AWS Boto3 – S3 Call

Firstly, Boto3 is the AWS SDK (Software Development Kit) for Python.

It allows you to write Python code that interacts with AWS services like:

S3 (Simple Storage Service), EC2 (Elastic Compute Cloud), Lambda, DynamoDB IAM, and many more.

Boto3 has 2 Types of Interfaces:

- Client: Low-level, direct api call and more advanced features.
- Resource: High-level, object-oriented, and for basic operations.
- Waiter: A waiter is a helper who waits for a specific AWS resource to enter a certain state.
- Paginator: A paginator is a tool that automatically handles multiple pages of responses from AWS API calls.

  Many AWS API responses (like listing S3 objects) return limited results per call (e.g., 1000 objects max per page). If you have more, you get them in pages, and must retrieve the next page using a Continuation Token.
  Paginators automate this for you.

Code Explanation:

- Importing the boto3 Loads the AWS SDK so you can interact with AWS services like S3, EC2, etc. Without it, you can't use .resource(), .client(), or AWS operations.
- OS gives access to the operating system (e.g., file creation, path handling, deleting files) Used to delete the test files after processing.

```
import boto3
import os
```

- Try: attempts to create a new bucket. If it succeeds, it prints success and continues.
- except s3_client.exceptions.BucketAlreadyOwnedByYou: This catches the case where the bucket already exists, you are the owner of the bucket, so no action is needed, the script can continue
- except s3_client.exceptions.BucketAlreadyExists: This catches the case where the bucket already exists, but another AWS account owns it, so you can't use this name, and the script exits.

We use try because create_bucket() can fail if the name is already taken, if the region is invalid, or if the user has no permission. Without try, your script would crash when this happens.

```
    s3 = boto3.client('s3', region_name=region)
    try:
        s3.create_bucket(Bucket=bucket_name)
        print(f" Bucket '{bucket_name}' created.")
    except s3.exceptions.BucketAlreadyOwnedByYou:
        print(f" Bucket '{bucket_name}' already exists.")
    except s3.exceptions.BucketAlreadyExists:
        print(f" Bucket name '{bucket_name}' already taken.")
        exit(1)
```

After creating it will wait after bucket is available

```
    waiter = s3_client.get_waiter("bucket_exists")
    print(" Waiting for bucket to become available...")
    waiter.wait(Bucket=bucket_name, WaiterConfig={'Delay': 10, 'MaxAttempts': 10})
    print(" Bucket is now ready.\n")
```

Then it will list the buckets available

```
s3_resource = boto3.resource("s3", region_name=region)
buckets = list(s3_resource.buckets.all())
print(" Available Buckets:")
for b in buckets:
    print(f"- {b.name}")
```

After the buckets are listed, it will create a dummy file and upload it.

```
    with open(dummy_file, "w") as f:
        f.write("This is a test dummy file.")
    bucket.upload_file(dummy_file, dummy_file)
    print(f" File '{dummy_file}' uploaded.")
```

It will then wait till the bucket is exist because

Uploading a file to S3 is eventually consistent sometimes, especially across regions or networks, it takes a few seconds before the file is visible.

Using a waiter ensures that your script waits patiently until the object appears in S3, and it avoids problems like trying to read or list the file before it's fully available.

```
waiter = s3_client.get_waiter("object_exists")
print(" Waiting for uploaded file to be available...")
waiter.wait(Bucket=bucket_name, Key=key, WaiterConfig={'Delay': 10, 'MaxAttempts': 10})
print(" Object is now available.")
```

Then it will list through the Paginator call type

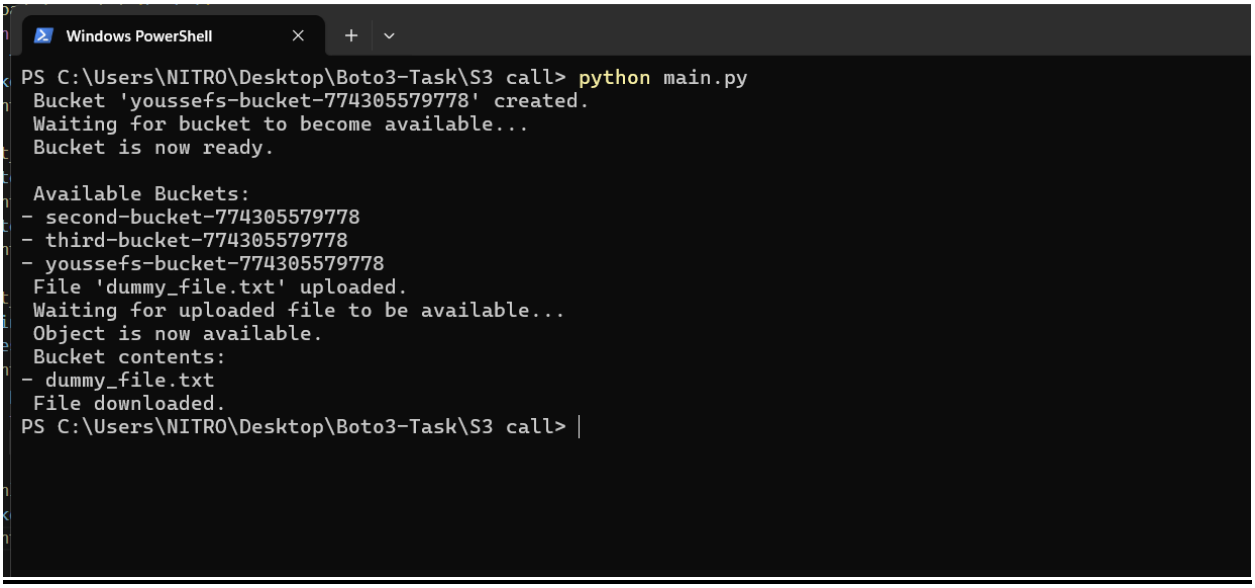
```
paginator = s3_client.get_paginator('list_objects_v2')
pages = paginator.paginate(Bucket=bucket_name)
print(" Bucket contents:")
for page in pages:
    for obj in page.get('Contents', []):
        print(f"- {obj['Key']}")
```

Finally, it will download the file 😊

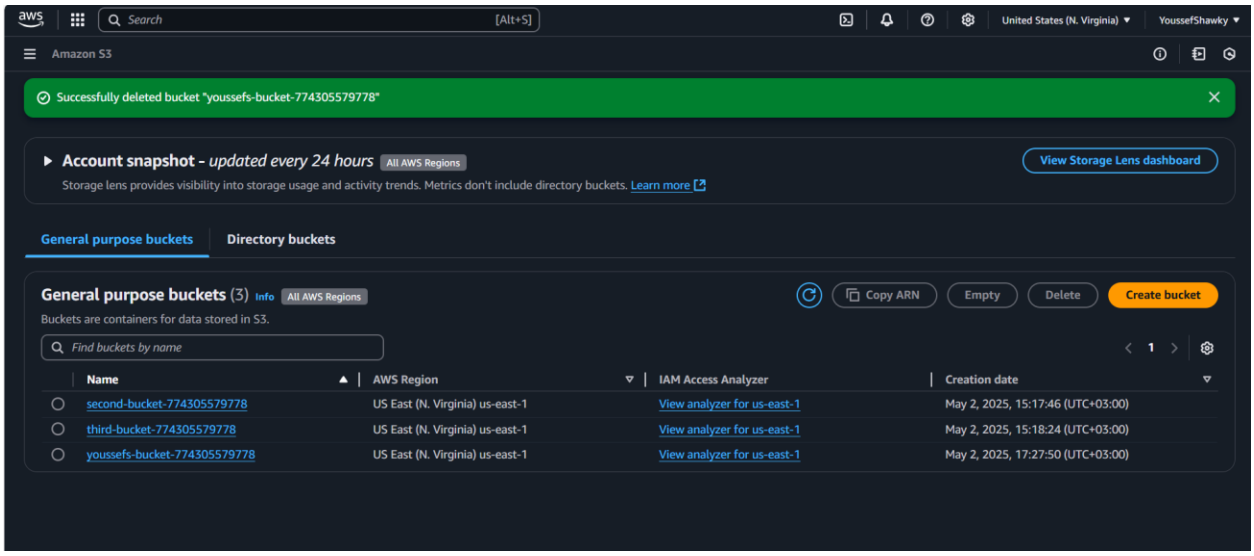
```
bucket.download_file(dummy_file, downloaded_file)
print(" File downloaded.")
```

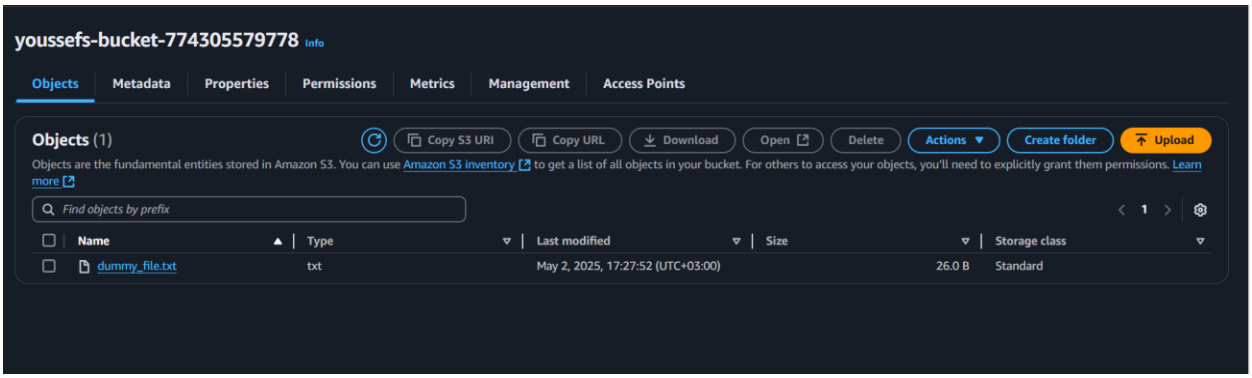
Output:

PowerShell



AWS Console





Bonus task:

So, instead of using a monolithic style, all logic is in one long main.py file. Code is executed top to bottom, no reuse, everything is hardcoded in the same flow

Script with Functions is more reusable, code is separated into named functions, and the Main section just calls those functions. You can import and reuse functions in other scripts.