AWS Serverless Image Recognition – Project Documentation

1. Introduction

This project is an event-driven serverless image recognition system built entirely on AWS. Users upload images via a static website, which are automatically analyzed using Amazon Rekognition. Detected labels are stored in DynamoDB, and notifications are sent via SNS.

Motivation:

Manual tagging of images is slow and error-prone. Serverless architecture ensures automation, scalability, and secure processing.

---

2. Problem Statement

Millions of images are uploaded daily on platforms like e-commerce websites. Manual tagging is time-consuming and often inaccurate. A serverless, automated solution is required to detect objects, people, and scenes in images.

---

3. Project Objectives

1. Build a **serverless image recognition system**.
2. Automate workflow: upload → recognition → store → notify.
3. Provide a **user-friendly frontend** to upload images and view results.
4. Ensure **security and scalability** using AWS services.

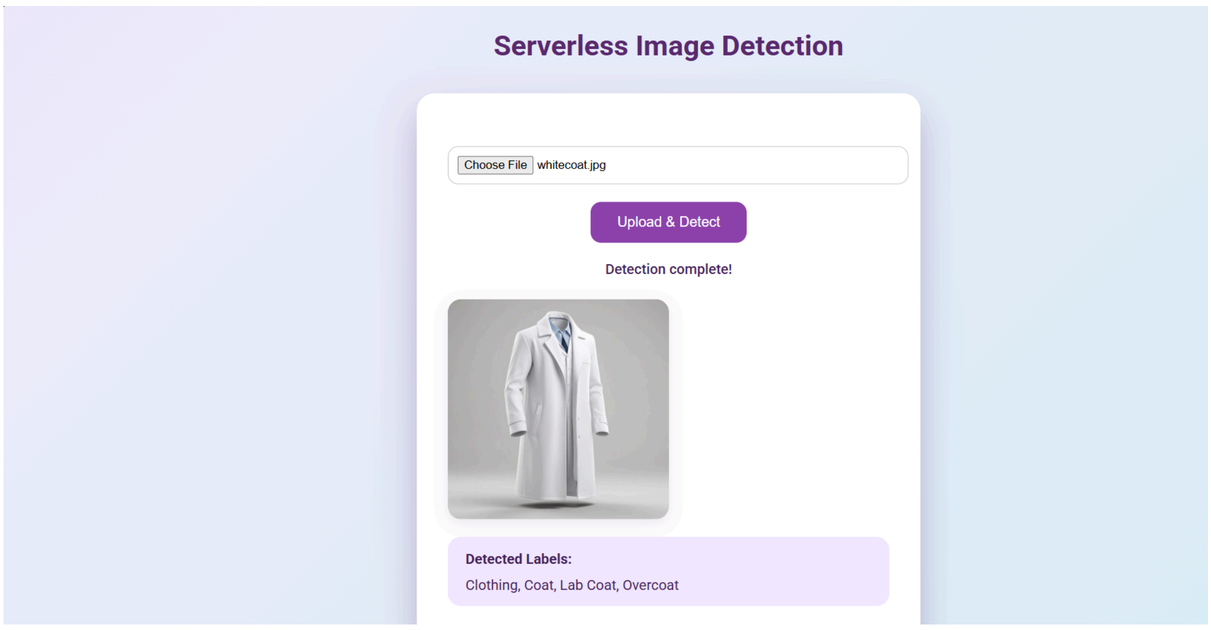
---

4. System Architecture

4.1 Architecture Overview

Frontend (S3 Static Website):

The frontend is a static website hosted on Amazon S3 where users can upload images and view the detected labels. The interface provides buttons to select and upload images, and it displays the recognition results directly on the page.

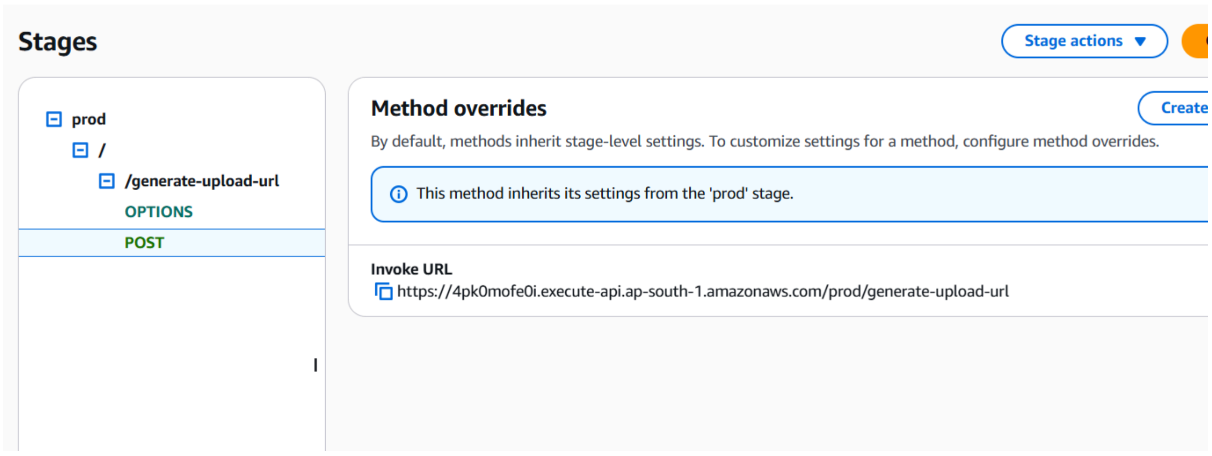


Detection complete!

Detected Labels:
Clothing, Coat, Lab Coat, Overcoat

API Gateway:

API Gateway exposes two endpoints: /generate-upload-url integrated with GenerateUploadURLLambda to provide secure pre-signed S3 URLs, and /detect-labels integrated with DetectLabelsLambda to fetch the labels stored in DynamoDB.



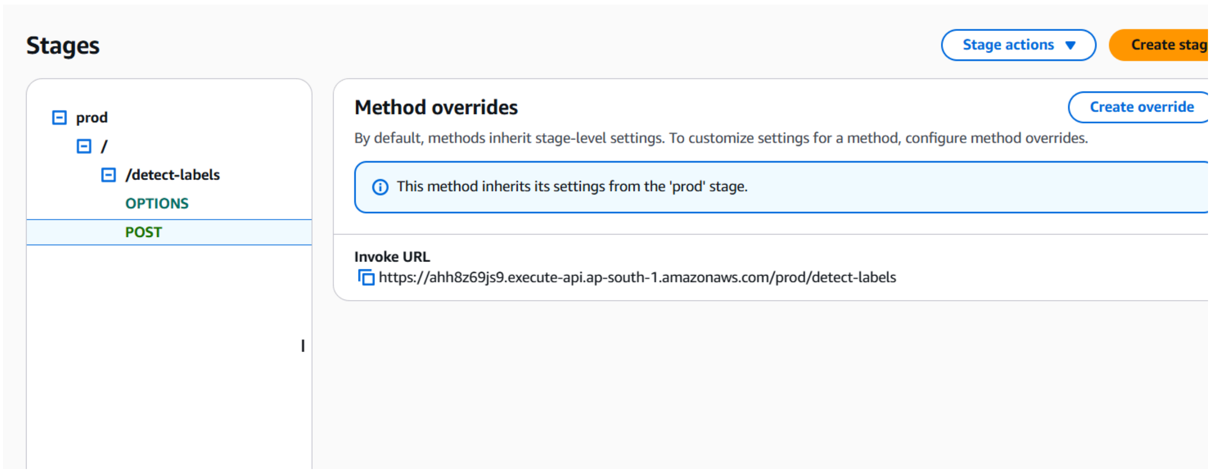
prod
/
/generate-upload-url
OPTIONS
POST

Method overrides

By default, methods inherit stage-level settings. To customize settings for a method, configure method overrides.

This method inherits its settings from the 'prod' stage.

Invoke URL
https://4pk0mofe0i.execute-api.ap-south-1.amazonaws.com/prod/generate-upload-url



prod
/
/detect-labels
OPTIONS
POST

Method overrides

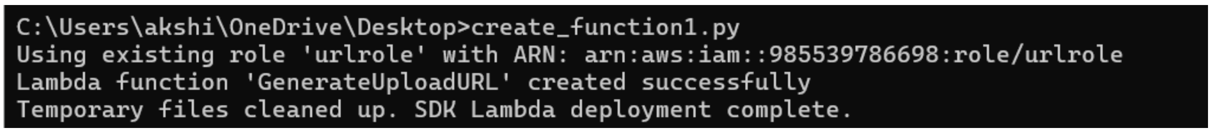
By default, methods inherit stage-level settings. To customize settings for a method, configure method overrides.

This method inherits its settings from the 'prod' stage.

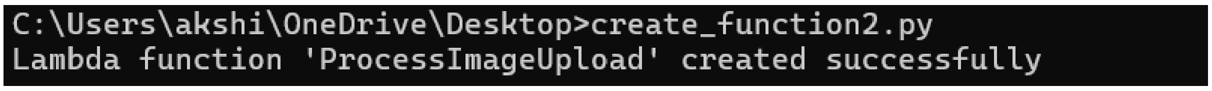
Invoke URL
https://ahh8z69js9.execute-api.ap-south-1.amazonaws.com/prod/detect-labels

Lambda Functions:

The system has three main Lambda functions: GenerateUploadURLLambda creates secure pre-signed URLs for S3 uploads; ProcessImageUploadLambda is triggered by S3, calls Rekognition to analyze the image, stores results in DynamoDB, and sends notifications via SNS; DetectLabelsLambda retrieves the stored labels and returns them to the frontend via API Gateway.



```
C:\Users\akshi\OneDrive\Desktop>create_function1.py
Using existing role 'urlrole' with ARN: arn:aws:iam::985539786698:role/urlrole
Lambda function 'GenerateUploadURL' created successfully
Temporary files cleaned up. SDK Lambda deployment complete.
```

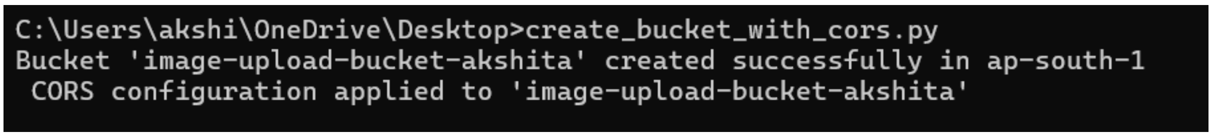


```
C:\Users\akshi\OneDrive\Desktop>create_function2.py
Lambda function 'ProcessImageUpload' created successfully
```

-DetectLabels deployed manually.

Amazon S3:

S3 stores all uploaded images and triggers the ProcessImageUploadLambda on new uploads. It provides durable, scalable storage and allows secure direct uploads using pre-signed URLs.



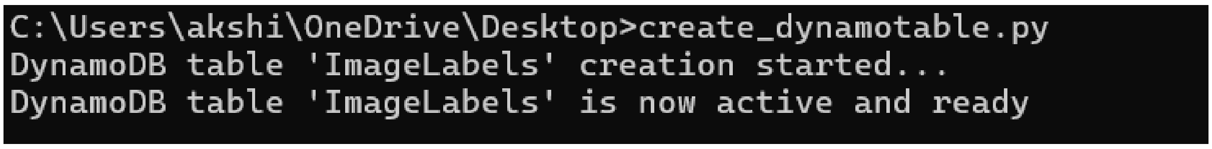
```
C:\Users\akshi\OneDrive\Desktop>create_bucket_with_cors.py
Bucket 'image-upload-bucket-akshita' created successfully in ap-south-1
 CORS configuration applied to 'image-upload-bucket-akshita'
```

Amazon Rekognition:

Rekognition analyzes uploaded images and generates labels for objects, people, and scenes with confidence scores. It provides automatic image recognition without manual intervention.

Amazon DynamoDB:

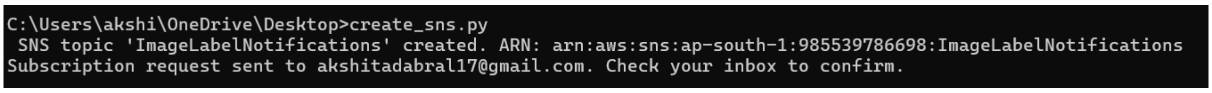
DynamoDB stores metadata for each uploaded image, including the file name, timestamp, and detected labels. This allows quick retrieval of labels when requested by the frontend.



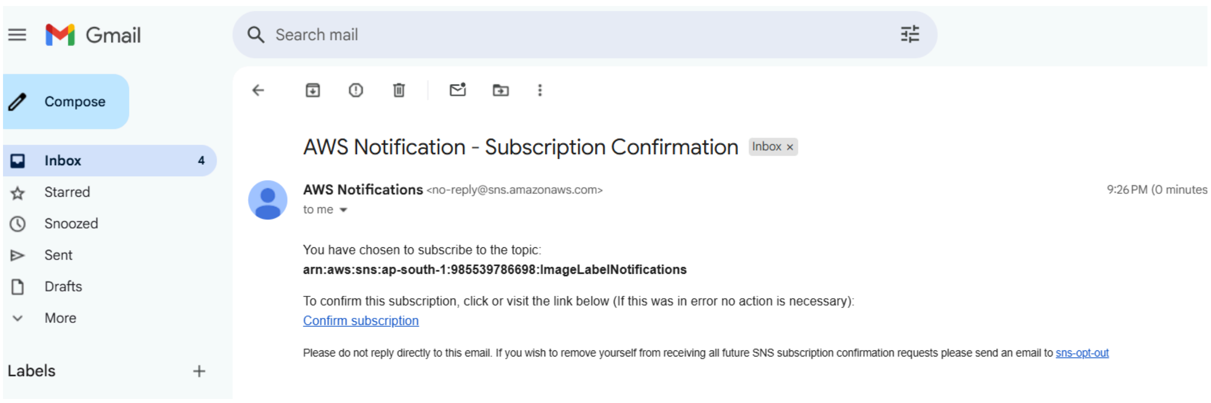
```
C:\Users\akshi\OneDrive\Desktop>create_dynamotable.py
DynamoDB table 'ImageLabels' creation started...
DynamoDB table 'ImageLabels' is now active and ready
```

Amazon SNS:

SNS sends email notifications to users with the detected labels for each uploaded image. This provides automated feedback to users without needing to check the website manually.



```
C:\Users\akshi\OneDrive\Desktop>create_sns.py
 SNS topic 'ImageLabelNotifications' created. ARN: arn:aws:sns:ap-south-1:985539786698:ImageLabelNotifications
Subscription request sent to akshitadabral17@gmail.com. Check your inbox to confirm.
```



AWS Notification - Subscription Confirmation Inbox ×

AWS Notifications <no-reply@sns.amazonaws.com>
to me

9:26 PM (0 minutes

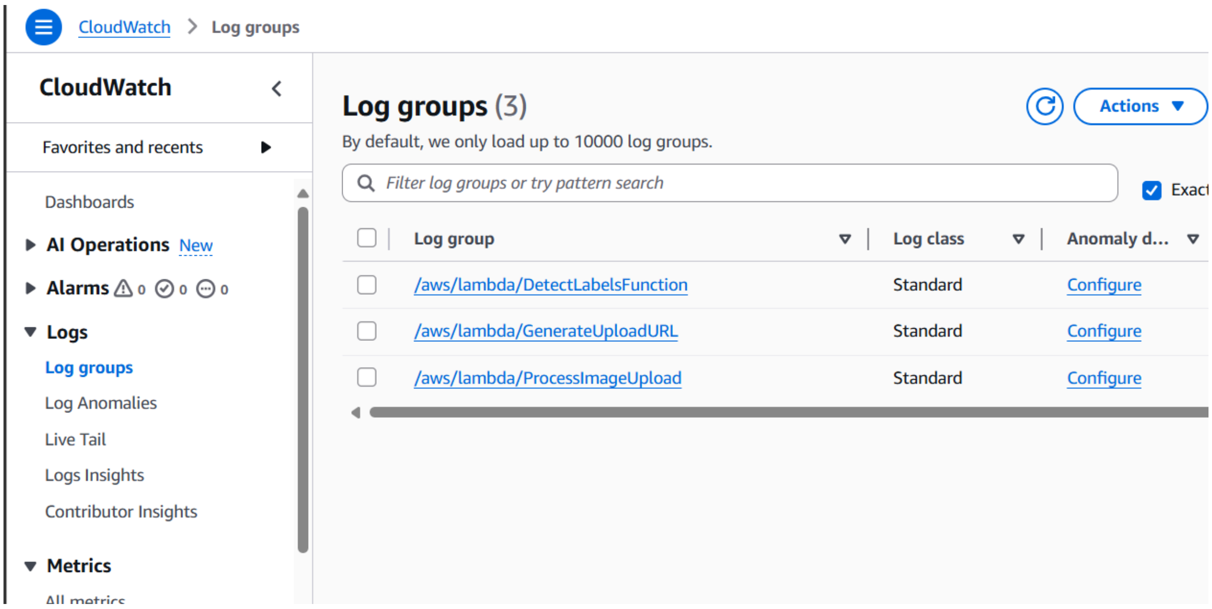
You have chosen to subscribe to the topic:
arn:aws:sns:ap-south-1:985539786698:ImageLabelNotifications

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):
Confirm subscription

Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to sns-opt-out

Amazon CloudWatch:

CloudWatch monitors all Lambda executions and helps debug issues in the system.


CloudWatch > Log groups

CloudWatch

Favorites and recents

Dashboards

AI Operations New

Alarms 0 0 0

Logs

Log groups

Log Anomalies

Live Tail

Logs Insights

Contributor Insights

Metrics

All metrics

Log groups (3)

Actions

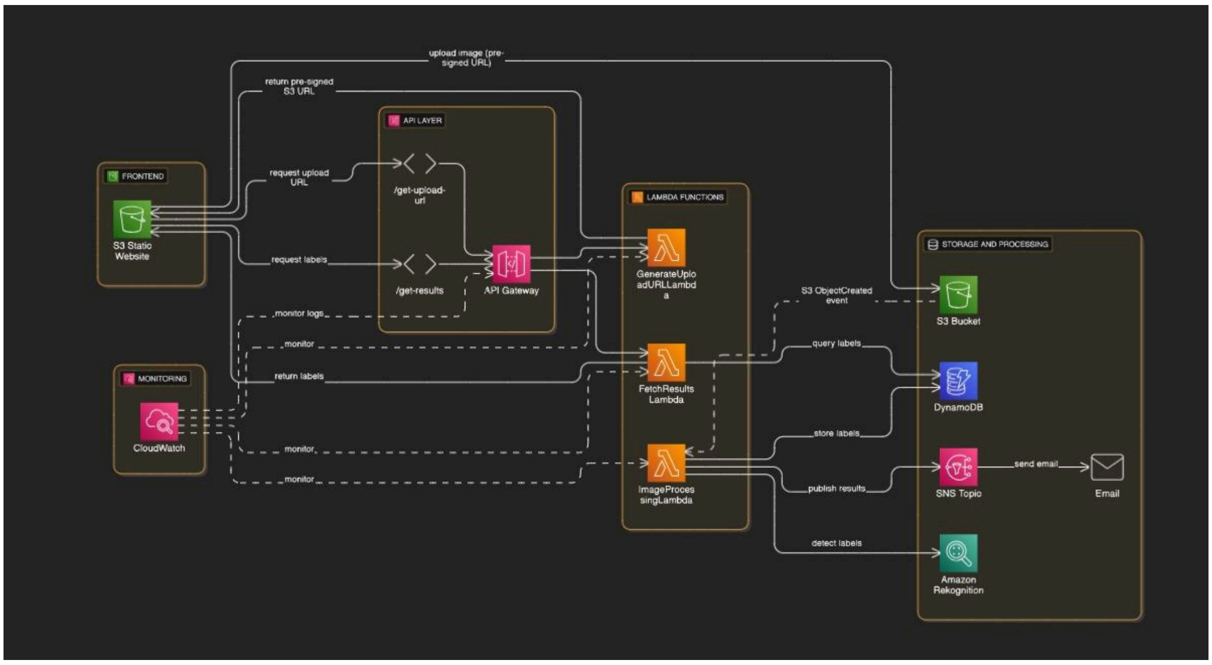
By default, we only load up to 10000 log groups.

Filter log groups or try pattern search

Exact

Log group	Log class	Anomaly d...
/aws/lambda/DetectLabelsFunction	Standard	Configure
/aws/lambda/GenerateUploadURL	Standard	Configure
/aws/lambda/ProcessImageUpload	Standard	Configure

Architecture Diagram



5. AWS Services Used

Amazon S3: Stores uploaded images and triggers Lambda functions on new uploads.

AWS Lambda: Processes uploaded images, calls Rekognition, stores results in DynamoDB, sends SNS notifications, and generates pre-signed URLs.

Amazon Rekognition: Detects objects, people, and scenes in uploaded images.

Amazon DynamoDB: Stores detected labels and image metadata for later retrieval.

Amazon SNS: Sends email notifications with recognition results to users.

Amazon API Gateway: Exposes APIs (/generate-upload-url and /detect-labels) for frontend interaction.

Amazon CloudWatch: Provides logging, monitoring, and alerts for Lambda and API Gateway.

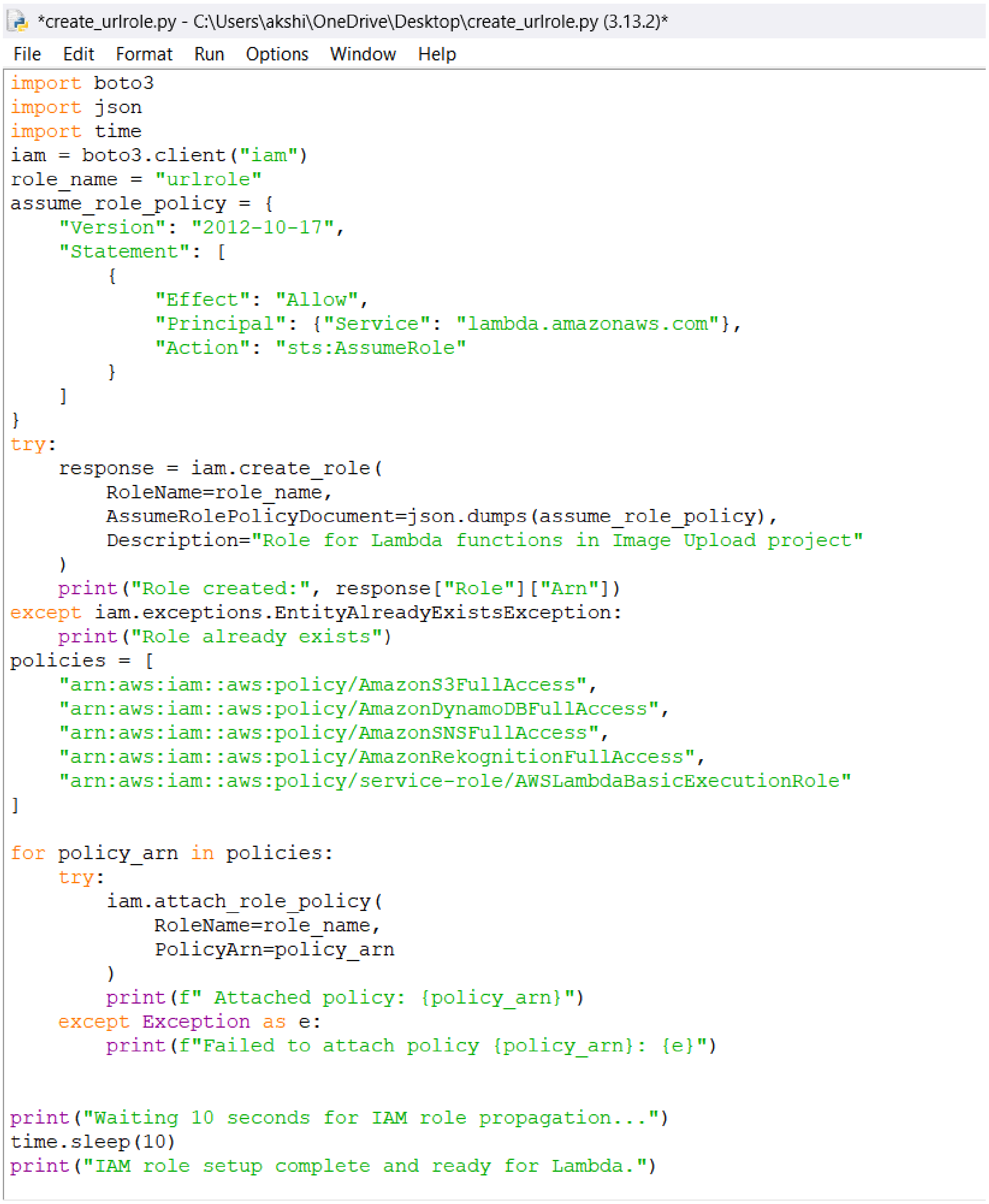
---

6. Deployment Steps

6.1 Backend Scripts (Automated via Python)

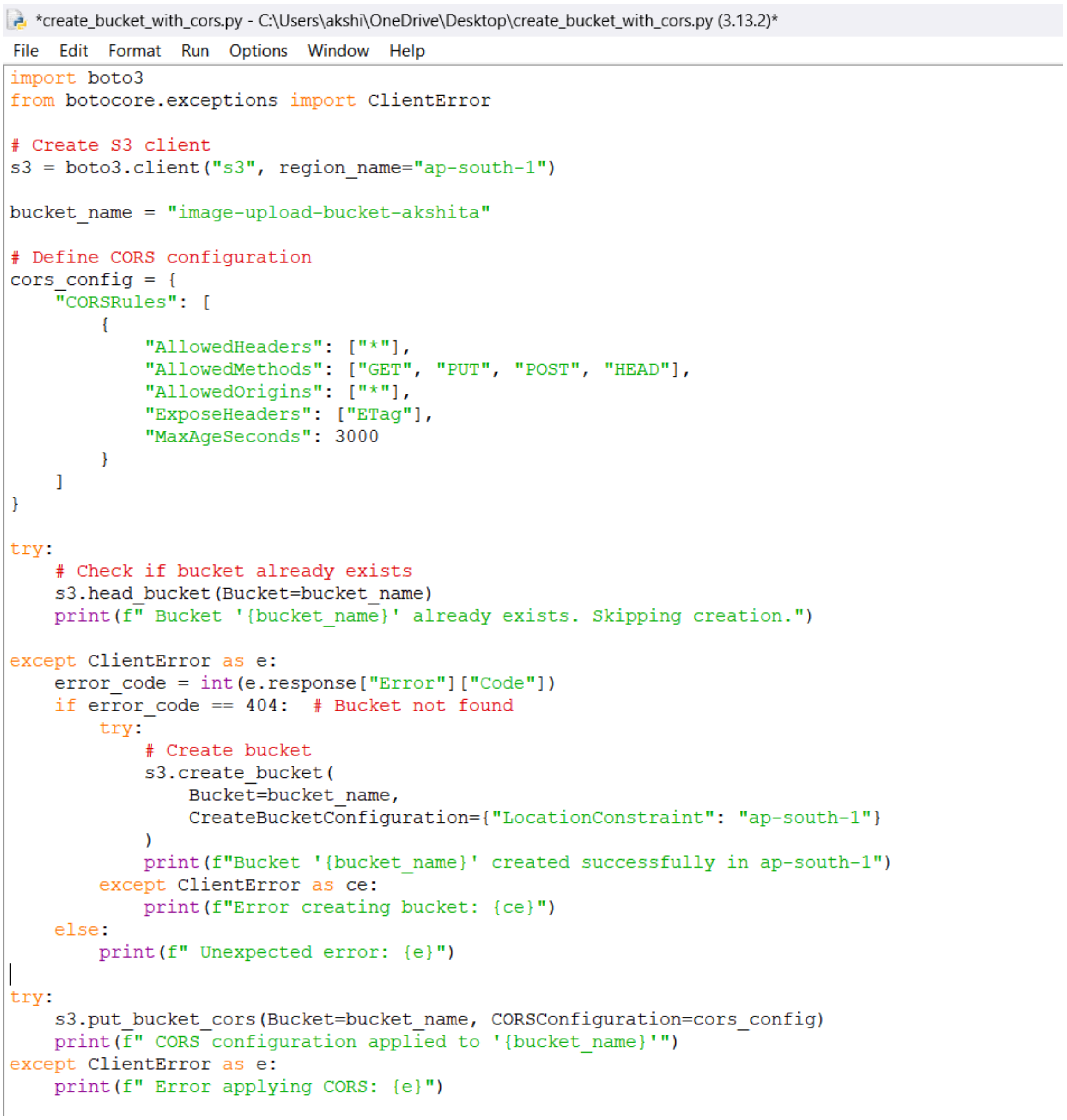
The backend infrastructure is mostly created using Python scripts with Boto3 for automation.

- **create_urlrole.py:** Creates an IAM role for Lambda functions with required permissions.


create_urlrole.py - C:\Users\akshi\OneDrive\Desktop\create_urlrole.py (3.13.2)
File Edit Format Run Options Window Help

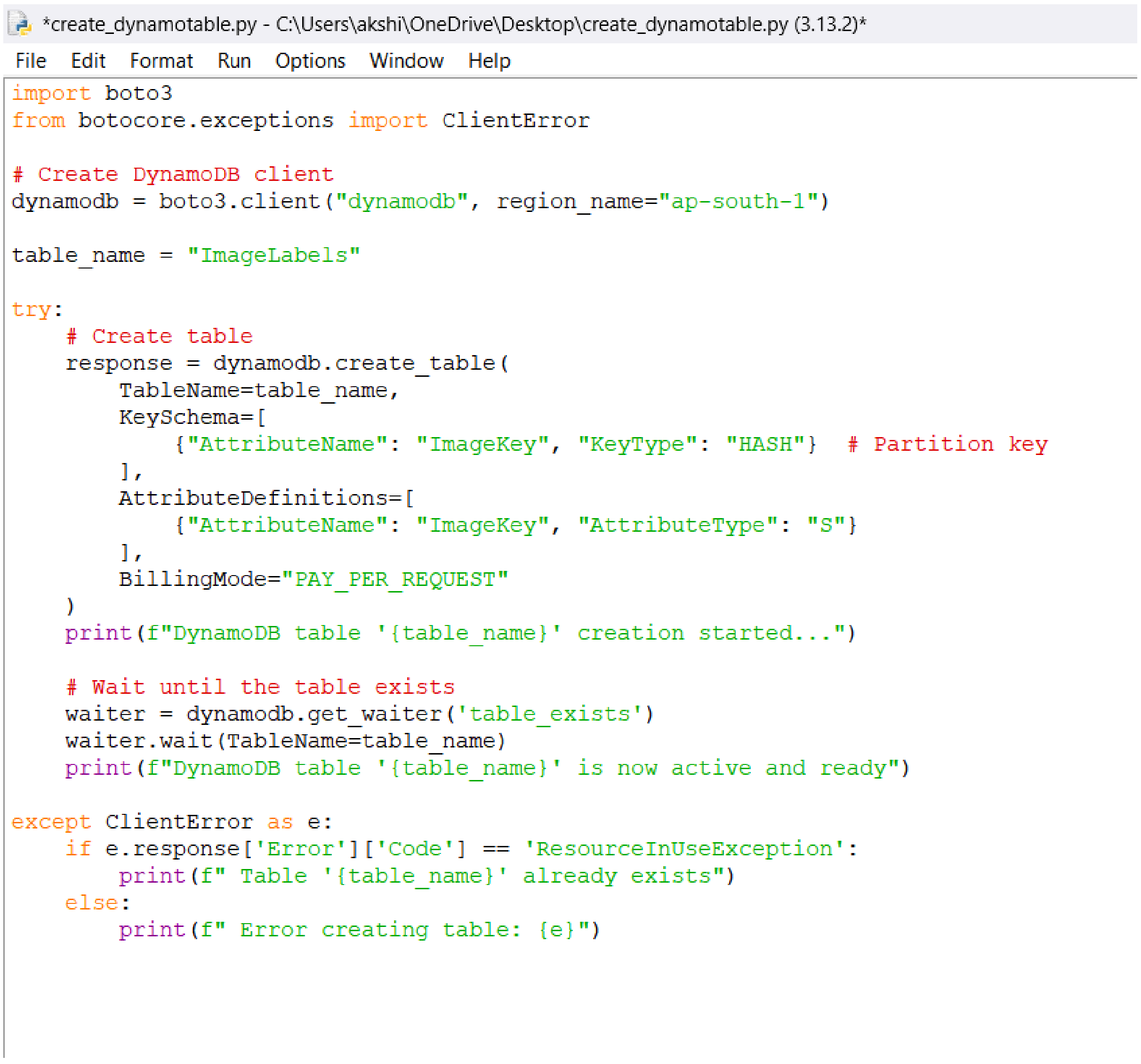
```python
import boto3
import json
import time
iam = boto3.client("iam")
role_name = "urlrole"
assume_role_policy = {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {"Service": "lambda.amazonaws.com"},
            "Action": "sts:AssumeRole"
        }
    ]
}
try:
    response = iam.create_role(
        RoleName=role_name,
        AssumeRolePolicyDocument=json.dumps(assume_role_policy),
        Description="Role for Lambda functions in Image Upload project"
    )
    print("Role created:", response["Role"]["Arn"])
except iam.exceptions.EntityAlreadyExistsException:
    print("Role already exists")
policies = [
    "arn:aws:iam::aws:policy/AmazonS3FullAccess",
    "arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess",
    "arn:aws:iam::aws:policy/AmazonSNSFullAccess",
    "arn:aws:iam::aws:policy/AmazonRekognitionFullAccess",
    "arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole"
]

for policy_arn in policies:
    try:
        iam.attach_role_policy(
            RoleName=role_name,
            PolicyArn=policy_arn
        )
        print(f" Attached policy: {policy_arn}")
    except Exception as e:
        print(f"Failed to attach policy {policy_arn}: {e}")


print("Waiting 10 seconds for IAM role propagation...")
time.sleep(10)
print("IAM role setup complete and ready for Lambda.")
```

- **create_bucket_with_cors.py:** Creates an S3 bucket and applies CORS configuration to allow frontend uploads.


create_bucket_with_cors.py - C:\Users\akshi\OneDrive\Desktop\create_bucket_with_cors.py (3.13.2)
File Edit Format Run Options Window Help

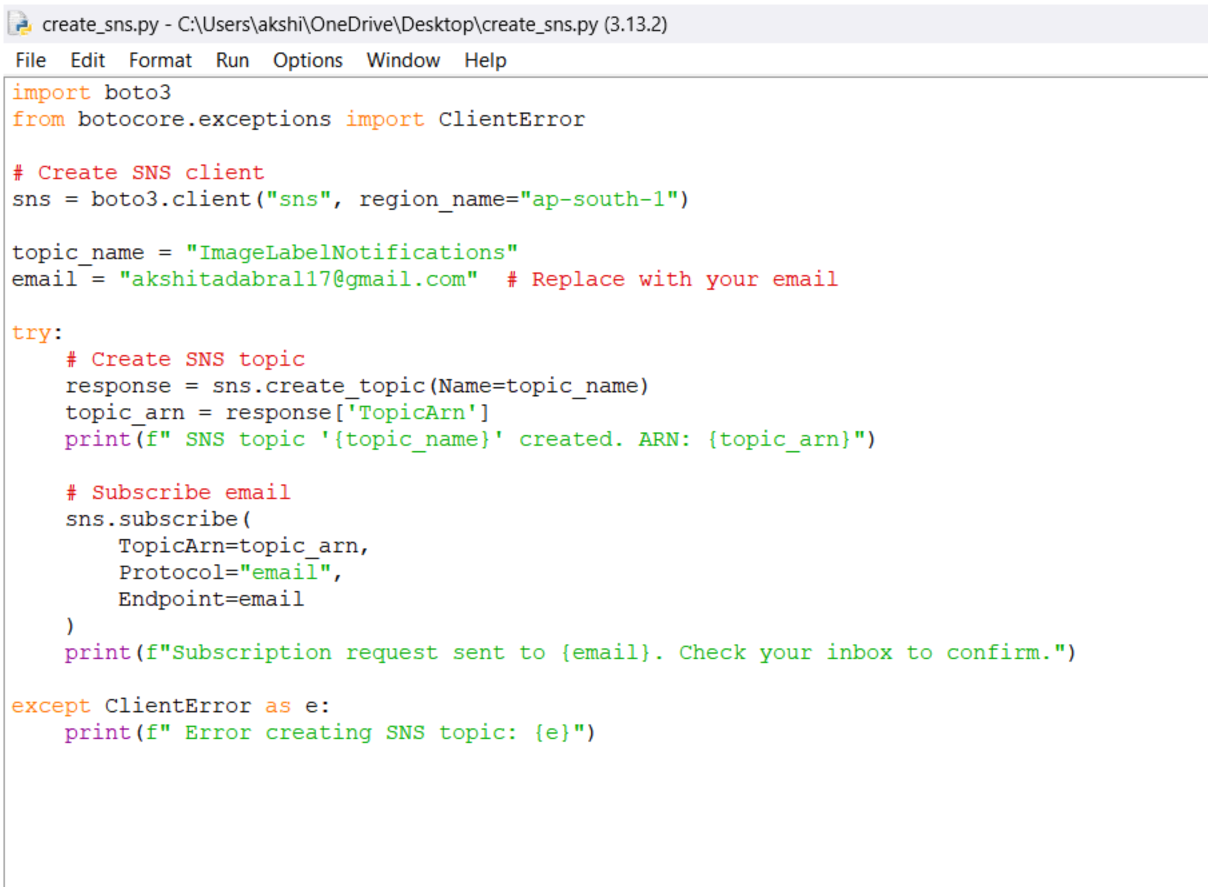
```python
import boto3
from botocore.exceptions import ClientError

# Create S3 client
s3 = boto3.client("s3", region_name="ap-south-1")

bucket_name = "image-upload-bucket-akshita"

# Define CORS configuration
cors_config = {
    "CORSRules": [
        {
            "AllowedHeaders": ["*"],
            "AllowedMethods": ["GET", "PUT", "POST", "HEAD"],
            "AllowedOrigins": ["*"],
            "ExposeHeaders": ["ETag"],
            "MaxAgeSeconds": 3000
        }
    ]
}

try:
    # Check if bucket already exists
    s3.head_bucket(Bucket=bucket_name)
    print(f" Bucket '{bucket_name}' already exists. Skipping creation.")

except ClientError as e:
    error_code = int(e.response["Error"]["Code"])
    if error_code == 404:  # Bucket not found
        try:
            # Create bucket
            s3.create_bucket(
                Bucket=bucket_name,
                CreateBucketConfiguration={"LocationConstraint": "ap-south-1"}
            )
            print(f"Bucket '{bucket_name}' created successfully in ap-south-1")
        except ClientError as ce:
            print(f"Error creating bucket: {ce}")
    else:
        print(f" Unexpected error: {e}")

try:
    s3.put_bucket_cors(Bucket=bucket_name, CORSConfiguration=cors_config)
    print(f" CORS configuration applied to '{bucket_name}'")
except ClientError as e:
    print(f" Error applying CORS: {e}")
```

- **create_dynamotable.py:** Creates a DynamoDB table to store image metadata and detected labels.


create_dynamotable.py - C:\Users\akshi\OneDrive\Desktop\create_dynamotable.py (3.13.2)
File Edit Format Run Options Window Help

```python
import boto3
from botocore.exceptions import ClientError

# Create DynamoDB client
dynamodb = boto3.client("dynamodb", region_name="ap-south-1")

table_name = "ImageLabels"

try:
    # Create table
    response = dynamodb.create_table(
        TableName=table_name,
        KeySchema=[
            {"AttributeName": "ImageKey", "KeyType": "HASH"}  # Partition key
        ],
        AttributeDefinitions=[
            {"AttributeName": "ImageKey", "AttributeType": "S"}
        ],
        BillingMode="PAY_PER_REQUEST"
    )
    print(f"DynamoDB table '{table_name}' creation started...")

    # Wait until the table exists
    waiter = dynamodb.get_waiter('table_exists')
    waiter.wait(TableName=table_name)
    print(f"DynamoDB table '{table_name}' is now active and ready")

except ClientError as e:
    if e.response['Error']['Code'] == 'ResourceInUseException':
        print(f" Table '{table_name}' already exists")
    else:
        print(f" Error creating table: {e}")
```

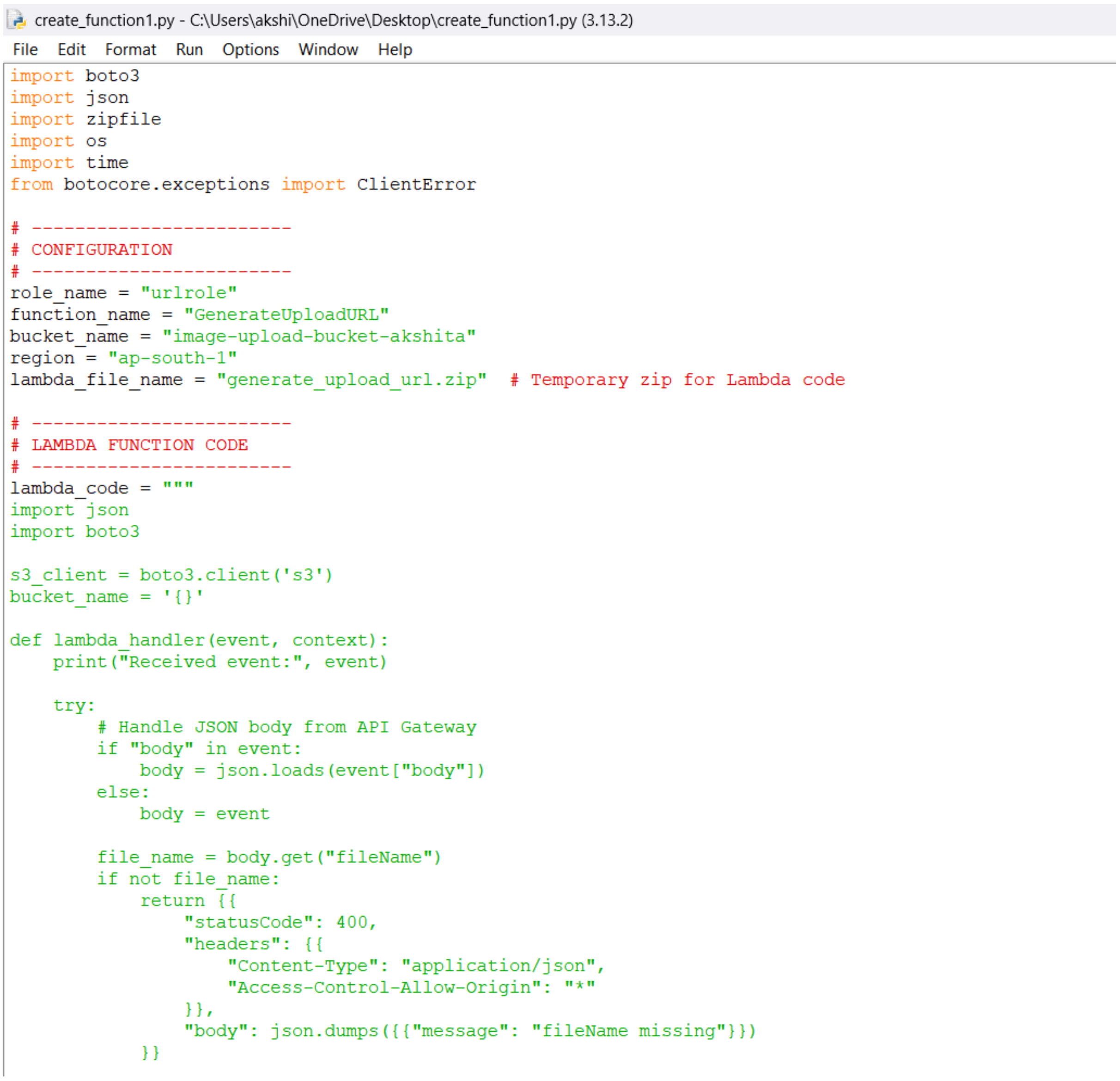
- **create_sns.py:** Creates an SNS topic and adds a subscription for email notifications.


create_sns.py - C:\Users\akshi\OneDrive\Desktop\create_sns.py (3.13.2)
File Edit Format Run Options Window Help

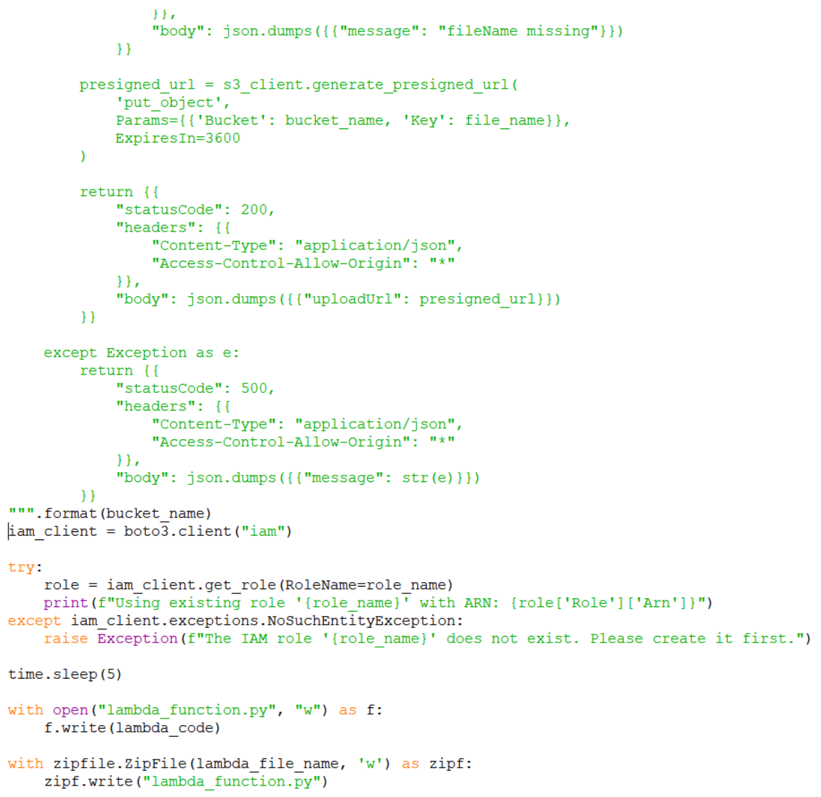
```python
import boto3
from botocore.exceptions import ClientError

# Create SNS client
sns = boto3.client("sns", region_name="ap-south-1")

topic_name = "ImageLabelNotifications"
email = "akshitadabral17@gmail.com"  # Replace with your email

try:
    # Create SNS topic
    response = sns.create_topic(Name=topic_name)
    topic_arn = response['TopicArn']
    print(f" SNS topic '{topic_name}' created. ARN: {topic_arn}")

    # Subscribe email
    sns.subscribe(
        TopicArn=topic_arn,
        Protocol="email",
        Endpoint=email
    )
    print(f"Subscription request sent to {email}. Check your inbox to confirm.")

except ClientError as e:
    print(f" Error creating SNS topic: {e}")
```

- **create_function1.py:**

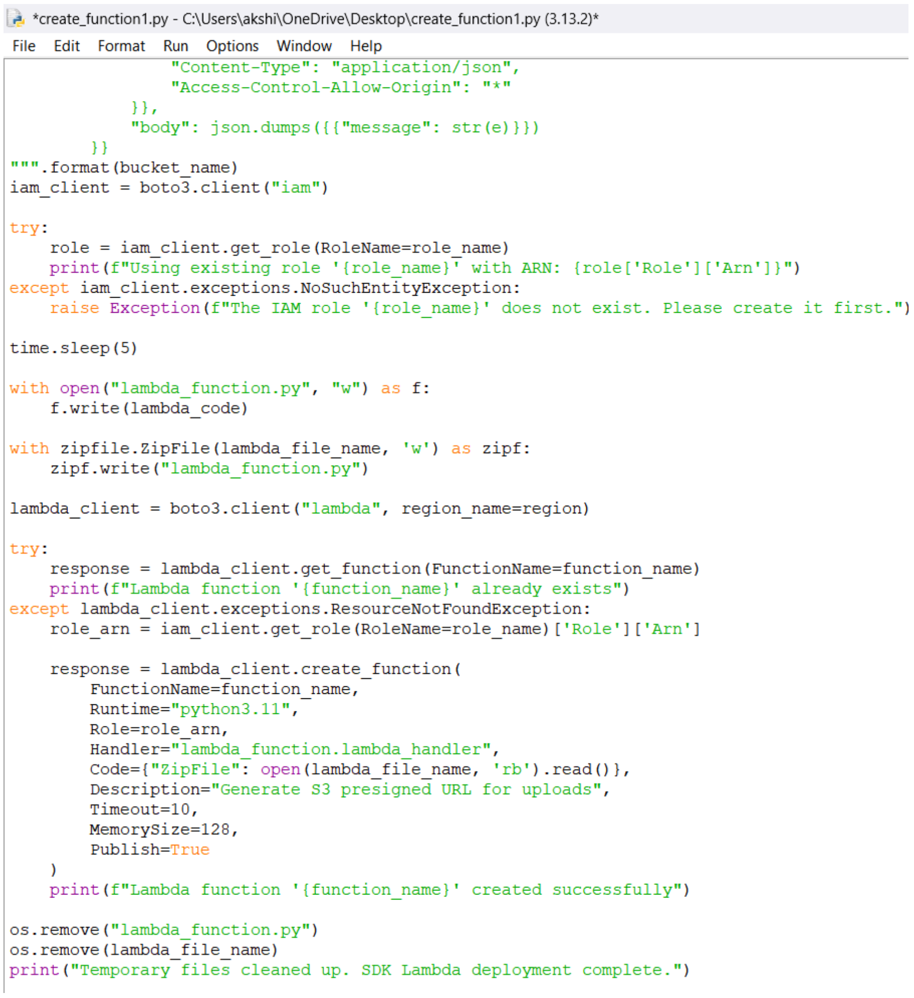
  Deploys the GenerateUploadURLLambda function for generating pre-signed URLs.


create_function1.py - C:\Users\akshi\OneDrive\Desktop\create_function1.py (3.13.2)
File Edit Format Run Options Window Help

```
import boto3
import json
import zipfile
import os
import time
from botocore.exceptions import ClientError

# ------------------------
# CONFIGURATION
# ------------------------
role_name = "urlrole"
function_name = "GenerateUploadURL"
bucket_name = "image-upload-bucket-akshita"
region = "ap-south-1"
lambda_file_name = "generate_upload_url.zip"  # Temporary zip for Lambda code

# ------------------------
# LAMBDA FUNCTION CODE
# ------------------------
lambda_code = """
import json
import boto3

s3_client = boto3.client('s3')
bucket_name = '{}'

def lambda_handler(event, context):
    print("Received event:", event)

    try:
        # Handle JSON body from API Gateway
        if "body" in event:
            body = json.loads(event["body"])
        else:
            body = event

        file_name = body.get("fileName")
        if not file_name:
            return {{
                "statusCode": 400,
                "headers": {{
                    "Content-Type": "application/json",
                    "Access-Control-Allow-Origin": "*"
                }},
                "body": json.dumps({{"message": "fileName missing"}})
            }}
```



```
                }},
                "body": json.dumps({{"message": "fileName missing"}})
            }}

        presigned_url = s3_client.generate_presigned_url(
            'put_object',
            Params={{'Bucket': bucket_name, 'Key': file_name}},
            ExpiresIn=3600
        )

        return {{
            "statusCode": 200,
            "headers": {{
                "Content-Type": "application/json",
                "Access-Control-Allow-Origin": "*"
            }},
            "body": json.dumps({{"uploadUrl": presigned_url}})
        }}

    except Exception as e:
        return {{
            "statusCode": 500,
            "headers": {{
                "Content-Type": "application/json",
                "Access-Control-Allow-Origin": "*"
            }},
            "body": json.dumps({{"message": str(e)}})
        }}
""".format(bucket_name)
iam_client = boto3.client("iam")

try:
    role = iam_client.get_role(RoleName=role_name)
    print(f"Using existing role '{role_name}' with ARN: {role['Role']['Arn']}")
except iam_client.exceptions.NoSuchEntityException:
    raise Exception(f"The IAM role '{role_name}' does not exist. Please create it first.")

time.sleep(5)

with open("lambda_function.py", "w") as f:
    f.write(lambda_code)

with zipfile.ZipFile(lambda_file_name, 'w') as zipf:
    zipf.write("lambda_function.py")
```

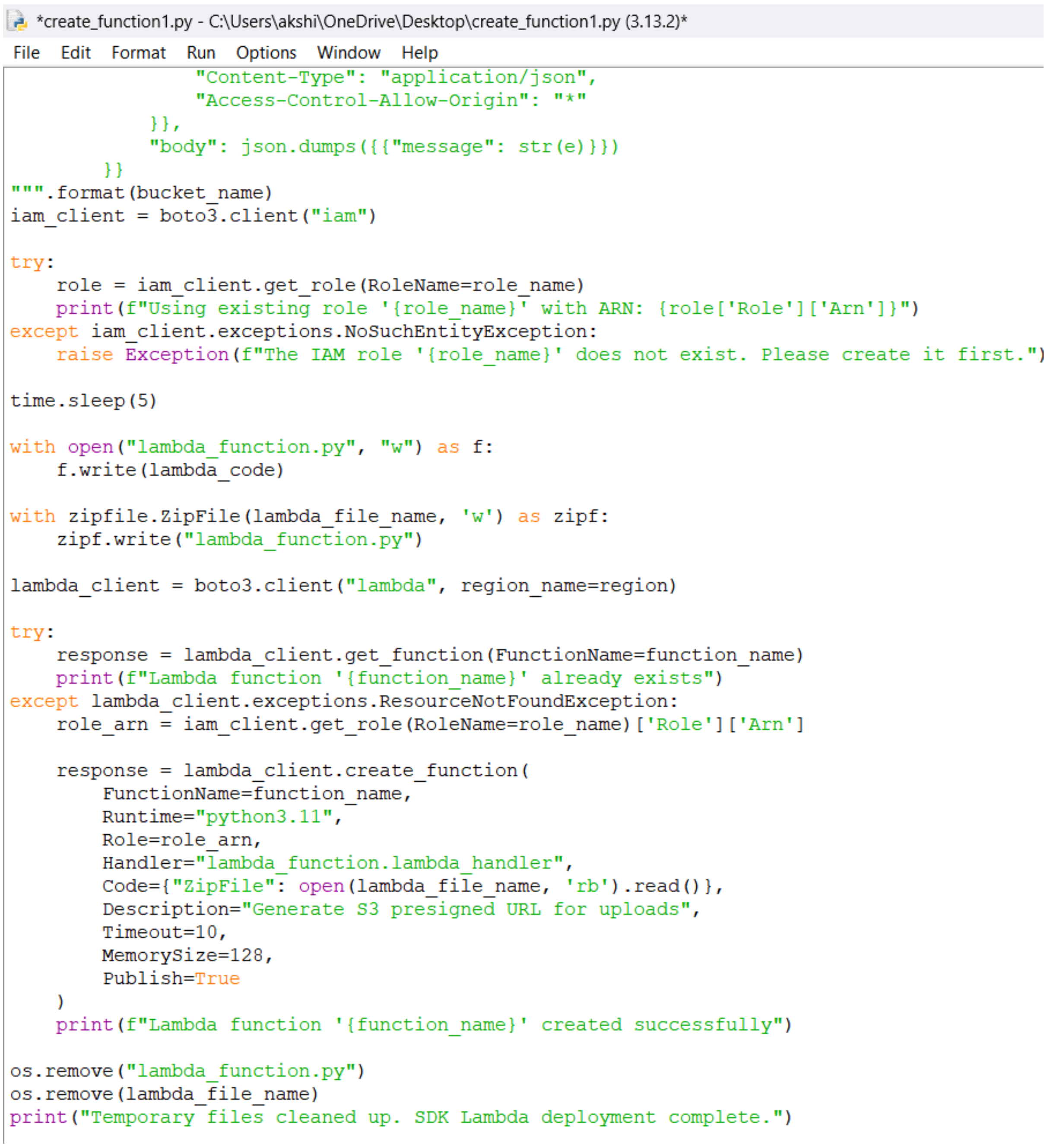


create_function1.py - C:\Users\akshi\OneDrive\Desktop\create_function1.py (3.13.2)

```
                "Content-Type": "application/json",
                "Access-Control-Allow-Origin": "*"
            }},
            "body": json.dumps({{"message": str(e)}})
        }}
""".format(bucket_name)
iam_client = boto3.client("iam")

try:
    role = iam_client.get_role(RoleName=role_name)
    print(f"Using existing role '{role_name}' with ARN: {role['Role']['Arn']}")
except iam_client.exceptions.NoSuchEntityException:
    raise Exception(f"The IAM role '{role_name}' does not exist. Please create it first.")

time.sleep(5)

with open("lambda_function.py", "w") as f:
    f.write(lambda_code)

with zipfile.ZipFile(lambda_file_name, 'w') as zipf:
    zipf.write("lambda_function.py")

lambda_client = boto3.client("lambda", region_name=region)

try:
    response = lambda_client.get_function(FunctionName=function_name)
    print(f"Lambda function '{function_name}' already exists")
except lambda_client.exceptions.ResourceNotFoundException:
    role_arn = iam_client.get_role(RoleName=role_name)['Role']['Arn']

    response = lambda_client.create_function(
        FunctionName=function_name,
        Runtime="python3.11",
        Role=role_arn,
        Handler="lambda_function.lambda_handler",
        Code={"ZipFile": open(lambda_file_name, 'rb').read()},
        Description="Generate S3 presigned URL for uploads",
        Timeout=10,
        MemorySize=128,
        Publish=True
    )
    print(f"Lambda function '{function_name}' created successfully")

os.remove("lambda_function.py")
os.remove(lambda_file_name)
print("Temporary files cleaned up. SDK Lambda deployment complete.")
```

- **create_function2.py:**

  Deploys the ProcessImageUploadLambda function, which analyzes uploaded images and stores results.

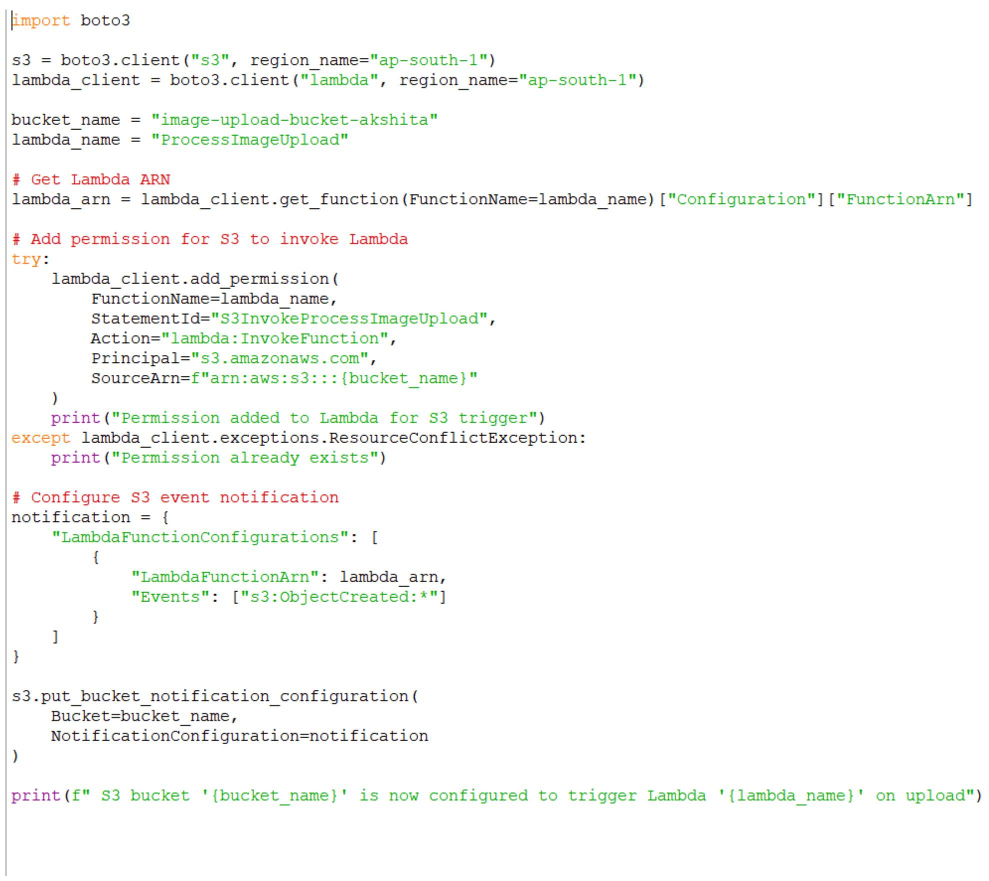

create_function1.py - C:\Users\akshi\OneDrive\Desktop\create_function1.py (3.13.2)
File Edit Format Run Options Window Help

```
                "Content-Type": "application/json",
                "Access-Control-Allow-Origin": "*"
            }},
            "body": json.dumps({{"message": str(e)}})
        }}
""".format(bucket_name)
iam_client = boto3.client("iam")

try:
    role = iam_client.get_role(RoleName=role_name)
    print(f"Using existing role '{role_name}' with ARN: {role['Role']['Arn']}")
except iam_client.exceptions.NoSuchEntityException:
    raise Exception(f"The IAM role '{role_name}' does not exist. Please create it first.")

time.sleep(5)

with open("lambda_function.py", "w") as f:
    f.write(lambda_code)

with zipfile.ZipFile(lambda_file_name, 'w') as zipf:
    zipf.write("lambda_function.py")

lambda_client = boto3.client("lambda", region_name=region)

try:
    response = lambda_client.get_function(FunctionName=function_name)
    print(f"Lambda function '{function_name}' already exists")
except lambda_client.exceptions.ResourceNotFoundException:
    role_arn = iam_client.get_role(RoleName=role_name)['Role']['Arn']

    response = lambda_client.create_function(
        FunctionName=function_name,
        Runtime="python3.11",
        Role=role_arn,
        Handler="lambda_function.lambda_handler",
        Code={"ZipFile": open(lambda_file_name, 'rb').read()},
        Description="Generate S3 presigned URL for uploads",
        Timeout=10,
        MemorySize=128,
        Publish=True
    )
    print(f"Lambda function '{function_name}' created successfully")

os.remove("lambda_function.py")
os.remove(lambda_file_name)
print("Temporary files cleaned up. SDK Lambda deployment complete.")
```

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

    response = rekognition.detect_labels(
        Image={'S3Object': {'Bucket': bucket, 'Name': key}},
        MaxLabels=10,
        MinConfidence=80
    )
    labels = [label['Name'] for label in response['Labels']]

    table.put_item(Item={
        'ImageKey': key,
        'Labels': labels,
        'UploadTime': datetime.utcnow().isoformat()
    })

    sns.publish(
        TopicArn=sns_topic_arn,
        Message=f"Labels detected for {key}: {', '.join(labels)}"
    )

    return {'statusCode': 200, 'body': f"Processed {key} successfully"}
"""

# Create in-memory zip
zip_buffer = io.BytesIO()
with zipfile.ZipFile(zip_buffer, 'w', zipfile.ZIP_DEFLATED) as zf:
    zf.writestr('process_image_upload.py', lambda_code_str)

zip_buffer.seek(0)

# Create Lambda function
try:
    lambda_client.get_function(FunctionName=function_name)
    print(f"Lambda function '{function_name}' already exists")
except lambda_client.exceptions.ResourceNotFoundException:
    response = lambda_client.create_function(
        FunctionName=function_name,
        Runtime='python3.11',
        Role=role_arn,
        Handler='process_image_upload.lambda_handler',  # matches file inside zip
        Code={'ZipFile': zip_buffer.read()},
        Timeout=15,
        MemorySize=128,
        Publish=True
    )
    print(f"Lambda function '{function_name}' created successfully")
```

- create_s3trigger.py:

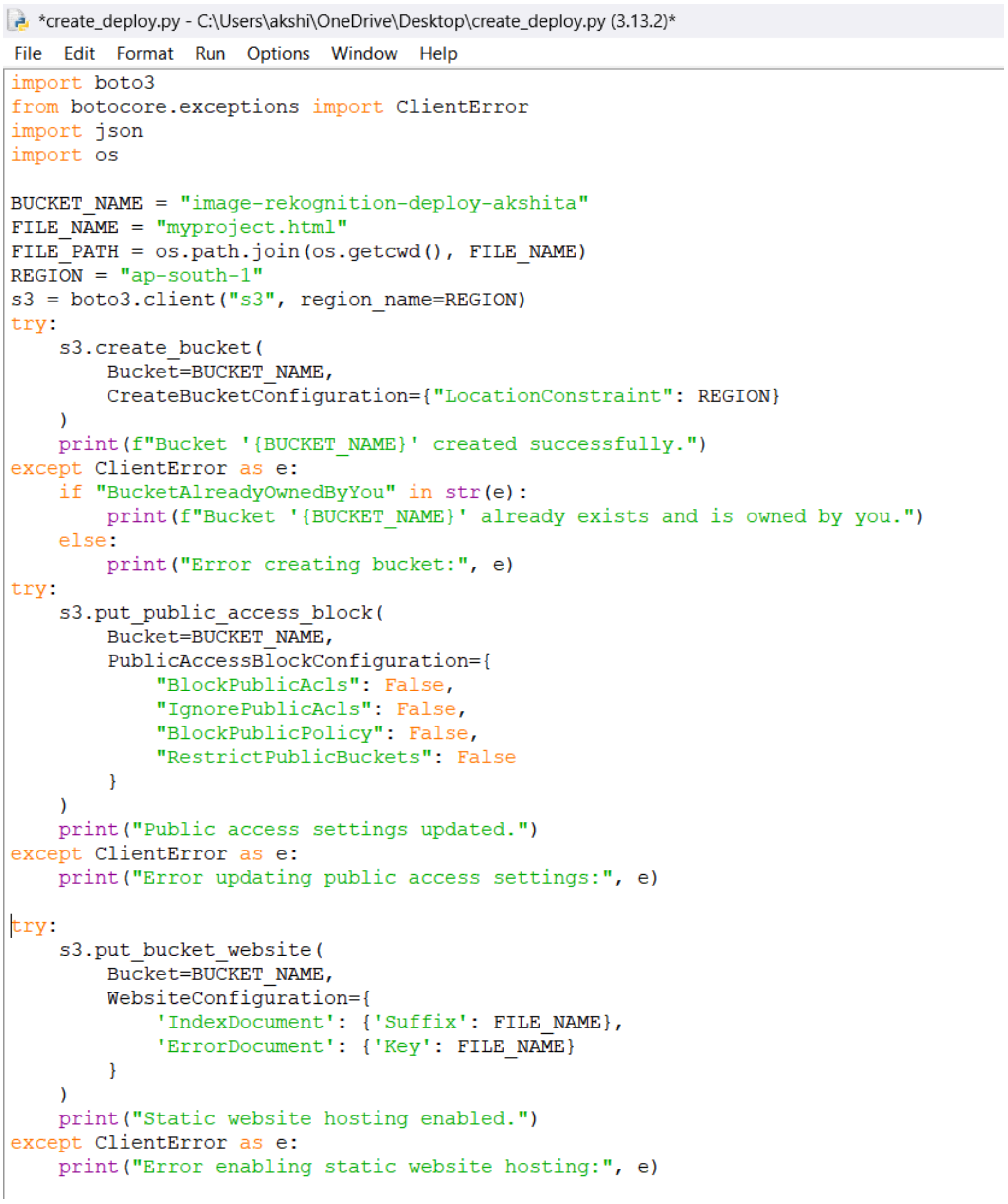
  Configures S3 bucket events to trigger ProcessImageUploadLambda automatically on image upload.



```python
import boto3

s3 = boto3.client("s3", region_name="ap-south-1")
lambda_client = boto3.client("lambda", region_name="ap-south-1")

bucket_name = "image-upload-bucket-akshita"
lambda_name = "ProcessImageUpload"

# Get Lambda ARN
lambda_arn = lambda_client.get_function(FunctionName=lambda_name)["Configuration"]["FunctionArn"]

# Add permission for S3 to invoke Lambda
try:
    lambda_client.add_permission(
        FunctionName=lambda_name,
        StatementId="S3InvokeProcessImageUpload",
        Action="lambda:InvokeFunction",
        Principal="s3.amazonaws.com",
        SourceArn=f"arn:aws:s3:::{bucket_name}"
    )
    print("Permission added to Lambda for S3 trigger")
except lambda_client.exceptions.ResourceConflictException:
    print("Permission already exists")

# Configure S3 event notification
notification = {
    "LambdaFunctionConfigurations": [
        {
            "LambdaFunctionArn": lambda_arn,
            "Events": ["s3:ObjectCreated:*"]
        }
    ]
}

s3.put_bucket_notification_configuration(
    Bucket=bucket_name,
    NotificationConfiguration=notification
)

print(f" S3 bucket '{bucket_name}' is now configured to trigger Lambda '{lambda_name}' on upload")
```

- create_deploy.py:

  Deploys the frontend static website to an S3 hosting bucket.


create_deploy.py - C:\Users\akshi\OneDrive\Desktop\create_deploy.py (3.13.2)
File Edit Format Run Options Window Help

```python
import boto3
from botocore.exceptions import ClientError
import json
import os

BUCKET_NAME = "image-rekognition-deploy-akshita"
FILE_NAME = "myproject.html"
FILE_PATH = os.path.join(os.getcwd(), FILE_NAME)
REGION = "ap-south-1"
s3 = boto3.client("s3", region_name=REGION)
try:
    s3.create_bucket(
        Bucket=BUCKET_NAME,
        CreateBucketConfiguration={"LocationConstraint": REGION}
    )
    print(f"Bucket '{BUCKET_NAME}' created successfully.")
except ClientError as e:
    if "BucketAlreadyOwnedByYou" in str(e):
        print(f"Bucket '{BUCKET_NAME}' already exists and is owned by you.")
    else:
        print("Error creating bucket:", e)
try:
    s3.put_public_access_block(
        Bucket=BUCKET_NAME,
        PublicAccessBlockConfiguration={
            "BlockPublicAcls": False,
            "IgnorePublicAcls": False,
            "BlockPublicPolicy": False,
            "RestrictPublicBuckets": False
        }
    )
    print("Public access settings updated.")
except ClientError as e:
    print("Error updating public access settings:", e)

try:
    s3.put_bucket_website(
        Bucket=BUCKET_NAME,
        WebsiteConfiguration={
            'IndexDocument': {'Suffix': FILE_NAME},
            'ErrorDocument': {'Key': FILE_NAME}
        }
    )
    print("Static website hosting enabled.")
except ClientError as e:
    print("Error enabling static website hosting:", e)
```

```
try:
    s3.put_bucket_website(
        Bucket=BUCKET_NAME,
        WebsiteConfiguration={
            'IndexDocument': {'Suffix': FILE_NAME},
            'ErrorDocument': {'Key': FILE_NAME}
        }
    )
    print("Static website hosting enabled.")
except ClientError as e:
    print("Error enabling static website hosting:", e)

bucket_policy = {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublicReadAccess",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:GetObject",
            "Resource": f"arn:aws:s3:::{BUCKET_NAME}/*"
        }
    ]
}

try:
    s3.put_bucket_policy(Bucket=BUCKET_NAME, Policy=json.dumps(bucket_policy))
    print("Bucket policy applied.")
except ClientError as e:
    print("Error setting bucket policy:", e)

if os.path.exists(FILE_PATH):
    try:
        s3.upload_file(FILE_PATH, BUCKET_NAME, FILE_NAME, ExtraArgs={'ContentType': 'text/html'})
        print(f"File '{FILE_NAME}' uploaded successfully.")
    except ClientError as e:
        print("Error uploading file:", e)
else:
    print(f"File '{FILE_PATH}' not found. Check the path.")

website_url = f"http://{BUCKET_NAME}.s3-website.{REGION}.amazonaws.com"
print(f"Website deployed! Access it at: {website_url}")
```

6.2 Manual Steps (AWS Console)

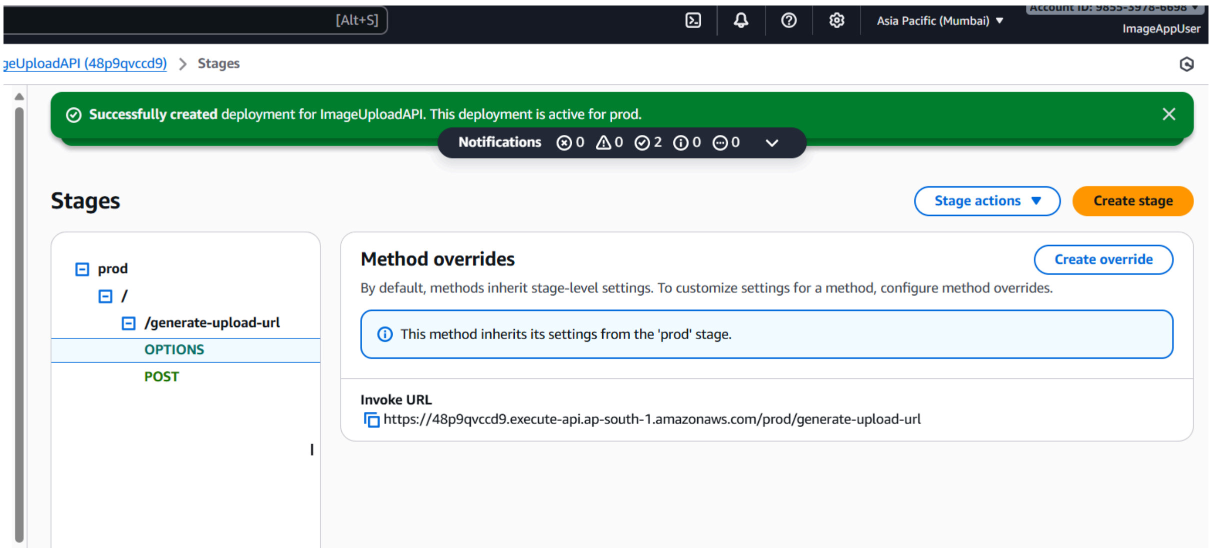
A few steps are done manually using the AWS Management Console.

- **DetectLabelsLambda:** Manually create the Lambda function and paste the detectlabels.py code into the editor:

```
import boto3
import json


rekognition = boto3.client("rekognition")


def lambda_handler(event, context):
    try:
        # Handle proxy or non-proxy event
        if "body" in event:
            body = json.loads(event["body"])
        else:
            body = event

        bucket = body["bucket"]
        key = body["key"]

        response = rekognition.detect_labels(
            Image={"S3Object": {"Bucket": bucket, "Name": key}},
            MaxLabels=10,
            MinConfidence=70
        )

        labels = [label["Name"] for label in response["Labels"]]

        return {
            "statusCode": 200,
            "headers": {
                "Access-Control-Allow-Origin": "*",
                "Content-Type": "application/json"
            },
            "body": json.dumps({"labels": labels})
        }

    except Exception as e:
        return {
            "statusCode": 500,
            "headers": {"Access-Control-Allow-Origin": "*"},
            "body": json.dumps({"error": str(e)})
        }
```

- **API Gateway (/generate-upload-url):**

  Create a resource, add a POST method, integrate with GenerateUploadURLLambda, and deploy to production.



[Alt+S]

Asia Pacific (Mumbai) ▼

ImageAppUser

geUploadAPI (48p9qvccd9) > Stages

Successfully created deployment for ImageUploadAPI. This deployment is active for prod.

Notifications ⊗ 0 ⚠ 0 ✓ 2 ⓘ 0 ⊖ 0

Stages

Stage actions ▼ | Create stage

prod
/
/generate-upload-url
OPTIONS
POST

Method overrides

Create override

By default, methods inherit stage-level settings. To customize settings for a method, configure method overrides.

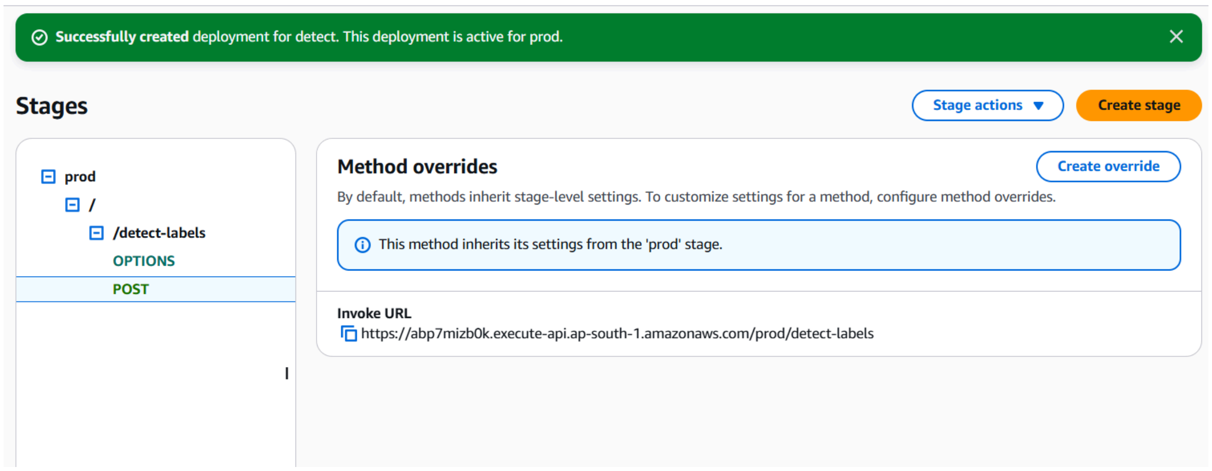
This method inherits its settings from the 'prod' stage.

Invoke URL

https://48p9qvccd9.execute-api.ap-south-1.amazonaws.com/prod/generate-upload-url

- **API Gateway (/detect-labels):**

  Create a resource, add a POST method, integrate with DetectLabelsLambda, enable CORS, and deploy.



Successfully created deployment for detect. This deployment is active for prod.

Stages

Stage actions ▼ | Create stage

prod
/
/detect-labels
OPTIONS
POST

Method overrides

Create override

By default, methods inherit stage-level settings. To customize settings for a method, configure method overrides.

This method inherits its settings from the 'prod' stage.

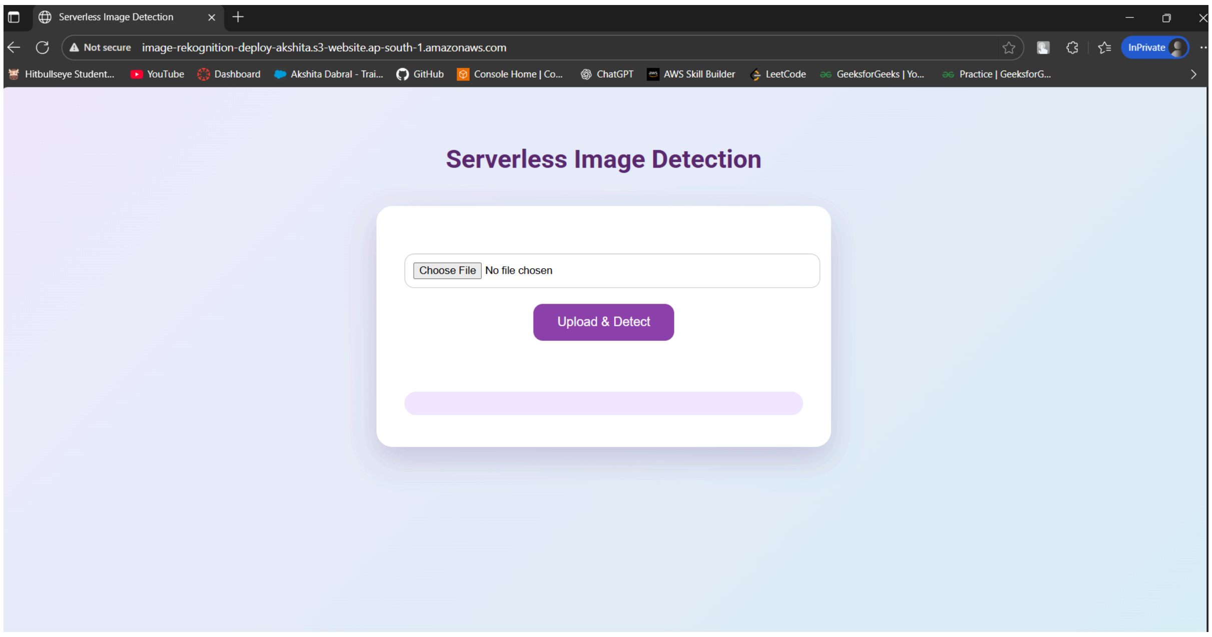
Invoke URL

https://abp7mizb0k.execute-api.ap-south-1.amazonaws.com/prod/detect-labels

---

7. User Workflow

1. Open the **static website** hosted on S3.

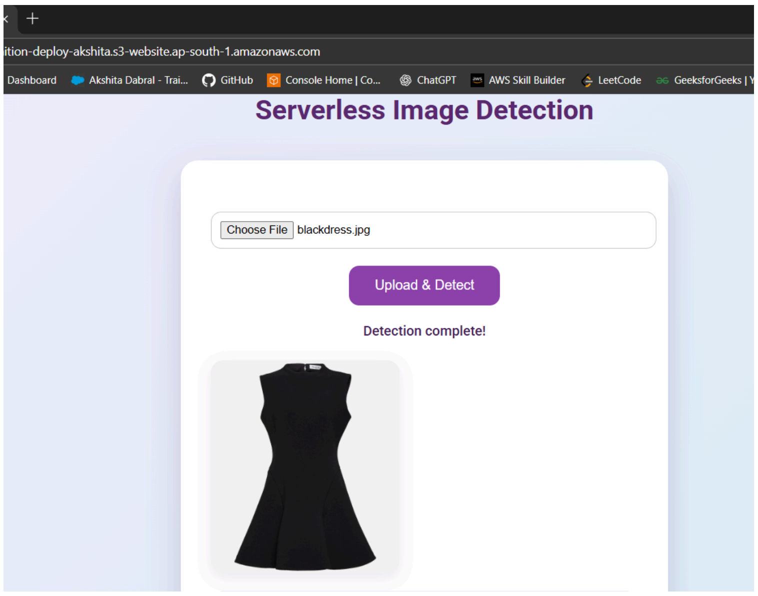

Serverless Image Detection
Not secure image-rekognition-deploy-akshita.s3-website.ap-south-1.amazonaws.com
Hitbullseye Student... YouTube Dashboard Akshita Dabral - Trai... GitHub Console Home | Co... ChatGPT AWS Skill Builder LeetCode GeeksforGeeks | Yo... Practice | GeeksforG...
InPrivate

Serverless Image Detection

Choose File No file chosen

Upload & Detect

2. Select and **upload an image**.


ition-deploy-akshita.s3-website.ap-south-1.amazonaws.com
Dashboard Akshita Dabral - Trai... GitHub Console Home | Co... ChatGPT AWS Skill Builder LeetCode GeeksforGeeks | Y

Serverless Image Detection

Choose File blackdress.jpg

Upload & Detect

Detection complete!

3. System automatically:
   - Uploads image to S3 securely using pre-signed URL.

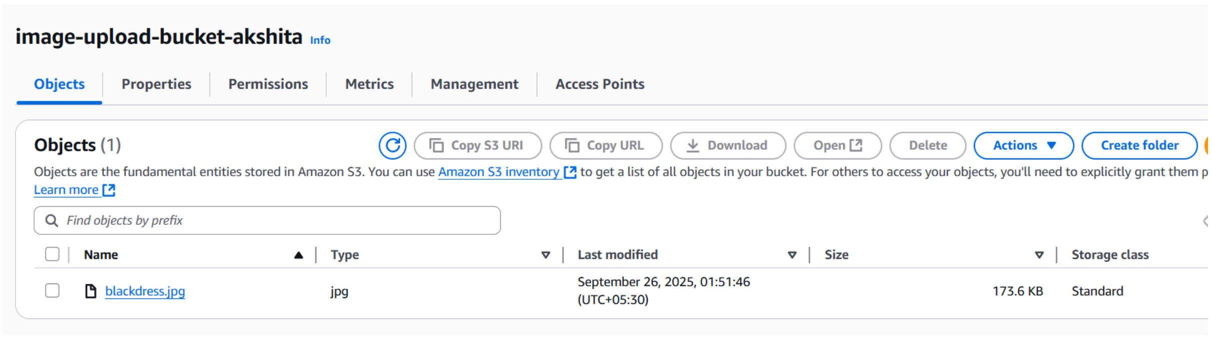


image-upload-bucket-akshita Info

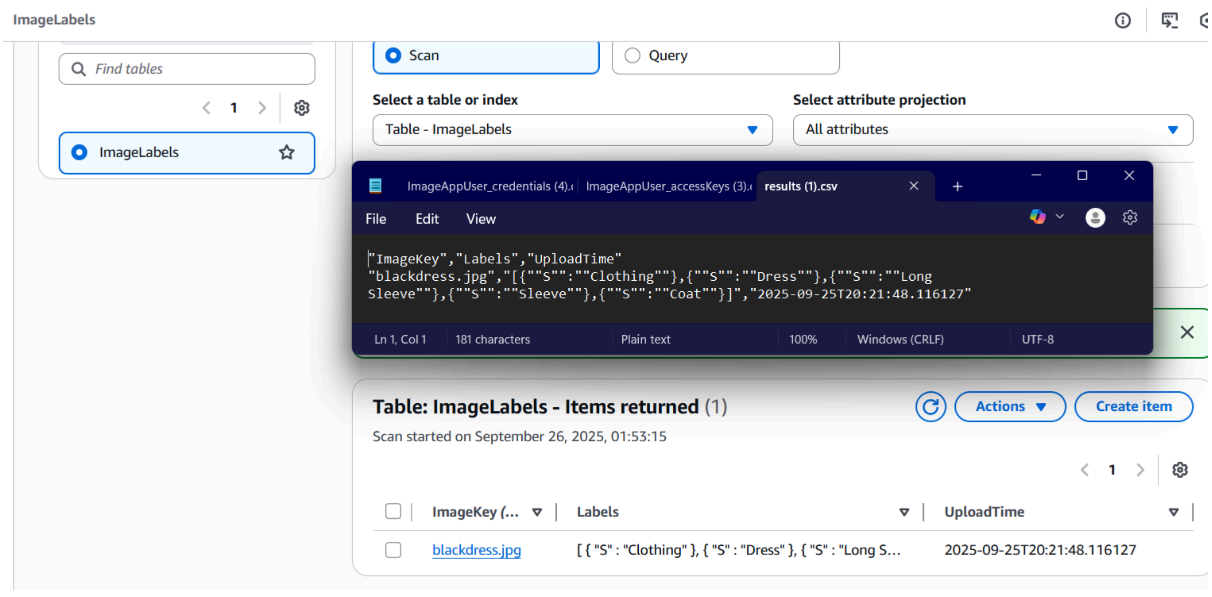
Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them p
Learn more

Find objects by prefix

Name	Type	Last modified	Size	Storage class
blackdress.jpg	jpg	September 26, 2025, 01:51:46 (UTC+05:30)	173.6 KB	Standard

- Analyzes image using Rekognition.
- Stores detected labels in DynamoDB.



ImageLabels

Select a table or index: Table - ImageLabels

Select attribute projection: All attributes

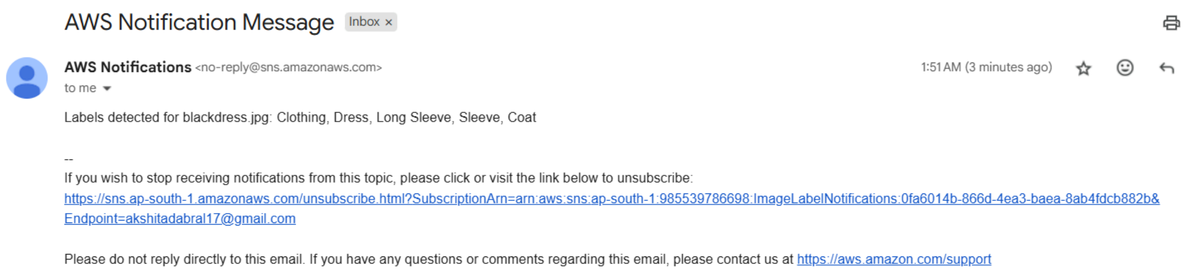
```
"ImageKey","Labels","UploadTime"
"blackdress.jpg","[{""S"":""Clothing""},{""S"":""Dress""},{""S"":""Long
Sleeve""},{""S"":""Sleeve""},{""S"":""Coat""}]","2025-09-25T20:21:48.116127"
```

Table: ImageLabels - Items returned (1)

Scan started on September 26, 2025, 01:53:15

ImageKey (...	Labels	UploadTime
blackdress.jpg	[{ "S" : "Clothing" }, { "S" : "Dress" }, { "S" : "Long S...	2025-09-25T20:21:48.116127

- Sends **email notification** via SNS.



AWS Notification Message Inbox ×

AWS Notifications <no-reply@sns.amazonaws.com>
to me

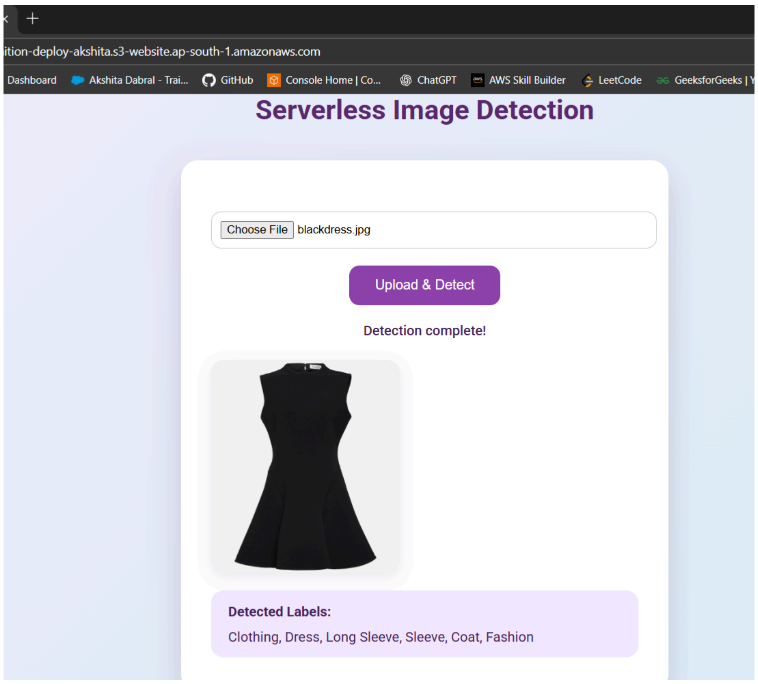
1:51 AM (3 minutes ago)

Labels detected for blackdress.jpg: Clothing, Dress, Long Sleeve, Sleeve, Coat

--
If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.ap-south-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:ap-south-1:985539786698:ImageLabelNotifications:0fa6014b-866d-4ea3-baea-8ab4fdcb882b&Endpoint=akshitadabral17@gmail.com

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at https://aws.amazon.com/support

4. User sees **detected labels** on the website.



Detection complete!

Detected Labels:
Clothing, Dress, Long Sleeve, Sleeve, Coat, Fashion

---

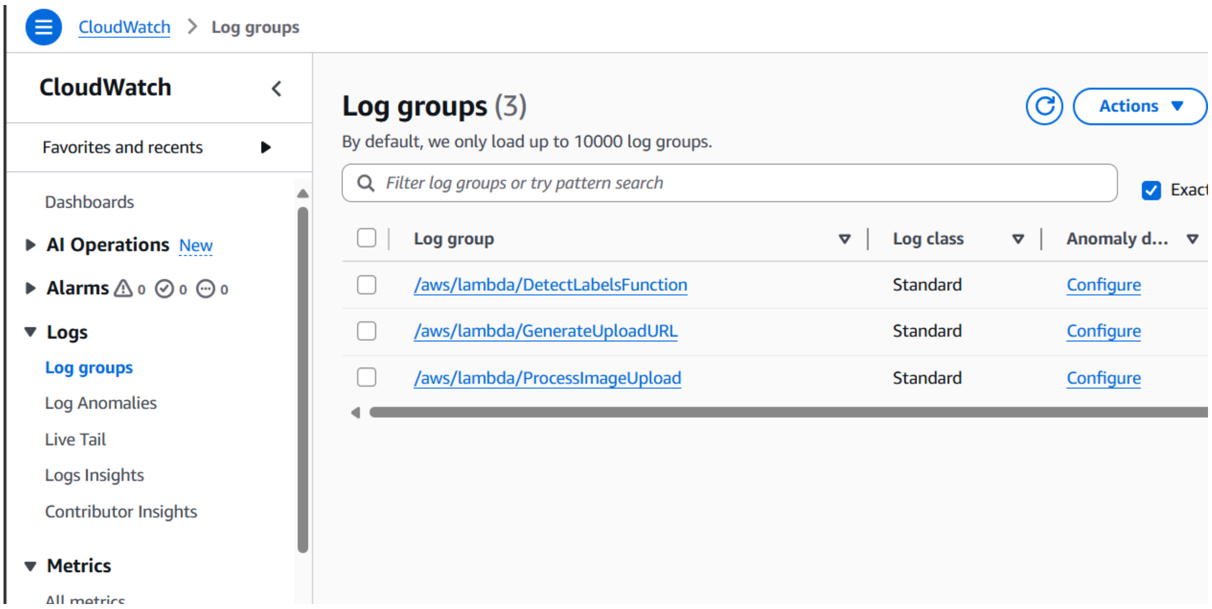
8. Security Measures

- **IAM Roles & Policies** – Least privilege applied.
- **S3 Security** – Public access blocked, uploads via pre-signed URLs.
- **API Gateway Security** – Only exposes Lambda endpoints securely.

---

9. Monitoring & Automation

- **CloudWatch** – Tracks Lambda executions, errors, and performance.
- **Event-driven Lambda** – Automatically triggers on S3 uploads.
- **Boto3 Scripts** – Automates backend resource creation.
- **SNS Notifications** – Sends alerts automatically after image analysis.



Log groups (3)

By default, we only load up to 10000 log groups.

Log group	Log class	Anomaly d...
/aws/lambda/DetectLabelsFunction	Standard	Configure
/aws/lambda/GenerateUploadURL	Standard	Configure
/aws/lambda/ProcessImageUpload	Standard	Configure

10. Repository Structure

```
AWS-Serverless-Image-Rekognition/
│
├── create_urlrole.py
├── create_bucket_with_cors.py
├── create_dynamotable.py
├── create_sns.py
├── create_function1.py
├── create_function2.py
├── create_s3trigger.py
├── create_deploy.py
├── detectlabels.py
├── myproject.html
├── Documentation
└── README.md
```