Use Case:

A healthcare company stores medical reports in S3. These reports include sensitive fields like name, SSN, or contact details. The company wants to:

Let internal users access full reports.

Let external auditors access redacted versions, where sensitive fields are hidden.

**Solution Using S3 Object Lambda**

Original reports are stored as-is in a private S3 bucket (e.g., s3://health-records/reports/12345.json).

An Object Lambda Access Point is created with an associated AWS Lambda function.

The Lambda function reads the original report, removes sensitive fields (e.g., "ssn" and "name"), and returns the sanitized version.

External users use a presigned URL or access point to retrieve the report via the Object Lambda.

**Behind the Scenes**

S3 calls the Lambda function when an object is accessed via Object Lambda.

Lambda reads the original object and modifies it.

The modified result is returned to the requester.

📦 Prerequisites

Ensure the Lambda has permissions:

s3:GetObject

s3:GetObjectVersion

+Log related, + Lambda GetObject

AmazonS3ReadOnlyAccess

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"s3:Get\*",

"s3:List\*",

"s3:Describe\*",

"s3-object-lambda:Get\*",

"s3-object-lambda:List\*"

],

"Resource": "\*"

}

]

}

AWSLambdaBasicExecutionRole

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"logs:CreateLogGroup",

"logs:CreateLogStream",

"logs:PutLogEvents"

],

"Resource": "\*"

}

]

}

LamPolicy

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"s3-object-lambda:WriteGetObjectResponse",

"s3:GetObject"

],

"Resource": "\*"

}

]

}

**✅ Step-by-Step Setup for S3 Object Lambda**

**🔹 1. Create the Original S3 Bucket**

Let’s say the bucket name is medical-records-anon.

Upload a sample object:

{

"name": "John Doe",

"ssn": "123-45-6789",

"diagnosis": "Hypertension"

}

Upload this JSON file as record1.json.

**🔹 2. Create the Lambda Function**

Paste the code provided earlier into a new Lambda function using Node.js 18.x or later.

**Required Lambda Role Permissions**

Attach the following inline IAM policy to the Lambda role:

json

CopyEdit

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"s3-object-lambda:WriteGetObjectResponse",

"s3:GetObject"

],

"Resource": "\*"

}

]

}

**🔹 3. Create an S3 Access Point**

In the S3 console:

1. Go to **Access Points** → **Create access point**
2. Name it: medical-access
3. Select your bucket: medical-records-anon
4. Leave the network settings as default (public or VPC as needed)
5. Click **Create access point**

**🔹 4. Create an Object Lambda Access Point**

1. Go to **S3 > Object Lambda Access Points** → **Create**
2. Name it: medical-redact-access
3. Source access point: select medical-access
4. Add your Lambda function under “Transformation configuration”
5. Select the “GetObject” transformation
6. Click **Create**

**🔹 5. Test the Flow**

You can access the redacted file via the Object Lambda Access Point using:

* **AWS CLI**
* Or SDK (e.g., boto3, AWS SDK for JavaScript)

bash

CopyEdit

aws s3api get-object \

--bucket arn:aws:s3-object-lambda:region:account-id:accesspoint/medical-redact-access \

--key record1.json \

output.json

🔸 output.json will not contain the name or ssn fields — redacted by Lambda.

**🚧 Gotchas / Tips**

* Ensure your Lambda and Access Points are in the same region.
* Lambda execution role must include both s3:GetObject and s3-object-lambda:WriteGetObjectResponse.
* Make sure the bucket policy or access point allows the Lambda to access the object.

Regions must be consistent

Command to invoke the lambda function

D:\Projects\aws>aws s3api get-object --bucket mumlambdaacpoint-fbmhicb1yaiobo6h73ysrbg3aps3a--ol-s3 --key arn:aws:s3:::mumbn/record1.json output.json

Object Lambda Access Point alias = mumlambdaacpoint-fbmhicb1yaiobo6h73ysrbg3aps3a--ol-s3

| **Component** | **Role in the Workflow** |
| --- | --- |
| Supporting Access Point | Provides access to the original object in the S3 bucket |
| Object Lambda Access Point | Intercepts the request, invokes Lambda, returns modified data |

So in short: the **Supporting Access Point** is the gateway to the raw data, while the **Object Lambda Access Point** is the smart filter that transforms it before delivery.

**AWS Lambda Billing Basics**

You are charged for:

**1. Number of Invocations**

* First **1 million invocations/month** are free.
* After that: **$0.20 per million requests**.

If your function runs **every minute**, that’s:

* 60 invocations/hour × 24 hours/day × 30 days/month = **~43,200 invocations/month**
* This is **well under the free tier** for invocations.

✅ **So you won’t be charged for invocations** unless you exceed 1M/month.

**2. Duration (GB-seconds)**

You pay for the **execution time**, based on:

* How long your function runs (in milliseconds)
* How much memory is allocated (e.g. 128MB, 512MB, etc.)

**Formula:**

(Duration in seconds) × (Memory in GB) × (Price per GB-second)

Example:

* 128MB = 0.125 GB
* Function runs 500ms
* Monthly cost (after free tier) = ~$0.000000208/100ms

You get **400,000 GB-seconds free/month**, then pay **$0.00001667 per GB-second**.

🟢 If your function is small (128MB, short duration), **it’s likely still free**.

**3. Additional Costs (if applicable)**

You may incur **other costs** if:

* It accesses **S3**, **DynamoDB**, or **other AWS services**.
* It sends data to **CloudWatch Logs** (there's a free tier, then cost).

**When to Use What**

* **Serverless Framework** → Best for **quick serverless app development** across clouds.
* **AWS SAM** → Best if you’re **all-in on AWS** and want **tight CloudFormation integration**.
* **Terraform** → Best for **large, multi-service infrastructure** or **multi-cloud strategy**.

👉 In practice:

* Startups / small teams often use **Serverless Framework** for speed.
* AWS-first teams use **SAM** for Lambda-centric projects.
* Enterprises with **complex infra** (VPCs, EC2, RDS, Kubernetes, etc.) go with **Terraform**.