DP900 Final Notes – part 3

Describe considerations for working with non-relational data on Azure (15—20%)

Describe capabilities of Azure storage

• Describe Azure Blob storage

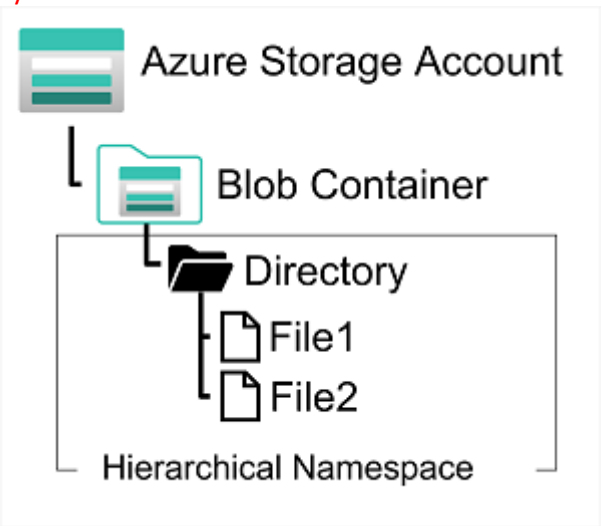
Block Blob
Page Blob
Append Blob
DataLake Gen 2 is Block Blob

Azure Blob Storage is Microsoft's object storage solution for the cloud. Blob Storage is optimized for storing massive amounts of unstructured data. Unstructured data is data that doesn't adhere to a particular data model or definition, such as text or binary data.

Blob Storage is designed for:
Serving images or documents directly to a browser.
Storing files for distributed access.
Streaming video and audio.
Writing to log files.
Storing data for backup and restore, disaster recovery, and archiving.
Storing data for analysis by an on-premises or Azure-hosted service.
Users or client applications can access objects in Blob Storage via HTTP/HTTPS, from anywhere in the world. Objects in Blob Storage are accessible via the Azure Storage REST API, Azure PowerShell, Azure CLI, or an Azure Storage client library. Client libraries are available for different languages, including: .NET, Java, Node.js, Python, Go, PHP, Ruby
Clients can also securely connect to Blob Storage by using SSH File Transfer Protocol (SFTP) and mount Blob Storage containers by using the Network File System (NFS) 3.0 protocol.

Azure Data Lake Storage Gen2

Azure Data Lake Storage Gen**2** is a newer version of this service that is integrated into Azure Storage; enabling you to take advantage of the scalability of blob storage and the cost-control of storage tiers, combined with the hierarchical file system capabilities and compatibility with major analytics systems of Azure Data Lake Store.



Systems like Hadoop in Azure HDInsight, Azure Databricks, and Azure Synapse Analytics can mount a distributed file system hosted in Azure Data Lake Store Gen2 and use it to process huge volumes of data.
To create an Azure Data Lake Store Gen2 files system, you must enable the **Hierarchical Namespace** option of an Azure Storage account.

Blob Storage supports Azure Data Lake Storage Gen2, Microsoft's enterprise big data analytics solution for the cloud. Azure Data Lake Storage Gen2 offers a hierarchical file system as well as the advantages of Blob Storage, including:
Low-cost, tiered storage
High availability
Strong consistency
Disaster recovery capabilities

Azure Blob Storage supports three different types of blob:
Block blobs. A block blob is handled as a set of blocks. Each block can vary in size, up to 100 MB. A block blob can contain up to 50,000 blocks, giving a maximum size of over 4.7 TB. The block is the smallest amount of data that can be read or written as an individual unit. Block blobs are best used to store discrete, large, binary objects that change infrequently.
Page blobs. A page blob is organized as a collection of fixed size 512-byte pages. A page blob is optimized to support random read and write operations; you can fetch and store data for a single page if necessary. A page blob can hold up to 8 TB of data. Azure uses page blobs to implement virtual disk storage for virtual machines.
Append blobs. An append blob is a block blob optimized to support append operations. You can only add blocks to the end of an append blob; updating or deleting existing blocks isn't supported. Each block can vary in size, up to 4 MB. The maximum size of an append blob is just over 195 GB.

• Describe Azure File storage
Stores data like a drive on an on premise server. SMB, NFS
Azure Files supports two common network file sharing protocols:

1. *Server Message Block* (SMB) file sharing is commonly used across multiple operating systems (Windows, Linux, macOS).
2. *Network File System* (NFS) shares are used by some Linux and macOS versions. To create an NFS share, you must use a premium tier storage account and create and configure a virtual network through which access to the share can be controlled.

Azure file sync option

Take advantage of fully managed file shares in the cloud that are accessible via the industry-standard SMB and NFS protocols. Azure Files shares can be mounted concurrently by cloud or on-premises deployments of Windows, Linux, and macOS. Azure Files shares can also be cached on Windows Servers with Azure File Sync for fast access near where the data is being used.
Migrate your Windows file server easily
Lift and shift your file shares to the cloud without breaking existing links non-disruptively:
Migrate existing servers with no downtime.
Use Storage Migration Service and Azure File Sync to make your migration a breeze.
Build Hybrid cloud file shares
Mount Azure Files shares from anywhere—from on premises to the cloud—giving you a truly hybrid experience.

Whether you mount your Azure Files share directly or use Azure File Sync to cache on premises, Azure Files is the natural choice for cloud file shares.

Persistent shared file storage for containers

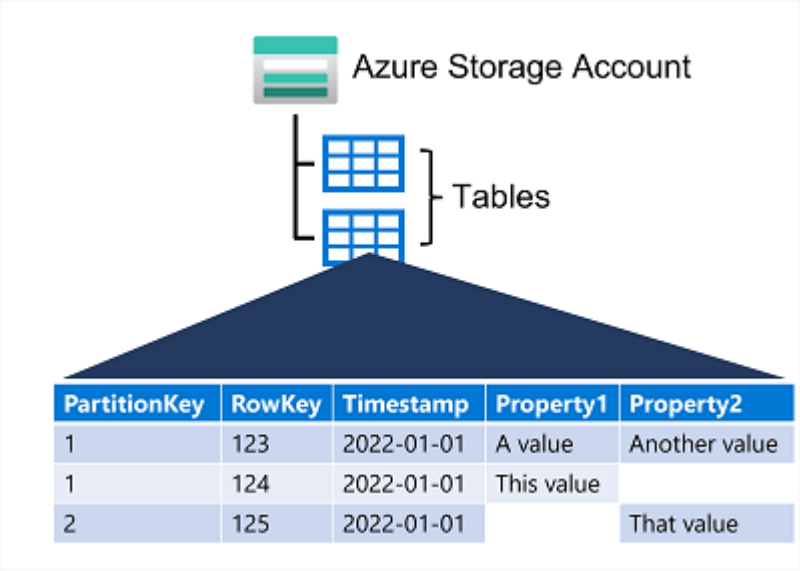
Easily share data between containers using NFS or SMB file shares. Azure Files is tightly integrated with Azure Kubernetes Service (AKS) for easy cloud file storage and management of your data.

Fully managed file shares in the cloud, accessible via the SMB and NFS protocols.

Azure Files shares can be mounted concurrently by cloud or on-premises deployments of Windows, macOS, and Linux. Enable file sharing between applications running in your virtual machines using familiar Windows APIs or the Azure Files REST API. Additionally, Azure File Sync allows caching and synchronization of Azure Files shares on Windows Servers for local access.

• Describe Azure Table storage

Azure Table Storage is a NoSQL storage solution that makes use of tables containing *key/value* data items. Each item is represented by a row that contains columns for the data fields that need to be stored.



PartitionKey	RowKey	Timestamp	Property1	Property2
1	123	2022-01-01	A value	Another value
1	124	2022-01-01	This value	
2	125	2022-01-01		That value

However, don't be misled into thinking that an Azure Table Storage table is like a table in a relational database. An Azure Table enables you to store semi-structured data. All rows in a table must have a unique key (composed of a partition key and a row key), and when you modify data in a table, a *timestamp* column records the date and time the modification was made; but other than that, the columns in each row can vary. Azure Table Storage tables have no concept of foreign keys, relationships, stored procedures, views, or other objects you might find in a relational database.

Data in Azure Table storage is usually denormalized, with each row holding the entire data for a logical entity. For example, a table holding customer information might store the first name, last name, one or more telephone numbers, and one or more addresses for each customer. The number of fields in each row can be different, depending on the number of telephone numbers and addresses for each customer, and the details recorded for each address. In a relational database, this information would be split across multiple rows in several tables.

To help ensure fast access, Azure Table Storage splits a table into partitions. Partitioning is a mechanism for grouping related rows, based on a common property or partition key. Rows that share the same partition key will be stored together. Partitioning not only helps to organize data, it can also improve scalability and performance in the following ways:

Partitions are independent from each other, and can grow or shrink as rows are added to, or removed from, a partition. A table can contain any number of partitions.

When you search for data, you can include the partition key in the search criteria. This helps to narrow down the volume of data to be examined, and improves performance by reducing the amount of I/O (input and output operations, or *reads* and *writes*) needed to locate the data.
The key in an Azure Table Storage table comprises two elements; the partition key that identifies the partition containing the row, and a row key that is unique to each row in the same partition. Items in the same partition are stored in row key order. If an application adds a new row to a table, Azure ensures that the row is placed in the correct position in the table. This scheme enables an application to quickly perform *point* queries that identify a single row, and *range* queries that fetch a contiguous block of rows in a partition.

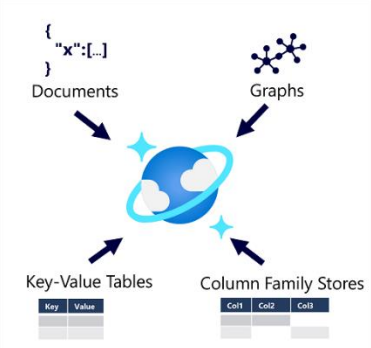
Azure Table storage is a service that stores non-relational structured data (also known as structured NoSQL data) in the cloud, providing a key/attribute store with a schemaless design. Because Table storage is schemaless, it's easy to adapt your data as the needs of your application evolve. Access to Table storage data is fast and cost-effective for many types of applications, and is typically lower in cost than traditional SQL for similar volumes of data.

You can use Table storage to store flexible datasets like user data for web applications, address books, device information, or other types of metadata your service requires. You can store any number of entities in a table, and a storage account may contain any number of tables, up to the capacity limit of the storage account.

Key value type of storage, but extended to allow for more fields to be written. Must have Partition key, Row key and Timestamp. Maximun 255 fields (including above 3) and max 1MB size (2MB for Cassandra API).

Describe capabilities and features of Azure Cosmos DB

• Identify use cases for Azure Cosmos DB



APPLIES TO: NoSQL MongoDB Cassandra Gremlin Table
This article provides an overview of several common use cases for Azure Cosmos DB. The recommendations in this article serve as a starting point as you develop your application with Azure Cosmos DB.

Introduction
Azure Cosmos DB is the Azure solution for a fast NoSQL database, with open APIs for any scale. The service is designed to allow customers to elastically (and independently) scale throughput and storage across any number of geographical regions. Azure Cosmos DB is the first globally distributed database service in the market today to offer comprehensive service level agreements encompassing throughput, latency, availability, and consistency.

Azure Cosmos DB is a global distributed, multi-model database that is used in a wide range of applications and use cases. It is a good choice for any serverless application that needs low order-of-millisecond response times, and needs to scale rapidly and globally. It supports multiple data models (key-value, documents, graphs and columnar) and many Azure Cosmos DB APIs for data access including API for MongoDB, API for NoSQL, API for Gremlin, and API for Table natively, and in an extensible manner.

The following are some attributes of Azure Cosmos DB that make it well-suited for high-performance applications with global ambition.

Azure Cosmos DB natively partitions your data for high availability and scalability. Azure Cosmos DB offers 99.99% guarantees for availability, throughput, low latency, and consistency on all single-region accounts and all multi-region accounts with relaxed consistency, and 99.999% read availability on all multi-region database accounts.

Azure Cosmos DB has SSD backed storage with low-latency order-of-millisecond response times.

Azure Cosmos DB's support for consistency levels like eventual, consistent prefix, session, and bounded-staleness allows for full flexibility and low cost-to-performance ratio. No database service offers as much flexibility as Azure Cosmos DB in levels consistency.

Azure Cosmos DB has a flexible data-friendly pricing model that meters storage and throughput independently.

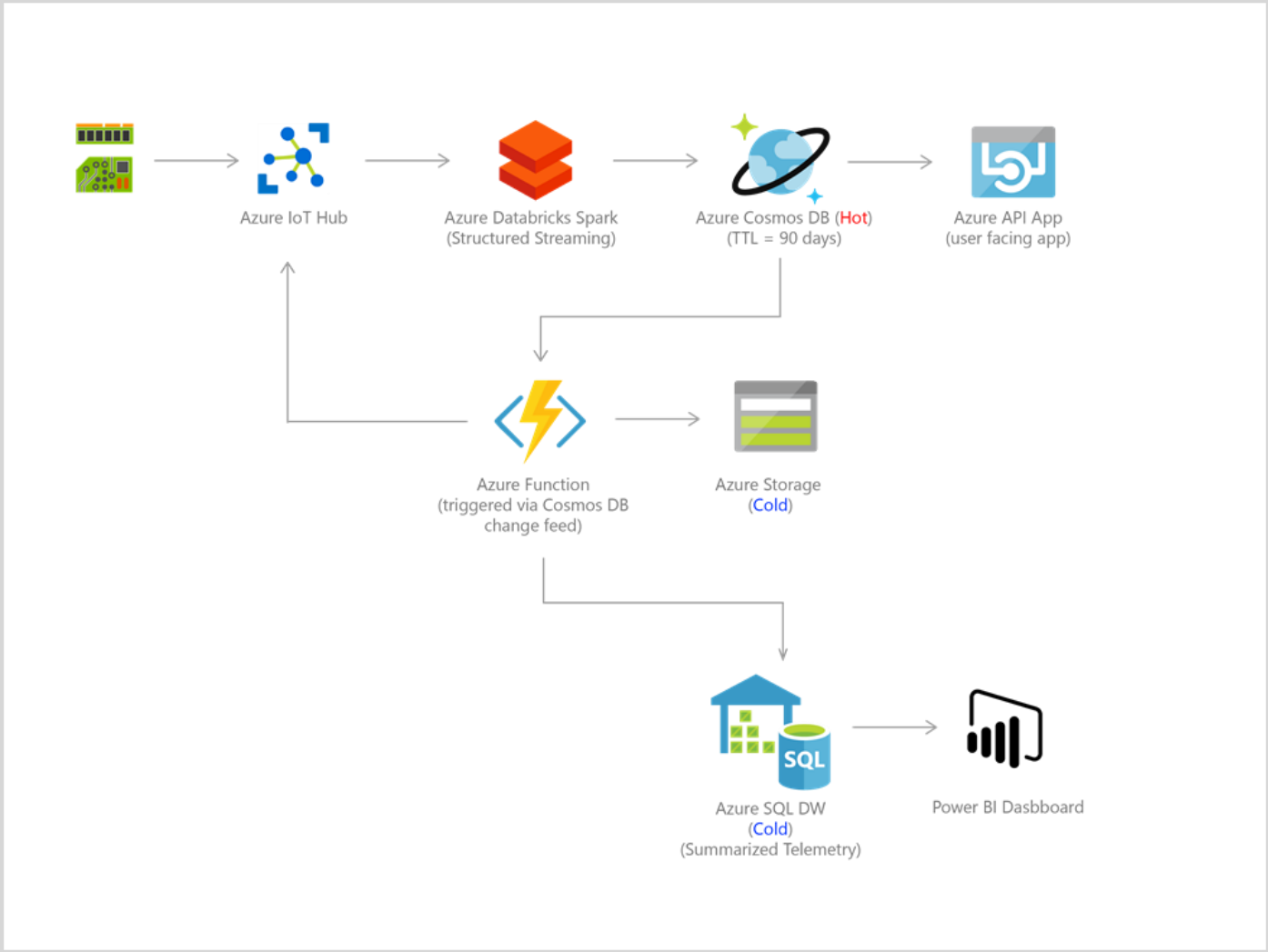
Azure Cosmos DB's reserved throughput model allows you to think in terms of number of reads/writes instead of CPU/memory/IOPs of the underlying hardware.

Azure Cosmos DB's design lets you scale to massive request volumes in the order of trillions of requests per day.

These attributes are beneficial in web, mobile, gaming, and IoT applications that need low response times and need to handle massive amounts of reads and writes.

IoT and telematics

IoT use cases commonly share some patterns in how they ingest, process, and store data. First, these systems need to ingest bursts of data from device sensors of various locales. Next, these systems process and analyze streaming data to derive real-time insights. The data is then archived to cold storage for batch analytics. Microsoft Azure offers rich services that can be applied for IoT use cases including Azure Cosmos DB, Azure Event Hubs, Azure Stream Analytics, Azure Notification Hub, Azure Machine Learning, Azure HDInsight, and Power BI.



Bursts of data can be ingested by Azure Event Hubs as it offers high throughput data ingestion with low latency. Data ingested that needs to be processed for real-time insight can be funneled to Azure Stream Analytics for real-time analytics. Data can be loaded into Azure Cosmos DB for adhoc querying. Once the data is loaded into Azure Cosmos DB, the data is ready to be queried. In addition, new data and changes to existing data can be read on change feed. Change feed is a persistent, append only log that stores changes to Azure Cosmos DB containers in sequential order. Then all data or just changes to data in Azure Cosmos DB can be used as reference data as part of real-time analytics. In addition, data can further be refined and processed by connecting Azure Cosmos DB data to HDInsight for Pig, Hive, or Map/Reduce jobs. Refined data is then loaded back to Azure Cosmos DB for reporting.

For a sample IoT solution using Azure Cosmos DB, Event Hubs and Apache Storm, see the hdinsight-storm-examples repository on GitHub.

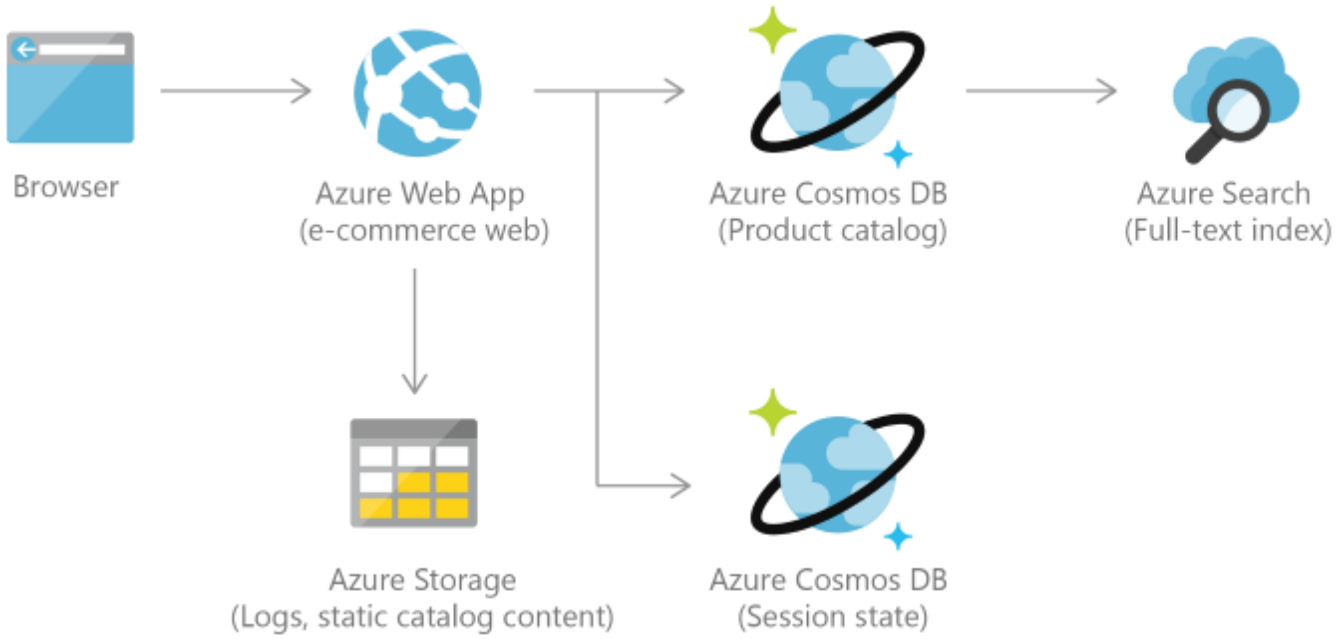
For more information on Azure offerings for IoT, see Create the Internet of Your Things.

Retail and marketing

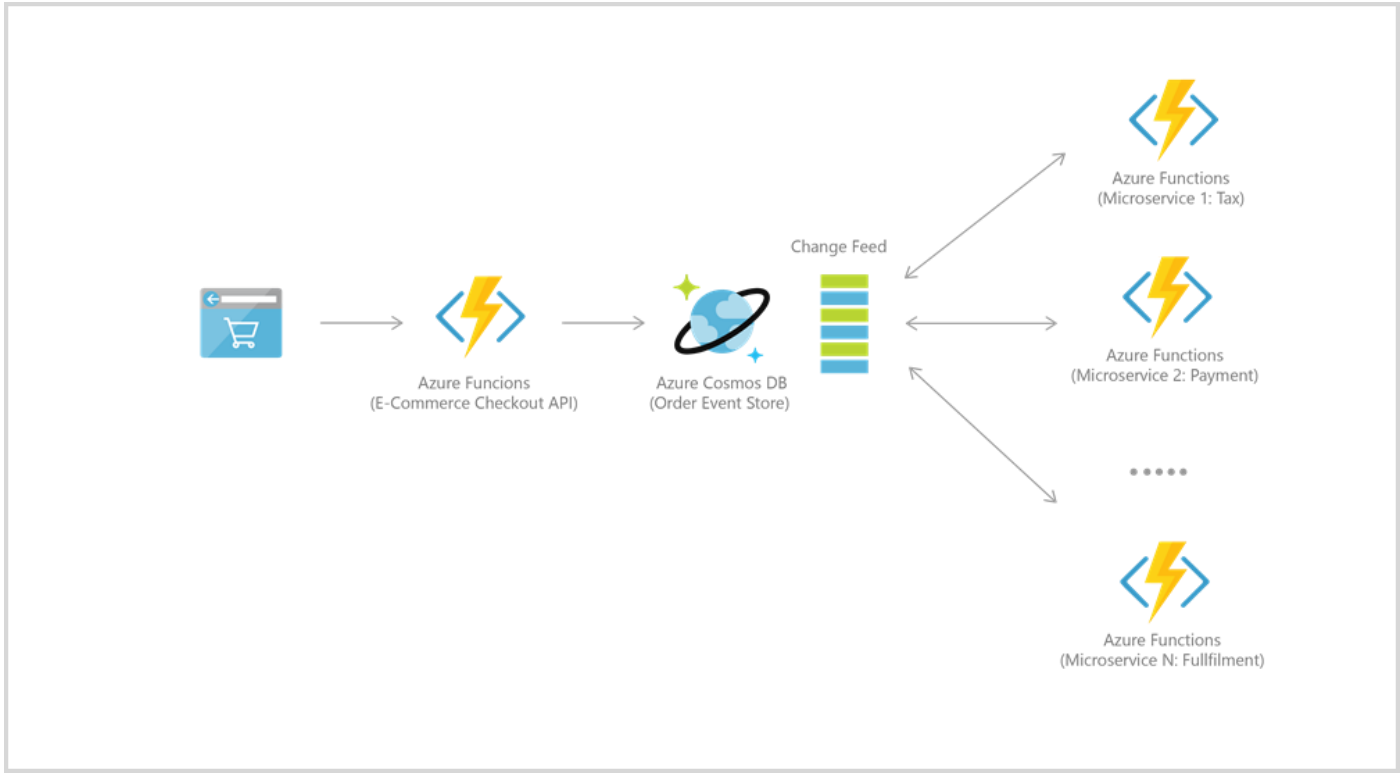
Azure Cosmos DB is used extensively in Microsoft's own e-commerce platforms, that run the Windows Store and XBox Live. It is also used in the retail industry for storing catalog data and for event sourcing in order processing pipelines.

Catalog data usage scenarios involve storing and querying a set of attributes for entities such as people, places, and products. Some examples of catalog data are user accounts, product catalogs, IoT device registries, and bill of materials systems. Attributes for this data may vary and can change over time to fit application requirements.

Consider an example of a product catalog for an automotive parts supplier. Every part may have its own attributes in addition to the common attributes that all parts share. Furthermore, attributes for a specific part can change the following year when a new model is released. Azure Cosmos DB supports flexible schemas and hierarchical data, and thus it is well suited for storing product catalog data.



Azure Cosmos DB is often used for event sourcing to power event driven architectures using its change feed functionality. The change feed provides downstream microservices the ability to reliably and incrementally read inserts and updates (for example, order events) made to an Azure Cosmos DB. This functionality can be leveraged to provide a persistent event store as a message broker for state-changing events and drive order processing workflow between many microservices (which can be implemented as serverless Azure Functions).



In addition, data stored in Azure Cosmos DB can be integrated with HDInsight for big data analytics via Apache Spark jobs. For details on the Spark Connector for Azure Cosmos DB, see Run a Spark job with Azure Cosmos DB and HDInsight.

Gaming

The database tier is a crucial component of gaming applications. Modern games perform graphical processing on mobile/console clients, but rely on the cloud to deliver customized and personalized content like in-game stats, social media integration, and high-score leaderboards. Games often require single-millisecond latencies for reads and writes to provide an engaging in-game experience. A game database needs to be fast and be able to handle massive spikes in request rates during new game launches and feature updates.

Azure Cosmos DB is used by games like The Walking Dead: No Man's Land by Next Games, and Halo 5: Guardians. Azure Cosmos DB provides the following benefits to game developers:

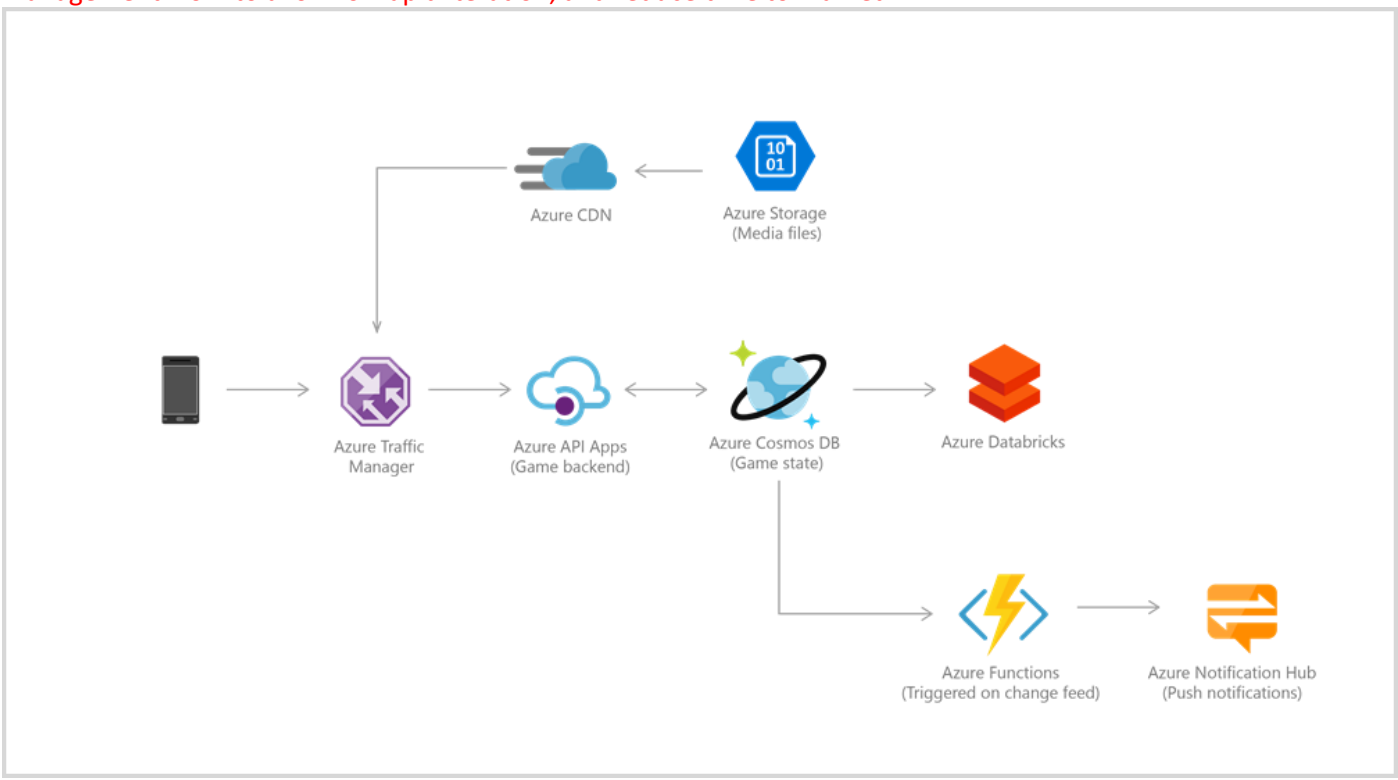
Azure Cosmos DB allows performance to be scaled up or down elastically. This allows games to handle updating profile and stats from dozens to millions of simultaneous gamers by making a single API call.

Azure Cosmos DB supports millisecond reads and writes to help avoid any lags during game play.

Azure Cosmos DB's automatic indexing allows for filtering against multiple different properties in real-time, for example, locate players by their internal player IDs, or their GameCenter, Facebook, Google IDs, or query based on player membership in a guild. This is possible without building complex indexing or sharding infrastructure.

Social features including in-game chat messages, player guild memberships, challenges completed, high-score leaderboards, and social graphs are easier to implement with a flexible schema.

Azure Cosmos DB as a managed platform-as-a-service (PaaS) required minimal setup and management work to allow for rapid iteration, and reduce time to market.



Web and mobile applications

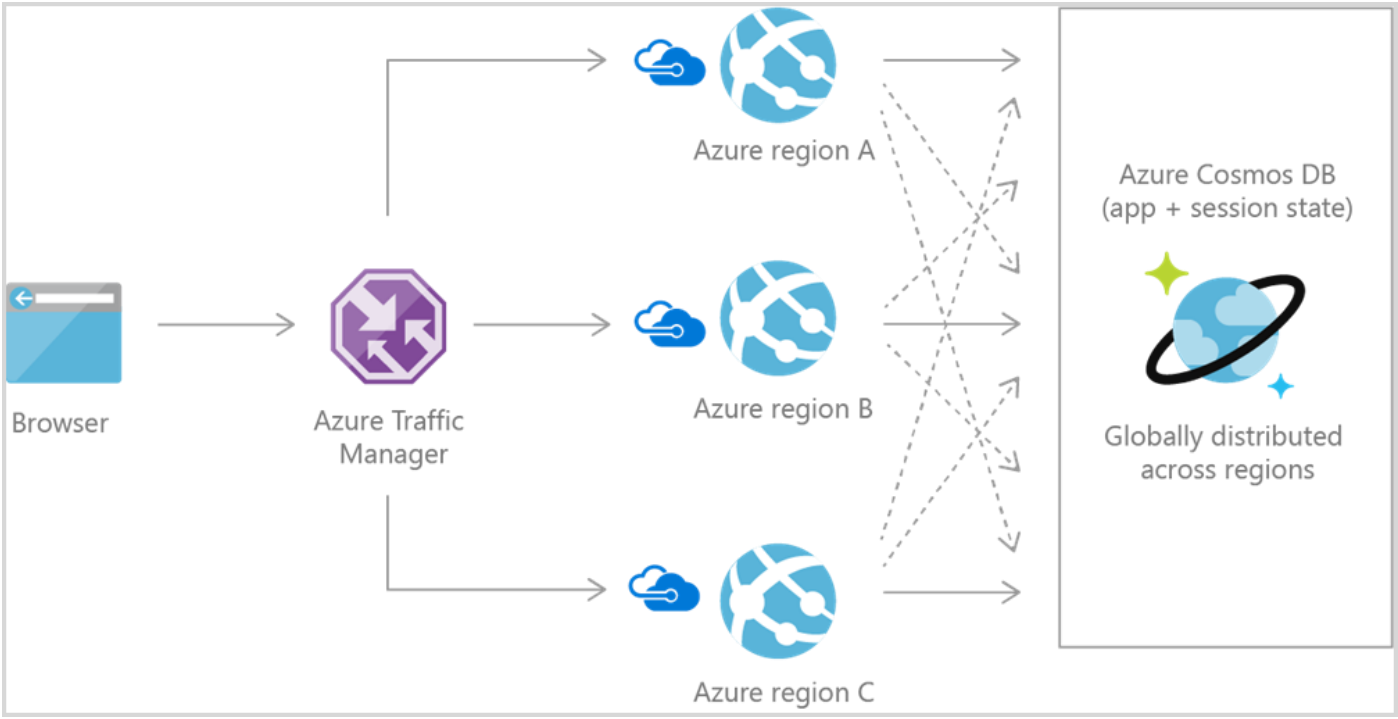
Azure Cosmos DB is commonly used within web and mobile applications, and is well suited for modeling social interactions, integrating with third-party services, and for building rich personalized experiences. The Azure Cosmos DB SDKs can be used build rich iOS and Android applications using the popular Xamarin framework.

Social Applications

A common use case for Azure Cosmos DB is to store and query user generated content (UGC) for web, mobile, and social media applications. Some examples of UGC are chat sessions, tweets, blog posts, ratings, and comments. Often, the UGC in social media applications is a blend of free form text, properties, tags, and relationships that are not bounded by rigid structure. Content such as chats, comments, and posts can be stored in Azure Cosmos DB without requiring transformations or complex object to relational mapping layers. Data properties can be added or modified easily to match requirements as developers iterate over the application code, thus promoting rapid development.

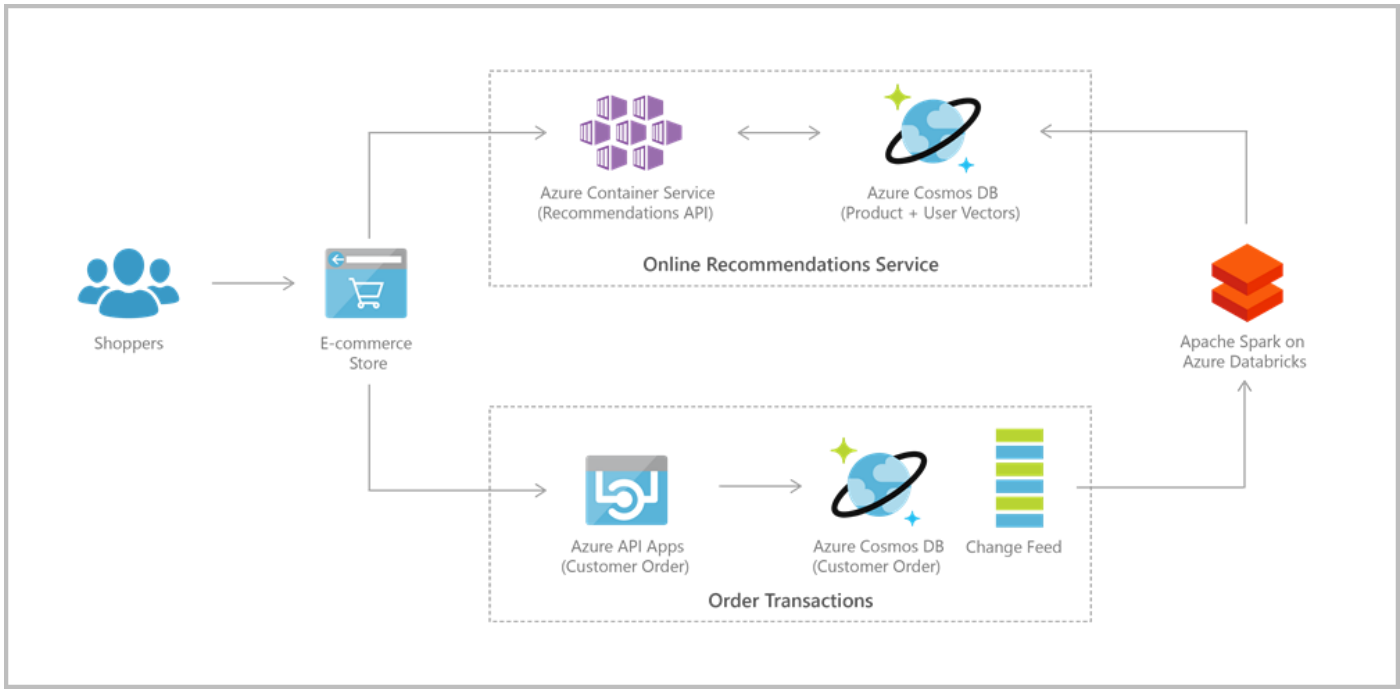
Applications that integrate with third-party social networks must respond to changing schemas from these networks. As data is automatically indexed by default in Azure Cosmos DB, data is ready to be queried at any time. Hence, these applications have the flexibility to retrieve projections as per their respective needs.

Many of the social applications run at global scale and can exhibit unpredictable usage patterns. Flexibility in scaling the data store is essential as the application layer scales to match usage demand. You can scale out by adding additional data partitions under an Azure Cosmos DB account. In addition, you can also create additional Azure Cosmos DB accounts across multiple regions. For Azure Cosmos DB service region availability, see Azure Regions.



Personalization

Nowadays, modern applications come with complex views and experiences. These are typically dynamic, catering to user preferences or moods and branding needs. Hence, applications need to be able to retrieve personalized settings effectively to render UI elements and experiences quickly. JSON, a format supported by Azure Cosmos DB, is an effective format to represent UI layout data as it is not only lightweight, but also can be easily interpreted by JavaScript. Azure Cosmos DB offers tunable consistency levels that allow fast reads with low latency writes. Hence, storing UI layout data including personalized settings as JSON documents in Azure Cosmos DB is an effective means to get this data across the wire.



• Describe Azure Cosmos DB APIs

PaaS offering. Need a CosmosDB account. Multi region. Can write to multiple regions based on Partition Key

---

Consistency Levels

Strong	Ensure all reads get up to date information
Bounded staleness	Quantity of version and Time. Strong consistency when data is read from region it was updated from
Session	Uses a token to ensure that reading and updating is from the same region
Consistent prefix	Only guarantees ordered updates
Eventual	Data will be synchronised eventually. May not be in order. For data that does not need grouping

---

High availability. Each partition has 4 copies of the data.

Request units. For billing. IKB = 1 RU, 1GB = 10 RU. Provisioned = Pay in advance, Auto Scale = used to limit, in Containier, Databases etc..

Identify Azure Cosmos DB APIs

Azure Cosmos DB is Microsoft's fully managed and serverless distributed database for applications of any size or scale, with support for both relational and non-relational workloads. Developers can build and migrate applications fast using their preferred open source database engines, including PostgreSQL, MongoDB, and Apache Cassandra. When you provision a new Cosmos DB instance, you select the database engine that you want to use. The choice of engine depends on many factors

including the type of data to be stored, the need to support existing applications, and the skills of the developers who will work with the data store.

Azure Cosmos DB for NoSQL

Azure Cosmos DB for NoSQL is Microsoft's native non-relational service for working with the document data model. It manages data in JSON document format, and despite being a NoSQL data storage solution, uses SQL syntax to work with the data.

A SQL query for an Azure Cosmos DB database containing customer data might look similar to this:

SQLCopy

```
SELECT *
FROM customers c
WHERE c.id = "joe@litware.com"
```

The result of this query consists of one or more JSON documents, as shown here:

JSONCopy

```
{
   "id": "joe@litware.com",
   "name": "Joe Jones",
   "address": {
       "street": "1 Main St.",
       "city": "Seattle"
    }
}
```

Azure Cosmos DB for MongoDB

MongoDB is a popular open source database in which data is stored in Binary JSON (BSON) format. Azure Cosmos DB for MongoDB enables developers to use MongoDB client libraries and code to work with data in Azure Cosmos DB.

MongoDB Query Language (MQL) uses a compact, object-oriented syntax in which developers use *objects* to call *methods*. For example, the following query uses the **find** method to query the **products** collection in the **db** object:

JavaScriptCopy

```
db.products.find({id: 123})
```

The results of this query consist of JSON documents, similar to this:

JSONCopy

```
{
   "id": 123,
   "name": "Hammer",
   "price": 2.99
}
```

Azure Cosmos DB for PostgreSQL

Azure Cosmos DB for PostgreSQL is a native PostgreSQL, globally distributed relational database that automatically shards data to help you build highly scalable apps. You can start building apps on a single node server group, the same way you would with PostgreSQL anywhere else. As your app's scalability and performance requirements grow, you can seamlessly scale to multiple nodes by transparently distributing your tables. PostgreSQL is a relational database management system (RDBMS) in which you define relational tables of data, for example you might define a table of products like this:

ProductID	ProductName
123	Hammer
162	Screwdriver

You could then query this table to retrieve the name and price of a specific product using SQL like this:

SQLCopy

```
SELECT ProductName, Price
```

```
FROM Products
WHERE ProductID = 123;
```

The results of this query would contain a row for product 123, like this:

ProductName	Price
Hammer	2.99

Azure Cosmos DB for Table

Azure Cosmos DB for Table is used to work with data in key-value tables, similar to Azure Table Storage. It offers greater scalability and performance than Azure Table Storage. For example, you might define a table named Customers like this:

PartitionKey	RowKey	Name	Email
1	123	Joe Jones	joe@litware.com
1	124	Samir Nadoy	samir@northwind.com

You can then use the Table API through one of the language-specific SDKs to make calls to your service endpoint to retrieve data from the table. For example, the following request returns the row containing the record for *Samir Nadoy* in the table above:

textCopy

```
https://endpoint/Customers(PartitionKey='1',RowKey='124')
```

Azure Cosmos DB for Apache Cassandra

Azure Cosmos DB for Apache Cassandra is compatible with Apache Cassandra, which is a popular open source database that uses a column-family storage structure. Column families are tables, similar to those in a relational database, with the exception that it's not mandatory for every row to have the same columns.

For example, you might create an **Employees** table like this:

ID	Name	Manager
1	Sue Smith	
2	Ben Chan	Sue Smith

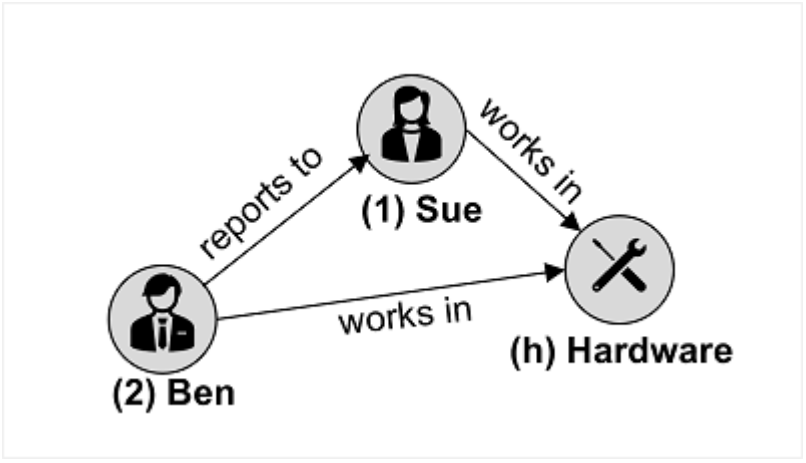
Cassandra supports a syntax based on SQL, so a client application could retrieve the record for *Ben Chan* like this:

SQLCopy

```
SELECT * FROM Employees WHERE ID = 2
```

Azure Cosmos DB for Apache Gremlin

Azure Cosmos DB for Apache Gremlin is used with data in a graph structure; in which entities are defined as vertices that form nodes in connected graph. Nodes are connected by edges that represent relationships, like this:



The example in the image shows two kinds of vertex (employee and department) and edges that connect them (employee "Ben" reports to employee "Sue", and both employees work in the "Hardware" department).

Gremlin syntax includes functions to operate on vertices and edges, enabling you to insert, update, delete, and query data in the graph. For example, you could use the following code to add a new employee named *Alice* that reports to the employee with ID **1** (*Sue*)

Copy

```
g.addV('employee').property('id', '3').property('firstName', 'Alice')
g.V('3').addE('reports to').to(g.V('1'))
```

The following query returns all of the *employee* vertices, in order of ID.

Copy

```
g.V().hasLabel('employee').order().by('id')
```

Cosmos DB API's - Document
MongoDB API for MongoDB - Document
Core (SQL) API is the default for CosmosDB. Data types include Undefined, Null, Boolean, String, Number and Object. JSON storage with SQL commands – Key and Partitions (Sharding means on separate locations)
Cassandra API is for Column storage
Gremlin API is for Graph storage
Azure Table API for Key Value
Also
Performance levels: Standard (old fashion hard drives, or Premium (SSD drives))
Account types: General Purpose v2, used for Data Lake. BlockBlobStorage is for Premium performance level with and without append. FileStorage seems same as above
Access Tiers: Hot, Access cheap, storage expensive. Cool, Storage cheaper but access expensive. Archive, Storage very cheap. Needs to copy data to a tier above before usage. May take time (offline) to copy. Life cycle management might move data between tiers (policies).
Replication Options (with Data centre and availability zone meaning the same thing)
Locally redundant storage (LRS) = 3 copies in same Data centre in one region
Zone redundant storage (ZRS) = 3 copies in different Data centres in one region
Geo-redundant storage (GRS) = As LRS except that another 3 copies are copied to another geographical region with 3 copies in one data centre
Geo-zone-redundant storage (GZRS) = As ZRS except that another 3 copies are copied to another geographical region with 3 copies in 3 different Data centres
RA-GRS allows read from replicas for GRS
RAGZRS allows read from replicas for GZRS