

Project 1: S3 to Lambda Thumbnail Generation with SNS Notifications

Table of Contents

- [By: Islam Zain](#)
- [Project Overview](#)
- [Prerequisites](#)
- [Step-by-Step Implementation](#)
- [Wait 30 seconds](#)
 - [Cost Optimization \(Staying in Free Tier\)](#)
 - [Troubleshooting Guide](#)
 - [Key Best Practices](#)
 - [Advanced Enhancements](#)
- [Convert all images to WebP for better compression](#)
 - [Cleanup Instructions](#)
- [Delete buckets and contents](#)
- [Delete Lambda function](#)
- [Delete SNS topic](#)
- [Delete IAM role](#)
 - [Conclusion](#)

By: Islam Zain

Project Overview

This lab demonstrates a **complete serverless image processing pipeline** on AWS. When images are uploaded to a source S3 bucket, a Lambda function automatically generates thumbnails and stores them in a destination bucket, while sending email notifications via SNS.

Architecture Components:

- **Source Bucket (Images):** Receives original images
- **Lambda Function:** Triggered by S3 events, resizes images to thumbnails
- **Destination Bucket (Thumbnails):** Stores resized images
- **SNS Topic:** Sends email notifications about processing

- **IAM Role:** Manages permissions for Lambda to access S3 and SNS
- **CloudWatch Logs:** Monitors Lambda execution

Prerequisites

1. AWS Account with Free Tier access
2. AWS CLI configured locally
3. Python 3.9+
4. Verified email for SNS subscription
5. Basic understanding of AWS services

Step-by-Step Implementation

Step 1: Create S3 Buckets

Using AWS Console:

1. Navigate to S3 service
2. Click "Create Bucket"
3. **Bucket 1 - Source (Images):**
 - Name: project1-images-[your-account-id]
 - Region: Select your closest region
 - Default settings OK
4. **Bucket 2 - Destination (Thumbnails):**
 - Name: project1-thumbnails-[your-account-id]
 - Same region as Bucket 1

Using AWS CLI:

```
aws s3 mb s3://project1-images-source --region us-east-1
aws s3 mb s3://project1-thumbnails-dest --region us-east-1
```

Step 2: Create SNS Topic and Email Subscription

Using AWS Console:

1. Navigate to SNS service
2. Click "Create Topic"
3. **Topic Details:**
 - Type: Standard

- Name: project1-image-processing

4. Click "Create Topic"

5. In the topic page, click "Create Subscription"

6. Subscription Details:

- Protocol: Email
- Endpoint: Your email address

7. Check your inbox and **confirm the subscription**

Using AWS CLI:

```
TOPIC_ARN=$(aws sns create-topic --name project1-image-processing --region us-east-1 --query TopicArn)
aws sns subscribe --topic-arn $TOPIC_ARN --protocol email --notification-endpoint your-email-address
echo "Check your email and confirm the subscription"
```

Step 3: Create IAM Role for Lambda

Using AWS Console:

1. Navigate to IAM → Roles

2. Click "Create Role"

3. Role Details:

- Trusted entity: AWS Lambda
- Role name: project1-lambda-s3-sns-role

4. Click Next

5. Attach Policies:

- AWSLambdaBasicExecutionRole (CloudWatch Logs)
- AmazonS3FullAccess (or create custom policy)
- AmazonSNSFullAccess (or create custom policy)

Custom Policy for S3 (More Secure):

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::project1-images-source/*",
        "arn:aws:s3:::project1-thumbnails-dest/*"
      ]
    }
  ]
}
```

```
]  
}
```

Step 4: Create Lambda Function

Using AWS Console:

1. Navigate to Lambda service

2. Click "Create Function"

3. **Function Details:**

- o Function name: project1-thumbnail-generator
- o Runtime: Python 3.9 or later
- o Role: Select the role created in Step 3

4. Click "Create Function"

Environment Variables (set in Lambda Console):

```
THUMBNAIL_BUCKET = project1-thumbnails-dest  
SNS_TOPIC_ARN = arn:aws:sns:us-east-1:ACCOUNT-ID:project1-image-processing
```

Step 5: Add Lambda Function Code

Function Code (Python):

```
import json  
import boto3  
import os  
from PIL import Image  
from io import BytesIO  
import logging  
  
s3_client = boto3.client('s3')  
sns_client = boto3.client('sns')  
  
logger = logging.getLogger()  
logger.setLevel(logging.INFO)  
  
SOURCE_BUCKET = os.environ.get('SOURCE_BUCKET')  
THUMBNAIL_BUCKET = os.environ.get('THUMBNAIL_BUCKET')  
SNS_TOPIC_ARN = os.environ.get('SNS_TOPIC_ARN')  
THUMBNAIL_SIZE = (200, 200)  
  
def lambda_handler(event, context):  
    """  
    Main Lambda handler to process S3 events  
    and generate thumbnails  
    """  
    try:  
        # Extract S3 event information
```

```

for record in event['Records']:
    bucket = record['s3']['bucket']['name']
    key = record['s3']['object']['key']

    logger.info(f"Processing image: {key} from bucket: {bucket}")

    # Download original image from S3
    response = s3_client.get_object(Bucket=bucket, Key=key)
    image_data = response['Body'].read()

    # Generate thumbnail
    image = Image.open(BytesIO(image_data))
    image.thumbnail(THUMBNAIL_SIZE, Image.Resampling.LANCZOS)

    # Save thumbnail to bytes
    thumbnail_buffer = BytesIO()
    image.save(thumbnail_buffer, format='JPEG', quality=80)
    thumbnail_buffer.seek(0)

    # Create thumbnail key
    thumbnail_key = f"thumbnails/{os.path.splitext(key)[0]}_thumb.jpg"

    # Upload thumbnail to destination bucket
    s3_client.put_object(
        Bucket=THUMBNAIL_BUCKET,
        Key=thumbnail_key,
        Body=thumbnail_buffer.getvalue(),
        ContentType='image/jpeg'
    )

    # Send SNS notification
    message = f"""
Image Processing Complete!

Original Image: {key}
Original Bucket: {bucket}
Thumbnail Created: {thumbnail_key}
Thumbnail Bucket: {THUMBNAIL_BUCKET}

Processing Time: Lambda execution
Status: SUCCESS
"""

    sns_client.publish(
        TopicArn=SNS_TOPIC_ARN,
        Subject='Image Thumbnail Generated Successfully',
        Message=message
    )

    logger.info(f"Thumbnail created: {thumbnail_key}")

return {
    'statusCode': 200,
    'body': json.dumps('Thumbnails generated successfully!')
}

```

```

        except Exception as e:
            logger.error(f"Error processing image: {str(e)}")

            # Send error notification
            sns_client.publish(
                TopicArn=SNS_TOPIC_ARN,
                Subject='Image Processing Error',
                Message=f'Error: {str(e)}'
            )

        return {
            'statusCode': 500,
            'body': json.dumps(f'Error: {str(e)}')
        }
    
```

Step 6: Add Python Dependencies (Pillow Library)

Since Pillow is not built into Lambda, use Lambda Layers:

Option A: Using AWS Console

1. Download Pillow layer: Use AWS managed layer or create custom
2. In Lambda function page → Layers → Add Layer
3. Select "AWS Managed Layers" → Choose Python image processing layer

Option B: Create Custom Layer (Recommended)

```

mkdir python/lib/python3.9/site-packages
pip install Pillow -t python/lib/python3.9/site-packages/
zip -r layer.zip python/
aws lambda publish-layer-version \
--layer-name project1-pillow-layer \
--zip-file fileb://layer.zip \
--compatible-runtimes python3.9 \
--region us-east-1

```

Step 7: Configure S3 Trigger

Using AWS Console:

1. In Lambda function page → Designer section
2. Click "Add Trigger"

3. Trigger Configuration:

- Source: S3
 - Bucket: project1-images-source
 - Event type: PUT (for object uploads)
 - Suffix: .jpg, .jpeg, .png (filter to images only)
4. Click "Add"

Note: Accept the permission prompt to allow S3 to invoke Lambda

Step 8: Test the Project

Test 1: Using AWS Console

1. Go to S3 → project1-images-source bucket
2. Click "Upload"
3. Choose a JPEG or PNG image
4. Wait 30 seconds
5. Check project1-thumbnails-dest bucket → Should see thumbnail
6. Check your email → SNS notification received

Test 2: Using AWS CLI

```
aws s3 cp your-image.jpg s3://project1-images-source/  
# Wait 30 seconds<a></a>  
aws s3 ls s3://project1-thumbnails-dest/
```

Test 3: Lambda Console Test Event

```
{  
  "Records": [  
    {  
      "s3": {  
        "bucket": {  
          "name": "project1-images-source"  
        },  
        "object": {  
          "key": "test-image.jpg"  
        }  
      }  
    }  
  ]  
}
```

Cost Optimization (Staying in Free Tier)

Monthly Free Tier Limits:

- **S3:** 5GB storage + 20,000 GET/PUT requests free
- **Lambda:** 1 million requests/month, 400,000 GB-seconds/month
- **SNS:** 1,000 email notifications/month
- **CloudWatch:** 10GB logs/month

Optimization Tips:

1. **Image Size:** Keep source images < 5MB
2. **Thumbnail Size:** Keep thumbnails to 200x200 pixels (reduces processing)
3. **Frequency:** Test with 10-20 images to stay well within limits
4. **Cleanup:** Delete old images regularly to manage storage
5. **S3 Lifecycle:** Set expiration policy after 30 days

Troubleshooting Guide

Issue 1: Lambda Timeout

Symptom: Function execution exceeds 3 seconds

Solution:

- Increase timeout in Lambda settings to 30 seconds
- Reduce image size before uploading
- Optimize Pillow operations

Issue 2: Permission Denied Error

Symptom: "AccessDenied" in CloudWatch logs

Solution:

- Verify IAM role has S3 and SNS permissions
- Check bucket names match in code
- Ensure SNS_TOPIC_ARN environment variable is correct

Issue 3: Pillow Import Error

Symptom: "ModuleNotFoundError: No module named 'PIL'"

Solution:

- Verify Lambda Layer is added to function
- Check layer compatible runtime matches Lambda runtime
- Recreate layer if necessary

Issue 4: SNS Email Not Received

Symptom: No email notifications arriving

Solution:

- Confirm SNS subscription in email
- Check spam folder

- Verify SNS_TOPIC_ARN is correct
- Check CloudWatch logs for errors

Issue 5: Thumbnail Not Created

Symptom: No files in destination bucket

Solution:

- Check S3 trigger is configured
- Verify image format is supported (.jpg, .png)
- Review CloudWatch logs for errors
- Test Lambda with console test event

Key Best Practices

1. **Error Handling:** Wrap operations in try-catch, send failure notifications
2. **Logging:** Use CloudWatch for debugging and monitoring
3. **Least Privilege:** Use specific IAM policies, not full access
4. **Versioning:** Enable S3 versioning for disaster recovery
5. **Testing:** Test with various image formats and sizes
6. **Monitoring:** Set up CloudWatch alarms for Lambda failures
7. **Security:** Never hardcode credentials; use environment variables
8. **Cleanup:** Implement S3 lifecycle policies to manage costs

Advanced Enhancements

Add Multiple Thumbnail Sizes

```
SIZES = {
    'small': (100, 100),
    'medium': (200, 200),
    'large': (400, 400)
}

for size_name, size_dims in SIZES.items():
    # Generate multiple thumbnails
```

Add Image Format Conversion

```
# Convert all images to WebP for better compression<a></a>
image.save(thumbnail_buffer, format='WEBP', quality=80)
```

Add DynamoDB Logging

```
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('project1-image-log')
table.put_item(Item={
    'image_key': key,
    'timestamp': datetime.now().isoformat(),
    'thumbnail_key': thumbnail_key,
    'status': 'SUCCESS'
})
```

Cleanup Instructions

To avoid unexpected charges:

```
# Delete buckets and contents<a></a>
aws s3 rm s3://project1-images-source --recursive
aws s3 rm s3://project1-images-source
aws s3 rm s3://project1-thumbnails-dest --recursive
aws s3 rm s3://project1-thumbnails-dest

# Delete Lambda function<a></a>
aws lambda delete-function --function-name project1-thumbnail-generator

# Delete SNS topic<a></a>
aws sns delete-topic --topic-arn arn:aws:sns:us-east-1:ACCOUNT-ID:project1-image-process

# Delete IAM role<a></a>
aws iam delete-role-policy --role-name project1-lambda-s3-sns-role --policy-name project1
aws iam delete-role --role-name project1-lambda-s3-sns-role
```

Conclusion

You've successfully built a **production-ready serverless image processing system!** This architecture is:

- **Scalable:** Handles unlimited concurrent uploads
- **Cost-Effective:** Charges only for actual usage
- **Automatic:** No manual intervention required
- **Resilient:** CloudWatch monitoring and error handling included
- **Professional:** Follows AWS best practices

This foundation can be extended with CloudFront caching, DynamoDB logging, advanced image processing, and more!

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10]

**

1. <https://docs.aws.amazon.com/lambda/latest/dg/with-s3-tutorial.html>
2. [https://stackoverflow.com/questions/54760261/what-is-the-best-approach-for-generating-thumbnails-and-uploading-to-s3-bucket-i](https://stackoverflow.com/questions/54760261/what-is-the-best-approach-for-generating-thumbnails-and-uploading-to-s3-bucket-in)
3. <https://www.linkedin.com/pulse/automating-aws-lambda-invocation-triggered-s3-event-sns-swati-kanungo-ggw3f>
4. <https://www.educative.io/cloudlabs/resizing-images-with-s3-batch-operations-and-aws-lambda>
5. <https://www.norrapscm.com/posts/2021-02-08-generate-thumbnails-in-lambda-from-s3-with-ffmpeg/>
6. <https://www.jeeviacademy.com/aws-s3-to-sns-simplifying-notification-delivery/>
7. <https://www.kubeblogs.com/resize-images-at-scale-without-breaking-a-sweat/>
8. <https://dev.to/nadaahmed/automating-image-thumbnails-with-aws-lambda-and-s3-a-step-by-step-guide-36dd>
9. <https://docs.aws.amazon.com/prescriptive-guidance/latest/patterns/subscribe-a-lambda-function-to-event-notifications-from-s3-buckets-in-different-aws-regions.html>
10. <https://stackoverflow.com/questions/60486873/resize-and-move-images-using-lambda-function-from-one-s3-to-another-s3>