A close up of text on a white background

Description automatically generatedA close up of text on a white background

Description automatically generatedA close up of text on a white background

Description automatically generatedA close up of text on a white background

Description automatically generatedGeneral purpose V2 Account

Azure table storage

Storage Account🡪 Table🡪 Entity 🡪 Properties

Partition Key: …

Row Key: … System Entities

Timestamp: …

Name: ….

Email: ….

(up to 252 custom properties)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

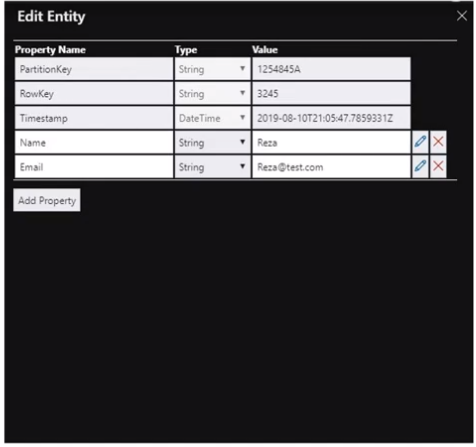
Reza’s

Users

Posts

Table Storage Concepts

* URL Format: <http://reza.table.core.windows.net>
* Account: Access possible through storage account
* Table: Collections of entities, no schema is enforced
* Entity: A set of properties, like a database row
* Properties: Name-value pairs. E.g. below



Properties:

* Name-value pair
* 1 entity can include max 252 properties to store data
* 3 System Properties: Partition key, row key, timestamp
* Can query entities with same partition key more quickly and insert/update them in atomic operations.

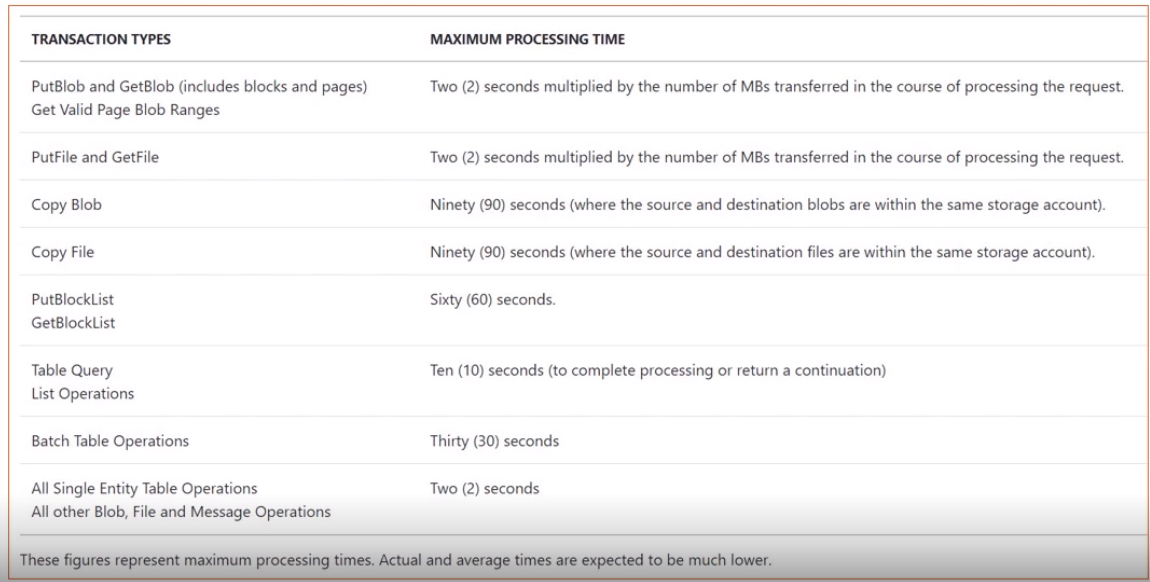
Table Storage System Properties

1. Partition Key: Developer is responsible for inserting and updating its value.
2. RowKey: Developer is responsible for inserting and updating its value.
3. Timestamp: The server manages its value & it cannot be manually modified.

Limitations

* Max size of entity in azure storage= 1MB

SLA for storage accounts



* For better SLA

Azure Cosmos DB table API (premium offering for azure table storage)

**Blob, Queue and File Storage**

Blob Storage: Azure Blob storage is optimized to store massive amounts of unstructured data in cloud.  
 Can automatically scale if demand increases

* Serving documents directly to a browser (storing file into blob and accessing them directly over browser🡪 each document has unique URL which can be used by clients with right key or token to the file)
* Storing data for backup and restore (backup till 10 years, azure sql db store backup in azure blob storage)
* Audio and video files can be uploaded.
* Storing data for analysis (e.g. storing huge amount of raw data in blob and configure azure stream analytics)
* Storing log files & files for distributed access.

Azure Queue Storage: It is a service for storing large numbers of messages. Access messages via authenticated calls using HTTP/HTTPS.

Use Case:

* Creating a backlog of work to process asynchronously.

e.g. Passing msgs from a web role to a worker role🡪 convert bitmap image into jpeg. Users can upload their file and they can get jpeg version of their image in their email (Time consuming process and might not be able to convert 1000 of files synchronously)   
Solution: To each bitmap request add msg to azure queue storage and you have your convertor worker node reading these msgs and do the conversion.

Azure Files: It offers fully managed file shares in the cloud that are accessible via the industry standard Server Message Block (SMB) protocol.

Use Cases:

* Replace on-premises file servers
* Lift and shift scenarios (lift from on premise and shift to cloud)

**Securing Storage Accounts**

Azure Storage Security

1. Storage Account Management Security

* These are operations that affect the storage account itself (create/delete storage account)
* RBAC roles for storage (owner, contributor, reader, etc.)
* Assigning apt. RBAC role to users, group or applications

1. Storage Account Data access security

* Blocked by default
* Storage account keys grants complete access to services within storage accounts
* Shared Access Signatures (SAS) tokens (Recommended): It gives permissions required for a limited amount of time

1. Encryption for data in transit

* Always use HTTPS (SAS token include option to enforce) when calling REST APIs or accessing objects in storage

1. Encryption for data at rest

* Referred to data stored in storage account on azure server. Two methods:
* Storage Service Encryption (SSE): Enabled for all storage accounts and cannot be disabled
* Client-side Encryption: Programmatically encrypt the data in a client application then send it across the wire.

1. Audit/monitor access

* Azure storage analytics to log authentication method used by clients.

1. CORS for browser clients

* When a web browser running in one domain makes an HTTP request for a different domain, this is called cross-origin HTTP request
* When calling Azure Table Storage API that return JSON data to be processed by the JavaScript client
* You can allow origins, methods and headers

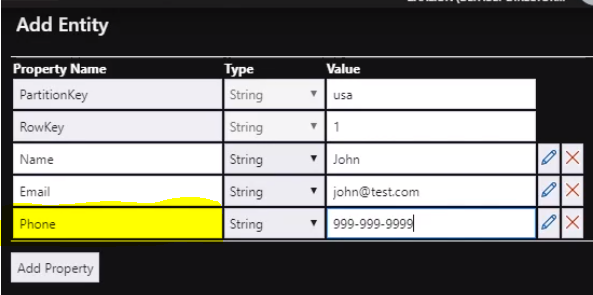
**Demo 1: Provisioning Table Storage in the Azure Portal and configuring Security**

1. **Create Storage account**
2. Choose subscription
3. Resource Group (option to create new)
4. Enter storage account name (only lower case characters & numbers)
5. Enter location.
6. Performance (Standard (data stored in magnetic drive)/Premium (data stored in solid state drive))
7. Account kind (Storage v2)
8. Choose Replication (LRS: Cheapest)
9. Choose access tier (Cool/Hot)
10. Review+Create
11. Now any storage type can be chosen out of (file,table,storage,blob)
12. **Security Setting** are done at storage account level which will be common for all storage type
13. Access Keys: 2 keys by default and can be regenerated if too old.
14. Shared access signature: allowed services (file, blob, queue, table), allowed resource types (service, container, object), allowed permissions (read, write, delete, list, add, create, update, process), start and expiry date/time, allowed IP address, allowed protocol (HTTP/HTTPS), signing key (key 1, key2, key3).
15. Generated SAS token, separate token for chosen service.
16. **Encryption:** Enabled by default, also option to use your own key (enter key url/ select from key vault)
17. **Firewall & Virtual networks:** By default all the services within all Virtual network can access my storage account/ Selected networks.
18. **CORS:**  Cross Origin request: specify cors for blob, table, queue, table separately

**Demo 2: Working with Azure Storage Explorer (Desktop/Online Version)**

1. Click on storage explorer and choose blob, file, queue or tables.
2. Enter table name.
3. Click on + to add entity
4. Add system properties (partition key=’canada’ , row key=’1’ and timestamp will be handled by azure)
5. Add custom properties (Name= ‘reza’, Email=’reza@123.com’)
6. Click on insert and new record will be created in table.
7. Click on add. Enter values similarly and add new property i.e. phoneno.
8. All entities should not share same schema. Further interaction can be done from storage explorer.

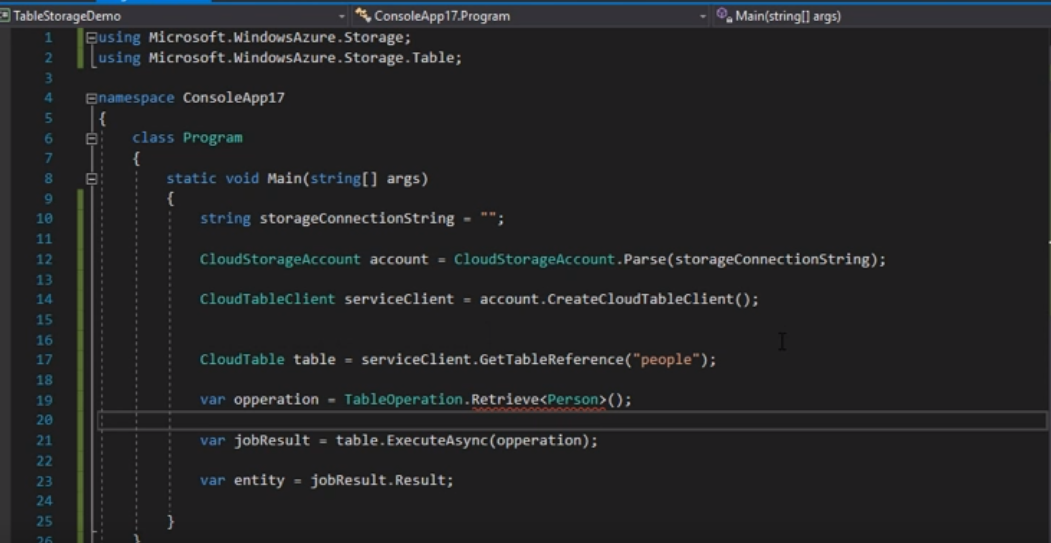




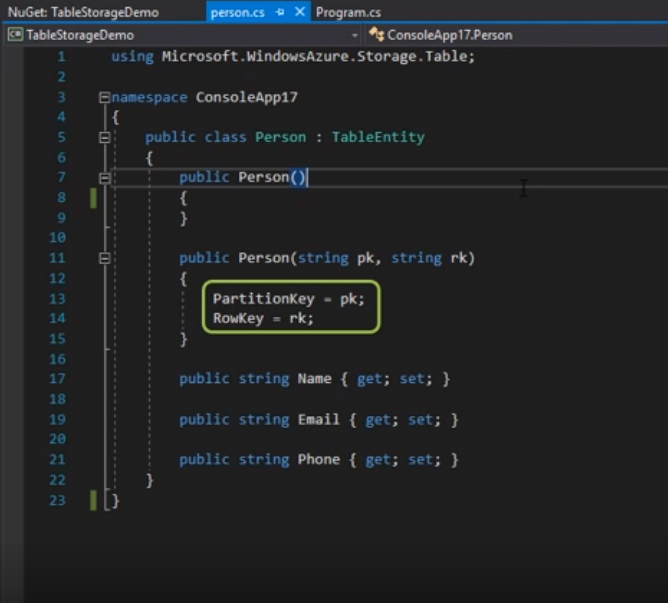
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| PARTITIONKEY | ROWKEY | TIMESTAMP | NAME | EMAIL | Phone |
| canada | 1 | ‘’ | reza | rez@123.com | null |
| Usa | 1 | ‘’ | John | john@test.com | 9999999999 |

**Demo 3: Working with Azure Table Storage (.net sdk)**

1. Create a small .net application



1. Click on project and then click on manage NuGet Packages and choose Windows Azure Storage and start writing code.
2. Extract connection string from access keys/ Shared access signature and paste it into code. In real it should not be hard coded and saved in application configuration.
3. Create a cloud storage account using this connection string.
4. Create service client. (Options: CreateBlobClient, CreateCloudClient, CreateCloudQueueClient, CreateCloudTableClient etc.) Here CreateCloudTableClient
5. Reference to table you are working on. i.e. person
6. Run Cloud Operation (create TableOperation object of type retrieve)
7. Retrieve function takes 2 args (PARTITIONKEY, ROWKEY) here🡪 (“canada”,”1”)
8. Pass retrieve table operation to table execute method.
9. Ctrl + Shift + B (Build)
10. Check Person class (f12 on person)



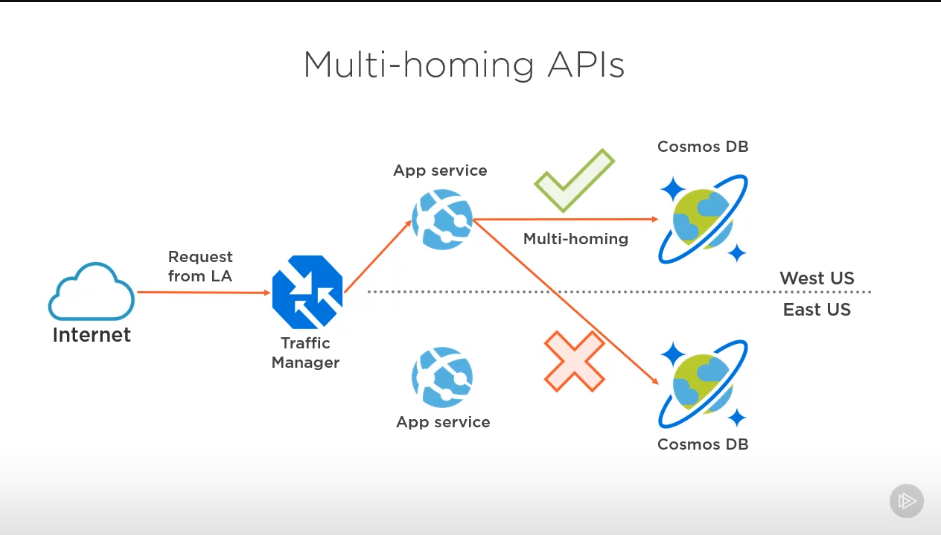
1. Run the application (object successfully retrieved from database)

**Module 4: Azure Cosmos DB**

Cosmos DB Concepts:

1. Global distribution and multi-homing

* Cosmos DB allows you to add/remove any of the azure regions to your cosmos account at any time with a click of a button.
* Choosing required regions depend on global reach of your application and where your users are located
* It replicates your data to all the regions associated with your cosmos account
* Multi Homing: Allows application to be highly available.
* Application is aware of the nearest region & request can be sent to that region
* Nearest region is identified without any configuration changes.
* Connection String remains same on adding/removing new regions.



Here: Two instance of app deployed in east & west us respectively.  
Traffic manager directs the request to the app service closest to that region.  
How does Traffic manager knows? When connection string is same for east & west us  
Job of Muti Homing API: It is going to figure out which instance of cosmos db is closest to my app service & redirects the data request to that instance.

1. Data Consistency Levels (Strong, Bounded staleness, session, consistent prefix, eventual)

* Cosmos DB gives the option to adjust the consistency level of database.
* Data consistency Vs data availability & latency
* Two levels: Strong and eventual
* Multiple Writes and Consistent prefix
* Strong: Always see latest version of data even if there is a delay (few sec/milisec)
* Eventual: No guarantee of latest record but returned quickly

Weakest consistency level which guarantees latest version of data

Might see out-of-order updates.

* Consistent Prefix: Guarantees that reads never see out-of-order writes.

Always see in-order-updates (If seeing update ‘n’ then will see all updates before n) but it might not be the latest version.

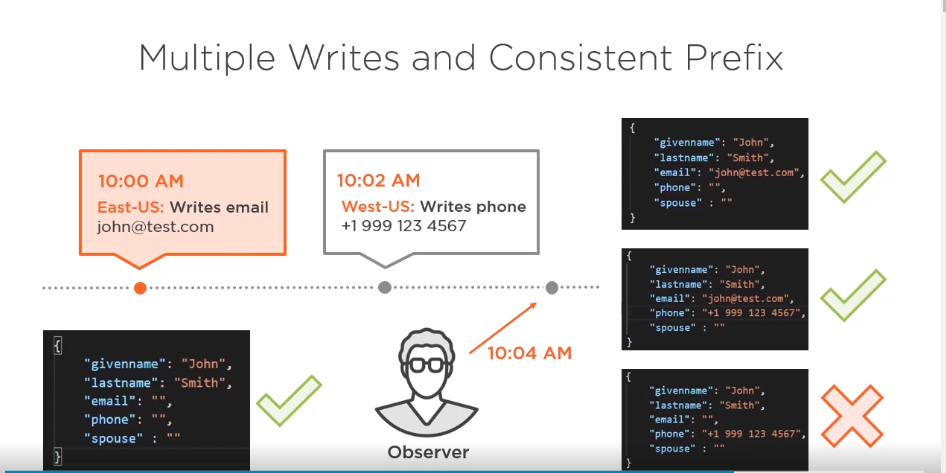
* Bounded Staleness: Specify window of staleness. Window is defined in time and no. of operations. (E.g. window of staleness for 10 mins and 1000 operations). So anyone accessing cosmos db outside this window will observe stronger consistency.  
  Accessing db in this window🡪 consistency level of consistent prefix(in-order updates)
* Session: Default consistency level when provisioning new instance of cosmos db

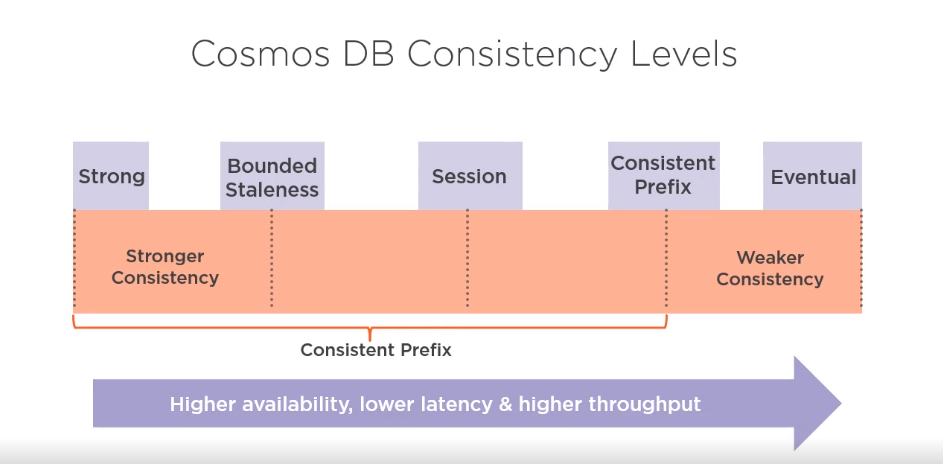
Two consistency levels and deal with session.

If accessing cosmos db in same session that writes data🡪 Strong Consistency

Outside session: like consistent prefix (in-order-updates)

Question: Which level to use :   
Accepted ans: Only see 1st update, see both updates  
Not Accepted ans: Only see 2nd update





1. Time-to-live (TTL)

* Set expiry time on cosmos db data items 🡪setting is called TTL and is set as seconds
* Cosmos DB automatically removes the items after this time period, since the last modified time

1. Data Partitioning

* Databases & Containers

Azure cosmos db is a unit of mgmt. for a set of containers

A db consists of a set of schema-agnostic containers

* Logical partition consists of set of items that have same partition key
* Data added to container is automatically partitioned across a set of logical partitions.
* Logical partitions are added to physical partitions that are distributed among several machines
* Throughput provisioned for a container is divided evenly among physical partitions.
* Batch update within a logical partition by using a transaction is possible.

**Data Partitioning best practices**

* Pick partition key that doesn’t result in hot spots within application
* Choose partition key that has wide range of values, so that data is evenly spread across logical partitions (e.g. choose city name if equal distribution)
* For partition keys, use properties that appear frequently as a filter in queries; including partition key in the filter predicate improves query performance
* Single logical partition has a limit of 20 GB of storage (earlier 10 GB)

Azure Cosmos DB APIs

1. SQL API: recommended for new applications
2. MongoDB API: Easy migration of existing apps
3. Table API: Easy migration from table storage
4. Cassandra API: Easy migration of existing apps
5. Gremlin API: Both new and existing apps using graph data

Security

1. RBAC: To grant permissions, groups or applications
2. Firewall: To limit clients who can access cosmos db
3. Deploy Cosmos DB into a VNet & use NSGs(to control inbound and outbound traffic)
4. CORS: client side app directly calling cosmos db, you can whitelist those apps in azure cosmos db so that all those requests can be made from browser
5. Read-only and read-write keys
6. Cosmos db now supports private endpoints to privately connect to a service or resource. 🡪 traffic doesn’t goes through public internet

**Demo 1: Provisioning new Cosmos DB instance**

1. Click on Azure comos DB and create new account.
2. Select Subscription and resource group (create new if not there).
3. Enter Account name and choose core (sql) api
4. Choose location and account type (Production, Non-Production: Use cosmos db at much cheaper price)
5. Click on networking tab and choose Networking Security: Connectivity method.
6. Click on Encryption (on by default)
7. Click on tags. Tags used for simplicity of management.
8. Click on review + create.
9. Azure Cosmos DB recommendation: Auotomatic recommendation about cosmos db account for sdk upgrades, suggestions about partitions and collections, query page size, incorrect sdk usagers, Lazy Indexing, Transient errors etc. (MS doc). Dashboard page🡪 Notification🡪 Recommendations
10. Add container. Create new database or choose existing.
11. Enter throughput, enter container ID and partition key.
12. Click on OK.
13. Click on scale and click on autoscale.
14. Create new item.
15. TTL: Click on settings. Enter TTL time and click save. If document not updated in TTL then it’ll be auto deleted.

**Demo 2: Configuring Cosmos DB Security**

1. Click on keys in security. Two types: read-write, read only. Two keys in each type to share with clients.
2. Click on Firewall and virtual networks🡪 All networks/Selected Networks (More details :up)
3. Click on CORS. Option to whitelist limited no. of domains. Javascript applications deployed to these domains can directly call azure cosmos db api and consume data at client side

**Module 5: Working with Azure Cosmos DB – SQL (Core) API**

Cosmos Db Container  
- Unit of scalability for throughput and storage  
- Container items & throughput are distributed across a set of logical partitions (created based on partition keys)  
- Azure Cosmos container can scale elastically  
  
**Implementing Azure Databricks Environment in MS Azure**