

# Lab 3

---

Saif Ali Athyaab - 23810756

Lab 3 covers 3 main topics

1. Uploading local directory to S3
2. Retrieving from S3 to local
3. Populating S3 details into DynamoDB

**Amazon S3** is an object storage service that provides industry-leading scalability, data availability, security, and performance. Amazon S3 can be used to store and retrieve any quantity of data at any time and from any location.

**Amazon DynamoDB** is a fully managed NoSQL database service with seamless scaling and quick and predictable performance.


Amazon DynamoDB's **core** components - Tables, objects, and attributes are the primary components with which you operate in DynamoDB. A **table** is made up of objects, and each item is made up of attributes. DynamoDB employs primary keys to uniquely identify each item in a table, as well as secondary indexes to increase query flexibility.

## Program

### [Step 1] Directory Preparation

I create a directory rootdir

 rootdir

Next I create a file in rootdir called rootfile.txt and put some content in it  poem Finally I create a second directory in rootdir called subdir and create another file subfile.txt with the same content as rootfile.txt

 subfile.txt

### [Step 2] Save to S3

Import libraries and save the necessary directory names into variables

```
import os
import boto3
ROOT_DIR = '/home/nsword/lab3'
ROOT_S3_DIR = '23810756-cloudstorage'
path_to_s3= ''
```

Initialize S3 and create bucket

```
s3 = boto3.client("s3")
bucket_config = {'LocationConstraint': 'ap-southeast-2'}
```

```
try:
    response = s3.create_bucket(Bucket=ROOT_S3_DIR,
                                CreateBucketConfiguration=bucket_config)
    print(response)
except Exception as error:
    pass
```

Construct the local path to replicate on S3


```
for root, dirs, files in os.walk(ROOT_DIR):
    for filename in files:
        # construct the full local path
        local_path = os.path.join(root, filename)
        # construct the full AWS path
        relative_path = os.path.relpath(local_path, ROOT_DIR)
        s3_path = os.path.join(path_to_s3, relative_path)
```


Search for the path and only then copy the path and files

```
try:
    s3.head_object(Bucket=ROOT_S3_DIR, Key=s3_path)

except:
    print("Uploading %s" % s3_path)
    s3.upload_file(local_path, ROOT_S3_DIR, s3_path)
```

## OUTPUT

 upload output

Directory Structure from the Console  Alt text

### [Step 3] Restore from S3

We start off by importing the relevant libraries and initialising s3 as the client

```
import os
import errno
import boto3

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

The below helper function identifies non-existent paths and raises exceptions

```
def assert_dir_exists(path):
    try:
        os.makedirs(path)
    except OSError as e:
        if e.errno != errno.EEXIST:
            raise
```

Our main getter function performs

1. Check for missing '/' sign
2. Gets the list of objects using paginator
3. Stores the relative paths for each file
4. Downloads the file to the right directory maintaining the structure

```
def download_dir(bucket, path, target):
    # Handle missing / at end of prefix
    if not path.endswith('/'):
        path += '/'



    paginator = client.get_paginator('list_objects_v2')
    for result in paginator.paginate(Bucket=bucket, Prefix=path):
        # Download each file individually
        for key in result['Contents']:
            # Calculate relative path
            rel_path = key['Key'][len(path):]
            # Skip paths ending in /
            if not key['Key'].endswith('/'):
                local_file_path = os.path.join(target, rel_path)
                # Make sure directories exist
                local_file_dir = os.path.dirname(local_file_path)
                assert_dir_exists(local_file_dir)
                client.download_file(bucket, key['Key'], local_file_path)
                print("Downloading %s" % local_file_path)
```

We call the function by passing in the destination path, bucket name, directory name


```
destin = '/home/nsword/rootdir'
ROOT_S3_DIR = 'rootdir'
buckt='23810756-cloudstorage'

download_dir(buckt, ROOT_S3_DIR, destin)
```

## OUTPUT

 restore op See below the downloaded folder  folder

[Step 4] Write information about files to DynamoDB

We install DynamoDb on the machine.  dynamo folder


When we run the docker command it

1. runs on port 8000:800 using the **--p** option
2. pulls the image of microsoft/dynamodb

## Table Creation

Using **aws create table** command. Pass in

1. Table Name
2. Attribute - userId
3. Primary Key - userID
4. Provisioning Type - Number of Reads + Writes

 table created

## Program

We start off with imports and setting the clients, table names and paginator to get all buckets(if any)

```
import os
import boto3

buckt = '23810756-cloudstorage'
s3 = boto3.client("s3")
dynamodb = boto3.client("dynamodb")
table_name = "23810756-CloudFiles"
response=s3.get_bucket_acl(Bucket=buckt)
```

We iterate through paginator and then through each file returned

```
paginator = s3.get_paginator('list_objects_v2')
page_iterator = paginator.paginate(Bucket=buckt)
for bucket in page_iterator:
    for file in bucket['Contents']:
```

Now we open **try** block. Here we perform


1. Get metadata for Owner name and Permission
2. Insert Record passing in all values
3. Print response
4. Except block prints error message if erroneous

```
try:
    metadata = s3.head_object(Bucket=buckt, Key=file['Key'])
```

```
insert_rec = dynamodb.put_item(
    TableName=table_name,
    Item={
        "userId": {"S": metadata['ResponseMetadata']['RequestId']},
        "fileName": {"S": os.path.basename(file['Key'])},
        "path": {"S": file['Key']},
        "lastUpdated": {"S": metadata['ResponseMetadata']['HTTPHeaders']
['last-modified']},
        "owner": {"S": response['Owner']['DisplayName']},
        "permissions": {"S": response['Grants'][0]['Permission']}

    })
print(insert_rec)
except:
    print("Failed {}".format(file['Key']))
```

## OUTPUT

data inserted

## Console

data from Console