Accessing Amazon S3 Using boto3

ARUN ARUNISTO

Boto3 is the Amazon Web Services (AWS) Software Development Kit (SDK) for Python, which allows Python developers to write software that makes use of services like **Amazon S3** and **Amazon EC2**. Boto3 provides an easy-to-use, object-oriented API as well as low-level access to AWS services.

Key Features of Boto3

- **Object-oriented Interface:** Boto3 provides resources, which are higher-level abstractions of AWS services. For example, an S3 resource might provide methods to create, list, and delete buckets and objects.
- **Low-level Access:** For more advanced users, Boto3 provides direct access to the AWS service APIs.
- **Sessions:** Boto3 allows for the creation of sessions that maintain AWS credentials, region settings, and other configurations, making it easy to interact with multiple services and regions.

- **Service Resource Abstractions:** These are high-level abstractions designed to make resource management simpler. For instance, interacting with an S3 bucket is as simple as using Python objects and methods.
- **Error Handling:** Boto3 has built-in support for handling errors and retrying requests automatically.

Before installing Boto3, install Python 3.8 or later; support for Python 3.7 and earlier is deprecated. After the deprecation date listed for each Python version, new releases of Boto3 will not include support for that version of Python.

Install the boto3 latest version using pip:

pip install boto3

Accessing Amazon S3

Amazon Simple Storage Service **(Amazon S3)** is an object storage service that offers scalability, data availability, security, and performance.

To use Amazon s3 first you have to configure the aws credentials, for that I created **.env** file and added the credentials like below:

```
AWS_ACCESS_KEY_ID = <access_key_id>
AWS_SECRET_ACCESS_KEY = <secret_key>
```

To use environment variables you have to install the **python-dotenv** package first. You can install the package using below command:

```
pip install python-dotenv
```

You can add **Python-dotenv** to your application to make it load the configuration from a .env file when it is present (e.g. in development) while remaining configurable via the environment:

```
from dotenv import load_dotenv

#this will help to load variables from .env.
load_dotenv()

#Your code goes here.
```

You can call your credentials using **os** module for that you have to import the **os** module on your file like below:

```
import os
```

import the boto3 and botocore.exceptions like below:

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

Then you have to configure the **client**

```
s3_client = boto3.client(
                's3',
                aws_access_key_id=os.environ.get(
                                "AWS_ACCESS_KEY_ID"
                ),
                aws_secret_access_key=os.environ.get(
                                "AWS_SECRET_ACCESS_KEY"
                ),
                region_name=os.environ.get(
                                "AWS_REGION"
               ),
                config=boto3.session.Config(
                                signature_version='s3v4'
            )
    )
```

Creating Amazon S3 Bucket

An **Amazon S3 bucket** is a storage location to hold files. S3 files are referred to as objects.

```
#creating a s3 bucket
def create_bucket(bucket_name):
    try:
        s3.client.create_bucket(
                Bucket=bucket_name
            )
    except ClientError as e:
        print(e)
        return False
    return True
```

And, also you can create bucket for mentioning the specific region. The name of an Amazon S3 bucket must be unique across all regions of the AWS platform. The bucket can be located in a specific region to minimize latency or to address regulatory requirements. For that you just need to add an extra argument like below:

```
#creating a s3 bucket in a specific region
def create_bucket(bucket_name):
   try:
      s3.client.create_bucket(
            Bucket=bucket_name,
            CreateBucketConfiguration={
                  'LocationConstraint': os.environ.get(
                     "AWS_REGION"
                     )
               }
         )
   except ClientError as e:
      print(e)
      return False
   return True
```

Listing the existing buckets

```
#retrieving the list of buckets
response = s3_client.list_buckets()
for bucket in response['Buckets']:
   print(bucket["Name"])
```

Uploading files to S3 buckets

```
#uploading file
def upload_file(filename, bucket, object_name=None):
    if object_name is None:
        object_name = os.path.basename(filename)
    try:
        response = s3_client.upload_file(filename,
                                         bucket,
                                         object_name)
    except ClientError as e:
        print(e)
        return False
    return True
```

Downloading files from s3

```
#downloading a file
def download_file(bucket, object_name, file_name):
    try:
        response = s3_client.download_file(
                        bucket_name,
                        object_name,
                        file_name
                    )
    except ClientError as e:
        print(e)
        return False
    return True
```

Presigned URLs

A user who does not have AWS credentials or permission to access an S3 object can be granted temporary access by using a presigned URL.

A presigned URL is generated by an AWS user who has access to the object.

The generated URL is then given to the unauthorized user. The presigned URL can be entered in a browser or used by a program or HTML webpage. The credentials used by the presigned URL are those of the AWS user who generated the URL.

```
#generating presigned URL
def create_presigned_url(bucket, object_name):
    try:
        response = s3_client.generate_presigned_url(
            'get_object',
            Params={
                "Bucket":bucket_name,
                "Key":object_name,
              }
          )

    except ClientError as e:
        return e
    return response
```

Presigned URL for uploading a file

```
#generating a presigned url to upload a file
def create_presigned_post(bucket, object_name):
    try:
        response = s3_client.generate_presigned_post(
                        bucket_name,
                        object_name
                    )
    except ClientError as e:
        return e
    return response
```

Access Control List

Getting bucket access control list

```
#access control list
result = s3_client.get_bucket_acl(
                Bucket=s3_bucket_name
        )
print(result)
```