

# Image Resize and Restore System Documentation

## Connections

AWS S3 Bucket Name: deepak-image-scaling

AWS Lambda Function Name: image-resize-restore

AWS Region: us-east-1

Streamlit Application File: app.py

## Introduction

This document describes the complete implementation of an Image Resize and Restore system using AWS services and a Streamlit based user interface. The goal of this project is to allow users to upload an original image, resize it automatically based on custom dimensions, and later restore the original image by uploading the resized image again.

## Problem Statement

Users often need to resize images for applications while still preserving the original version. Additionally, they may need to retrieve the original image later. This project solves that problem using a serverless and automated architecture.

## System Architecture Overview

The system consists of a Streamlit user interface, an AWS S3 bucket with structured folders, and an AWS Lambda function. The Streamlit UI handles user interactions. AWS S3 acts as the storage layer and event trigger. AWS Lambda performs image processing and restoration logic.

## Folder Structure

The S3 bucket contains three main folders.

The original folder stores original uploaded images.

The resized folder stores resized images.

The restored folder stores restored original images.

## Detailed Workflow

When a user uploads an original image through the UI, the file is uploaded to the original folder with width and height metadata. This upload triggers the Lambda function. The Lambda function reads the metadata, resizes the image using Pillow, and saves the resized image to the resized folder.

When a user uploads a resized image through the UI, the image is uploaded to the resized folder. This triggers the Lambda function again. The function detects that the upload came from the resized folder, finds the corresponding original image with the same filename in the original folder, and copies it into the restored folder.

## Streamlit Application Description

The Streamlit application provides two tabs. One tab is used for resizing images and the

other for restoring images. The resize tab allows users to upload an image and specify custom width and height. The restore tab allows users to upload a resized image to restore its original version.

#### AWS Lambda Function Description

The Lambda function is triggered by S3 object creation events. It checks the folder prefix of the uploaded file. If the file is uploaded to the original folder, it performs resizing. If the file is uploaded to the resized folder, it performs restoration.

#### Image Processing Logic

The Lambda function uses the Pillow library to open and process images. The resizing operation uses a high quality resizing algorithm. The original image format is preserved.

#### Security and Permissions

The Lambda function requires permissions to read and write objects in the S3 bucket. The Streamlit application requires AWS credentials with access to upload files to S3.

#### Error Handling

The system includes error handling for missing files, unsupported formats, and permission issues. Logs are generated in CloudWatch for debugging.

#### Performance Considerations

Lambda memory should be set to at least 512 MB and timeout to at least 15 seconds. This ensures smooth image processing.

#### Testing Strategy

The system can be tested by uploading images via the Streamlit UI and verifying outputs in the S3 folders. Lambda test events can also be used for validation.

#### Deployment Steps

Create the S3 bucket and folders.

Deploy the Lambda function.

Configure S3 triggers.

Set up IAM roles.

Run the Streamlit application.

#### Limitations

The system relies on filename matching for restoration. If filenames differ, restoration will fail.

#### Future Enhancements

Support for image versioning.

Support for multiple resized versions.

Database based image mapping.

# #APP.PY

```
import streamlit as st  
import boto3  
from botocore.exceptions import ClientError  
  
AWS_REGION = "us-east-1"  
BUCKET = "deepak-image-scaling"  
s3 = boto3.client("s3", region_name=AWS_REGION)  
  
st.set_page_config(page_title="Image Resize & Restore", layout="centered")  
st.title(" Image Resize & Restore UI")  
  
tab1, tab2 = st.tabs(["Resize Image (Upload Original)", "Restore Image (Upload Resized)"])
```

with tab1:

```
    st.header("Upload ORIGINAL Image → Resize Automatically")  
    uploaded_file = st.file_uploader(  
        "Choose an image to resize",  
        type=["jpg", "jpeg", "png", "webp", "gif"],  
        key="original")  
    )
```

```
col1, col2 = st.columns(2)
```

with col1:

```
    width = st.number_input("Width (px)", min_value=50, value=300)
```

with col2:

```

height = st.number_input("Height (px)", min_value=50, value=300)

if uploaded_file is not None:
    st.image(uploaded_file, caption="Preview Original", use_column_width=True)
    if st.button("Upload & Resize Original"):
        try:
            s3.put_object(
                Bucket=BUCKET,
                Key=f"original/{uploaded_file.name}",
                Body=uploaded_file.read(),
                ContentType=uploaded_file.type,
                Metadata={"width": str(width), "height": str(height)}
            )
            st.success(f" Uploaded to original/{uploaded_file.name}\n"
                      f"Lambda will resize it automatically → resized/{uploaded_file.name}")
        except ClientError as e:
            st.error(f"AWS Error: {e}")

```

with tab2:

```

st.header("Upload RESIZED Image → Restore Original Automatically")

uploaded_file2 = st.file_uploader(
    "Choose a resized image",
    type=["jpg", "jpeg", "png", "webp", "gif"],
    key="resized"
)

```

```
if uploaded_file2 is not None:  
    st.image(uploaded_file2, caption="Preview Resized", use_column_width=True)  
    if st.button("Upload & Restore Original"):  
        try:  
            s3.put_object(  
                Bucket=BUCKET,  
                Key=f"resized/{uploaded_file2.name}",  
                Body=uploaded_file2.read(),  
                ContentType=uploaded_file2.type  
            )  
            st.success(f" Uploaded to resized/{uploaded_file2.name}\n"  
                      f"Lambda will restore the original → restored/{uploaded_file2.name}")  
        except ClientError as e:  
            st.error(f"AWS Error: {e}")
```

---

```
#LAMBDA FUNCTION  
  
import boto3  
from PIL import Image, ImageFile  
import io  
  
ImageFile.LOAD_TRUNCATED_IMAGES = True  
  
s3 = boto3.client("s3")
```

```
BUCKET = "deepak-image-scaling"
```

```
ORIGINAL_PREFIX = "original/"
```

```
RESIZED_PREFIX = "resized/"
```

```
RESTORED_PREFIX = "restored/"
```

```
DEFAULT_WIDTH = 300
```

```
DEFAULT_HEIGHT = 300
```

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

```
    try:
```

```
        record = event["Records"][0]
```

```
        key = record["s3"]["object"]["key"]
```

```
        bucket = record["s3"]["bucket"]["name"]
```

```
        print(f"Triggered for key: {key}")
```

```
        if key.startswith(ORIGINAL_PREFIX):
```

```
            return resize_image(bucket, key)
```

```
        elif key.startswith(RESIZED_PREFIX):
```

```
            return restore_image(bucket, key)
```

```
    else:
```

```
        return {"message": "Ignored file"}
```

```
except Exception as e:
```

```
print("ERROR:", str(e))

return {"error": str(e)}

# ----- RESIZE -----

def resize_image(bucket, key):

    filename = key.replace(ORIGINAL_PREFIX, "")

    obj = s3.get_object(Bucket=bucket, Key=key)

    body = obj["Body"].read()

    metadata = obj.get("Metadata", {})

    width = int(metadata.get("width", DEFAULT_WIDTH))

    height = int(metadata.get("height", DEFAULT_HEIGHT))

    img = Image.open(io.BytesIO(body))

    img.load()

    # Resize image

    resized = img.resize((width, height), Image.LANCZOS)

    buffer = io.BytesIO()

    fmt = img.format or "PNG"

    resized.save(buffer, fmt)

    buffer.seek(0)

    resized_key = f"{RESIZED_PREFIX}{filename}"
```

```
# Upload resized image

s3.put_object(
    Bucket=bucket,
    Key=resized_key,
    Body=buffer,
    ContentType=f'image/{fmt.lower()}"'
)

print(f'Resized image saved to {resized_key}')

return {"message": "Image resized", "resized_key": resized_key}

def restore_image(bucket, key):
    import os

    filename = os.path.basename(key)

    original_key = f'{ORIGINAL_PREFIX}{filename}'
    restored_key = f'{RESTORED_PREFIX}{filename}'

    try:
        original_obj = s3.get_object(Bucket=bucket, Key=original_key)
        original_body = original_obj["Body"].read()

        s3.put_object(
```

```
        Bucket=bucket,
        Key=restored_key,
        Body=original_body,
        ContentType=original_obj.get("ContentType", "image/png")
    )

print(f"Original restored to {restored_key}")

return {"message": "Original restored", "restored_key": restored_key}

except s3.exceptions.NoSuchKey:

    print(f"Original image not found for {filename}")

    return {"error": f"Original image {filename} not found in original/ folder"}

except Exception as e:

    print("Restore failed:", str(e))

    return {"error": str(e)}
```

---

```
#lambda—added policy in IAM
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3>ListBucket"
```

```

        ],
        "Resource": [
            "arn:aws:s3:::deepak-image-scaling",
            "arn:aws:s3:::deepak-image-scaling/*"
        ]
    }
]
}

```

## #IAM ROLE --IMG

The screenshot shows the AWS IAM Roles page. On the left, there's a navigation sidebar with options like Dashboard, Access management (selected), Roles, Policies, and Access reports. The main area is titled 'Summary' for the role 'lambda-image-scaling-role'. It shows the creation date (December 12, 2025, 13:59 (UTC+05:30)), last activity (5 hours ago), ARN (arn:aws:iam::049061800282:role/lambda-image-scaling-role), and maximum session duration (1 hour). Below this, the 'Permissions' tab is selected, showing three attached policies: 'AmazonS3FullAccess', 'image-scaling', and 'lambda'. There are buttons for 'Simulate', 'Remove', and 'Add permissions'.

Policy name	Type	Attached entities
AmazonS3FullAccess	AWS managed	3
image-scaling	Customer inline	0
lambda	Customer inline	0

## #LAMBDA ---IMG

The screenshot shows the AWS Lambda console. The top navigation bar includes the AWS logo, a search bar, and account information (Account ID: 0490-6180-0282, Region: Asia Pacific (Sydney)). The main title is 'image-resize-restore'. On the left, there's a 'Function overview' section with tabs for 'Diagram' (selected) and 'Template'. The diagram shows a single layer named 'image-resize-restore' and an S3 trigger. Buttons for '+ Add destination' and '+ Add trigger' are visible. To the right, there are sections for 'Description', 'Last modified' (15 hours ago), 'Function ARN' (arn:aws:lambda:ap-southeast-2:049061800282:function:image-resize-restore), and 'Function URL' (Info). Below the main content are tabs for 'Code' (selected), 'Test', 'Monitor', 'Configuration', 'Aliases', and 'Versions'.

## #S3 BUCKET---IMG

The screenshot shows the AWS S3 console. The top navigation bar includes the AWS logo, a search bar, and account information (Account ID: 0490-6180-0282, Region: Asia Pacific (Sydney)). The main title is 'deepak-image-scaling'. On the left, there's a sidebar with sections for 'Amazon S3' (Buckets, Access management and security, Storage management and insights), 'General purpose buckets' (Directory buckets, Table buckets, Vector buckets), and links for 'Account and organization settings' and 'AWS Marketplace for S3'. The main content area shows the 'Objects' tab selected for the 'deepak-image-scaling' bucket. It displays three objects: 'original/' (Folder), 'resized/' (Folder), and 'restored/' (Folder). There are buttons for 'Copy S3 URI', 'Copy URL', 'Download', 'Open', 'Delete', 'Actions', 'Create folder', and 'Upload'.