# Task 3

## Implementation

| AWS | Azure |
|---|---|
| - S3 API extremely easy to use<br>- Short, easy to understand commands (E.g. s3.create_bucket())<br>- DynamoDB very easy to use<br>- noSQL difficult learning curve<br>- Simple commands such as table.Scan()<br>- Filtering is done intuitively with dynamodb conditions (Key, Attr, And)<br>- Sorting is done AFTER in python | - More complete and layered blob API<br>- BlobServiceClient<br>- ContainerClient<br>- BlobClient<br>- Easy commands within clients (E.g. blob_service_client.create_container)<br>- Cosmos DB quick start guide is very useful<br>- Lots of resources for SQL queries<br>- Filtering and sorting is done in the SQL query, which makes it more advantageous than dynamoDB |

## Performance

| AWS | Azure |
|---|---|
| Slower than Azure for task 1<br>- 0.61s user<br>- 0.25s system<br>Faster than Azure for task 2<br>- 24.75s for building and populating db in task 2 | Faster than AWS for task 1<br>- 0.43s user<br>- 0.24s system<br>Slower than AWS for task 2<br>- 28.4s for building and populating db in task 2 |

## Permissions

| AWS | Azure |
|---|---|
| - Must include a credentials file in your '~/.aws/credentials'<br>- Aws requires unique names for buckets<br>- Default access is **<u>public</u>** when creating containers through code | - Must create environment variable and use it in the code<br>- 'AZURE_STORAGE_CONNECTION_STRING'<br>- Azure seems to have less unique name requirements<br>- Default access is **<u>private</u>** when creating containers through code |

**Available documentation**

| AWS | Azure |
|---|---|
| - Quickstart guide<br>- Boto3 API docs<br>    - S3 docs<br>    - DynamoDB docs<br>- Extensive online forums (E.g. Stackoverflow) | - Quickstart Guide<br>- Azure Storage docs<br>    - Blob API docs<br>    - Table API docs<br>- Less online forums than AWS |

**Tutorials**

| AWS | Azure |
|---|---|
| - Clear step by step instructions in quick-start guide<br>- Python specific example code for S3 in boto3 docs | - Clear step by step instructions in quick-start guide<br>- Python specific example code for blob storage API in docs |

**References**

- “Amazon Dynamodb API Reference,” *Amazon AWS Docs*, 15-Jan-2020. [Online]. Available: https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/Welcome.html. [Accessed: 27-Jan-2020].
- “Get Started With S3,” *Amazon AWS Docs*, 12-Jan-2020. [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/gsg/GetStartedWithS3.html. [Accessed: 27-Jan-2020].
- “How can I convert JSON to CSV?,” *Stack Overflow*, 01-Dec-1959. [Online]. Available: https://stackoverflow.com/questions/1871524/how-can-i-convert-json-to-csv. [Accessed: 27-Jan-2020].
- Normesta, “Azure Blob storage documentation,” *Microsoft Docs*. [Online]. Available: https://docs.microsoft.com/en-us/azure/storage/blobs/. [Accessed: 27-Jan-2020].

- “python sort list of json by value,” *Stack Overflow*, 01-Nov-1964. [Online]. Available: https://stackoverflow.com/questions/26924812/python-sort-list-of-json-by-value. [Accessed: 27-Jan-2020].

## Task 4

**Task 1 Implementation -** Below there is a brief step by step guide to the implementation of the task which involved creating buckets/containers and objects/blobs on AWS and Azure and performing basic API commands against them

| Step 1: Setting up the environment | |
|---|---|
| AWS | Azure |
| - Create buckets directly in S3 using API | - Must create storage account before being able to add containers using the interface<br>- Storage account names must be unique |
| - Must include aws credentials in order to use api. Done by adding credentials in '~/.aws/credentials' file<br>- Find the credentials in labs.vocareum -> Account details -> show<br>- Credentials expire every 3 hours | - Uses a connection that is stored as an environment variable on your machine to authenticate. It is called 'AZURE_STORAGE_CONNECTION_STRING' in my program<br>- Find connection string in 'access keys' section of storage account<br>- Set by running 'export AZURE_STORAGE_CONNECTION_STRING=""' in terminal<br>- Connection does not expire<br>- connect_str = os.getenv('AZURE_STORAGE_CONNECTION_STRING') |
| - Can include region in '~/.aws/config' | - Defaults region to East US for you |

| Step 2: Creating the containers/buckets | |
|---|---|
| AWS | Azure |
| - Create buckets using s3 API 's3.create_bucket(Bucket=name)'<br>- Container names must be unique | - Create the equivalent of buckets, which are containers, by using blob Api's 'blob_service_client.create_container(name)'<br>- Container names do not need to be unique |

| Step 3: Creating the objects/blobs | |
|---|---|
| AWS | Azure |
| - Create objects using ‘s3.Bucket(name).put_object(Key, Body)’ in s3 API<br>- Object names must be unique inside containers<br>- Object names do not need to be unique from container to container | - Create blobs using ‘blob_client.upload_blob(file)’<br>- Blob names must be unique inside containers<br>- Blob names do not need to be unique from container to container |

| Step 4: Display buckets/containers | |
|---|---|
| AWS | Azure |
| - Get all buckets in aws using API s3.buckets.all() | - Get all containers in Azure using blob API ‘blob_service_client.list_containers’ |

| Step 5: Display objects/blobs | |
|---|---|
| AWS | Azure |
| - All buckets returned from previous step have an objects attribute<br>- Access using “bucket.objects.all()” | - Must create a containerClient using the connection string and then get a list of blobs from that<br>- ContainerClient.from_connection_string(connect_str, container_name=container['name'])<br>- Access using ‘container_client.list_blobs()’ |

| Step 6: Download objects/blobs | |
|---|---|
| AWS | Azure |
| - s3.Bucket(bucketName).download_fileobj(objName, filePathToDownloadTo) | - Create blob_client<br>- file.write(blob_client.download_blob().readall()) |

**Task 2 Implementation** - Below there is a brief step by step guide to the implementation of the task which involved building a movie database in AWS and Azure and then querying that database

| Step 1: Building the database | |
|---|---|
| AWS | Azure |
| - Create the database using boto3 resource 'dynamodb'<br><br>import boto3<br><br>dynamodb = boto3.resource('dynamodb', region_name='us-east-1') | - Currently incomplete. Will finish by Friday, Jan 31st<br><br>from azure.cosmos import CosmosClient, PartitionKey<br><br>client = CosmosClient(endpoint, key)<br><br>database = client.create_database_if_not_exists |
| - Must include aws credentials in order to use api. Done by adding credentials in '~/.aws/credentials' file<br>- Find the credentials in labs.vocareum -> Account details -> show<br>- Credentials expire every 3 hours, which is very annoying | - Set the environment variables for endpoint and key<br>- These are found in the database ui under keys -> URI for endpoint and keys -> Primary key for key<br><br>export AZURE_COSMOSDB_URI_STRING="..."<br><br>export AZURE_COSMOSDB_PRIMARY_KEY_STRING="..." |
| - Can include region in '~/.aws/config' | - Region is set default for you |

| Step 2: Create table | |
|---|---|
| AWS | Azure |
| dynamodb.create_table(<br>TableName,<br>KeySchema, AttributeDefinitions,<br>ProvisionedThroughput) | table = database.create_container_if_not_exists(<br>id=TableName,<br>partition_key=PartitionKey(path="/year"),<br>offer_throughput=400) |

| Step 3: Load data into table | |
|---|---|
| AWS | Azure |
| - Load in json file using<br><br>movies = json.load(json_file, parse_float = decimal.Decimal)<br><br>- Add data to table using<br><br>table = dynamodb.Table('Movies')<br>table.put_item(Item) | - Load in json file using<br><br>movies = json.load(json_file)<br><br>- Add data to table using<br><br>table.create_item(body={id, year, title... })<br><br>*Must include a unique id in the body* |

| Step 4: Get query from user | |
|---|---|
| AWS | Azure |
| - Get filters for primary key, secondary key or three info columns (rank, rating or running_time_secs)<br>- Get column to sort by and fields to display<br>- Use input() as well as many while loops and try except statements to error check input | - Get filters for primary key, secondary key or three info columns (rank, rating or running_time_secs)<br>- Get column to sort by and fields to display<br>- Use input() as well as many while loops and try except statements to error check input |

| Step 5: Query database | |
|---|---|
| AWS | Azure |
| - Use Scan function<br><br>table.scan(FilterExpression=fe)<br><br>- FilterExpression is a combination of searches on primary key, secondary key, or the other filters mentioned in the previous step<br>- Example of fe being constructed: | - Sorting and querying done in the same step<br>- SQL query<br><br>items = list(table.query_items(query=query, enable_cross_partition_query=True)) |

| | |
|---|---|
| fe=Key('year').eq(‘1992’)<br>fe=And(fe, Key('title').eq(‘Movie title’)) | |

| Step 6: Sort result | |
|---|---|
| AWS | Azure |
| - Use sorted function<br><br>Ex. sortedResponse = sorted(response['Items'], key=lambda k: k.get(‘year’, 0), reverse=True) | - Done in step 5 for azure |

| Step 7: Display fields/print to csv | |
|---|---|
| AWS | Azure |
| - Split fields included by comma<br>- Create csv writer using:<br><br>csvwriter = csv.writer(csvFile, delimiter=",")<br><br>- Write a row to the csv file using:<br><br>csvwriter.writerow(ArrayOfItemsInRow) | - Split fields included by comma<br>- Create csv writer using:<br><br>csvwriter = csv.writer(csvFile, delimiter=",")<br><br>- Write a row to the csv file using:<br><br>csvwriter.writerow(ArrayOfItemsInRow) |