Mohammadamin Sheikhtaheri
0930853

Jan 27, 2020

Cloud Computing Assignment 1 - Task 4

Connection to Storage Services:

For AWS:

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

AWS simply calls for the storage resource using one function, which returns the resource handle to be used in later tasks.

For Azure:

```
blob_service_client = BlobServiceClient.from_connection_string(connect_str)
```

Azure uses the *from_connection_string* function provided by BlobServiceClient to connect to a storage account that has been previously created.

Unmappable Aspects:

The method of which Azure connects to the cloud services is different in the way that it requires a connection string that identifies a specific storage account, whereas AWS requires these configurations to be done beforehand by providing the access keys to a file named “credentials” within a folder named “.aws”

Creating & Populating Storage Containers:

For AWS:

```
data = open('4010Lecture1.pdf.pdf', 'rb')
s3.create_bucket(Bucket=’cis4010’)
s3.Bucket(‘cis4010’).put_object(Key='4010Lecture1.pdf', Body=data)
```

AWS uses the functions *create_bucket* and *put_object* to create and populate a container, neither function returns anything necessary.

For Azure:

```
data = open('4010Lecture1.pdf.pdf', 'rb')
containerClient = blob_service_client.create_container(‘cis4010’)
blobClient = blob_service_client.get_blob_client(container=’cis4010’,
                                                  blob='4010Lecture1.pdf')
```

```
blobClient.upload_blob(data)
```

Azure uses the functions *create_container*, *get_blob_client*, to create a container with an EMPTY blob, and *upload_blob* to upload data to that blob. *Create_container* and *get_blob_client* both return the client handle to be stored in a variable.

Unmappable Aspects:

Azure uses client handle variables, which it uses in order to communicate with its cloud services such as container and blob creation, whereas AWS simply calls functions to perform the task, returning nothing afterwards.

Accessing Containers & Listing Objects:

For AWS:

```
for bucket in s3.buckets.all():
  for bucket_object in bucket.objects.all():
      print(bucket_object.key)
```

AWS uses *s3.buckets.all()* in order to get access to all buckets that are available, and the function *bucket.objects.all()* gives access to all of the objects present within that specific bucket/container.

For Azure:

```
containerList = blob_service_client.list_containers()
  for container in containerList:
      containerClient = blob_service_client.get_container_client(container)
      blobList = containerClient.list_blobs()
      for blob in blobList:
        print(blob.name)
```

Azure used a bit more complex system where the containers could be retrieved using the *list_containers()* function, and then the container client could be retrieved using *get_container_client(container),* and finally the blobs could be listed using *list_blobs().*

Unmappable Aspects:

For Azure, instead of simply doing something similar to AWS such as *container.list_blobs()*, the container itself does not have access to these functions, instead the container’s CLIENT must be retrieved by using *get_container_client(container),* which then can be used to list the blobs. AWS is much simpler where every bucket has access to all of the bucket functions such as *bucket.objects.all()*.

Downloading Objects:

For AWS:

```
s3.meta.client.download_file(bucket.name, objectName, fileLocation)
```

AWS simply uses the *download_file()* function along with the name of the container, name of the object, and the file location in order to download an object from the cloud.

For Azure:

```
with open(fileLocation, "wb") as download_file:
    download_file.write(blobClient.download_blob().readall())
```

Azure opens the download file for writing, and writes to it with the contents provided by the function *download_blob().readall()*.

Unmappable Aspects:

The key difference between AWS and Azure here is that the function provided by Azure does not create the download file manually, it simply writes the contents of the specified blob to the file created by the developer. AWS does all of this automatically when calling *download_file().*

Creating & Populating a Database:

For AWS:

```
dynamoDB = boto3.resource('dynamodb', region_name='us-east-1')
movieTable = dynamoDB.create_table(tableName, KeySchema, AttributeDefinitions,
                                                        ProvisionedThroughput)
movieTable.put_item(Item)
```

The AWS DynamoDB resource must be retrieved from boto3 in order to create any table. The *create_table()* function is used to create tables. This function requires the name of the table, a list of keys and their key types (year, title -> partition, sort), the attribute types of the keys (year = number, title = string), and the desired throughput of the table. Once the table is created, the *put_item()* function can be used to insert an object into the table. The object must include the partition and sort keys, and can include any extra information (ex rating).

For Azure:

```
table_service = TableService(connection_string=connection_str)
table_service.create_table(tableName)
table_service.insert_entity(tableName, Entity)
```

Azure first establishes the connection with the azure cosmosDB account using a connection string, similar to the storage account. Then the table can be created using the *create_table()* function which accepts a table name. New objects can be added to this table using the *insert_entity()* function provided by the table service. This function accepts the name of the table, and an object which contains the required keys and values (Entity.Partitionkey = year). Any extra information can be added to the object by adding a new variable to the object (ex. Entity.rating = 8.9)

Unmappable Aspects:

When creating a table in Azure, only the table name is required, whereas in AWS, the *create_table()* function requires much more information. The AWS SDK allows for the throughput to be changed, which is not an option when creating the table using the Azure SDK.

Querying a Database:

For AWS:

```
movies = movieTable.scan(FilterExpression = expression)
```

The *scan()* function uses an expression to search through the table, and returns a list of all results that matched the expression. An example of an expression is Key('year').eq(1997), which would search and return all objects with the year being equal to 1997.

For Azure:

```
movies = table_service.query_entities(table_name, filter = query_string)
```

The *query_entities()* function uses a table name and a query string to return a list of all results that matched the query string. An example of a query string is "rating gt 8", which would return all objects within the table with a rating above 8.

Unmappable Aspects:

When using the *scan()* function in AWS, there is a max result list that can be returned which equals roughly 2200. This means that queries with results longer than this amount must be scanned multiple times, using the previous key to pick up where the previous scan left off.

References:

[1] Amazon AWS, "DynamoDB Getting Started Python," https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GettingStarted.Python.04.html [Accessed 24 Jan. 2020].
[2] Microsoft, "Table Storage How to Use Python," https://docs.microsoft.com/en-us/azure/cosmos-db/table-storage-how-to-use-python [Accessed 25 Jan. 2020]