**Build a Table API App with Python and Azure Cosmos DB**

Problem: At your company, Globomantics, your team needs to maintain a low-latency, high-throughput application that enables writes all over the world. Within the request flow, there is a need to maintain a key-value store for the state of incoming requests that offers an SLA of uptime and latency. You have chosen Azure CosmosDB Table API for such requirements, as it excels in providing enterprise capabilities and latency with a simple-to-use API. You now need to create the Python microservice that will create tables, set entities, and query entities using the Azure CosmosDB Table API.

**Authenticate Against CosmosDB Table API in Python**

1. Click on the **Open Azure portal** link and use the provided credentials to log in.
2. Open the **Azure Cloud Shell** button, which is located immediately to the right of the search bar at the top of the page.figure﻿
3. Click on **Bash**.
4. At the **You have no storage mounted** panel, click the **Show advanced settings** link.
5. For **Cloud Shell region**, choose **East US**.
6. For **Resource group**, choose **pluralsight-resource-group**.
7. For **Storage account**, leave Create new selected, and enter a unique name.

A screenshot of a computer

Description automatically generated

Note: If you don't choose a unique name, or if you don't use a name three to 24 characters long made of solely lowercase letters and numbers, an error message will be displayed later when you click Create storage. To help avoid this, use tableapilabsa<random-numbers>, where you choose the random-numbers suffix. For example, tableapilabsa1341998

1. For **File share** leave Create new selected, and enter fileshare.
2. Click **Create storage,** then wait for a minute or two until the Bash command prompt appears.

Note: If you didn't choose a proper or unique Storage account name, you will get an error message, prompting you to choose a different name. See the error messages, and previous Note, for guidance.

1. Once you have access to the Bash command prompt, execute the following commands in the Cloud Shell:

pip install azure-cosmosdb-table

mkdir app

touch app/main.py

cd app

This will install the azure-cosmosdb-table Python SDK and create a Python file to create the code.

1. Use the search bar at the top of the Azure portal to navigate to the **Azure Cosmos DB** page, then click the **Name** link of the CosmosDB resource listed.
2. From the navigation panel on the left, select **Quick start**, click the **Python** tab,and then copy the **CONNECTION STRING.**
3. In the Cloud Shell, execute the following command after replacing the <placeholder> text with the connection string:

export COSMOSDB\_CONNECTION\_STRING="<placeholder>"

A screenshot of a computer

Description automatically generated

1. Next, execute the command code main.py.

A screenshot of a computer

Description automatically generated

This will bring up a new Visual Studio Code window to write our main script. You may rearrange the sliders to make the Cloud Shell fit your screen if necessary.

1. Inside the Visual Studio Code window, paste the following snippet and then save the file:

from azure.cosmosdb.table.tableservice import TableService

from azure.cosmosdb.table.models import Entity

from azure.cosmosdb.table.tablebatch import TableBatch

import os

connection\_string = os.environ['COSMOSDB\_CONNECTION\_STRING']

table\_service = TableService(connection\_string=connection\_string)

print('Connected to CosmosDB')

**Note:** To save a file in Visual Studio Code, you can press **Cmd/Ctrl + s.**

This fetches the connection string from the environment variable you exported and uses the **TableService** module to connect to CosmosDB

1. In the Cloud Shell run python main.pyA screenshot of a computer

   Description automatically generated

**Add an Entity to Azure CosmosDB Table Api with Python**

We now have a Python program that can authenticate against Azure CosmosDB Table API, it is time to use it to generate some entities.

1. Add this line of code to the last empty line of the main.py file:

table\_service.create\_table('tasktable')

This will create a table **tasktable.** Recall that a *Table* in CosmosDB Table API is the analogue of a *Container* in the SQL API and holds *Entities* that are accessed as a key-value store.

1. Below the previously-added code, paste the following snippet:

task = Entity()

task.PartitionKey = 'tasksSeattle'

task.RowKey = '002'

task.description = 'Wash the car'

task.priority = 100

table\_service.insert\_entity('tasktable', task)

This creates an **Entity** object that can set any field in the **Table** as properties; in the end, use the insert\_entity method to insert it.

**Note:**

* + You must specify both a **PartitionKey** and a **RowKey** property for every entity. These are the unique identifiers of your entities, as together they form the primary key of an entity. You can query using these values much faster than you can query any other entity properties because only these properties are indexed.
  + The Table service uses **PartitionKey** to intelligently distribute table entities across storage nodes. Entities that have the same **PartitionKey** are stored on the same node. **RowKey** is the unique ID of the entity within the partition it belongs to.

1. You may also insert **Entities** from dictionaries and the **Entity** object will get auto-generated for you. Next, continue the main.py file with the following:

task = {'PartitionKey': 'tasksSeattle', 'RowKey': '001',

'description': 'Take out the trash', 'priority': 200}

table\_service.insert\_entity('tasktable', task)

1. To update the last *Entity* in the table, add the following:

task = {'PartitionKey': 'tasksSeattle', 'RowKey': '001',

'description': 'Take out the trash', 'priority': 150}

table\_service.insert\_or\_replace\_entity('tasktable', task)

**Note:** The insert\_or\_replace\_entity method will search the entity inserted with the combination **PartitionKey + RowKey**. If it is not found then it will insert a new element, but if it is found it will modify it in a **non-atomic** manner.

1. Finally, one can also insert elements in **Batch** **atomically** as a transaction. In the main.py file, add the following:

**Note:** Every **Batch** operation, as is **atomic**, need to happen within the same **PartitionKey**

1. Save the file, then from the Cloud Shell, run python main.py*.*
2. Back in the Azure portal, select **Data Explorer** from the left-hand menu, then click **TablesDB > tasktable > Entites**. You should see the following 4 Entities:

A screenshot of a computer

Description automatically generated

**Query for a Group of Entities in Azure CosmosDB Table API with Python**

1. In the file main.py, comment out the lines of code referring to *insertions*. That means everything *below* the following line:

print('Connected to CosmosDB')

**Note:** To comment a block of code in Visual Studio Code, you can highlight the block and type **Cmd/Ctrl + /**

1. Below the commented code, add the following:

tasks = table\_service.query\_entities('tasktable', filter="PartitionKey eq 'tasksSeattle'", select='description')

for task in tasks:

print(f'Task Description found: {task.description}')

This will return the descriptions of all the **Entities** in the **tasksSeattle** **Partition**.

1. Save the file and then execute it with python main.py.

You should see the following output:

A screen shot of a computer

Description automatically generated

1. To filter the results by **PartitionKey** and **RowKey,** add the following to the end of main.py:

print('Filtering by RowKey')

tasks\_filtered = table\_service.query\_entities('tasktable', filter="PartitionKey eq 'tasksSeattle' and RowKey ge '003'", select='description')

for task\_new in tasks\_filtered:

print(f'Task Description found: {task\_new.description}')

A screenshot of a computer

Description automatically generated

1. Finally, to delete an **Entity**, add the below line to the file:

table\_service.delete\_entity('tasktable', 'tasksSeattle', '001')

**Note:** The **delete\_entity** method needs the **Table**, **PartitionKey**, and **RowKey** to find the **Entity** to delete.

A screenshot of a computer

Description automatically generated