Module – 3 Microsoft Azure Data and Storage

Applications need data, and different kinds of applications need different kinds of data. Because of this, Azure provides several different ways to store and manage data. Azure provides many storage options, but all are designed for very durable storage. With any of these options, there are always 3 copies of your data kept in sync across an Azure datacenter -- 6 if you allow Azure to use geo-redundancy to back up to another datacenter at least 300 miles away.

Agenda:

- Importance of Azure Data & Storage
- Azure SQL Database Services
- Azure Storage: Blobs, Tables, Queue and Files
- Azure Cosmos DB (No SQL Database)

Importance of Azure Data & Storage

Azure storage is one of the cloud computing PAAS(Platform as a service) service provided by the Microsoft azure.

The storage option is one of the best computing service provided by Azure as it supports both legacy application development using Azure SQL and modern application development using Azure No-SQL table storage.

Storage in azure can be broadly classified into two categories based on the type of data that we are going to save.

1. Relational data Storage
2. NonRelational data storage

Relational Data Storage:

Relational data can be saved in the cloud using Azure SQL storage.

Non-Relational Data Storage:

This kind of cloud storage option enables the users to store their documents, media files NoSQL data over the cloud that can be accessed using REST APIs.

Azure provides four types of storage options based on the data type.

1. Blob storage
2. Queue storage
3. Table Storage
4. File storage
Azure storage have following advantages:

**Scalability:**
We can start with small size blob and we can increase the size as per the demand to an infinite number without affecting production environment.

**Secure and Reliable:**
Security for azure storage data can be provided in two ways, By using Storage Account access keys and by server level and client level encryption.

**Durability & High availability:**
Replication concept has been used for azure storage in order to give high availability (99.99 % uptime) and durability. This replication concept maintains different copies of your data to different location or region based on the replication option [Locally redundant storage, Zone-redundant storage, Geo-redundant storage, Read-access geo-redundant storage] at the time of creating a storage account.

## Azure SQL Database Services

SQL Database is a general-purpose relational database service in Microsoft Azure that supports structures such as relational data, JSON, spatial, and XML. It delivers dynamically scalable performance and provides options such as columnstore indexes for extreme analytic analysis and reporting, and in-memory OLTP for extreme transactional processing. Microsoft handles all patching and updating of the SQL code base seamlessly and abstracts away all management of the underlying infrastructure.

SQL Database delivers predictable performance at multiple service levels that provides dynamic scalability with no downtime, built-in intelligent optimization, global scalability and availability, and advanced security options — all with near-zero administration. These capabilities allow you to focus on rapid app development and accelerating your time to market, rather than allocating precious time and resources to managing virtual machines and infrastructure. The SQL Database service is currently in 38 data centers around the world, with more data centers coming online regularly, which enables you to run your database in a data center near you.

### Scalable performance and pools

With SQL Database, each database is isolated from each other and portable, each with its own service tier with a guaranteed performance level. SQL Database provides different performance levels for different needs, and enables databases to be pooled to maximize the use of resources and save money.

### Adjust performance and scale without downtime

SQL Database offers four service tiers to support lightweight to heavyweight database workloads: Basic, Standard, Premium, and Premium RS. You can build your first app on a small, single database at a low cost per month and then change its service tier manually or programmatically at any time to meet the needs of your solution. You can adjust performance without downtime to
your app or to your customers. Dynamic scalability enables your database to transparently respond to rapidly changing resource requirements and enables you to only pay for the resources that you need when you need them.

Elastic pools to maximize resource utilization

For many businesses and applications, being able to create single databases and dial performance up or down on demand is enough, especially if usage patterns are relatively predictable. But if you have unpredictable usage patterns, it can make it hard to manage costs and your business model. Elastic pools are designed to solve this problem. The concept is simple. You allocate performance resources to a pool rather than an individual database, and pay for the collective performance resources of the pool rather than for single database performance.

With elastic pools, you don’t need to focus on dialing database performance up and down as demand for resources fluctuates. The pooled databases consume the performance resources of the elastic pool as needed. Pooled databases consume but don’t exceed the limits of the pool, so your cost remains predictable even if individual database usage doesn’t. What’s more, you can add and remove databases to the pool, scaling your app from a handful of databases to thousands, all within a budget that you control.
You can also control the minimum and maximum resources available to databases in the pool to ensure that no database in the pool uses all the pool resources and that every pooled database has a guaranteed minimum amount of resources. To learn more about design patterns for SaaS applications using elastic pools, see Design Patterns for Multi-tenant SaaS Applications with SQL Database.

**Blend single databases with pooled databases**

Either way you go — single databases or elastic pools — you are not locked in. You can blend single databases with elastic pools, and change the service tiers of single databases and elastic pools quickly and easily to adapt to your situation. With the power and reach of Azure, you can mix-and-match other Azure services with SQL Database to meet your unique app design needs, drive cost and resource efficiencies, and unlock new business opportunities.

**Extensive monitoring and alerting capabilities**

But how can you compare the relative performance of single databases and elastic pools? How do you know the right click-stop when you dial up and down? You use the built-in performance monitoring and alerting tools, combined with the performance ratings based on Database Transaction Units (DTUs) for single databases and elastic DTUs (eDTUs) for elastic pools. Using these tools, you can quickly assess the impact of scaling up or down based on your current or project performance needs. See SQL Database options and performance: Understand what’s available in each service tier for details.

Additionally, SQL Database can emit metrics and diagnostic logs for easier monitoring. You can configure SQL Database to store resource usage, workers and sessions, and connectivity into one of these Azure resources:

- **Azure Storage**: For archiving vast amounts of telemetry for a small price
- **Azure Event Hub**: For integrating SQL Database telemetry with your custom monitoring solution or hot pipelines
- **Azure Log Analytics**: For built-in monitoring solution with reporting, alerting, and mitigating capabilities
Availability capabilities

Azure's industry leading 99.99% availability service level agreement (SLA), powered by a global network of Microsoft-managed datacenters, helps keep your app running 24/7. In addition, SQL Database provides built-in business continuity and global scalability features, including:

- **Automatic backups**: SQL Database automatically performs full, differential, and transaction log backups.
- **Point-in-time restores**: SQL Database supports recovery to any point in time within the automatic backup retention period.
- **Active geo-replication**: SQL Database allows you to configure up to four readable secondary databases in either the same or globally distributed Azure data centers. For example, if you have a SaaS application with a catalog database that has a high volume of concurrent read-only transactions, use active geo-replication to enable global read scale and remove bottlenecks on the primary that are due to read workloads.
- **Failover groups**: SQL Database allows you to enable high availability and load balancing at global scale, including transparent geo-replication and failover of large sets of databases and elastic pools. Failover groups and active geo-replication enables creation of globally distributed SaaS applications with minimal administration overhead leaving all the complex monitoring, routing, and failover orchestration to SQL Database.

Built-in intelligence

With SQL Database, you get built-in intelligence that helps you dramatically reduce the costs of running and managing databases and maximizes both performance and security of your application. Running millions of customer workloads around-the-clock, SQL Database collects and processes a massive amount of telemetry data, while also fully respecting customer privacy behind
the scenes. Various algorithms are continuously evaluating the telemetry data so that the service can learn and adapt with your application. Based on this analysis, the service comes up with performance improving recommendations tailored to your specific workload.

**Automatic performance monitoring and tuning**

SQL Database provides detailed insight into the queries that you need to monitor. SQL Database's learns about your database patterns and enables you to adapt your database schema to your workload. SQL Database provides performance tuning recommendations, where you can review tuning actions and apply them.

**Adaptive query processing**

Azure also adding the adaptive query processing family of features to SQL Database, including interleaved execution for multi-statement table-valued functions, batch mode memory grant feedback, and batch mode adaptive joins. Each of these adaptive query processing features applies similar "learn and adapt" techniques, helping further address performance issues related to historically intractable query optimization problems.

**Intelligent threat detection**

SQL Threat Detection leverages SQL Database auditing to continuously monitor Azure SQL databases for potentially harmful attempts to access sensitive data. SQL threat detection provides a new layer of security, which enables customers to detect and respond to potential threats as they occur by providing security alerts on anomalous activities.

**Advanced security and compliance**

SQL Database provides a range of built-in security and compliance features to help your application meet various security and compliance requirements.

- Auditing for compliance and security
- Data encryption at rest
- Data encryption in motion
- Dynamic data masking
- Row-level security
- Azure Active Directory integration and multi-factor authentication
- Compliance certification

**Easy-to-use tools**

SQL Database makes building and maintaining applications easier and more productive. SQL Database allows you to focus on what you do best: building great apps. You can manage and develop in SQL Database using tools and skills you already have.

- **The Azure portal**: A web-based application for managing all Azure services
• **SQL Server Management Studio**: A free, downloadable client application for managing any SQL infrastructure, from SQL Server to SQL Database

• **SQL Server Data Tools in Visual Studio**: A free, downloadable client application for developing SQL Server relational databases, Azure SQL databases, Integration Services packages, Analysis Services data models, and Reporting Services reports.

• **Visual Studio Code**: a free, downloadable, open source, code editor for Windows, macOS, and Linux that supports extensions, including the mssql extension for querying Microsoft SQL Server, Azure SQL Database, and SQL Data Warehouse.

### Choosing a service tier

Azure SQL Database offers **Basic**, **Standard**, **Premium**, and **Premium RS** service tiers for both single databases and elastic pools. Service tiers are primarily differentiated by a range of performance level and storage size choices, and price. All service tiers provide flexibility in changing performance level and storage size. Single databases and elastic pools are billed hourly based on service tier, performance level, and storage size.

<table>
<thead>
<tr>
<th>Target workload</th>
<th>Basic</th>
<th>Standard</th>
<th>Premium</th>
<th>Premium RS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development and production</td>
<td>Development and production</td>
<td>Development and production</td>
<td>Development and production</td>
<td>Workload that can tolerate data loss up to 5-minutes due to service failures</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Basic</th>
<th>Standard</th>
<th>Premium</th>
<th>Premium RS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uptime SLA</td>
<td>99.99%</td>
<td>99.99%</td>
<td>99.99%</td>
<td>N/A while in preview</td>
</tr>
<tr>
<td>Backup retention</td>
<td>7 days</td>
<td>35 days</td>
<td>35 days</td>
<td>35 days</td>
</tr>
<tr>
<td>CPU</td>
<td>Low</td>
<td>Low, Medium, High</td>
<td>Medium, High</td>
<td>Medium</td>
</tr>
<tr>
<td>IO throughput</td>
<td>Low</td>
<td>Medium</td>
<td>Order of magnitude higher than Standard</td>
<td>Same as Premium</td>
</tr>
<tr>
<td>IO latency</td>
<td>Higher than Premium</td>
<td>Higher than Premium</td>
<td>Lower than Basic and Standard</td>
<td>Same as Premium</td>
</tr>
<tr>
<td></td>
<td>Basic</td>
<td>Standard</td>
<td>Premium</td>
<td>Premium RS</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------</td>
<td>----------</td>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>Columnstore indexing and in-memory OLTP</td>
<td>N/A</td>
<td>N/A</td>
<td>Supported</td>
<td>Supported</td>
</tr>
</tbody>
</table>

**Use SQL Server Management Studio to connect and query data**

Get the connection information needed to connect to the Azure SQL database. You will need the fully qualified server name, database name, and login information in the next procedures.

Use SQL Server Management Studio to establish a connection to your Azure SQL Database server.

1. Open SQL Server Management Studio.
2. In the **Connect to Server** dialog box, enter the following information:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Suggested value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Server type</strong></td>
<td>Database engine</td>
<td>This value is required.</td>
</tr>
<tr>
<td><strong>Server name</strong></td>
<td>The fully qualified server name</td>
<td>The name should be something like this: mytestserver.database.windows.net.</td>
</tr>
<tr>
<td><strong>Authentication</strong></td>
<td>SQL Server Authentication</td>
<td>SQL Authentication is the only authentication type that we have configured in this tutorial.</td>
</tr>
<tr>
<td><strong>Login</strong></td>
<td>The server admin account</td>
<td>This is the account that you specified when you created the server.</td>
</tr>
<tr>
<td><strong>Password</strong></td>
<td>The password for your server admin account</td>
<td>This is the password that you specified when you created the server.</td>
</tr>
</tbody>
</table>
3. Click **Options** in the **Connect to server** dialog box. In the **Connect to database** section, enter **mySampleDatabase** to connect to this database.

4. Click **Connect**. The Object Explorer window opens in SSMS.
5. In Object Explorer, expand **Databases** and then expand **mySampleDatabase** to view the objects in the sample database.

### Query data

Use the following code to query for the top 20 products by category using the `SELECT` Transact-SQL statement.

1. In Object Explorer, right-click **mySampleDatabase** and click **New Query**. A blank query window opens that is connected to your database.
2. In the query window, enter the following query:

   ```sql
   SELECT pc.Name as CategoryName, p.name as ProductName 
   FROM [SalesLT].[ProductCategory] pc 
   JOIN [SalesLT].[Product] p 
   ON pc.productcategoryid = p.productcategoryid;
   ```

3. On the toolbar, click **Execute** to retrieve data from the Product and ProductCategory tables.
Azure Storage

Microsoft Azure Storage is a Microsoft-managed cloud service that provides storage that is highly available, secure, durable, scalable, and redundant. Microsoft takes care of maintenance and handles critical problems for you.

Azure Storage consists of three data services: Blob storage, File storage, and Queue storage. Blob storage supports both standard and premium storage, with premium storage using only SSDs for the fastest performance possible. Another feature is cool storage, allowing you to store large amounts of rarely accessed data for a lower cost.

Introducing the Azure Storage services

To use any of the services provided by Azure Storage -- Blob storage, File storage, Table storage and Queue storage -- you first create a storage account, and then you can transfer data to/from a specific service in that storage account.

Blob storage
Blobs are basically files like those that you store on your computer (or tablet, mobile device, and so on). They can be pictures, Microsoft Excel files, HTML files, virtual hard disks (VHDs), big data such as logs, database backups -- pretty much anything. Blobs are stored in containers, which are similar to folders.

After storing files in Blob storage, you can access them from anywhere in the world using URLs, the REST interface, or one of the Azure SDK storage client libraries. Storage client libraries are available for multiple languages, including Node.js, Java, PHP, Ruby, Python, and .NET.

There are three types of blobs: block blobs, page blobs (used for VHD files), and append blobs.

- **Block blobs** are used to hold ordinary files up to about 4.7 TB.
- **Page blobs** are used to hold random access files up to 8 TB in size. These are used for the VHD files that back VMs.
- **Append blobs** are made up of blocks like the block blobs, but are optimized for append operations. These are used for things like logging information to the same blob from multiple VMs.

For very large datasets where network constraints make uploading or downloading data to Blob storage over the wire unrealistic, you can ship a set of hard drives to Microsoft to import or export data directly from the data center.

**Blob service concepts**

The Blob service contains the following components:

- **Storage Account**: All access to Azure Storage is done through a storage account. This storage account can be a *General-purpose storage account* or a *Blob storage account* that is specialized for storing objects/blobs.
- **Container**: A container provides a grouping of a set of blobs. All blobs must be in a container. An account can contain an unlimited number of containers. A container can store an unlimited number of blobs. Note that the container name must be lowercase.
- **Blob**: A file of any type and size.

![Diagram of account, container, and blob relationships](image-url)
File Storage

Azure Files offers fully managed file shares in the cloud that are accessible via the industry standard Server Message Block (SMB) protocol (also known as Common Internet File System or CIFS). Azure File shares can be mounted concurrently by cloud or on-premises deployments of Windows, Linux, and macOS. Additionally, Azure File shares can be cached on Windows Servers with Azure File Sync (preview) for fast access near where the data is being used.

Why Azure Files is useful

Azure File shares can be used to:

- **Replace or supplement on-premises file servers:**
  Azure Files can be used to completely replace or supplement traditional on-premises file servers or NAS devices. Popular operating systems such as Windows, macOS, and Linux can directly mount Azure File shares wherever they are in the world. Azure File shares can also be replicated with Azure File Sync to Windows Servers, either on-premises or in the cloud, for performant and distributed caching of the data where it's being used.

- **"Lift and Shift" applications:**
  Azure Files makes it easy to "lift and shift" applications to the cloud that expect a file share to store file application or user data. Azure Files enables both the "classic" lift and shift scenario, where both the application and it's data are moved to Azure, and the "hybrid" lift and shift scenario, where the application data is moved to Azure Files, and the application continues to run on-premises.

- **Simplify Cloud Development:**
  Azure Files can also be used in numerous ways to simplify new cloud development projects. For example:
  - **Shared Application Settings:**
    A common pattern for distributed applications is to have configuration files in a centralized location where they can be accessed from many application instances. Application instances can load their configuration through the File REST API, and humans can access them as needed by mounting the SMB share locally.
  - **Diagnostic Share:**
    An Azure File share is a convenient place for cloud applications to write logs, metrics, and crash dumps. Logs can be written by the application instances via the File REST API, and developers can access them by mounting the file share on their local machine. This enables great flexibility, as developers can embrace cloud development without having to abandon any existing tooling they know and love.
  - **Dev/Test/Debug:**
    When developers or administrators are working on VMs in the cloud, they often need a set of tools or utilities. Copying such utilities and tools to each VM can be a time consuming exercise. By mounting an Azure File share locally on the VMs, a developer and administrator can quickly access their tools and utilities, no copying required.

Key Benefits
• **Shared access.** Azure File shares support the industry standard SMB protocol, meaning you can seamlessly replace your on-premises file shares with Azure File shares without worrying about application compatibility. Being able to share a file system across multiple machines, applications/instances is a significant advantage with Azure Files for applications that need shareability.

• **Fully Managed.** Azure File shares can be created without the need to manage hardware or an OS. This means you don’t have to deal with patching the server OS with critical security upgrades or replacing faulty hard disks.

• **Scripting and Tooling.** PowerShell cmdlets and Azure CLI can be used to create, mount, and manage Azure File shares as part of the administration of Azure applications. You can create and manage Azure file shares using Azure portal and Azure Storage Explorer.

• **Resiliency.** Azure Files has been built from the ground up to be always available. Replacing on-premises file shares with Azure Files means you no longer have to wake up to deal with local power outages or network issues.

• **Familiar Programmability.** Applications running in Azure can access data in the share via file system I/O APIs. Developers can therefore leverage their existing code and skills to migrate existing applications. In addition to System IO APIs, you can use Azure Storage Client Libraries or the Azure Storage REST API.

---

**Queue Storage**

Azure Queue storage is a service for storing large numbers of messages that can be accessed from anywhere in the world via authenticated calls using HTTP or HTTPS. A single queue message can be up to 64 KB in size, and a queue can contain millions of messages, up to the total capacity limit of a storage account.

**Common uses**

Common uses of Queue storage include:

- Creating a backlog of work to process asynchronously
- Passing messages from an Azure web role to an Azure worker role

**Queue service concepts**

The Queue service contains the following components:
• **Storage account:** All access to Azure Storage is done through a storage account
• **Queue:** A queue contains a set of messages. All messages must be in a queue. Note that the queue name must be all lowercase.
• **Message:** A message, in any format, of up to 64 KB. The maximum time that a message can remain in the queue is seven days.

**Table Storage**

Azure Table storage is a service that stores 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.

Common uses of Table storage include:

- Storing TBs of structured data capable of serving web scale applications
- Storing datasets that don't require complex joins, foreign keys, or stored procedures and can be denormalized for fast access
- Quickly querying data using a clustered index
- Accessing data using the OData protocol and LINQ queries with WCF Data Service .NET Libraries

You can use Table storage to store and query huge sets of structured, non-relational data, and your tables will scale as demand increases.
Table storage concepts

Table storage contains the following components:

- **Storage Account**: All access to Azure Storage is done through a storage account.
- **Table**: A table is a collection of entities. Tables don’t enforce a schema on entities, which means a single table can contain entities that have different sets of properties. The number of tables that a storage account can contain is limited only by the storage account capacity limit.
- **Entity**: An entity is a set of properties, similar to a database row. An entity can be up to 1MB in size.
- **Properties**: A property is a name-value pair. Each entity can include up to 252 properties to store data. Each entity also has three system properties that specify a partition key, a row key, and a timestamp. Entities with the same partition key can be queried more quickly, and inserted/updated in atomic operations. An entity’s row key is its unique identifier within a partition.

Table Entities

Table entities represent the units of data stored in a table and are similar to rows in a typical relational database table. Each entity defines a collection of properties. Each property is key/value pair defined by its name, value, and the value’s data type. Entities must define the following three system properties as part of the property collection:

- **PartitionKey** – The PartitionKey property stores string values that identify the partition that an entity belongs to. Partitions, as discussed later, are integral to the scalability of the table. Entities with the same PartitionKey values are stored in the same partition.
- **RowKey** – The RowKey property stores string values that uniquely identify entities within each partition. The PartitionKey and the RowKey together form the primary key for the entity.
- **Timestamp** – The Timestamp property provides traceability for an entity. A timestamp is a DateTime value that tells you the last time the entity was modified. A timestamp is sometimes referred to as the entity’s version. Modifications to timestamps are ignored because the table service maintains the value for this property during all inserts and update operations.
Table Partitions

Partitions represent a collection of entities with the same PartitionKey values. Partitions are always served from one partition server and each partition server can serve one or more partitions. A partition server has a rate limit of the number of entities it can serve from one partition over time. Specifically, a partition has a scalability target of 500 entities per second. This throughput may be higher during minimal load on the storage node, but it will be throttled down when the node becomes hot or very active. To better illustrate the concept of partitioning, the following figure illustrates a table that contains a small subset of data for footrace event registrations. It presents a conceptual view of partitioning where the PartitionKey contains three different values comprised of the event’s name and distance. In this example, there are two partition servers. Server A contains registrations for the half-marathon and 10 Km distances while Server B contains only the full-marathon distances. The RowKey values are shown to provide context but are not meaningful for this example. A table with three partitions:

<table>
<thead>
<tr>
<th>PartitionKey</th>
<th>RowKey</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>2011 New York City Marathon Full</td>
<td>1234 John M_55</td>
<td>-</td>
</tr>
<tr>
<td>2011 New York City Marathon Full</td>
<td>1235 Jane F_25</td>
<td>-</td>
</tr>
<tr>
<td>2011 New York City Marathon Half</td>
<td>1236 Jack M_26</td>
<td>-</td>
</tr>
<tr>
<td>2011 New York City Marathon Half</td>
<td>1237 Bob M_21</td>
<td>-</td>
</tr>
<tr>
<td>2011 New York City Marathon Half</td>
<td>1238 Jill F_31</td>
<td>-</td>
</tr>
<tr>
<td>2011 New York City Marathon 10K</td>
<td>1239 Julie F_22</td>
<td>-</td>
</tr>
</tbody>
</table>

Shared access signatures (SAS)

A shared access signature (SAS) provides you with a way to grant limited access to objects in your storage account to other clients, without exposing your account key.

A shared access signature provides delegated access to resources in your storage account. With a SAS, you can grant clients access to resources in your storage account, without sharing your account keys. This is the key point of using shared access signatures in your applications—a SAS is a secure way to share your storage resources without compromising your account keys.

A SAS gives you granular control over the type of access you grant to clients who have the SAS, including:

- The interval over which the SAS is valid, including the start time and the expiry time.
- The permissions granted by the SAS. For example, a SAS for a blob might grant read and write permissions to that blob, but not delete permissions.
- An optional IP address or range of IP addresses from which Azure Storage will accept the SAS. For example, you might specify a range of IP addresses belonging to your organization.
• The protocol over which Azure Storage will accept the SAS. You can use this optional parameter to restrict access to clients using HTTPS.

When should you use a shared access signature?

A common scenario where a SAS is useful is a service where users read and write their own data to your storage account. In a scenario where a storage account stores user data, there are two typical design patterns:

1. Clients upload and download data via a front-end proxy service, which performs authentication. This front-end proxy service has the advantage of allowing validation of business rules, but for large amounts of data or high-volume transactions, creating a service that can scale to match demand may be expensive or difficult.

2. A lightweight service authenticates the client as needed and then generates a SAS. Once the client receives the SAS, they can access storage account resources directly with the permissions defined by the SAS and for the interval allowed by the SAS. The SAS mitigates the need for routing all data through the front-end proxy service.

Types of shared access signatures

You can create two types of shared access signatures:

• **Service SAS.** The service SAS delegates access to a resource in just one of the storage services: the Blob, Queue, Table, or File service.
• **Account SAS.** The account SAS delegates access to resources in one or more of the storage services. All of the operations available via a service SAS are also available via an account SAS. Additionally, with the account SAS, you can delegate access to operations that apply to a given service, such as **Get/Set Service Properties** and **Get Service Stats**. You can also delegate access to read, write, and delete operations on blob containers, tables, queues, and file shares that are not permitted with a service SAS.

How a shared access signature works
A shared access signature is a signed URI that points to one or more storage resources and includes a token that contains a special set of query parameters. The token indicates how the resources may be accessed by the client. One of the query parameters, the signature, is constructed from the SAS parameters and signed with the account key. This signature is used by Azure Storage to authenticate the SAS.

Here's an example of a SAS URI, showing the resource URI and the SAS token:

```
https://storageexample.blob.core.windows.net/sample-container/sampleBlob.bf?sv=2015-07-08&sn=399up8pfikyhuwM9JW984Dze7w06wRgQy8GA95d%3D&sr=c&sp=rwdl&se=2016-10-18T13%3A51%3A37Z&spw&st=2016-10-18T12%3A49%3A30Z&spr&sig=up8pfikyhuwM9JW984Dze7w06wRgQy8GA95d%3D
```

The SAS token is a string you generate on the client side (see the SAS examples section for code examples). A SAS token you generate with the storage client library, for example, is not tracked by Azure Storage in any way. You can create an unlimited number of SAS tokens on the client side.

When a client provides a SAS URI to Azure Storage as part of a request, the service checks the SAS parameters and signature to verify that it is valid for authenticating the request. If the service verifies that the signature is valid, then the request is authenticated. Otherwise, the request is declined with error code 403 (Forbidden).

**Azure Cosmos DB**

Azure Cosmos DB is Microsoft’s globally distributed, multi-model database. With the click of a button, Azure Cosmos DB enables you to elastically and independently scale throughput and storage across any number of Azure’s geographic regions. It offers throughput, latency, availability, and consistency guarantees with comprehensive service level agreements (SLAs), something no other database service can offer.
You can Try Azure Cosmos DB for free without an Azure subscription, free of charge and commitments.

Solutions that benefit from Azure Cosmos DB

Any web, mobile, gaming, and IoT applications that need to handle massive amounts of reads and writes on a global scale with low response times for a variety of data will benefit from Azure Cosmos DB’s guaranteed availability, high throughput, low latency, and tunable consistency.

Key capabilities

As a globally distributed database service, Azure Cosmos DB provides the following capabilities to help you build scalable, highly responsive applications:

- **Turnkey global distribution**
  - You can distribute your data to any number of Azure regions, with the click of a button. This enables you to put your data where your users are, ensuring the lowest possible latency to your customers.
  - Using Azure Cosmos DB’s multi-homing APIs, the app always knows where the nearest region is and will send requests to the nearest data center. All of this is possible with no config changes, you set your write region and as many read regions as you want and the rest is handled for you.

- **Multiple data models and popular APIs for accessing and querying data**
  - The atom-record-sequence (ARS) based data model that Azure Cosmos DB is built on natively supports multiple data models, including but not limited to document, graph, key-value, table, and columnar data models.
  - APIs for the following data models are supported with SDKs available in multiple languages:
    - DocumentDB API
    - MongoDB API
    - Table API
    - Graph (Gremlin) API
- Additional data models coming soon
- **Elastically scale throughput and storage on demand, worldwide**
  - Easily scale database throughput at a per second granularity, and change it anytime you want.
  - Scale storage size transparently and automatically to handle your size requirements now and forever.
- **Build highly responsive and mission-critical applications**
  - Azure Cosmos DB guarantees end-to-end low latency at the 99th percentile to its customers.
  - For a typical 1 KB item, Cosmos DB guarantees end-to-end latency of reads under 10 ms and indexed writes under 15 ms at the 99th percentile, within the same Azure region. The median latencies are significantly lower (under 5 ms).
- **Ensure "always on" availability**
  - 99.99% availability within a single region.
  - Deploy to any number of Azure regions for higher availability.
  - Simulate a failure of one or more regions with zero-data loss guarantees.
- **Write globally distributed applications, the right way**
  - Five consistency models provide a spectrum of strong SQL-like consistency all the way to NoSQL-like eventual consistency, and every thing in between.
- **Money back guarantees**
  - Your data gets there fast, or your money back.
  - Service level agreements for availability, latency, throughput, and consistency.
- **No database schema/index management**
  - Stop worrying about keeping your database schema and indexes in-sync with your application’s schema. We’re schema-free.
  - Azure Cosmos DB’s database engine is fully schema-agnostic – it automatically indexes all the data it ingests without requiring any schema or indexes and serves blazing fast queries.
- **Low cost of ownership**
  - Five to ten times more cost effective than a non-managed solution.
  - Three times cheaper than DynamoDB.

### Resource Model and API Projections

Developers can start using Azure Cosmos DB by provisioning a *database account* using their Azure subscription. A database account manages one or more *databases*. An Azure Cosmos DB database in-turn manages *users, permissions* and *containers*. An Azure Cosmos DB *container* is a schema-agnostic container of arbitrary user-generated entities and *stored procedures, triggers* and *user-defined functions* (UDFs). Entities under the customer’s database account – databases, users, permissions, containers etc., are referred to as *resources* as illustrated in Figure.
All the data within an Azure Cosmos DB container (e.g., collection, table, graph etc.) is horizontally partitioned and transparently managed by resource partitions as illustrated in Figure 3. A resource partition is a consistent and highly available container of data partitioned by a customer specified partition-key; it provides a single system image for a set of resources it manages and is a fundamental unit of scalability and distribution. Azure Cosmos DB is designed for customer to elastically scale throughput based on the application traffic patterns across different geographical regions to support fluctuating workloads varying both by geography and time. The system manages the partitions transparently without compromising the availability, consistency, latency or throughput of an Azure Cosmos DB container.
Customers can elastically scale throughput of a container by programmatically provisioning throughput at a second or minute granularity on an Azure Cosmos DB container. Internally, the system transparently manages resource partitions to deliver the throughput on a given container. Elastically scaling throughput using horizontal partitioning of resources requires that each resource partition is capable of delivering the portion of the overall throughput for a given budget of system resources. Since an Azure Cosmos DB container is globally distributed, Azure Cosmos DB ensures that the throughput of a container is available for use across all the regions where the container is distributed within a few seconds of the change in its value. Customers can provision throughput (measured in using a currency unit called, Request Unit or RU) on an Azure Cosmos DB container at both, second and at the minute granularities. The provisioned throughput at the minute granularity is used to effectively manage the unexpected spikes in the workload occurring at a second granularity. As an example consider a customer who has provisioned 10K RU/s and 100K RU/min on a container for a given hour. As seen in Figure 4 that the spikes occurring in the workload within any given minute are smoothened out by the RU/m provisioned for that minute. In this example, the customer was able to save the overall cost of provisioned throughput by as much as 73%.

---

**Global Distribution from the Ground-Up**

As illustrated in Figure 5, a customer’s resources are distributed along two dimensions: within a given region, all resources are horizontally partitioned using resource partitions (local distribution). Each resource partition is also replicated across geographical regions (global distribution).
When customers elastically scale throughput or storage, Azure Cosmos DB transparently performs partition management operations across all the regions. Independent of the scale, distribution, or failures, Azure Cosmos DB continues to provide a single system image of the globally-distributed resources. Global distribution of resources in Azure Cosmos DB is turnkey: at any time with a few button clicks (or programmatically with a single API call), customers can associate any number of geographical regions with their database account. Regardless of the amount of data or the number of regions, Azure Cosmos DB guarantees each newly associated region to start processing client requests in under an hour at the 99th percentile. This is done by parallelizing the seeding and copying data from all the source resource partitions to the newly associated region. Customers can also remove an existing region or take a region that was previously associated with their database account “offline”.

APIs for the following data models are supported with SDKs available in multiple languages,

- DocumentDB API (This API will be used in the article)
- MongoDB API
- Table API
- Graph (Gremlin) API