


Laboratory 2: Programming Cloud Services - Storage Ser- vices

M7024E, Cloud Services - (2023)

Full Name	Student ID
Argaw Mahlet	maharg-3
Omidiora Joseph	Josomi-3

Tutor: Karan Mitra



January 16, 2024

Contents

1	Introduction	i
1.1	Objectives	i
2	Exercises	i

1 | Introduction

In this laboratory, we used the application programming interfaces (APIs) provided by Amazon Web Services (AWS) to program Cloud services that require storage, such as the Amazon S3 block storage service.

1.1 | Objectives

The objective of this lab is to:

- Setup a programming environment for building (using programming tools and languages) Cloud services for a major Cloud provider, for example, Amazon Web Services (AWS);
- Develop cloud services for file storage, listing, and retrieval using the APIs provided by AWS.

2 | Exercises

- Identify ways of creating Amazon S3 service clients.

To create Amazon S3 service clients, we use

- SDKs provided by AWS. These SDKs are available for various programming languages, which include Python, .NET, Java, PHP, Kotlin and more.
- AWS Command Line Interface (CLI): The command line can be used to intricate with Amazon S3. In its background, the CLI uses the SDK provided by AWS.
- Management Console: S3 services can also be created graphically using the management console.

- Create a Python program to create a bucket in three regions of your choice.

a. Explain in detail the steps involved and explain the output.

The Python script created uses the Boto3 library to interact with Amazon Simple Storage Service (S3) in AWS. It is designed to create S3 buckets in different AWS regions. The following breaks down the code step by step:

- import libraries

```
1  #Language: Python 3
2  #import libraries
3  ∨ import boto3
4  import os
```

Figure 2.1: Libraries Import

This code imports the required libraries: boto3 for AWS SDK and os for accessing environment variables.

- Prompt the user for AWS access key and secret key

```
6  # Prompt the user for AWS access key and secret key
7  aws_access_key_id = os.environ['aws_access_key_id']
8  aws_secret_access_key = os.environ['aws_secret_access_key']
9
```

Figure 2.2: AWS Access Key

The script prompts the user for their AWS access key ID and secret access key. It assumes that these credentials are available as environment variables.

- AWS region configuration.

```
9
10 # Specify your AWS region
11 aws_region = ['eu-west-1']
12
```

Figure 2.3: AWS Region Configuration

The script specifies the AWS regions in which S3 buckets will be created. In this case, it only contains the 'eu-west-1' region.

- AWS Session Initialization using the provided credentials

```
13 # Initialize the AWS client using the provided credentials
14 session = boto3.Session(
15     aws_access_key_id=aws_access_key_id,
16     aws_secret_access_key=aws_secret_access_key,
17     region_name=aws_region
18 )
19
```

Figure 2.4: AWS Session Initialization

It initializes an AWS session using the provided access key ID, secret access key, and region.

- Specify your S3 bucket name

```
20 # Specify your S3 bucket name
21 bucket_prefix = 'mj-accesslog' # this is to make all buckets created universally unique
22
```

Figure 2.5: Set Bucket Prefix

This is a prefix used for naming the S3 buckets. It is added to create universally unique bucket names.

- Bucket Creation Loop

```
for region in aws_region:
```

Figure 2.6: AWS Region Loop

It iterates over each AWS region specified earlier.

- S3 Client Initialization:

```
5     s3 = boto3.client(aws_secret_access_key: str, aws_access_key_id=aws_access_key_id,
6                       aws_secret_access_key=aws_secret_access_key)
```

Figure 2.7: S3 Client Initialization

It initializes an S3 client for the current region using the AWS credentials.

- Bucket Naming

```
27     bucket_name = f"{bucket_prefix}-{region}"
28
```

Figure 2.8: Set Bucket Naming

The script constructs a unique bucket name by combining the bucket prefix and the current region.

□ Bucket Creation

```

30     try:
31         if region == 'eu-north-1':
32             s3.create_bucket(Bucket=bucket_name)
33         else:
34             s3.create_bucket(
35                 Bucket=bucket_name,
36                 CreateBucketConfiguration={'LocationConstraint': region}
37             )
38         print(f"S3 bucket created successfully: {bucket_name}")
39     except Exception as e:
40         print(f"Error creating S3 bucket: {e}")

```

Figure 2.9: Bucket Creation

It attempts to create an S3 bucket in the specified region. If the region is 'eu-north-1', it creates the bucket without specifying a location constraint. Otherwise, it includes the location constraint based on the region. The success or failure message is printed accordingly.

□ Output Explanation:

If the S3 bucket creation is successful, it prints a message like: S3 bucket created successfully: mj-accesslog-eu-west-1. If there is an error during the bucket creation, it prints an error message.

- Create a Python program that lists your buckets in the region of your choice.

```

1  # language: Python 3.8
2  """
3  ~/Documents/GENIAL/LTU/Cloud Services/
4  Cloud Services lab/3 -Programming Compute
5  Services/listing.py - Untracked
6  """
7  # Prompt the user for AWS access key, secret key and region
8  aws_access_key_id = os.environ['aws_access_key_id']
9  aws_secret_access_key = os.environ['aws_secret_access_key']
10 aws_account_id = os.environ['aws_account_id']
11
12 # Initialize the AWS client using the provided credentials
13 session = boto3.Session(
14     aws_access_key_id=aws_access_key_id,
15     aws_secret_access_key=aws_secret_access_key,
16 )
17
18 # Specify your AWS region
19 region = 'eu-west-1' # Change this to your desired region
20
21 # Create an S3 client
22 s3 = boto3.client('s3', region_name=region, aws_access_key_id=aws_access_key_id,
23                 aws_secret_access_key=aws_secret_access_key)
24
25 # List S3 buckets
26 try:
27     response = s3.list_buckets()
28     buckets = response['Buckets']
29
30     print("List of S3 buckets in " + region + ":")
31     for bucket in buckets:
32         res = s3.get_bucket_location(
33             Bucket=bucket['Name'],
34             ExpectedBucketOwner="aws_account_id"
35         )
36
37         if res['LocationConstraint'] == region:
38             print(bucket["Name"])
39
40 except Exception as e:
41     print(f"Error listing S3 buckets: {e}")

```

Figure 2.10: Python Script for listing S3 Buckets

The code file is attached to the submission

- Create a Python program to upload objects in your newly created bucket.

```

1  # language: Python 3.8
2  #!/usr/bin/env python
3  # ~/Documents/GENIAL/LTU/Cloud Services/
4  # Cloud Services lab/3 -Programming Compute
5  # Services/listing.py - Untracked
6  # Import the boto3 module, secret key and region
7  aws_access_key_id = os.environ['aws_access_key_id']
8  aws_secret_access_key = os.environ['aws_secret_access_key']
9  aws_account_id = os.environ['aws_account_id']
10
11 # Initialize the AWS client using the provided credentials
12 session = boto3.Session(
13     aws_access_key_id=aws_access_key_id,
14     aws_secret_access_key=aws_secret_access_key,
15 )
16
17 # Specify your AWS region
18 region = 'eu-west-1' # Change this to your desired region
19
20 # Create an S3 client
21 s3 = boto3.client('s3', region_name=region, aws_access_key_id=aws_access_key_id,
22                 aws_secret_access_key=aws_secret_access_key)
23
24 # List S3 buckets
25 try:
26     response = s3.list_buckets()
27     buckets = response['Buckets']
28
29     print("List of S3 buckets in " + region + ":")
30     for bucket in buckets:
31         res = s3.get_bucket_location(
32             Bucket=bucket['Name'],
33             ExpectedBucketOwner=aws_account_id
34         )
35
36         if res['LocationConstraint'] == region:
37             print(bucket['Name'])
38
39 except Exception as e:
40     print(f"Error listing S3 buckets: {e}")
41

```

Figure 2.11: Python Script for Uploading objects to S3

The code file is attached to the submission

- Create a Python program to delete particular objects from your newly created bucket.

```

1  #Language: Python 3
2  import boto3
3  import os
4
5  # Prompt the user for AWS access key and secret key
6  aws_access_key_id = os.environ['aws_access_key_id']
7  aws_secret_access_key = os.environ['aws_secret_access_key']
8  aws_region = input("Enter your AWS region (default: eu-north-1): ")
9
10 # Initialize the AWS client using the provided credentials
11 session = boto3.Session(
12     aws_access_key_id=aws_access_key_id,
13     aws_secret_access_key=aws_secret_access_key,
14     region_name=aws_region
15 )
16
17 # Specify your S3 bucket name
18 bucket_name = input("Enter your S3 bucket name: ")
19
20 # Specify the key (object name) under which the file will be stored in the bucket
21 s3_key_to_delete = input("Specify the key (object name) to delete: ")
22
23 # Create an S3 client
24 s3 = boto3.client('s3', region_name=aws_region, aws_access_key_id=aws_access_key_id,
25                 aws_secret_access_key=aws_secret_access_key)
26
27 # Delete the object from the S3 bucket
28 try:
29     s3.delete_object(Bucket=bucket_name, Key=s3_key_to_delete)
30     print(f"Object deleted successfully from S3 bucket: {bucket_name}/{s3_key_to_delete}")
31 except Exception as e:
32     print(f"Error deleting object from S3 bucket: {e}")
33

```

Figure 2.12: Python Script for deleting S3 Buckets

The code file is attached to the submission

2.0.1 | Latency Measurement

- Create a Python program to upload and download objects (sizes 1MB, 10 MB, 100 MB and 500MB) from the three regions used in the above exercise. Measure the object upload and download latency from these regions. Plot and explain the results in your report. Think about statistical analysis and any assumptions made.

```
import boto3
import time
import matplotlib.pyplot as plt
import os

# AWS credentials
# Prompt the user for AWS access key and secret key
aws_access_key_id = os.environ['aws_access_key_id']
aws_secret_access_key = os.environ['aws_secret_access_key']

# S3 regions and bucket names
regions = ['us-east-1', 'us-west-2', 'eu-west-1']
bucket_names = ['buc-mj-us-east-1', 'buc-mj-us-west-2', 'buc-mj-eu-west-1']

# Object sizes in MB
object_sizes = [1, 10, 100, 500]

def upload_object(s3, bucket, key, data):
    start_time = time.time()
    s3.put_object(Bucket=bucket, Key=key, Body=data)
    end_time = time.time()
    return end_time - start_time

def download_object(s3, bucket, key):
    start_time = time.time()
    s3.get_object(Bucket=bucket, Key=key)
    end_time = time.time()
    return end_time - start_time

def main():
    latencies = {'Upload': {size: [] for size in object_sizes},
                'Download': {size: [] for size in object_sizes}}

    for region, bucket_name in zip(regions, bucket_names):
        s3 = boto3.client('s3', region_name=region, aws_access_key_id=aws_access_key_id,
                          aws_secret_access_key=aws_secret_access_key)

        for size in object_sizes:
            data = b'0' * (size * 1024 * 1024) # Creating data of specified size in MB

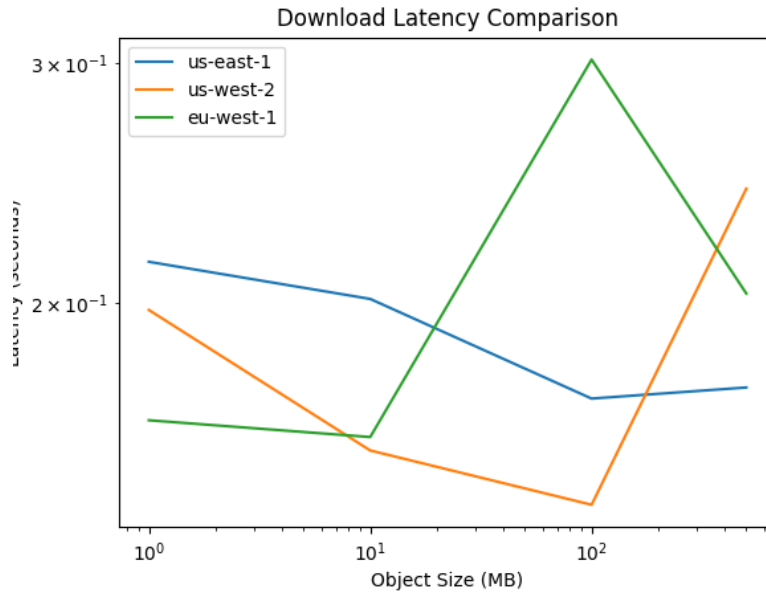
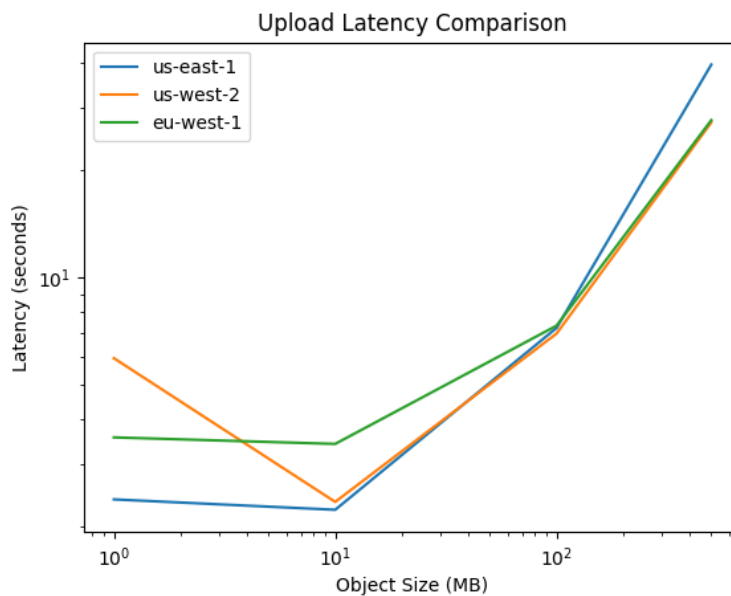
            upload_latency = upload_object(s3, bucket_name, f'{size}MB_object', data)
            download_latency = download_object(s3, bucket_name, f'{size}MB_object')

            latencies['Upload'][size].append(upload_latency)
            latencies['Download'][size].append(download_latency)

    # Plotting results
    for operation in ['Upload', 'Download']:
        for size in object_sizes:
            plt.plot(regions, latencies[operation][size], label=f'{size}MB')
```

Figure 2.13: Python Script for measuring upload and download latency

The output of the script above is presented and explained below.

**Figure 2.14:** download latency**Figure 2.15:** Upload latency

The graph, depicted in Figure 2.14, illustrates the download latency trends across three distinct AWS regions: us-east-1, us-west-2, and eu-west-1, as a function of varying file sizes. Notably, in us-east-1, a consistent and gradual decline in latency is observed with increasing file sizes, indicative of a scalable and efficient data retrieval system. In contrast, us-west-2 exhibits a similar descending latency trend until the 500MB file size, where an unexpected spike in latency occurs, prompting further investigation. The eu-west-1 region displays an intricate latency pattern, with a marginal reduction in latency for a 1MB object compared to a 10MB one, followed by a rise for the 100MB file and a subsequent latency decrease for the 500MB file.

The graphical representation, as illustrated in Figure 2.15, delineates the latency dynamics during upload sessions across three AWS regions: us-east-1, us-west-2, and eu-west-1. A uniform and gradual increase in latency is observed across all regions, indicating a general trend of increase

in upload time irrespective of the AWS region. However, a notable departure from this pattern is observed in us-west-2, where a decrease in latency is evident, specifically at the 10MB file size. Moreover, the performance differentials between us-east-1 and us-west-2 are accentuated at smaller object sizes, where us-east-1 demonstrates superior latency management. Intriguingly, as object sizes expand, the latency roles are reversed, with us-east-1 exhibiting higher latencies at significantly larger file sizes.

The observed variations in both upload and download latencies can be attributed to a myriad of factors, encompassing geographical distance, client network latency, and region-specific optimizations or infrastructure enhancements. Notably, the performance discrepancies witnessed in regions such as us-east-1 and us-west-2 are indicative of nuanced dynamics. For instance, us-east-1, being one of the earliest AWS regions after us-west-1, exhibits superior performance despite geographical distances. This superiority can be attributed to strategic optimizations and enhancements implemented within the region. These intricate regional variabilities underscore the critical importance of considering both file size and geographical location when optimizing upload and download latencies in the complex landscape of diverse cloud computing environments. It is worth noting that, in this lab, it is assumed that the test environment has minimum network interference and is relatively stable.