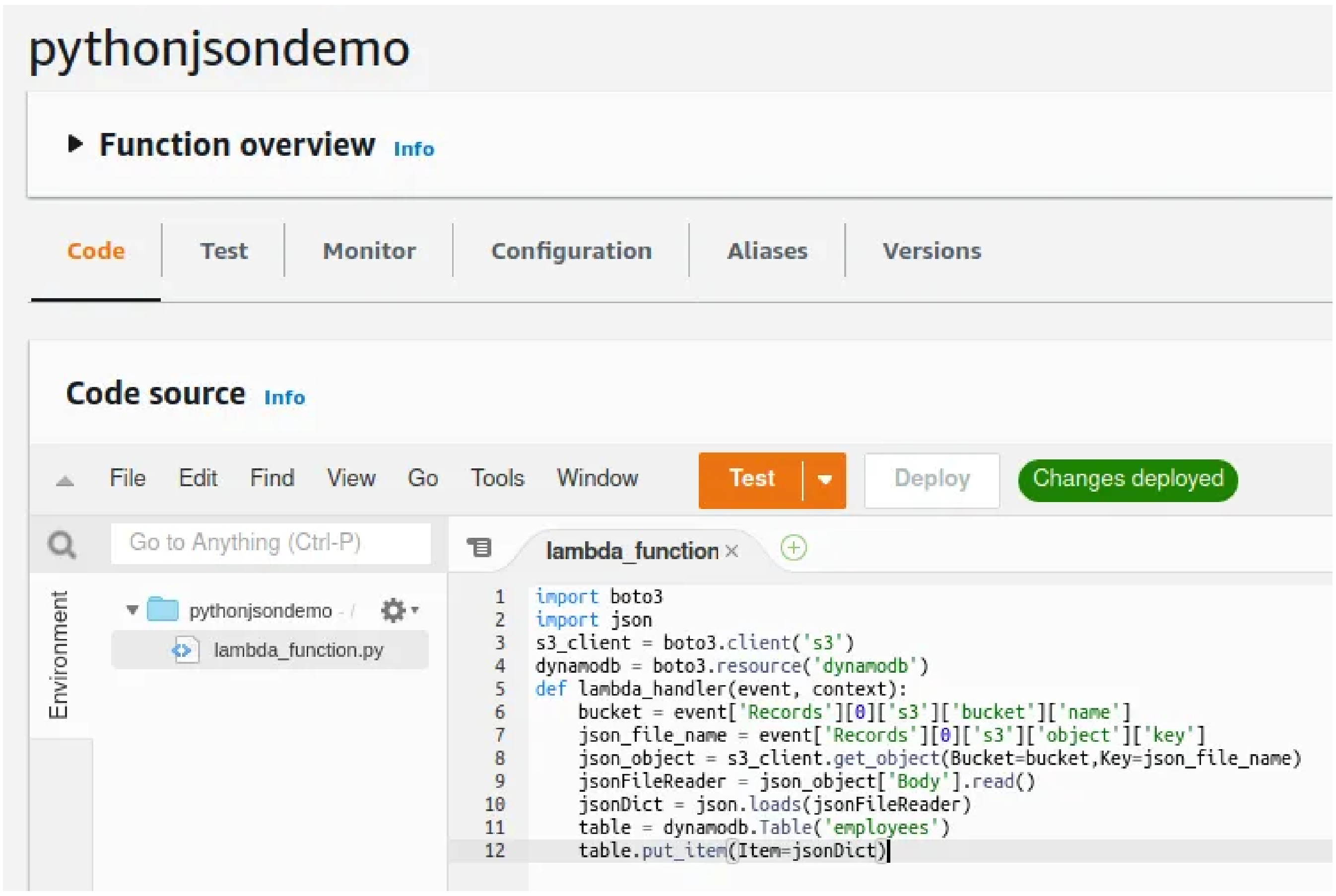
Recursive invocation
If your function writes objects to an S3 bucket, ensure that you are using different S3 buckets for input and output. Writing to the same bucket increases the risk of creating a recursive invocation, which can result in increased Lambda usage and increased costs. Learn more

✓ 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.

Cancel Add

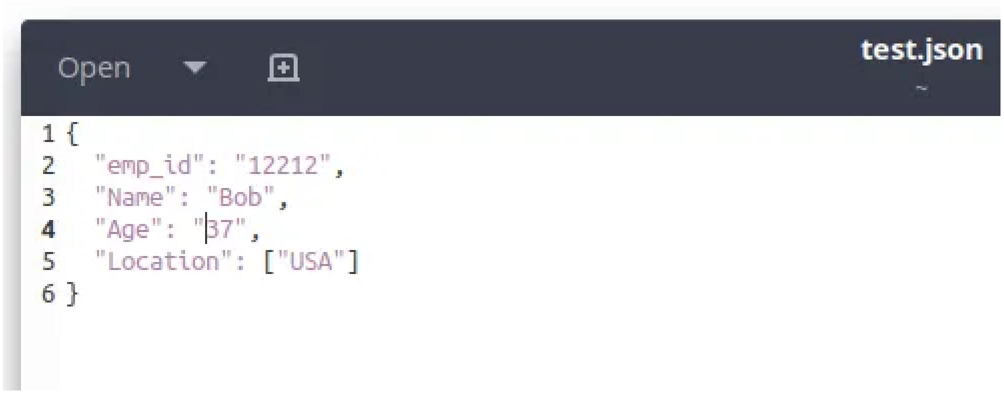
Here is our code for the lambda function. Let's break down exactly what we're doing. First, we're importing the boto3 and json Python modules. boto3 is the AWS SDK for Python. Directing our function to get the different properties our function will need to reference such as bucket name from the s3 object,etc. Essentially telling our modules where to collect all of the information to reference, and what dynamoDB table to use and what to move. Be sure to update it with your DyanamoDB table name.

```python
import boto3
import json
s3_client = boto3.client('s3')
dynamodb = boto3.resource('dynamodb')
def lambda_handler(event, context):
    bucket = event['Records'][0]['s3']['bucket']['name']
    json_file_name = event['Records'][0]['s3']['object']['key']
    json_object = s3_client.get_object(Bucket=bucket,Key=json_file_name)
    jsonFileReader = json_object['Body'].read()
    jsonDict = json.loads(jsonFileReader)
    table = dynamodb.Table('employees')
    table.put_item(Item=jsonDict)
```



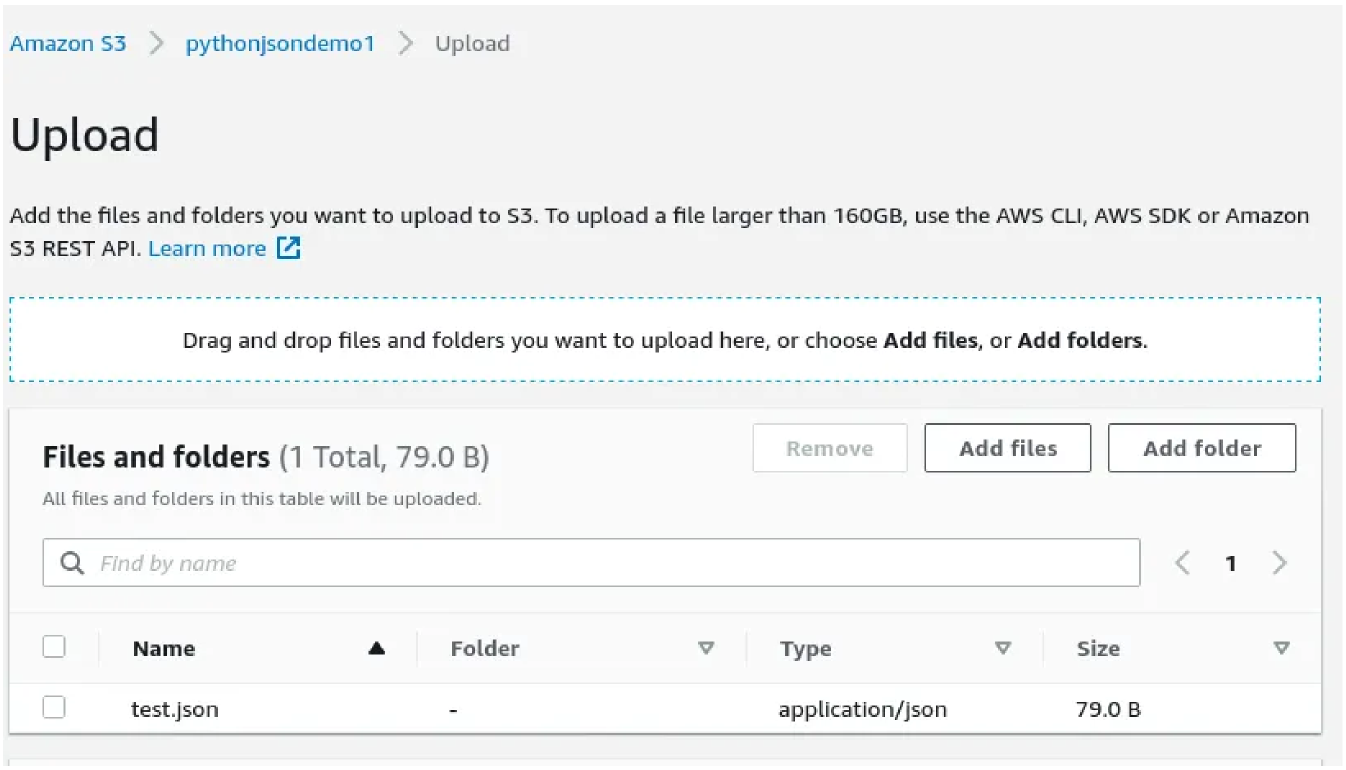
```python
import boto3
import json
s3_client = boto3.client('s3')
dynamodb = boto3.resource('dynamodb')
def lambda_handler(event, context):
    bucket = event['Records'][0]['s3']['bucket']['name']
    json_file_name = event['Records'][0]['s3']['object']['key']
    json_object = s3_client.get_object(Bucket=bucket,Key=json_file_name)
    jsonFileReader = json_object['Body'].read()
    jsonDict = json.loads(jsonFileReader)
    table = dynamodb.Table('employees')
    table.put_item(Item=jsonDict)
```

Now let’s test, upload a json file to the s3 bucket. Some misc data to be entered into our DynamoDB table.



```
{
  "emp_id": "12212",
  "Name": "Bob",
  "Age": "37",
  "Location": ["USA"]
}
```

Our new employee Bob needs his info entered



Amazon S3 > pythonjsondemo1 > Upload

Upload

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

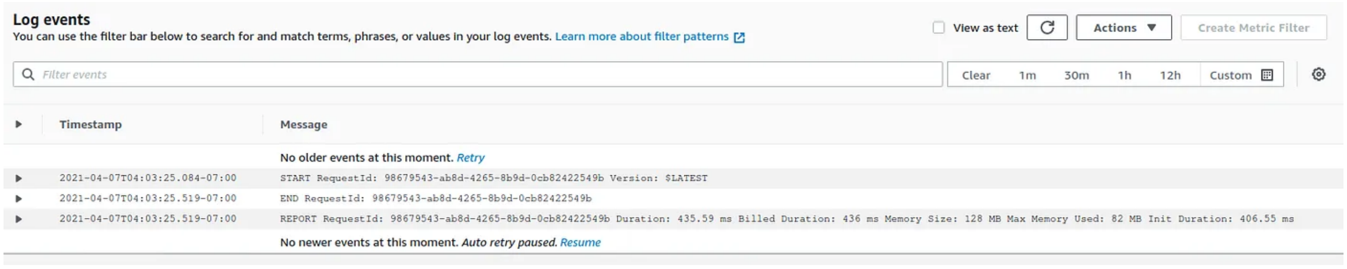
Drag and drop files and folders you want to upload here, or choose **Add files**, or **Add folders**.

Files and folders (1 Total, 79.0 B)

All files and folders in this table will be uploaded.

Name	Folder	Type	Size
test.json	-	application/json	79.0 B

Check our CloudWatch Logs to make sure everything ran in our code without error. Go under log events and you will see the Lambda function.



Log events

You can use the filter bar below to search for and match terms, phrases, or values in your log events. Learn more about filter patterns

```
Timestamp                       Message
                                No older events at this moment. Retry
2021-04-07T04:03:25.084-07:00   START RequestId: 98679543-ab8d-4265-8b9d-0cb82422549b Version: $LATEST
2021-04-07T04:03:25.519-07:00   END RequestId: 98679543-ab8d-4265-8b9d-0cb82422549b
2021-04-07T04:03:25.519-07:00   REPORT RequestId: 98679543-ab8d-4265-8b9d-0cb82422549b Duration: 435.59 ms Billed Duration: 436 ms Memory Size: 128 MB Max Memory Used: 82 MB Init Duration: 406.55 ms
                                No newer events at this moment. Auto retry paused. Resume
```

Lastly we check DynamoDB. If everything went as expected, our info has now been added to the table.