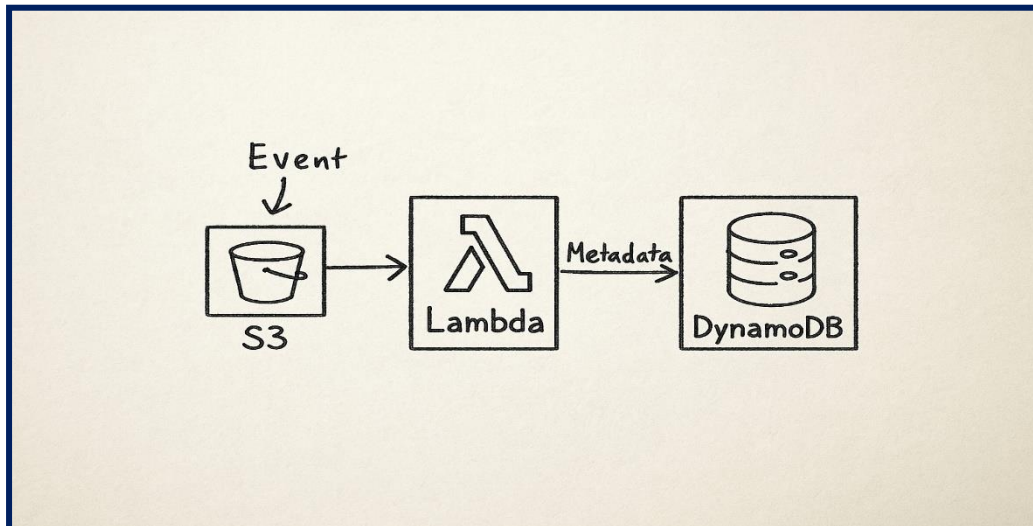


Automating Metadata Flow from S3 to DynamoDB with AWS Lambda.

This project automates the workflow of capturing metadata from newly uploaded objects in an S3 bucket. Upon each upload, an AWS Lambda function is triggered to extract the object's metadata and store it in a DynamoDB table—enabling seamless, serverless metadata tracking.

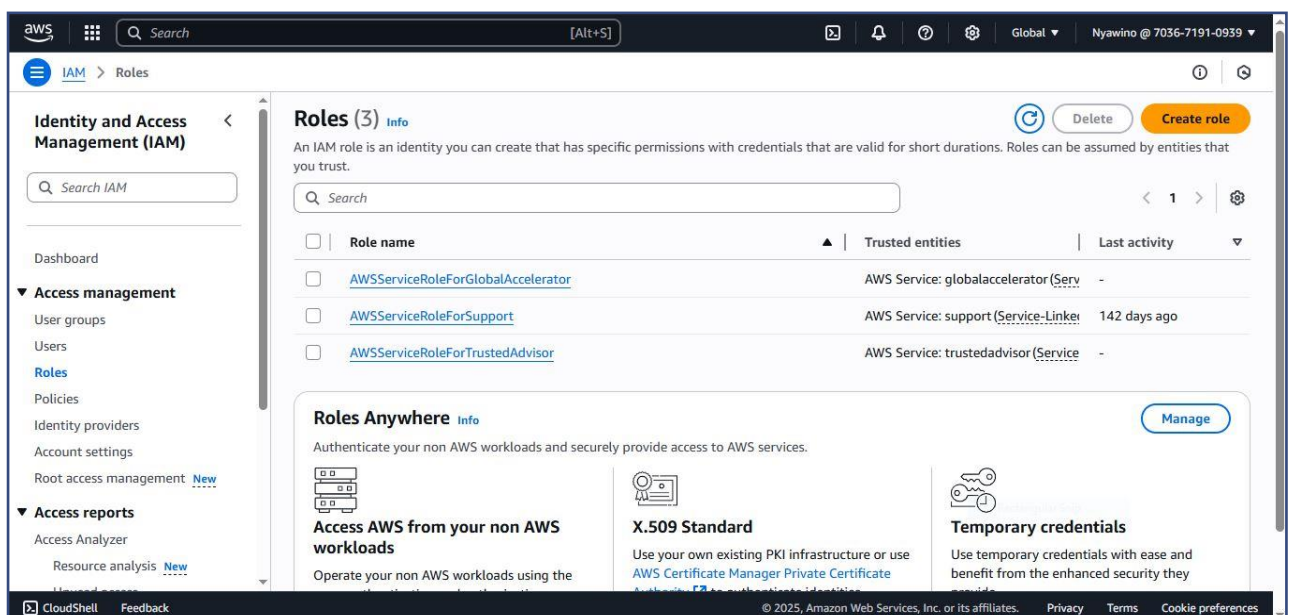


Architecture Overview

1. **Trigger:** S3 Event (e.g., `s3:ObjectCreated:*`)
2. **Action:** Lambda Function (Python)
3. **Target:** DynamoDB Table

Step 1: Create an IAM Role for Lambda

- Go to the **IAM Console**
- Click **Roles** —> **Create Role**



- Use Case → **Lambda**. Click Next

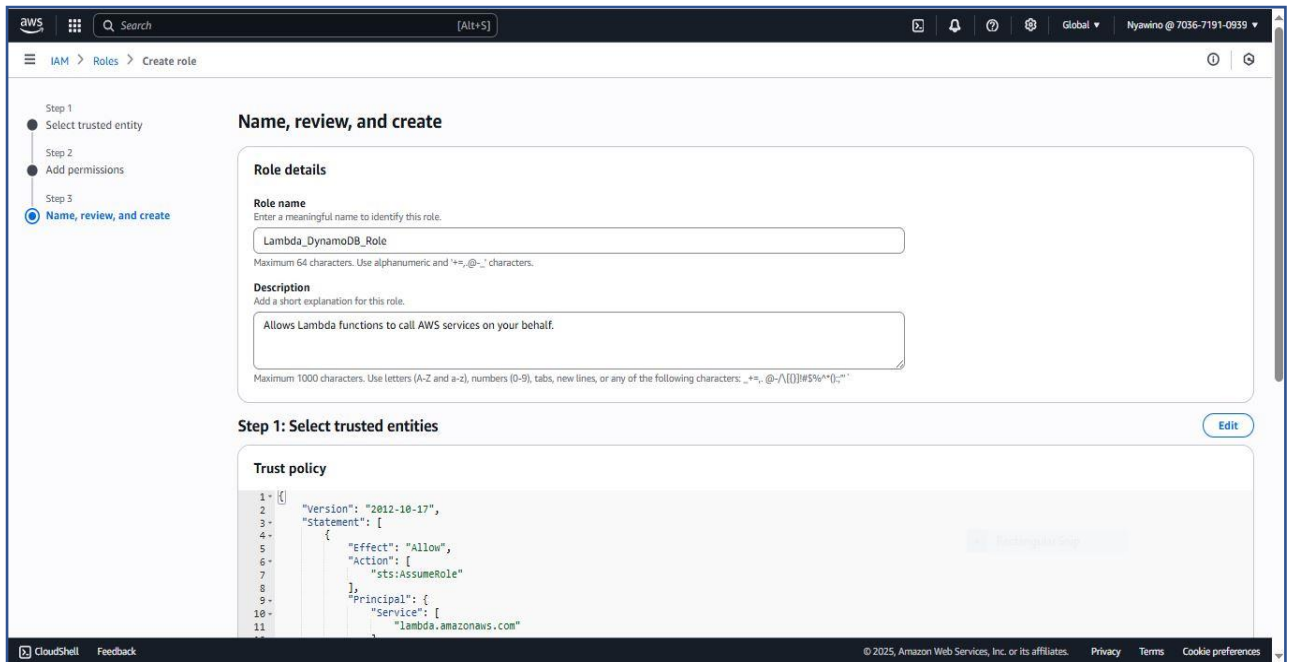
The screenshot shows the 'Select trusted entity' step in the AWS IAM console. The left sidebar indicates the current step is 'Step 1: Select trusted entity'. The main content area is titled 'Select trusted entity' and includes a 'Trusted entity type' section with five options: 'AWS service' (selected), 'AWS account', 'Web identity', 'SAML 2.0 federation', and 'Custom trust policy'. Below this is a 'Use case' section with a dropdown menu set to 'Lambda' and a list of use cases, with 'Lambda' (allows Lambda functions to call AWS services on your behalf) selected. At the bottom right are 'Cancel' and 'Next' buttons.

- **Add Permissions:**
 - ❖ *AmazonDynamoDBFullAccess*
 - ❖ *AmazonS3ReadOnlyAccess*
- Click Next

The screenshot shows the 'Add permissions' step in the AWS IAM console. The left sidebar indicates the current step is 'Step 2: Add permissions'. The main content area is titled 'Add permissions' and includes a search bar with 'Dy' entered, showing 6 matches. A table lists several policies, with 'AmazonDynamoDBFullAccess' selected. The table has columns for 'Policy name', 'Type', and 'Description'. Below the table is a section for 'Set permissions boundary - optional'. At the bottom right are 'Cancel', 'Previous', and 'Next' buttons.

Policy name	Type	Description
<input checked="" type="checkbox"/> AmazonDynamoDBFullAccess	AWS managed	Provides full access to Amazon Dynamo...
<input type="checkbox"/> AmazonDynamoDBFullAccess_v2	AWS managed	Provides full access to Amazon DynamoDB
<input type="checkbox"/> AmazonDynamoDBFullAccessWithDataPipeline	AWS managed	This policy is on a deprecation path. See ...
<input type="checkbox"/> AmazonDynamoDBReadOnlyAccess	AWS managed	Provides read only access to Amazon Dyn...
<input type="checkbox"/> AWSLambdaDynamoDBExecutionRole	AWS managed	Provides list and read access to Dynamo...
<input type="checkbox"/> AWSLambdaInvocation-DynamoDB	AWS managed	Provides read access to DynamoDB Strea...

- **Role Name** for example: *Lambda_DyanmoDB_Role*
- Review the Role.
- Click **Create Role**

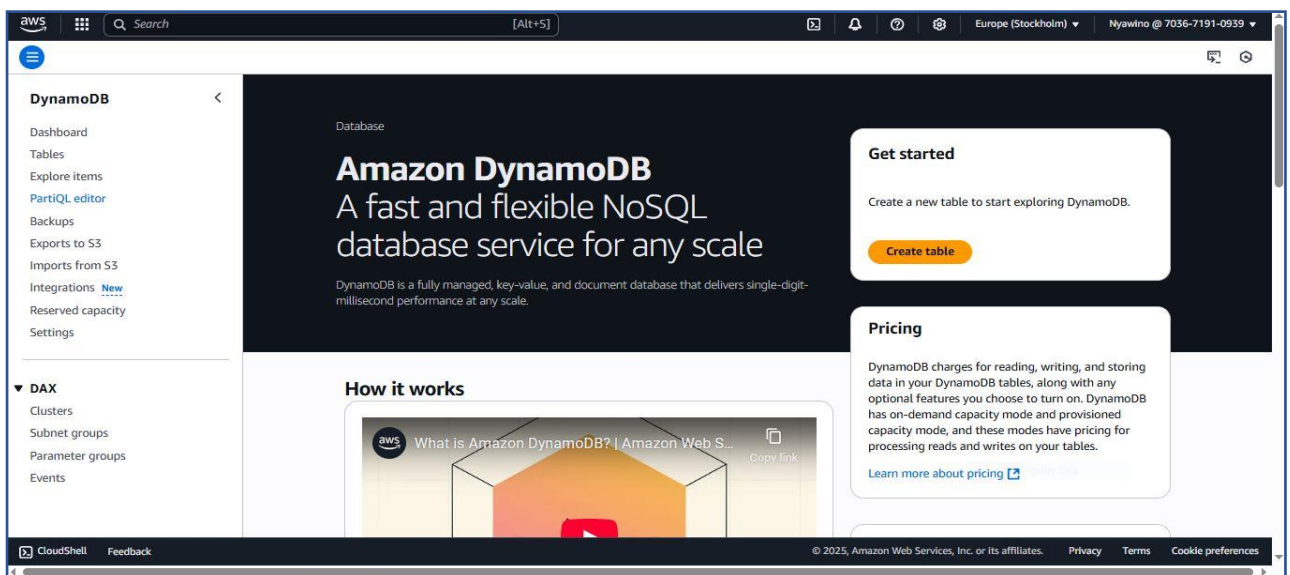


Step 2: Create an S3 Bucket.

- On the AWS Console: Search for **S3**
- Click **Create a Bucket**.
- Fill in:
 - ❖ Bucket Name: for example, *mys3bucket*
 - ❖ Object Ownership: ACL enabled
 - ❖ Disable Block Public Access
- Click **Create a Bucket**.

Step 3: Create a DynamoDB Table

- On the AWS Console, search for **AWS DynamoDB**
- Click **Create Table**



- Create a table and use default settings
- Fill in:
 - ❖ Table Name: e.g., *Newdynamodbtable*
 - ❖ Partition Key: unique
 - ❖ Table Settings: Leave as Default

The screenshot shows the 'Create table' page in the AWS Management Console. The page is titled 'Create table' and has a breadcrumb trail: 'DynamoDB > Tables > Create table'. The main content area is divided into three sections: 'Table details', 'Partition key', and 'Table settings'.

Table details (Info icon): 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. The input field contains 'S3DynamoDBtable'. Below the input field, it says: '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. The input field contains 'unique'. To the right, there is a dropdown menu set to 'String'. Below the input field, it says: '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. The input field is empty with the placeholder text 'Enter the sort key name'. To the right, there is a dropdown menu set to 'String'. Below the input field, it says: '1 to 255 characters and case sensitive'.

Table settings: There are two radio buttons. The first is 'Default settings' (selected), with the description: 'The fastest way to create your table. You can modify most of these settings after your table has been created. To modify these settings now, choose "Customize settings".' The second is 'Customize settings', with the description: 'Use these advanced features to make DynamoDB work better for your needs.'

The footer of the console shows 'CloudShell', 'Feedback', and copyright information: '© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences'.

- Click **Create Table**

Step 4: Create a Lambda Function.

AWS Lambda is a serverless computing services that allows you to run code without provisioning or managing services.

- On the AWS Console, search for **AWS Lambda**
- Click **Create Function**

The screenshot shows the 'Get started' page for AWS Lambda in the AWS Management Console. The page has a dark header with the AWS logo and a search bar. Below the header, the main content area has a dark background with the text 'Compute' and 'AWS Lambda lets you run code without thinking about servers.' Below this, it says: '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.'

On the right side, there is a white box titled 'Get started' with the text: 'Author a Lambda function from scratch, or choose from one of many preconfigured examples.' Below this text is an orange button that says 'Create a function'.

At the bottom of the page, there is a section titled 'How it works' with a 'Run' button. Below this, there are tabs for different runtimes: '.NET', 'Java', 'Node.js' (selected), 'Python', 'Ruby', and 'Custom runtime'. Below the tabs, there is a code editor showing the following code:

```
1 * exports.handler = async (event) => {
2   console.log(event);
}
```

The footer of the console shows 'CloudShell', 'Feedback', and copyright information: '© 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences'.

- Choose:
 - ❖ Author from Scratch
 - ❖ Name: S3ToDynamoFunction
 - ❖ Runtime: Python 3.10/3.9/3.11)
 - ❖ Permissions: Use existing role: select Lambda_DynamoDB_Role
- Click **Create function**

Step 5: Add the Lambda Code.

In the Lambda editor, **paste this code**:

```
import boto3
from uuid import uuid4

s3 = boto3.client('s3')
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('S3DynamoDBtable')

def lambda_handler(event, context):
    for record in event['Records']:
        bucket_name = record['s3']['bucket']['name']
        object_key = record['s3']['object']['key']
        event_name = record['eventName']
        event_time = record['eventTime']

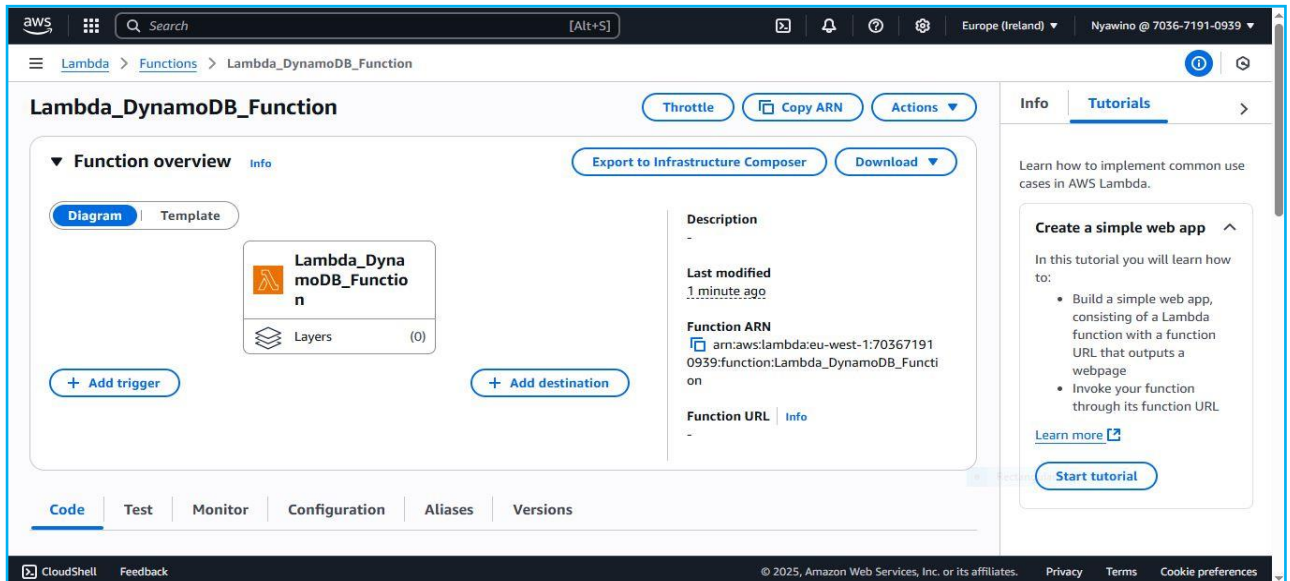
        # Get object metadata
        try:
            meta = s3.head_object(Bucket=bucket_name, Key=object_key)
            size = meta['ContentLength']
        except Exception as e:
            print(f"Failed to get metadata: {e}")
            size = -1

        table.put_item(Item={
            'unique': str(uuid4()),
            'Bucket': bucket_name,
            'Object': object_key,
            'Size': size,
            'Event_name': event_name,
            'Event_time': event_time
        })
```

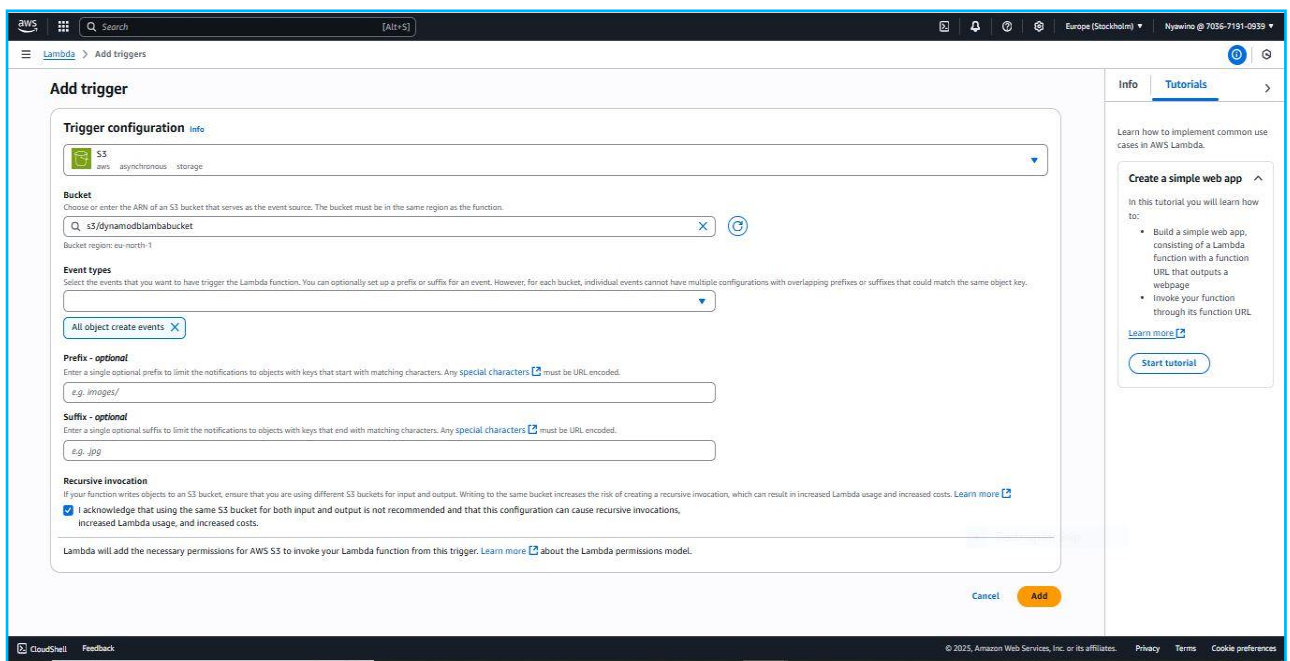
- Click **Deploy**

Step 6: Add an S3 Trigger

- On the same **Lambda Function** page, go to “**Configuration → Triggers**”
- Click “**Add trigger**”



- Choose:
 - ❖ **Trigger type:** S3
 - ❖ **Bucket:** your bucket name
 - ❖ **Event type:** PUT or All object create events
 - ❖ **Prefix/suffix:** leave blank unless filtering



Step 8: Test the Automation.

- Go to your S3 bucket
- Upload any test file (e.g., `test.txt`)
- Go to **CloudWatch Logs** > **/aws/lambda/S3ToDynamoFunction**
 - ❖ Confirm your function was triggered
 - ❖ Check for errors or success messages
- Go to **DynamoDB** → **Tables** → **S3DynamoDBtable** → **Explore Table Items**
- You should see a new row with the metadata.

The screenshot shows the AWS DynamoDB console interface. On the left is a navigation menu with options like Dashboard, Tables, Explore Items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. The main area is titled 'S3DynamoDBtable' and includes a 'Tables (1)' sidebar with a search bar and a 'Find tables' button. The main content area has a 'Scan or query items' section with a 'Scan' button selected. Below this, a status bar indicates 'Completed · Items returned: 1 · Items scanned: 1 · Efficiency: 100% · RCUs consumed: 2'. A table titled 'Table: S3DynamoDBtable - Items returned (1)' shows the scan results, with columns for unique (String), Bucket, Event_name, Event_time, Object, and Size. The first row contains the values: 43f34b47-980c-4764..., dnnewbuck..., ObjectCreat..., 2025-07-24..., Chrome+Ch..., and -1.

unique (String)	Bucket	Event_name	Event_time	Object	Size
43f34b47-980c-4764...	dnnewbuck...	ObjectCreat...	2025-07-24...	Chrome+Ch...	-1