

Automated File Processing with S3 Event Notifications and Lambda function

You can turn on S3 event notifications to send notifications on S3-related events, such as object uploads or deletions. These notifications can be configured to target AWS services like Lambda functions, SQS queues, or SNS topics

Objectives

Imagine you're a solutions architect at a financial company. Your job is to provide the audit team with the transaction records they need for their checks. But there's a catch due to privacy rules: they can't see full credit card numbers. So, your challenge is to make sure they only get to see the last four digits of each card number. This way, they have just enough information to do their job without compromising customer data.

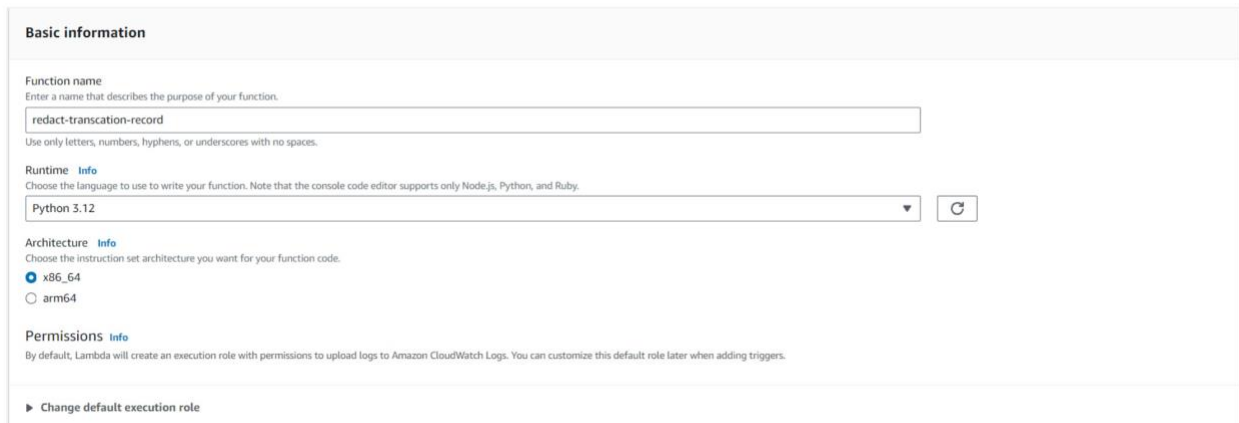
In this lab, you'll set up this system. You'll create an S3 bucket, configure its event notifications to trigger a Lambda function, and write the function logic to process and store a redacted version of the uploaded file in a designated folder within the same bucket.

In this lab, you will:

- Learn how to enable S3 event notifications to invoke a Lambda function
- Learn how to read from and upload files to an S3 bucket using the AWS SDK for Python (Boto3)

Creating the Lambda function

1. Create a Lambda function called **redact-transaction-record** and choose **Python** as the runtime.



The screenshot shows the 'Basic information' tab of the AWS Lambda console. It includes fields for 'Function name' (set to 'redact-transaction-record'), 'Runtime' (set to 'Python 3.12'), and 'Architecture' (set to 'x86_64'). There are also links for 'Permissions' and a 'Change default execution role' button.

Basic information

Function name
Enter a name that describes the purpose of your function.

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime [Info](#)
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Architecture [Info](#)
Choose the instruction set architecture you want for your function code.
☒ x86_64
☐ arm64

Permissions [Info](#)
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

[► Change default execution role](#)

2. For the execution role, click the **Use an existing role** option, then choose **PlayCloud-Sandbox**.

▼ 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.

PlayCloud-Sandbox

[View the PlayCloud-Sandbox role](#) on the IAM console.

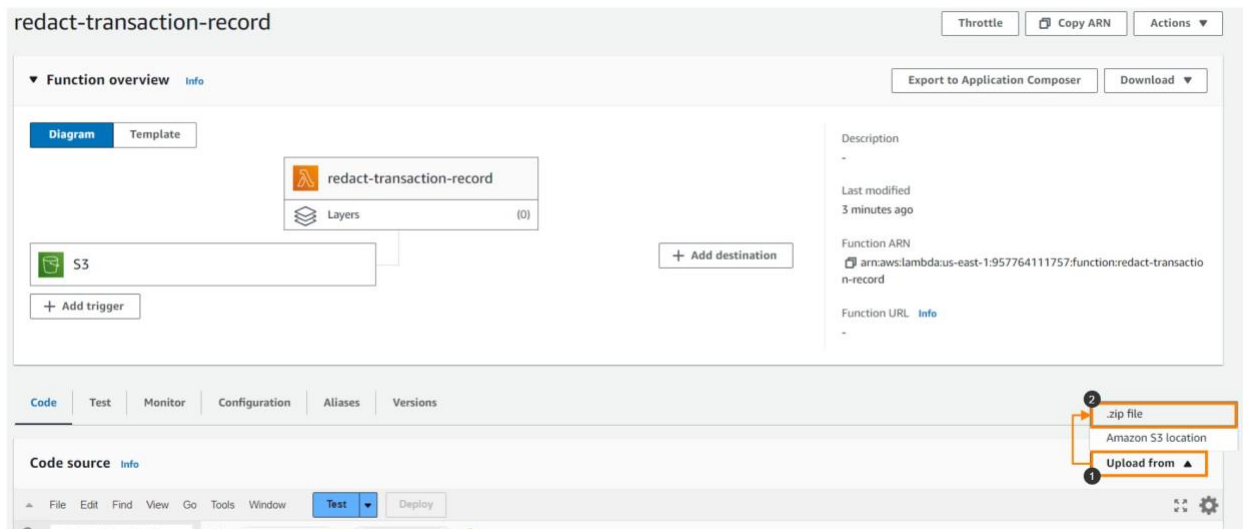
3. Click the **Create function**.

Deploying the Lambda function code

1. Download the following ZIP file on your local machine.

<https://media.tutorialsdojo.com/public/redact-transaction-record.zip>

2. On the AWS Lambda console, click the **Upload from** dropdown box, then click **.zip file**.



3. A dialog box will appear. Upload the ZIP file you downloaded and click on **Save**.

Upload a .zip file

When you upload a new .zip file package, it overwrites the existing code.

Upload

For files larger than 10 MB, consider uploading using Amazon S3.

Cancel

Save

Configuring S3 Event notification

1. Create an S3 bucket with the name **transactions-<your-name>** (for example, **transactions-carlo**) in the **N. Virginia** region. S3 bucket names need to be globally unique. Include your name or any arbitrary characters to ensure that your bucket name won't conflict with existing ones.
2. Once created, click the **Properties** section of your bucket.

Amazon S3 > Buckets > transactions-carlo

transactions-carlo

Objects

Properties

Permissions

Metrics

Management

Access Points

Bucket overview

AWS Region US East (N. Virginia) us-east-1	Amazon Resource Name (ARN) arn:aws:s3::transactions-carlo	Creation date January 12, 2024, 15:29:57 (UTC+08:00)
---	--	---

Bucket Versioning

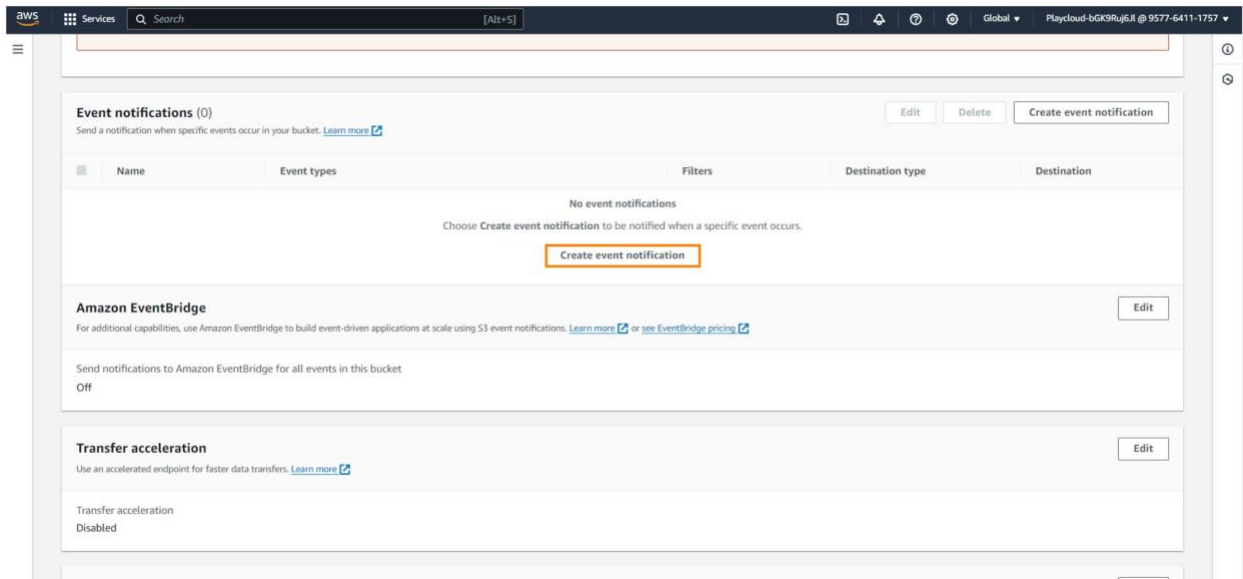
Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. [Learn more](#)

Bucket Versioning
Disabled
Multi-factor authentication (MFA) delete
An additional layer of security that requires multi-factor authentication for changing Bucket Versioning settings and permanently deleting object versions. To modify MFA delete settings, use the AWS CLI, AWS SDK, or the Amazon S3 REST API. [Learn more](#)
Disabled

Tags (0)

You can use bucket tags to track storage costs and organize buckets. [Learn more](#)

3. Scroll down the **Event notifications** setting and click **Create event notification**.



4. In the **General configuration** section, enter the following details:

[Amazon S3](#) > [Buckets](#) > [transactions-carlo](#) > Create event notification

Create event notification [Info](#)

To enable notifications, you must first add a notification configuration that identifies the events you want Amazon S3 to publish and the destinations where you want Amazon S3 to send the notifications.

General configuration

Event name

Event name can contain up to 255 characters.

Prefix - optional
Limit the notifications to objects with key starting with specified characters.

Suffix - optional
Limit the notifications to objects with key ending with specified characters.

5. For Event Types, tick the **Put** checkbox.

Event types

Specify at least one event for which you want to receive notifications. For each group, you can choose an event type for all events, or you can choose one or more individual events.

Object creation

☐ All object create events
s3:ObjectCreated:*

☒ Put
s3:ObjectCreated:Put

☐ Post
s3:ObjectCreated:Post

☐ Copy
s3:ObjectCreated:Copy

☐ Multipart upload completed
s3:ObjectCreated:CompleteMultipartUpload

Object removal

☐ All object removal events
s3:ObjectRemoved:*



☐ Permanently deleted
s3:ObjectRemoved:Delete

☐ Delete marker created
s3:ObjectRemoved:DeleteMarkerCreated


Object restore

6. For **Destination**, choose the **redact-transaction-record** Lambda function. Then, click on **Save changes**.

Destination

 Before Amazon S3 can publish messages to a destination, you must grant the Amazon S3 principal the necessary permissions to call the relevant API to publish messages to an SNS topic, an SQS queue, or a Lambda function. [Learn more](#) 

Destination

Choose a destination to publish the event. [Learn more](#) 

☒ **Lambda function**
Run a Lambda function script based on S3 events.

☐ **SNS topic**
Fanout messages to systems for parallel processing or directly to people.

☐ **SQS queue**
Send notifications to an SQS queue to be read by a server.

Specify Lambda function

☒ Choose from your Lambda functions

☐ Enter Lambda function ARN

Lambda function

redact-transaction-record ▼

Cancel

Save changes

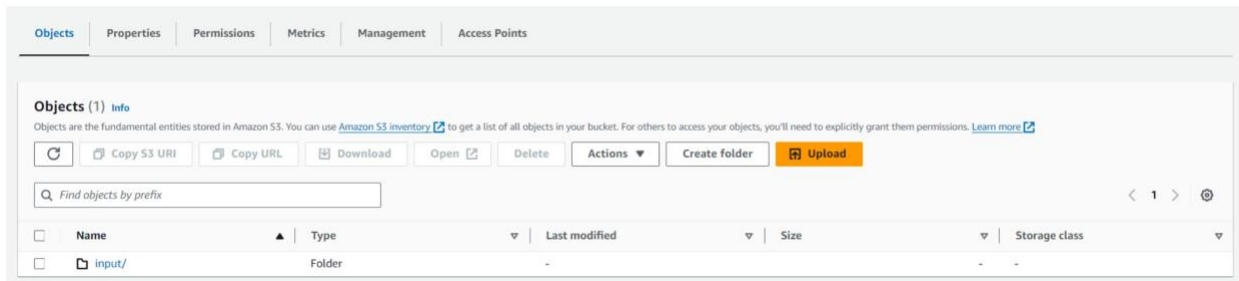
Testing the Configuration

Now that we're done setting up all the required resources, let's do a quick test.

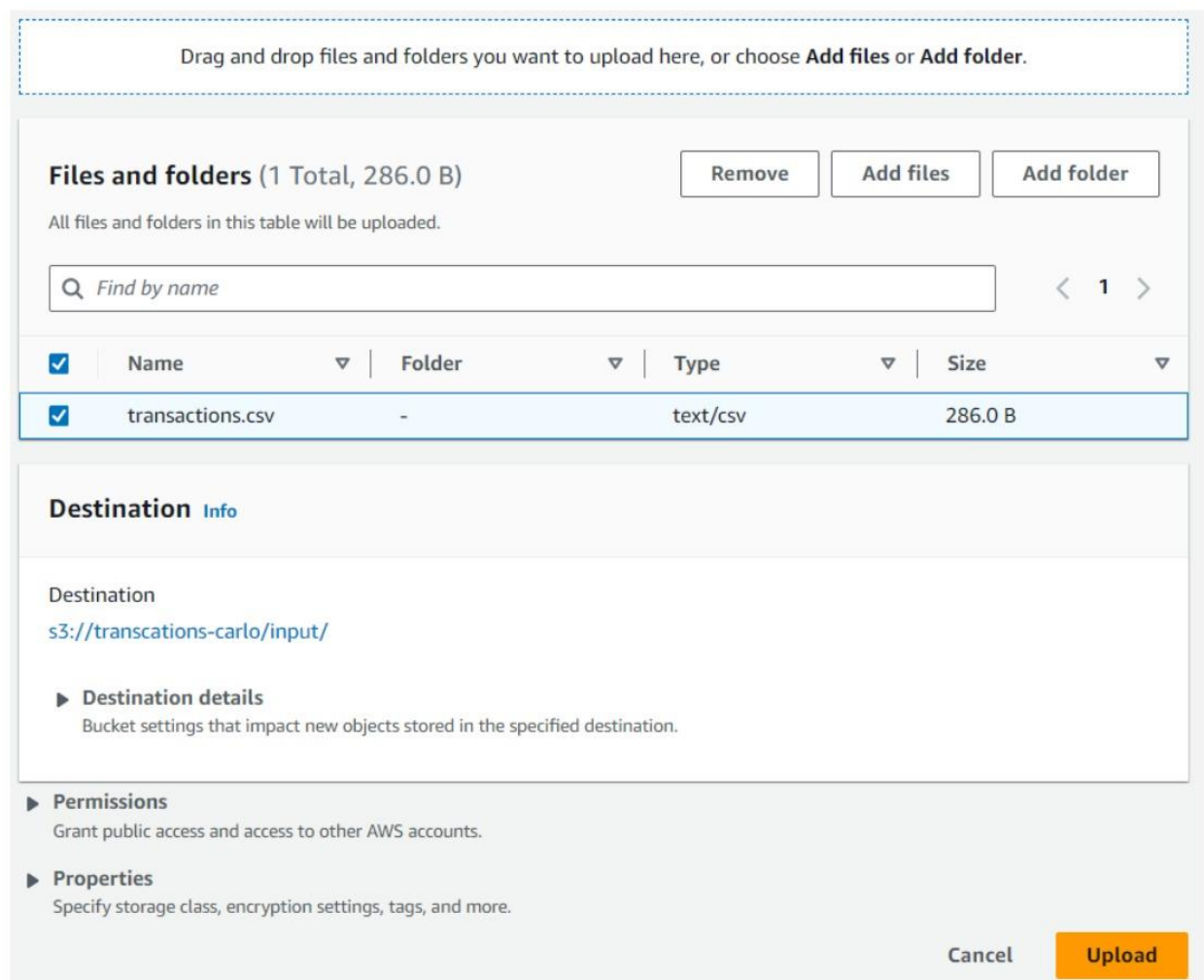
When we created the event notification, note that we have configured both a Prefix and a Suffix. In this setup, 'input/' is used as the Prefix, which means the event will only trigger for files uploaded to the 'input' folder within the bucket. The Suffix '.csv' ensures that the event is triggered only for files with a '.csv' extension. We use these filters to prevent unnecessary S3 notifications to our Lambda functions, ensuring they run only in response to relevant files.

To start the test:

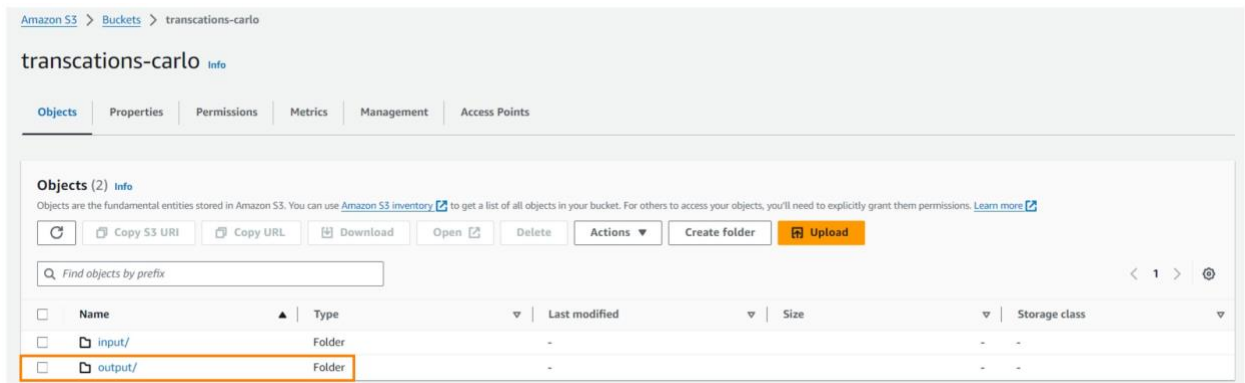
1. Create a folder named **'input'** in the S3 bucket.



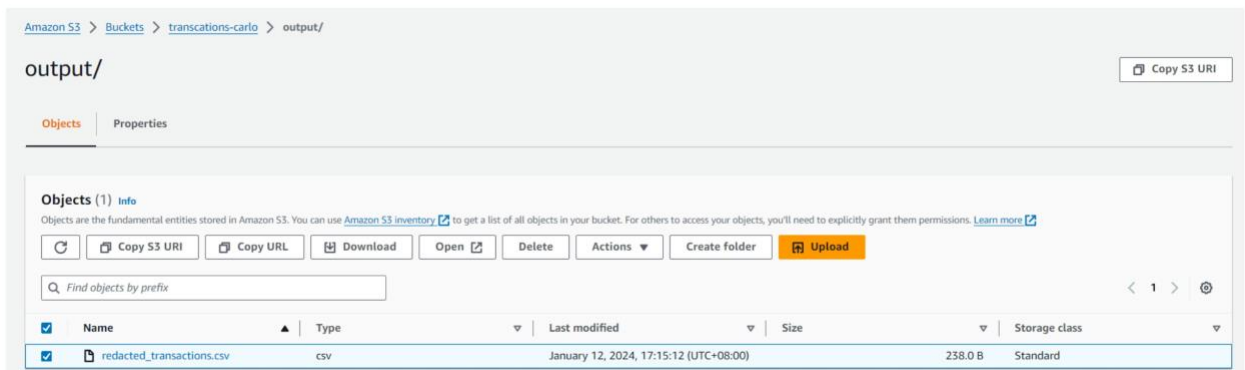
2. Download the [transactions.csv](#) file into your computer.
3. Open up the **transactions.csv**. You'll notice that the file contains 5 transactions, with the card numbers fully displayed.
4. Upload the **transactions.csv** to the **input** folder.:



5. Go back to the **Objects** section of your S3 bucket. You'll find a new **output** folder created in the bucket.



6. Click the **output** folder.



7. You should see a new file called **redacted_transactions.csv**. Download it to your local folder and open it. This file contains the same transaction records, but with the first 12 digits of each credit card number redacted

Transactions.csv

Date	Amount	CardNumber	Merchant
2024-01-12	\$150.00	1234567890123456	Coffee Shop
2024-01-13	\$45.00	6543210987654321	Bookstore
2024-01-14	\$78.25	8706543211234677	Electronics Store
2024-01-15	\$22.50	3216549870654321	Grocery Store
2024-01-16	\$130.00	4561237890654321	Online Marketplace

Lambda_function.py

```
import boto3
```

```
import csv
```

```
import re
```

```
import os
```

```
s3_client = boto3.client('s3')
```

```
def lambda_handler(event, context):
```

```
    # Get bucket name and object key from the event
```

```
    bucket_name = event['Records'][0]['s3']['bucket']['name']
```

```
    object_key = event['Records'][0]['s3']['object']['key']
```

```
    # Download the file to /tmp directory
```

```
    download_path = '/tmp/' + os.path.basename(object_key)
```

```
    s3_client.download_file(bucket_name, object_key, download_path)
```

```
    # Read, process, and redact the CSV file
```

```
    redacted_data = read_csv(download_path)
```

```
    # Write redacted content to a new CSV file in /tmp
```

```
    redacted_file_path = '/tmp/redacted_' + os.path.basename(object_key)
```

```
    write_csv(redacted_data, redacted_file_path)
```

```
# Upload the redacted file to the 'output/' folder

output_key = 'output/' + 'redacted_' + os.path.basename(object_key)

s3_client.upload_file(redacted_file_path, bucket_name, output_key)
```

```
def redact(row):
```

```
    """
```

```
    Redacts credit card numbers, leaving the last four digits visible.
```

```
    Assumes credit card number is in the third column (index 2).
```

```
    """
```

```
    row[2] = re.sub(r'\d{12}(\d{4})', r'*\1', row[2])
```

```
    return row
```

```
def read_csv(file_path):
```

```
    redacted_data = []
```

```
    with open(file_path, mode='r', newline='') as file:
```

```
        reader = csv.reader(file)
```

```
        for row in reader:
```

```
            redacted_data.append(redact(row))
```

```
    return redacted_data
```

```
def write_csv(data, file_path):
```

```
    with open(file_path, mode='w', newline='') as file:
```

```
        writer = csv.writer(file)
```

```
        writer.writerows(data)
```