# Data abundance

Data can be structured, unstructured, or aggregated. For structured databases, data architects define the structure (schema) as they create the data storage in platform technologies such as Azure SQL Database and Azure SQL Data Warehouse. For unstructured (NoSQL) databases, each data element can have its own schema at query time. Data can be stored as a file in Azure Blob storage or as NoSQL data in Azure Cosmos DB or Azure HDInsight.

Data engineers must maintain data systems that are accurate, highly secure, and constantly available. The systems must comply with applicable regulations such as GDPR (General Data Protection Regulation) and industry standards such as PCI DSS (Payment Card Industry Data Security Standard). International companies might also have special data requirements that conform to regional norms such as the local language and date format. Data in these systems can be located anywhere. It can be on-premises or in the cloud, and it can be processed either in real time or in a batch.

Azure provides a comprehensive and rich set of data technologies that can store, transform, process, analyze, and visualize a variety of data formats in a secure way. As data formats evolve, Microsoft continually releases new technologies to the Azure platform. Azure customers can explore these new technologies in preview mode. Using the on-demand Azure subscription model, customers can minimize costs, paying only for what they consume and only when they need it.

### **Total cost of ownership**

The cost of operating an on-premises server system rarely aligns with the actual usage of the system. In cloud systems, the cost usually aligns more closely with the actual usage.

Organizations can reduce the costs of underutilization by adopting a best practice to provision production instances only after their developers are ready to deploy an application to production. Developers can use tools like the **Azure Cosmos DB emulator or the Azure Storage emulator** to develop and test cloud applications without incurring production costs.

The lift-and-shift strategy provides immediate benefits. These benefits include higher availability, lower operational costs, and the ability to transfer workloads from one datacenter to another. The disadvantage is that the application can't take advantage of the many features available within Azure.

**The lift-and-shift strategy** provides immediate benefits. These benefits include higher availability, lower operational costs, and the ability to transfer workloads from one datacenter to another. The disadvantage is that the application can't take advantage of the many features available within Azure.

Consider using the migration as an opportunity to transform your business practices by creating new versions of your applications and databases. Your rearchitected application can take advantage of Azure offerings such as Cognitive Services, Bot Service, and machine learning capabilities.

You load Transformed data into DWH using ETL process.

ELT is used for large repositories such as Cosmos D, ADL. Data transformation can be done as soon as the data load is complete.

# Use cases for the cloud

Web, Healthcare, IoT solutions.

Azure Databricks to accelerate big-data analytics and AI solutions.

Azure IOT hub, you design a data solution arechitecture that captures information from IoT devices so that the information can be analysed.

## **Structured data**

Relational systems typically use a querying language such as Transact-SQL (T-SQL).

## **Nonstructured data – NoSQL systems**

Examples of nonstructured data include binary, audio, and image files. Nonstructured data is stored in nonrelational systems, commonly called unstructured or NoSQL systems. In nonrelational systems, the data structure isn't defined at design time, and data is typically loaded in its raw format. The data structure is defined only when the data is read. The difference in the definition point gives you flexibility to use the same source data for different outputs. Nonrelational systems can also support semistructured data such as JSON file formats.

The open-source world offers four types of NoSQL databases:

1. **Key-value store**: Stores key-value pairs of data in a table structure.
2. **Document database**: Stores documents that are tagged with metadata to aid document searches.
3. **Graph database**: Finds relationships between data points by using a structure that's composed of vertices and edges.
4. **Column database**: Stores data based on columns rather than rows. Columns can be defined at the query's runtime, allowing flexibility in the data that's returned performantly.

Azure storage can be used when data is provisioned into a data platform technology such as Azure Data lake storage and HDInsight. But storage can also be used for standalone purpose by provisioning Azure Blob store wither as standard storage in the form of magnetic disk storage or as premium storage in the form of solid-state drives (SSDs).

***When to use Blob storage:*** you will need to store the data and don’t query, your cheapest option is to set up a storage account with a Blob store. Works well with images and unstructured data. it is the **cheapest way to store the data**.

Azure storage provides REST API and SDKs in various languages. Supported are: .NET, Java, Node.js, Python, PHP, Ruby and Go. Azure Storage also supports scripting in Azure PowerShell and the Azure CLI.

**Data Ingestion:** To ingest data into your system, use Azure Data Factory, Storage Explorer, the AzCopy tool, PowerShell, or Visual Studio. If you use the file Upload feature to import file sizes above 2 GB, use PowerShell or Visual Studio. AzCopy supports a max file size of 1 TB and automatically splits data files that exceeds 200 GB.

You cant query the data in Azure data store account as Blob Store. To query it move the data to other stores and or move to Data lake store.

Data Security in Azure Store: data is encrypted. Data is secured by using keys or shared access signatures.

Azure Resource Manager (ARM) provides a permissions model that uses role based access control (RBAC).

**Azure Data Lake Storage**

Azure Data lake Storage is a Hadoop-compatible data repository that can store any size or type of data. Gen2 has advantage of Azure Blob Storage, a hierarchical file system, and performance tuning that helps them process big-data analytics solutions. In Gen2, developers can access data through either the Blob API or the Data lake file API. Gen2 can also act as a storage layer for a wide range of compute platforms, including Azure Databricks, Hadoop, and Azure HDInsight, but data does not need to be loaded into compute platforms.

The compute aspects that sits above this storage can vary:

1. HDInSight
2. Hadoop
3. Cloudera
4. Azure Databriks
5. Hortonworks

## Key features

Here are the key features of Data Lake Storage:

* Unlimited scalability
* Hadoop compatibility
* Security support for both access control lists (ACLs)
* POSIX compliance
* An optimized Azure Blob File System (ABFS) driver that's designed for big-data analytics
* Zone-redundant storage
* Geo-redundant storage

Queries: in Gen 1 use U-SQL and In Gen 2, use the Azure Blob Storage API or the Azure Data Lake System (ADLS) API.

Supports Azure Active Directory Acitve Control Lists (AAD ACLs), Active Directory Secuirty Groups. RBAC is supported by Gen1. Built in Secuirty groups – ReadOnlyUsers, WriteAccess users, and Full access users.

**AZ Cosmos DB**

Azure Cosmos DB is a globally distributed, **multimodal database.** Can be deployed by using several API models

SQL API, MangoDB API(for semi structured data), Cassandra API(Wide columnar data), Gremlin API(graphs db), Table API. When you move the data to comsmos DB form these DB’s the applications that built using these data bases are continue to operate in cosmos DB.

Currently Azure Cosmos DB supports five – nines uptime (99.999 percent). It can support response times below 10 ms when it’s provisioned correctly.

Eg: T&S deals application is being operated in US and Europe. Data center is located in gemany. For US users performance of application is poor. So managedment decided to replicate the data to MS azure US datacentre to cosmos DB using SQL API– this solution improves the application performance for Americal users. The data can be stored in Germnany and replicated to US to improve throughput times.

Multi replication in Cosmos DB – helps in response times anywhere in the world for both read and writes.

Data ingestion into Cosmos DB

1. ADF 2) and APP which can write data to Cosmos DB through its API, 3) Upload JSON documents 4) directly edit the document.

Query the data from Cosmos DB: stored proc, triggers, and user defined functions (UDF). Java script query API, or in Data Explorer component you can use the graph visualization pane.

Data Security:

Cosmos DB supports: 1) Automatic data encryption 2) IP firewall config and access from vNets 3) User authentication is based on Tokens, and AAD – RBAC

AZ Cosomos DB – Meets many security compliance certifications, including

HIPAA(Health insurance portability and accountability Act)

FedRAMP: The Federal Risk and Authorization Management Program

SOCS

HITRUST: data protection standards development cert org.

**Azure SQL DB**

Is a relational DB and can support structured data such as relational and unstructured formats such as spatial and XML data.

**When to use** to scale up and scale down OLTP systems on demand. SQL DB is backed up by the Azure Service-level agreement. It can provide dynamic scalbility with no downtime, built-in intelligent optimization, and advanced security options.

These capabilities let you focus on rapid app development and on speeding up your time to market. You no longer have to devote precious time and resources to manage VMs and infrastructure. SQL DB is a PaaS service

You can ingest data through application integration from a wide range of developers SDKs. Beyond applications you can also ingest through Transact-SQL and ADF.

Data security: 1) Advanced Threat protection 2) SQL DB auditing 3) Data encryption 4) AAD authentication 5)Multifactor authentication 6) Compliance certification

**Azure SQL DWH**

It can process massive amounts of data and answer complex business questions.

Organizations that look for petabyte scale that doesn’t involve complex installations and configurations and provide business intelligence reports faster. SQL DWH, Data management services(DMS) coordinate and transports data between compute nodes as necessary. SQL DWH supports 2 types of distributed tables: hash and round-robin.

SQL DWH can also pause and resume the compute layer. That means you pay only for the computation you use.

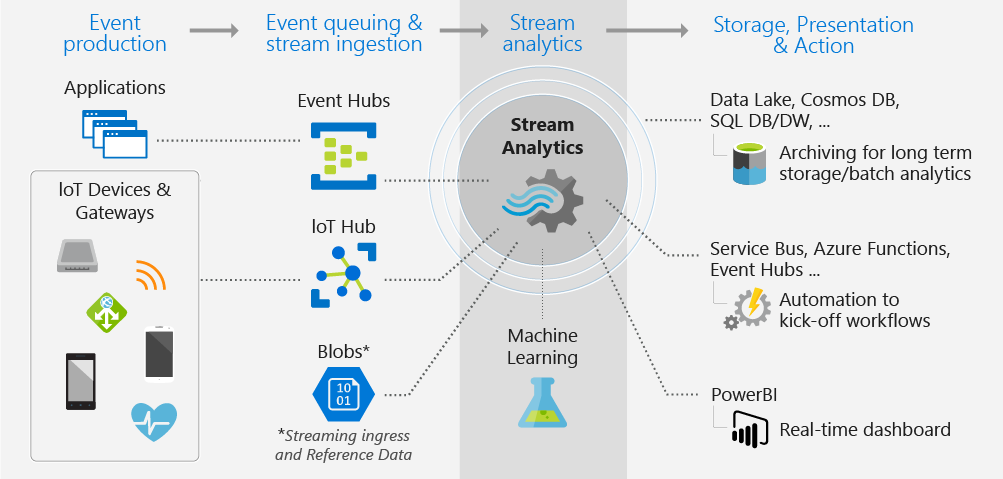
Data ingestion: use bulk copy tools such as: bcp and SQLBulkCopy API. PolyBase(for big data ingetion) also use ADF.

Secuirty: AAD, multifactor authentication, it supports security at the level of both columns and rows.

**Azure Steam Analytics**

Applications, Sensors, Monitoring devices, and gateways broadcast continuous event data known as **data streams**. Streaming data is high volume and has a lighter payload than no streaming systems.

Data engineers use Azure Stream analytics to process streaming data and respond to data anomalies in real time. You can use stream analytics for IoT monitoring, web logs, remote patient monitoring, and point of sale(POS) sytems.



**When to use Stream Analytics:**

If an org respond to data events in real time or analyse large batches of data in a continuous time-bound stream, Stream Analytics is a good solution. Organization must decide whether to work with streaming data or batch data.

In real time, data ingested from apps or IoT devices and gateways into an event hub or IoT hub. The event hub or IoT hub then streams the data into stream analytics for real-time analytics

Batch systems process groups of data that are stored in an Azure Blob store. They do this in a single job that runs at a predefined interval. Don’t use batch systems for a fraud-detection system must decline a questionable financial transaction in real time.

Azure Event Hubs provides big-data streaming services. Its designed for high data throughput, allowing customers to send billions of requests per day. Event Hubs provide authentication through a shared key.

**Queries**: Stream Analytics query language like SQL for queries and analytics.

**Data Security**: Stream analytics handles security at the transport layer between devices and IoT hub. Event hub uses a shared key to secure the data transfer.

**Azure HDInsight**

Azure HDInsight provides technologies to help you ingest, process and analyse big data. It supports batch processing, data warehousing, IoT, and data science.

## Key features

HDInsight is a low-cost cloud solution. It includes Apache Hadoop, Spark, Kafka, HBase, Storm, and Interactive Query.

* **Hadoop** includes Apache Hive, HBase, Spark, and Kafka. Hadoop stores data in a file system (HDFS). Spark stores data in memory. This difference in storage makes Spark about 100 times faster.
* **HBase** is a NoSQL database built on Hadoop. It's commonly used for search engines. HBase offers automatic failover.
* **Storm** is a distributed real-time streamlining analytics solution.
* **Kafka** is an open-source platform that's used to compose data pipelines. It offers message queue functionality, which allows users to publish or subscribe to real-time data streams.

Hive does ETL can be integrated with ADF

## Data processing

In Hadoop, use Java and Python to process big data. Mapper consumes and analyzes input data. It then emits tuples that Reducer can analyze. Reducer runs summary operations to create a smaller combined result set.

Spark processes streams by using Spark Streaming. For machine learning, use the 200 preloaded Anaconda libraries with Python. Use GraphX for graph computations.

Developers can remotely submit and monitor jobs from Spark. Storm supports common programming languages like Java, C#, and Python.

## Queries

In Hadoop supports Pig and HiveQL languages. In Spark, data engineers use Spark SQL.

## Data security

Hadoop supports encryption, Secure Shell (SSH), shared access signatures, and Azure Active Directory security.

More about Apache HBase on Apache Hadoop:

[Apache HBase](https://hbase.apache.org/) is an open-source, NoSQL database that is built on [Apache Hadoop](https://hadoop.apache.org/) and modeled after [Google BigTable](https://cloud.google.com/bigtable/). HBase provides random access and strong consistency for large amounts of unstructured and semistructured data in a schemaless database organized by column families.

From user perspective, HBase is similar to a database. Data is stored in the rows and columns of a table, and data within a row is grouped by column family. HBase is a schemaless database in the sense that neither the columns nor the type of data stored in them need to be defined before using them. The open-source code scales linearly to handle petabytes of data on thousands of nodes. It can rely on data redundancy, batch processing, and other features that are provided by distributed applications in the Hadoop ecosystem.

[Apache Phoenix](https://phoenix.apache.org/) is a SQL query engine for Apache HBase. It is accessed as a JDBC driver, and it enables querying and managing HBase tables by using SQL.

From user perspective, HBase is similar to a database. Data is stored in the rows and columns of a table, and data within a row is grouped by column family. HBase is a schemaless database in the sense that neither the columns nor the type of data stored in them need to be defined before using them. The open-source code scales linearly to handle petabytes of data on thousands of nodes. It can rely on data redundancy, batch processing, and other features that are provided by distributed applications in the Hadoop ecosystem.

HDInsight HBase is offered as a managed cluster that is integrated into the Azure environment. The clusters are configured to store data directly in [Azure Storage](https://docs.microsoft.com/en-us/azure/hdinsight/hdinsight-hadoop-use-blob-storage) which provides low latency and increased elasticity in performance and cost choices. This enables customers to build interactive websites that work with large datasets, to build services that store sensor and telemetry data from millions of end points, and to analyze this data with Hadoop jobs. HBase and Hadoop are good starting points for big data project in Azure; in particular, they can enable real-time applications to work with large datasets.

The HDInsight implementation leverages the scale-out architecture of HBase to provide automatic sharding of tables, strong consistency for reads and writes, and automatic failover. Performance is enhanced by in-memory caching for reads and high-throughput streaming for writes. HBase cluster can be created inside virtual network. For details, see [Create HDInsight clusters on Azure Virtual Network](https://docs.microsoft.com/en-us/azure/hdinsight/hbase/apache-hbase-provision-vnet).

## **Databricks**

Databricks is a serverless platform that's optimized for Azure. It provides one-click setup, streamlined workflows, and an interactive workspace for Spark-based applications.

Databricks adds capabilities to Apache Spark, including fully managed Spark clusters and an interactive workspace. You can use REST APIs to program clusters.

In Databricks notebooks you'll use familiar programming tools such as R, Python, Scala, and SQL. Role-based security in Azure Active Directory and Databricks provides enterprise-grade security.

## **Data Factory**

Data Factory is a cloud-integration service. It orchestrates the movement of data between various data stores.

As a data engineer, you can create data-driven workflows in the cloud to orchestrate and automate data movement and data transformation. Use Data Factory to create and schedule data-driven workflows (called pipelines) that can ingest data from data stores.

Data Factory processes and transforms data by using compute services such as Azure HDInsight, Hadoop, Spark, and Azure Machine Learning. Publish output data to data stores such as Azure SQL Data Warehouse so that business intelligence applications can consume the data. Ultimately, you use Data Factory to organize raw data into meaningful data stores and data lakes so your organization can make better business decisions.

## **Data Catalog**

Analysts, data scientists, developers, and others use Data Catalog to discover, understand, and consume data sources. Data Catalog features a crowdsourcing model of metadata and annotations. In this central location, an organization's users contribute their knowledge to build a community of data sources that are owned by the organization.

Data Catalog is a fully managed cloud service. Users discover and explore data sources, and they help the organization document information about their data sources.

data wrangling is the process for getting data, inget, transform, validate, and clean up data to meet business requirements.

Roles

**Data Engineer:** Primarily provision data stores. They make sure that massive amounts of data are securely and cost effectively extracted, loaded and transformed.

**AI Engineer:** AI engineer add the intelligent capabilities of vision, voice, language and knowledge to applications. To do this they use the cognitive services offerings that are available out of the box.

**Data Scientist:** Develop machine learning models and customize components for an AI engineer’s application.

**ETL:**

**Extract :** a) Define data source b) Define the data

**Transform**: Define the data transformation

**Load:** Define the destingation, Strart the job, Monitor the job.

ELT: Extraction Load and transformation. Technologies that can handle unstructured data at an unlimited scale. this chaged the the approach from ETL to ELT.

Phases of Data engineering project:

1. Source b) ingest c) Prepare d) analyse e) consume.

Contoso Health Network recently deployed IoT devices to its intensive care unit (ICU). Here are the goals of the project:

* Capture data on patient biometric monitoring in real time to help physicians treat their patients.
* Store the biometric data so that Contoso's research center can further analyze it in the future.
* Use Azure Machine Learning to understand which treatments improve the quality of care and reduce the likelihood that a patient will be readmitted to the hospital.
* Create a visualization of the data's history for Contoso's chief medical officer.

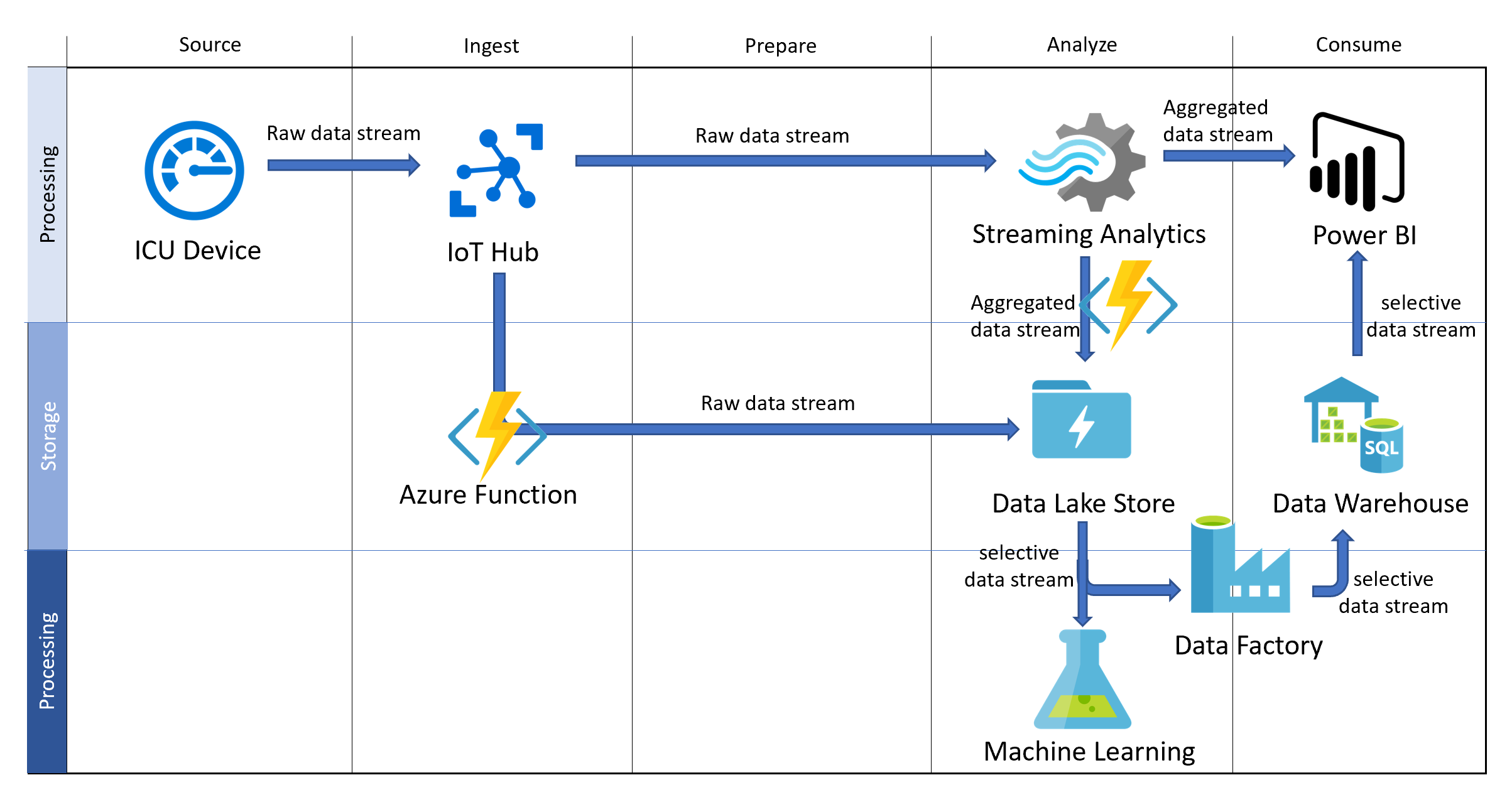
After reviewing the business case, Contoso's technical architect proposes the following technologies:

* **Azure IoT Hub** to capture real-time data from the ICU's IoT devices.
* **Azure Stream Analytics** to stream and enrich the IoT data, to create windows and aggregations, and to integrate Azure Machine Learning.
* **Azure Data Lake Storage Gen2** to store the biometric data at high speed.
* **Azure Data Factory** to perform the extract, load, transform, and load (ELTL) process to move the data from the data lake store to Azure SQL Data Warehouse.
* **Azure SQL Data Warehouse** to provide data warehousing services to support the chief medical officer's needs.
* **Power BI** to create the patient dashboard. Part of the dashboard will show real-time telemetry about the patient's condition. The other part will show the patient's recent history.
* **Azure Machine Learning** to process both raw and aggregated data. Researchers will use this to perform predictive analytics on patient readmittance.

ELTL 🡪 Extract Load Transform and Load.

The holistic workflow:

1. Set up Azure IoT Hub to capture data form the ICU IoT devices
2. Connect Azure IoT Hub to Azure Stream Analytics
3. Set up Azure functions to store the Azure Stream Analytics aggregates in Azure Data Lake Storage Gen2.
4. Use ADF to load data from ADL to SQL DWH
5. In parallel connect the azure ML service to ADL to perform predictive analytics
6. Connect power BI to stream analytics to pull the real-time aggregates for the patient data. and connect SQL DWH to pull the historical data to create a combined dashboard.



**Azure Data Store**

No SQL Data bases – Cosmos DB can store – Structured, semi structured and unstructured data.

It stores: 1) Document databases 2) Key value stores 3) Column family stores 4) Graph databases.

**Structured data:** Forcing a consistent structure also means evolution of the data is more difficult as each record has to be updated to fit to the new structure.

**Semi-structured data** is less organized than structured data for eg: key-value pairs. Semi-structured data is also referred to as non-relational or NoSQL data. the structure of the data is defined by **serialization language.**

Serialization languages:

a) **XML**

<Person Age="23"> 🡪 attribute

<FirstName>John</FirstName> 🡪 element

<LastName>Smith</LastName>

<Hobbies>

<Hobby Type="Sports">Golf</Hobby> 🡪 child element

<Hobby Type="Leisure">Reading</Hobby>

<Hobby Type="Leisure">Guitar</Hobby>

</Hobbies>

</Person>

XML is flexible and can express complex data easily. However it tends to be larger to store, process or pass over a network.

1. **JSON** : Javascript object notation, has a lightwight specification and relies on curly braces to indicate data structure. JSON is frequently used by web services to return data.

{

"firstName": "John",

"lastName": "Doe",

"age": "23",

"hobbies": [

{ "type": "Sports", "value": "Golf" },

{ "type": "Leisure", "value": "Reading" },

{ "type": "Leisure", "value": "Guitar" }

]

}

🡪 key/value pair model. JSON is Very popular for webdevelopment

c) YAML (Ain’t Markup Language): is relatively new data language that’s growing quickly in popularity in part due to its human-friendliness.

firstName: John

lastName: Doe

age: 23

hobbies:

- type: Sports

value: Golf

- type: Leisure

value: Reading

- type: Leisure

value: Guitar

NoSQL: growing relational DB results in complex queries and poor performance. NoSQL can be easy to use and scalability.

NoSQL data models: Key-Value, graph and document.

**Key-Vlaue data bases** store data in key-value pairs. Now query languages – Get/Put/deletes

Graph Databases: they use nodes, links of relation ships

Document database – stores the data in JSON or XML format – you can store any data or structure.

**NoSQL db: MS azure cosmos DB**

**Unstructured data:**  organization of unstructured data is ambiguous. Unstructured data is often delivered in files, such as photos or videos.

Examples of unstructured data: 1) Media files: a) Photos b) Videos c) audio d) word doc e) text file f) log file.

Examples: Product catalog data: semi structured. Photos and Videos: Unstructured. Business Data (Sales performance data): Structured.

Case study: Retail business:

* Will you be doing simple lookups using an ID?
* Do you need to query the database for one or more fields?
* How many create, update, and delete operations do you expect?
* Do you need to run complex analytical queries?
* How quickly do these operations need to complete?

When deciding what storage solution to use, think about how your data will be used. How often will your data be accessed? Is your data read-only? Does query time matter? The answers to these questions will help you decide on the best storage solution for your data.

Transactions are often defined by a set of four requirements, referred to as ACID guarantees. ACID stands for **A**tomicity, **C**onsistency, **I**solation, and **D**urability:

* **Atomicity** means a transaction must execute exactly once and must be atomic; either all of the work is done, or none of it is. Operations within a transaction usually share a common intent and are interdependent.
* **Consistency** ensures that the data is consistent both before and after the transaction.
* **Isolation** ensures that one transaction is not impacted by another transaction.
* **Durability** means that the changes made due to the transaction are permanently saved in the system. Committed data is saved by the system so that even in the event of a failure and system restart, the data is available in its correct state.

When a database offers ACID guarantees, these principles are applied to any transactions in a consistent manner.

Requirement: Semi Structured data, High throughput and low latency, high number of read/write operations 🡪 **go for Cosmos DB, is a recommended DB service. It is ACID-Compliant. Global replication of data. you can determine trade offs to make between consistency, availability, latency, and throughput. You can scale up to handle higher customer demand during peak shopping times or scale down during times to conserve cost.**

**Cosomos DB is better choice for highly unstructured and variable data where you cant predict which properties are to be indexed.**

|  |  |
| --- | --- |
| **Azure SQL DB** | **Azure Cosmos DB** |
| Can have structured data in the columns and semi structured data stored as JSON columns that can be easily extended. | Can have both structured and unstructured data |
| It provides little benefit if the structre of the data changes in different entities and cant pre-define a set of common properties that repeated in the most of the entries. | Best DB when there are changes to the data new entities are being added or delted… |
| Requires Explicit definition of properties to beindexed in semi-structrured documents. | Every property in the JSON document is indexed by default |
|  |  |

**Photos and Videos:**

* Only need to be retrieved by ID
* Customers require a high number of read operations with low latency
* Creates and updates will be somewhat infrequent and can have higher latency than read operations.

**Latency & Throughput:** Retrievals by ID need to support low latency and high throughput. Creates and updates can have higher latency than read operations.

Recommended Service: **Azure Blob storage**

Azure Blob Storage also works with CDN – Content delivery network by caching the most frequently used content and storing it on edge servers. Azure CDN reduces latency in service up those images to your users.

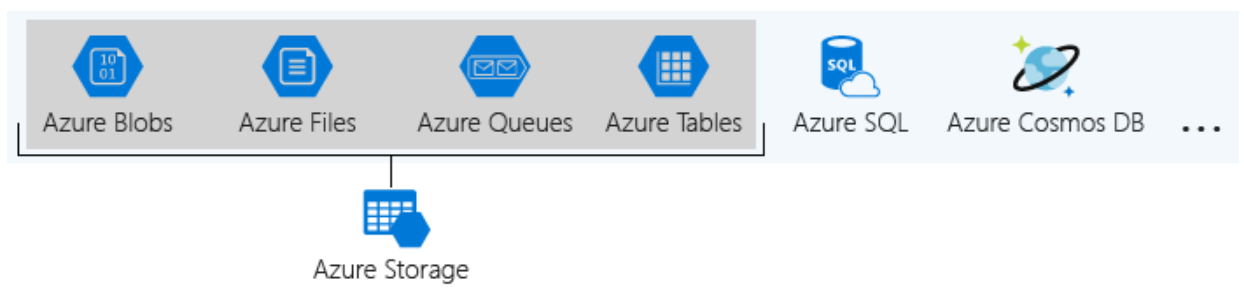
By using Azure Blob storage, you can also move images from the hot storage tier to the cool or archive storage tier, to reduce costs and focus throughput on the most frequently viewed images and videos.

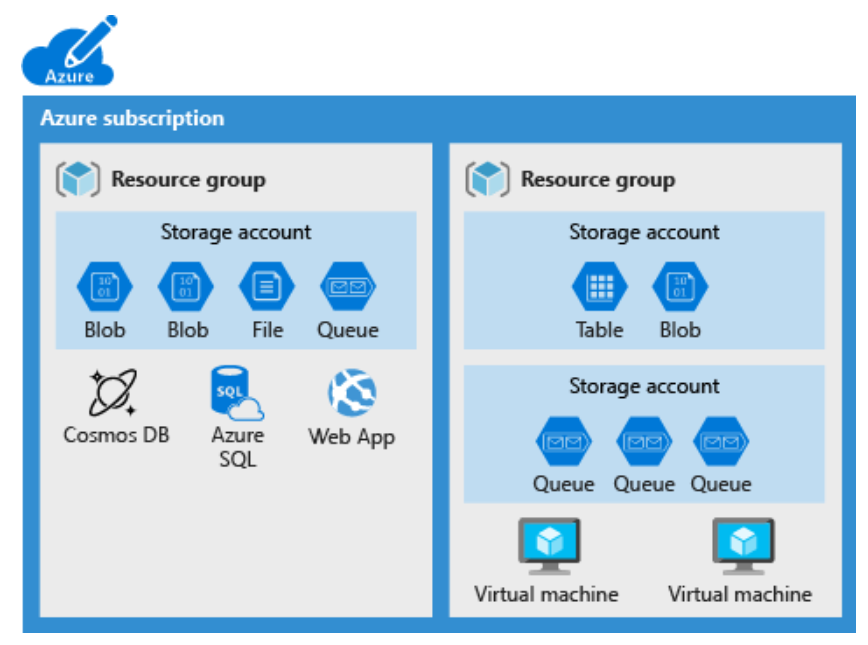
You can upload your images to Azure app service, so that the same server that is running your app is serving up your images. This solution will work if your solution didn’t have many files, but if you have lots of files, and global audience you will get more performant results by using Azure Blob storage with CDN.

Business Data: Business analysts require to query the data across multiple data bases which SQL DWH cant have multi DB. Stream analytics is best for streaming and real time data.

Azure analysis services – you can use a direct query to the SQL DB.

**Summary:** Each type of data has different storage requirements, and it's your job to figure out which solution is best. Always consider the type of data, the operations required, expected latency, and the need for transactional support.





**Note:** Azure Data services like SQL DB, Cosmos DB, DWH etc… cant be included azure storage account. Various Azure services can be bundled together in a resource group.

**A storage account is an Azure resource and is included in a Resource group.**

**Performance**

**Standard(magnetic disk drive)** allows you to have any data service (Blob, File, Queue, Table) and uses magnetic disk drives. **Premium** limits you to one specific type of blob called a page blob and uses solid-state drives (SSD) for storage.

**Replication**

The minimum replication/redundancy (automatically maintain a copies of your data in data centre associated with your storage account. This call Locally-redundant storage (LRS). It cant protect data center disaster/failure. You can upgrade to one of the other options such as Geo-Redundant storage(GRS) to get replication at different data centers across the world.

**Access tier** controls how quickly you will be able to access the blobs in this storage account. this restricted to only blobs. Hot is quicker than cool

**Secure transfer required:** enabled requires HTTPs while disabled allows HTTP.

**Virtual Networks:** enables data access within vNets.

A typical strategy is to start with an analysis of your data and create partitions that share characteristics like location, billing, and replication strategy, and then create one storage account for each partition.

Storage account:

Name: should be unique globally in Azure. Only lowercase letter – use letters and digits and between 3 and 24.

Deploymet -> Resource manger -> Resource group (MS recommended)

Account Kind:

StorageV2 (general purpose V2) – it supports all kind of storage types (MS recommended)

Storgage (general purpose V1) – a legacy kind that supports all storage types but may not support all features.

Blob storage: only supports block blobs and append blobs

Creation of storage account can be done by using one the below tools:

* Azure Portal
* Azure CLI (Command-line interface)
* Azure PowerShell
* Management client libraries

Apart from portal – other tools supports in automation. Azure CLI and power shell allows you to create scripts, where as management libraries allow you to incorporate the creation into a client app. One time creation – portal is convenient.

Secure transfer required:

If this option is enabled, it will enforce some additional restrictions. Azure files service connections without encryption will fail, including scenarios using SMB 2.1 or 3.0 on Linux. Because Azure storage doesn’t support SSL for custom domain names, this option cannot be used with a custom domain name.

Deleting resource group will delete all the resource/services permanently.

# Connect an app to Azure Storage

A single Azure subscription can host up to 200 storage accounts, each of which can hold 500 TB of data. with special approval it can reach to 250 storage accounts in a subscription which pushes your max storage up to **125 Petabytes!**

* **Blobs**: A massively scalable object store for text and binary data.
* **Files**: Managed file shares for cloud or on-premises deployments.
* **Queues**: A messaging store for reliable messaging between application components.
* **Tables**: A NoSQL store for schemaless storage of structured data. This service has been replaced by Azure Cosmos DB and will not be discussed here.

**Blob Storage**

* Serving images or documents directly to a browser, including full static websites.
* Storing files for distributed access.
* Streaming video and audio.
* Storing data for backup and restoration, disaster recovery, and archiving.
* Storing data for analysis by an on-premises or Azure-hosted service.

| **Blob type** | **Description** |
| --- | --- |
| **Block blobs** | Block blobs are used to hold text or binary files up to ~5 TB (50,000 blocks of 100 MB) in size. The primary use case for block blobs is the storage of files that are read from beginning to end,  such as media files or image files for websites. They are named block blobs because files larger than 100 MB must be uploaded as small blocks, which are then consolidated (or committed)  into the final blob. |
| **Page blobs** | Page blobs are used to hold random-access files up to 8 TB in size. Page blobs are used primarily as the backing storage for the VHDs used to provide durable disks for  Azure Virtual Machines (Azure VMs). They are named page blobs because they provide random read/write access to 512-byte pages. |
| **Append blobs** | Append blobs are made up of blocks like block blobs, but they are optimized for append operations. These are frequently used for logging information from one or more sources into the same blob.  For example, you might write all of your trace logging to the same append blob for an application running on multiple VMs. A single append blob can be up to 195 GB. |

Azure Storage supports three kinds of blobs:

### **Files**

Azure Files enables you to set up highly available network file shares that can be accessed by using the standard Server Message Block (SMB) protocol. This means that multiple VMs can share the same files with both read and write access. You can also read the files using the REST interface or the storage client libraries. You can also associate a unique URL to any file to allow fine-grained access to a private file for a set period of time. File shares can be used for many common scenarios:

* Storing shared configuration files for VMs, tools, or utilities so that everyone is using the same version.
* Log files such as diagnostics, metrics, and crash dumps.
* Shared data between on-premises applications and Azure VMs to allow migration of apps to the cloud over a period of time.

### **Queues**

The Azure Queue service is used to store and retrieve messages. Queue messages can be up to 64 KB in size, and a queue can contain millions of messages. Queues are generally used to store lists of messages to be processed asynchronously.

VM Disks(Page blob?) can be stored in the storage account.

If your development language is C# then then application will be based on .Net. if language is JavaScript the app is based on Node.js

Eg: JavaScript, Node.js

Do the code in Azure CLI (Command Line interface - could shell editor)

Eg code: JavaScript +Node.js C# + .Net

Create a package: *mkdir PhotoSharingApp (cd ..)* dotnet new console *--name PhotoSharingApp*

Change to folder: *cd* *PhotoSharingApp cd* *PhotoSharingApp*

Create a new source file index.js this is where code will go

*Touch index.js dotnet run*

Open the index.js file with an editor in could shell type *code .*

Past below code:

*#!/usr/bin/env node*

*function main() {*

*console.log('Hello, World!');*

*}*

*main();*

save the file

run the app: *node index.js*

Create Storage Account using Azure CLI

*az storage account create \*

*--resource-group* ***learn-e52c6058-af0e-4d0a-9aaa-38d67bd3953b \***

*--kind* ***StorageV2 \***

*--sku* ***Standard\_LRS*** *\*

*--access-tier* ***Cool \***

*--name* ***sandhri03***

# Interact with the Azure Storage APIs#

* **Blobs** for unstructured data such as binary and text files.
* **Queues** for persistent messaging.
* **Tables** for structured storage of key/values.
* **Files** for traditional SMB file shares.

Using REST API

GET https://[url-for-service-account]/?comp=list&include=metadata

Client libraries can save significant amount of work for application developers because the API is tested, and it often provides nicer wrappers around the data models sent and received by the Rest API.

* .NET
* Java
* Python
* Node.js
* Go

For example to retrieve the same list of blobs in C# below is the code snippet:

CloudBlobDirectory directory = ...;

foreach (IEnumerable<IListBlobItem> blob in directory.ListBlobs(

useFlatBlobListing: true,

blobListingDetails: BlobListingDetails.All))

{

// Work with blob item .. could be page blob, block blob, etc.

}

Javascript:

const containerName = "...";

const blobService = storage.createBlobService();

blobService.listBlobsSegmented(containerName, null, function (error, results) {

if (results) {

for (var i = 0, blob; blob = results.entries[i]; i++) {

// Work with blob item .. could be page blob, block blob, etc.

}

}

});

Java script:

Add azure-storage package to the application and use –save option so it persists to packages.json

**npm install azure-storage --save**

**node index.js (to make sure that every this ready to go)**

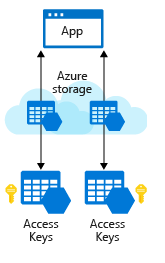
# Connect to your Azure storage account

**To connect to storage account your app will need two pieces of data:**

* 1. **An access key 2) The REST API endpoint**

**Security access keys:**

Each storage account has two unique access keys that are used to secure the storage account. If your app needs to connect to multiple storage accounts, then your app will require an access key for each storage account.



**In addition to access key for authentication to storage accounts, your app will need to know the storage service endpoints to issue the REST requests.**

The REST endpoint is a combination of your storage account *name*, the data type, and a known domain. For example:

| **Data type** | **Example endpoint** |
| --- | --- |
| Blobs | https://sandhri.blob.core.windows.net/ |
| Queues | https://[name].queue.core.windows.net/ |
| Table | https://[name].table.core.windows.net/ |
| Files | https://[name].file.core.windows.net/ |

**Connection string syntax**

DefaultEndpointsProtocol=https;AccountName={your-storage};

AccountKey={your-access-key};

EndpointSuffix=core.windows.net

Security: Typically, storage account connectivity information(access keys) is stored within an environment variable, database, or configuration file.

Access keys can be stored in Azure key Valult to store the access key for you. Key Vaults include support to synchronize directly to the storage Account and automatically rotate the keys periodically. Using keyvault provides an additional layer of security, so your app never has to work directly with an access key.

### Access keys – gives full access to the users on the storage account. using **Shared access signatures (SAS) you can grant limited access.**

**Add Azure Storage configuration to your app**

## Create a .env configuration file

**Touch .env**

Code . to open the shell editor

Open .env file and save AZURE\_STORAGE\_CONNECTION\_STRING=<value>

Now run

az storage account show-connection-string \

--resource-group Ravi\_RG \

--query connectionString \

--name ravistorageac01

Below is the connection string

"DefaultEndpointsProtocol=https;EndpointSuffix=core.windows.net;AccountName=ravistorageac01;AccountKey=2LSetTLY5b196bSydOX9F9iOlCh3egEVOddSmSC4F46klpC4wHQE2Ufw8kR2kyOZNTuAkC4WsdCuCbWush7oUA==” (minus quotes)

Copy this connection string in the .env file.

Note: Node.js apps can include support to read from the **.env** file by adding the **dotenv** package.

npm install dotenv –save

#!/usr/bin/env node

require('dotenv').config();

// ... more code follows

const util = require('util');

const storage = require('azure-storage');

const blobService = storage.createBlobService();

const createContainerAsync = util.promisify(blobService.createContainerIfNotExists).bind(blobService);

const uploadBlobAsync = util.promisify(blobService.createBlockBlobFromLocalFile).bind(blobService);

async function main() {

try {

// This makes an actual service call to the Azure Storage service.

// Unless this call fails, the container will have been created.

await createContainerAsync(containerName);

// This transfers data in the file to the blob on the service.

var uploadResult = await uploadBlobAsync(containerName, "myphoto", "photo.png");

if (uploadResult) {

console.log("blob uploaded");

}

}

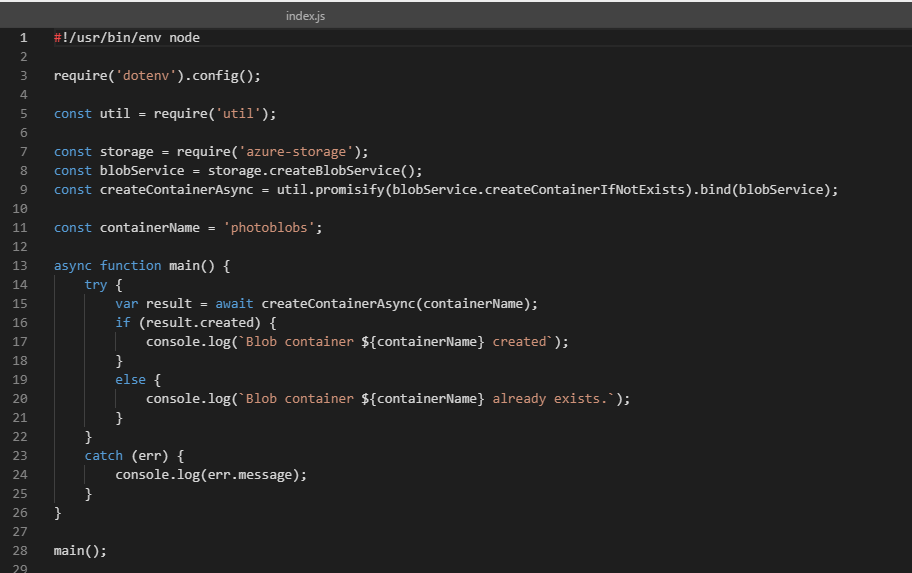
catch (err) {

console.log(err.message);

}

}

main();





Note: make sure that azure connection string is configured before you run index.js

**Data Storage Security**

**Cloud provides:**

* Protect the data at rest
* Protect the data in transit
* Support browser cross-domain access
* Control who can access data
* Audit storage access

**Encryption at rest:** All data written to Azure Storage is automatically encrypted by Storage Service Encryption (SSE) with a 256-bit Advanced Encryption Standard (AES) cipher. SSE automatically encrypts data when writing it to Azure Storage. When you read data from Azure Storage, Azure Storage decrypts the data before returning it. This process incurs no additional charges and doesn't degrade performance. It can't be disabled.

For virtual machines (VMs), Azure lets you encrypt virtual hard disks (VHDs) by using Azure Disk Encryption. This encryption uses BitLocker for Windows images, and it uses dm-crypt for Linux.

Azure Key Vault stores the keys automatically to help you control and manage the disk-encryption keys and secrets. So even if someone gets access to the VHD image and downloads it, they can't access the data on the VHD.

## **Encryption in transit**

Keep your data secure by enabling transport-level security between Azure and the client. Always use HTTPS to secure communication over the public internet. When you call the REST APIs to access objects in storage accounts, you can enforce the use of HTTPS by requiring [secure transfer](https://docs.microsoft.com/azure/storage/storage-require-secure-transfer) for the storage account. After you enable secure transfer, connections that use HTTP will be refused. This flag will also enforce secure transfer over SMB by requiring SMB 3.0 for all file share mounts.

## **CORS support**

Contoso stores several website asset types in Azure Storage. These types include images and videos. To secure browser apps, Contoso locks GET requests down to specific domains.

Azure Storage supports cross-domain access through cross-origin resource sharing (CORS). CORS uses HTTP headers so that a web application at one domain can access resources from a server at a different domain. By using CORS, web apps ensure that they load only authorized content from authorized sources.

CORS support is an optional flag you can enable on Storage accounts. The flag adds the appropriate headers when you use HTTP GET requests to retrieve resources from the Storage account.

## **Role-based access control**

To access data in a storage account, the client makes a request over HTTP or HTTPS. Every request to a secure resource must be authorized. The service ensures that the client has the permissions required to access the data. You can choose from several access options. Arguably, the most flexible option is role-based access.

Azure Storage supports Azure Active Directory and role-based access control (RBAC) for both resource management and data operations. To security principals, you can assign RBAC roles that are scoped to the storage account. Use Active Directory to authorize resource management operations, such as configuration. Active Directory is supported for data operations on Blob and Queue storage.

To a security principal or a managed identity for Azure resources, you can assign RBAC roles that are scoped to a subscription, a resource group, a storage account, or an individual container or queue.

## **Auditing access**

Auditing is another part of controlling access. You can audit Azure Storage access by using the built-in Storage Analytics service.

Storage Analytics logs every operation in real time, and you can search the Storage Analytics logs for specific requests. Filter based on the authentication mechanism, the success of the operation, or the resource that was accessed.

**Shared access signatures:**

For untrusted clients, use a *shared access signature* (SAS). A shared access signature is a string that contains a security token that can be attached to a URI. Use a shared access signature to delegate access to storage objects and specify constraints, such as the permissions and the time range of access.

Currently, Advanced Threat Protection for Azure Storage is available for the Blob service. Security alerts are integrated with Azure Security Center. The alerts are sent by email to subscription admins.

**Azure Data Lake Storage Gen2 provides** a first-class data lake solution that allows enterprises to pull together their data. It's built on Azure Blob storage, so it inherits all of the security features we've reviewed in this module.

Along with role-based access control (RBAC), Azure Data Lake Storage Gen2 provides access control lists (ACLs) that are POSIX-compliant and that restrict access to only authorized users, groups, or service principals. It applies restrictions in a way that's flexible, fine-grained, and manageable. Azure Data Lake Storage Gen2 authenticates through Azure Active Directory OAuth 2.0 bearer tokens. This allows for flexible authentication schemes, including federation with Azure AD Connect and multifactor authentication that provides stronger protection than just passwords.

**Blobs give you file storage in the cloud and an API that lets you build apps to access the data.**

* + **Unstructured data**
  + **Unqueriable data**
  + **Unindexable data.**

**It can be used in the combination of database.**

**For example employee data and profile picture url can be in SQL db where as actual photos are stored in azure blob storage.**

Many Azure components use blobs behind the scenes. For example, Azure Cloud Shell stores your files and configuration in blobs, and Azure Virtual Machines uses blobs for hard-disk storage.

in Blob storage, every blob lives inside a blob container. You can store an unlimited number of blobs in a container and an unlimited number of containers in a storage account. Containers are "flat" — they can only store blobs, not other containers.

Blobs in a container configured for public access can be downloaded without any kind of authentication or auditing by anyone who knows their storage URLs. Never put blob data in a public container that you don't intend to share publicly.

In blob container you can’t create sub containers but can give the naming conventions in such a way that they resembles folder structure. virtual directories

## **Initialize the Blob storage object model**

CloudStorageAccount storageAccount = CloudStorageAccount.Parse(connectionString); // or TryParse()

CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

CloudBlobContainer container = blobClient.GetContainerReference(containerName);

How to get a connection string:

$ az storage account show-connection-string --name ravistorageac01

**My first app:** <https://ravistorageapp07.azurewebsites.net/>

# Work with relational data in Azure

# Provision an Azure SQL database to store application data

AZ SQL DB : Azure provides Platform as a Service (PaaS) services to help you manage all kinds of data, from highly structured relational data to unstructured data.

DTU stands for **Database Transaction Unit**, and is a combined measure of compute, storage, and IO resources. Think of the DTU model as a simple, preconfigured purchase option. DTU and vCore are used for purchasing models.

While the ***DTU*** model provides fixed combinations of compute, storage, and IO resources, the **vCore** model enables you to configure resources independently. For example, with the vCore model you can increase storage capacity but keep the existing amount of compute and IO throughput.

SQL elastic pools relate to **eDTUs.** They enable you to buy a set of compute and storage resources that are **shared among all the databases in the pool**. Each database can use the resources they need, within the limits you set, depending on current load.

**Collation** helps you define sorting rules when case sensitivity, accent marks, and other language characteristics are important.

default DB collation, **SQL\_Latin1\_General\_CP1\_CI\_AS**, means.

* **Latin1\_General** refers to the family of Western European languages.
* **CP1** refers to code page 1252, a popular character encoding of the Latin alphabet.
* **CI** means that comparisons are case insensitive. For example, "HELLO" compares equally to "hello".
* **AS** means that comparisons are accent sensitive. For example, "résumé" doesn't compare equally to "resume".

SQL server is logical server (it is **not** VM machine), connection string credentials for SQL DB is based on the server. It is a Azure PaaS.

**Server name: ravisqldbserv01.database.windows.net**

**DB: Ravi\_Logistics**

**User name: ravilogistics**

**Password: Ravi@Logistics**

**Location: Central US**

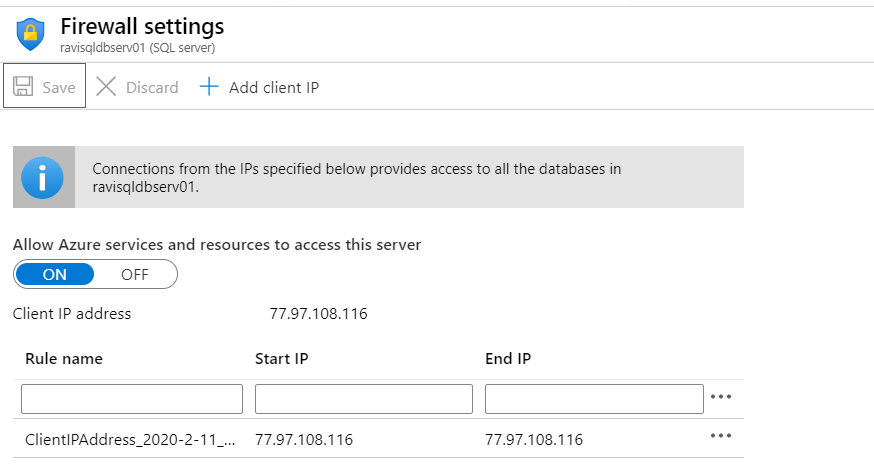
**Configure Database:**

DTUs based (no of DTUs): Basic, Standard, Premium – if you increase no of DTUs – cost will increase. Premium usually used of IO-intensive workloads.

vCore based Database(no of : vCores and DB size – based on this cost varies): a) General purpose, b)Hyperscale, c)Business Critical. (you can configure comupation power in vCore)

Database can be created: blank DB, Backup(from other DB) or Sample.

Set Firewall settings 🡪 Add client IP. 1) Rule Name 2) Start IP 3) End IP.



SQL DB is managed using Visual Studio, SQL server management Studio, Azure Data studio.

CRUD operations 🡪 Create, Read, update and delete.

In Azure Command line interface AZ-CLI, you can execute data base commands using **slqcmd** on your sqlserver. **az** utility to execute any command on azure resources. **jq** utility is a command-line JSON parser. You can pipe output from az commands to this tool to extract important fields from JSON output.

Eg: az sql db list | jq '[.[] | {name: .name}]' 🡪 it will list out only the data bases. Big block of json is piped through jq command line .

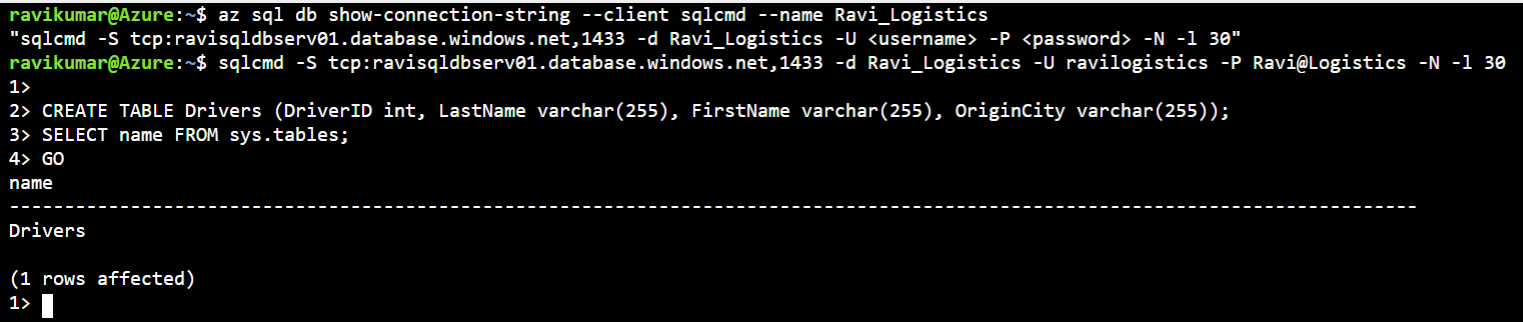
How to start interactive session in AZ CLI.

sqlcmd -S tcp:ravisqldbserv01.database.windows.net,1433 -d Ravi\_Logistics -U ravilogistics -P Ravi@Logistics -N -l 30

SELECT name FROM sys.tables;

**GO 🡪** use GO to run T-SQL statements.

Working screen



Try to connet using SQL server management studio, if you are not vpn, get ip address and add it to DB firewall settings (add ip).

Use exit; to come out of sql interactive mode in cli

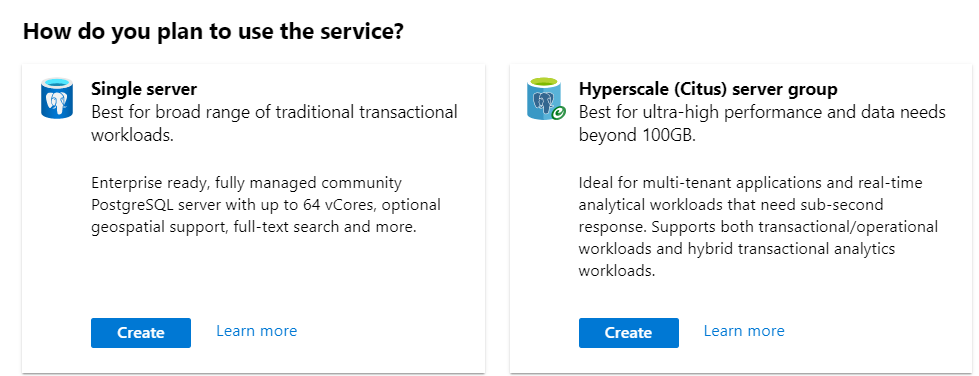
## **The Azure Database for PostgreSQL**

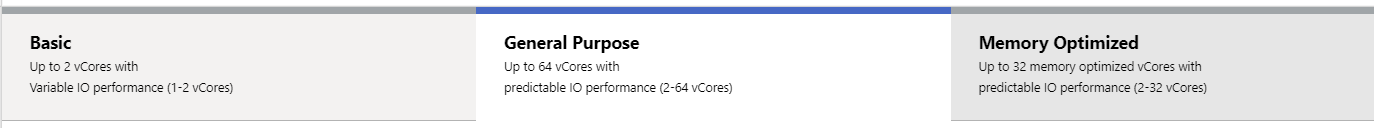
Azure Database for PostgreSQL is a relational database service in the Microsoft cloud. The server software is based on the community version of the open-source PostgreSQL database engine. Your familiarity with tools and expertise with PostgreSQL is applicable when using Azure Database for PostgreSQL.

Moreover, Azure Database for PostgreSQL delivers the following benefits:

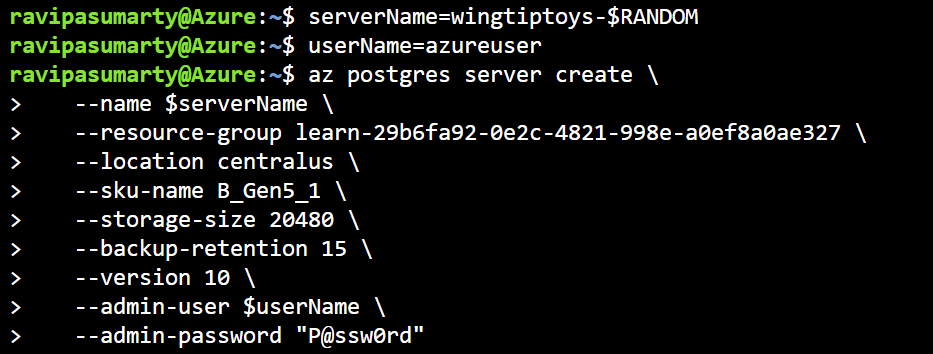
* Built-in high availability compared to on-premises resources. There is no additional configuration, replication, or cost required to make sure your applications are always available.
* Simple and flexible pricing. You have predictable performance based on a selected pricing tier choice that includes software patching, automatic backups, monitoring, and security.
* Scale up or down as needed within seconds. You can scale compute or storage independently as needed, to make sure you adapt your service to match usage.
* Adjustable automatic backups and point-in-time-restore for up to 35 days.
* Enterprise-grade security and compliance to protect sensitive data at-rest and in-motion that covers data encryption on disk and SSL encryption between client and server communication.

**Create Postgre SQL server**





User Azure CLI.



Using psql requires a successful connection to a PostgreSQL server. There are a number of command-line parameters available for use when working with psql.

* --host - The host to which you'd like to connect.
* --username - The user name/ID with which to connect.
* --dbname - The name of the database to connect to.

**SQL DB Elastic pools:** SQL database elastic pools are a cost-effective service that can manage and scale multiple Azure SQL databases that have varying and unpredictable resource requirements.

SQL elastic pools are ideal when you have several SQL databases that have a low average utilization, but have infrequent, high utilization spikes. In this scenario, you can allocate enough capacity in the pool to manage the spikes for the group, but the total resources can be less than the sum of all of the peak demand of all of the databases. Since the spikes are infrequent, a spike from one database will be unlikely to impact the capacity of the other databases in the pool.

Total demand = 1.5\*Elastic pool is cost effective approach.

**Secure SQL DB:**

* Control network access to your Azure SQL Database using firewall rules
* Control user access to your Azure SQL Database using authentication and authorization
* Protect your data in transit and at rest
* Audit and monitor your Azure SQL Database for access violations
* **Server-level firewall rules**
  + Allow access to Azure services
  + IP address rules
  + Virtual network rules
* **Database-level firewall rules**
  + IP address rules

SQL Data Warehouse only supports server-level IP firewall rules, and not database-level IP firewall rules.

**Authentication**

Authentication is the process of verifying an identity. This identity could be a user, a service running on a system, or a system itself (such as a virtual machine). Through the process of authentication, we ensure that the person or system is who they claim to be. SQL Database supports two types of authentication: SQL authentication and Azure Active Directory authentication.

**SQL Authentication: by creating user id/password at logical server and DB level.**

**AAD:**

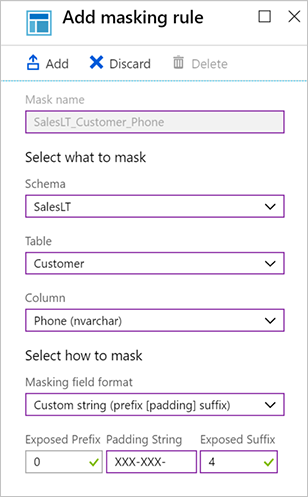
### **Azure Active Directory authentication**

This authentication method uses identities managed by Azure Active Directory (AD) and is supported for managed and integrated domains. Use Azure AD authentication (integrated security) whenever possible. With Azure AD authentication, you can centrally manage the identities of database users and other Microsoft services in one central location. Central ID management provides a single place to manage database users and simplifies permission management. If you want to use Azure AD authentication, you must create another server admin called the "Azure AD admin," which is allowed to administer Azure AD users and groups. This admin can also perform all operations that a regular server admin can.

Authorization refers to what an identity can do within an Azure SQL Database. This is controlled by permissions granted directly to the user account and/or database role memberships. A database role is used to group permissions together to ease administration, and a user is added to a role to be granted the permissions the role has.

## Dynamic data masking rules:

Add masking rules to a table and a column



sqlcmd -S tcp:serverNNNN.database.windows.net,1433 -d marketplaceDb -U 'ApplicationUser' -P '[password]' -N -l 30

Azure SQL Database auditing

By enabling auditing, operations that occur on the database are stored for later inspection or to have automated tools analyze them.

it's recommended you enable only server-level blob auditing and leave the database-level auditing disabled for all databases. Unless one database is to audited which is different from rest of the dbs

**for Auditing information can be stored blob storage or V1 or V2.**

Audit Log destination can be: Storage or Log analytics or Event Hub

**Advanced Data Security for Azure SQL Database**

**Data discovery & classification**

**Vulnerability assessment**

**Advanced Threat Protection**

Transparent data encryption: it encrypts your databases, backups, and logs at rest without any changes to your applicaton.

Which of the following is the most efficient way to secure a database to allow only access from a VNet while restricting access from the internet?

**A server-level virtual network rule will allow you to allow connectivity from specific Azure VNet subnets, and will block access from the internet. This is the most efficient manner to secure this configuration.**

Import data into SQL DB

* SQL Server Integration Services (SSIS)
* The SQL *BULK INSERT* statement
* The Bulk Copy Program (bcp) utility

Bcp can be run in the command line

bcp <database>.dbo.mytable format nul -c -f mytable.fmt -t, -S <server>.database.windows.net -U <username> -P <password>

how to download data files and application specific code from git

git clone https://github.com/MicrosoftDocs/mslearn-develop-app-that-queries-azure-sql education

mv ~/education/data ~/educationdata 🡪 move the content

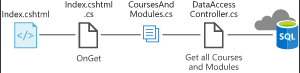
cd ~/educationdata

ls 🡪 to list data

cat courses.csv 🡪 to view a file data

## **Connect an ASP.NET application to Azure SQL Database**

Using System.Data.SqlClient .NET apps can connect.



Once code changes are done – code has to be built and deploy.

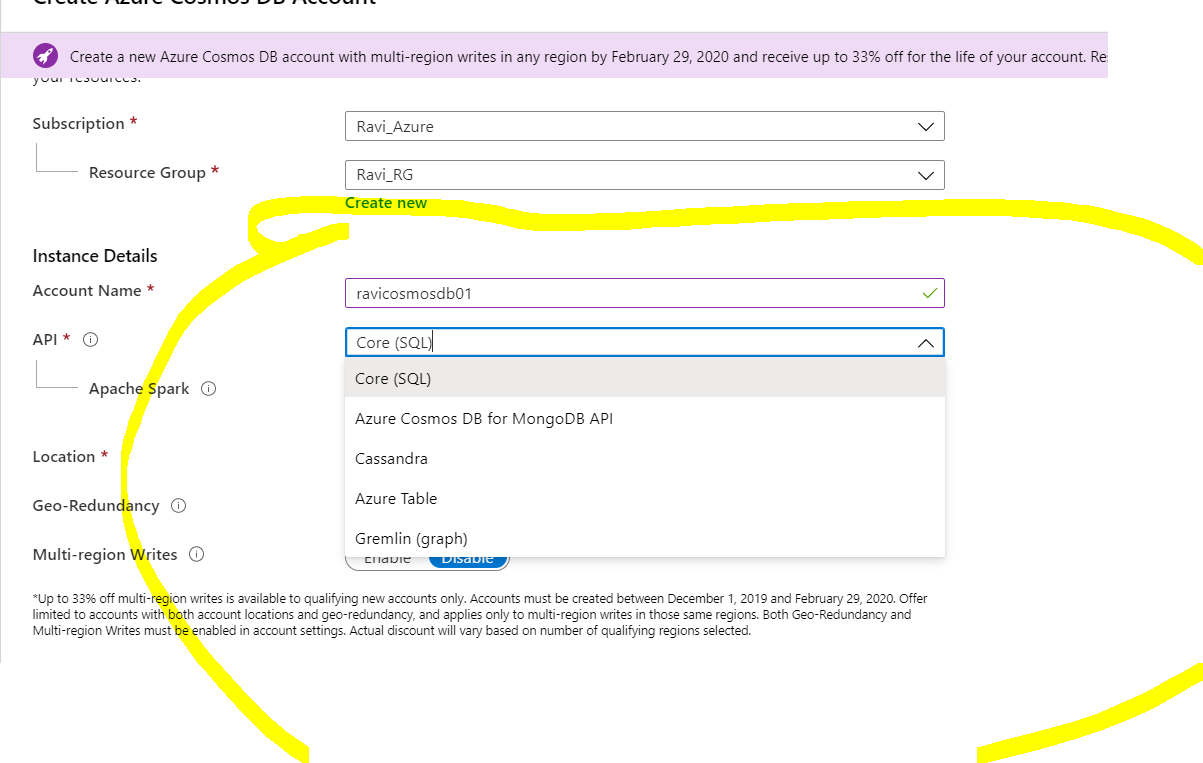
**Cosmos DB**

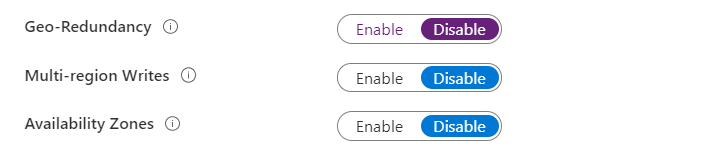
Each Azure Cosmos DB account is associated with one of the several data models Azure Cosmos DB supports, and you can create as many accounts as you need.

SQL API is the preferred data model if you are creating a new application.

If you are working with graphs or tables or migrating your MangoDB or Cassandra data to Azure, create additional accounts and select relevant data models.

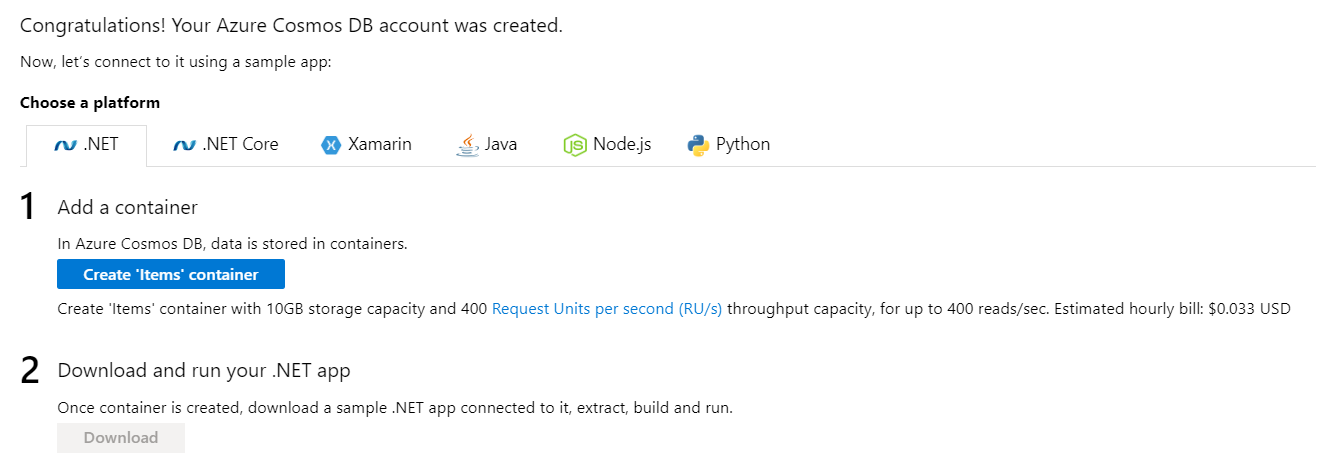
List of APIs available





Request Unit (RU):

Azure Cosmos DB measures throughput using something called a **request unit (RU)**. Request unit usage is measured per second, so the unit of measure is **request units per second (RU/s)**.



A single request unit, 1 RU, is equal to the approximate cost of performing a single GET request on a 1-KB document using a document's ID. Performing a GET by using a document's ID is an efficient means for retrieving a document, and thus the cost is small. Creating, replacing, or deleting the same item requires additional processing by the service, and therefore requires more request units.

You provision the number of RUs for your application on a per-second basis in increments of 100 RUs per second. To scale the provisioned throughput for your application, you can increase or decrease the number of RUs at any time. You can scale in increments or decrements of 100 RUs. You can make your changes either programmatically or by using the Azure portal. You are billed on an hourly basis.

**In Cosmos DB data is stored in the containers**

When you create an account, you can provision a minimum of 400 RU/s, or a maximum of 250,000 RU/s in the portal. If you need even more throughput, fill out a ticket in the Azure portal.

Partition key: unique fields like customer id, product id – data distributes across the data base container(cosmos db) . each partition key cant be exceed 10 GB. Which is size of one physical partition in cosmos DB. Using composite key so record would be smaller and partition can be achieved efficiently.

**Best practices**

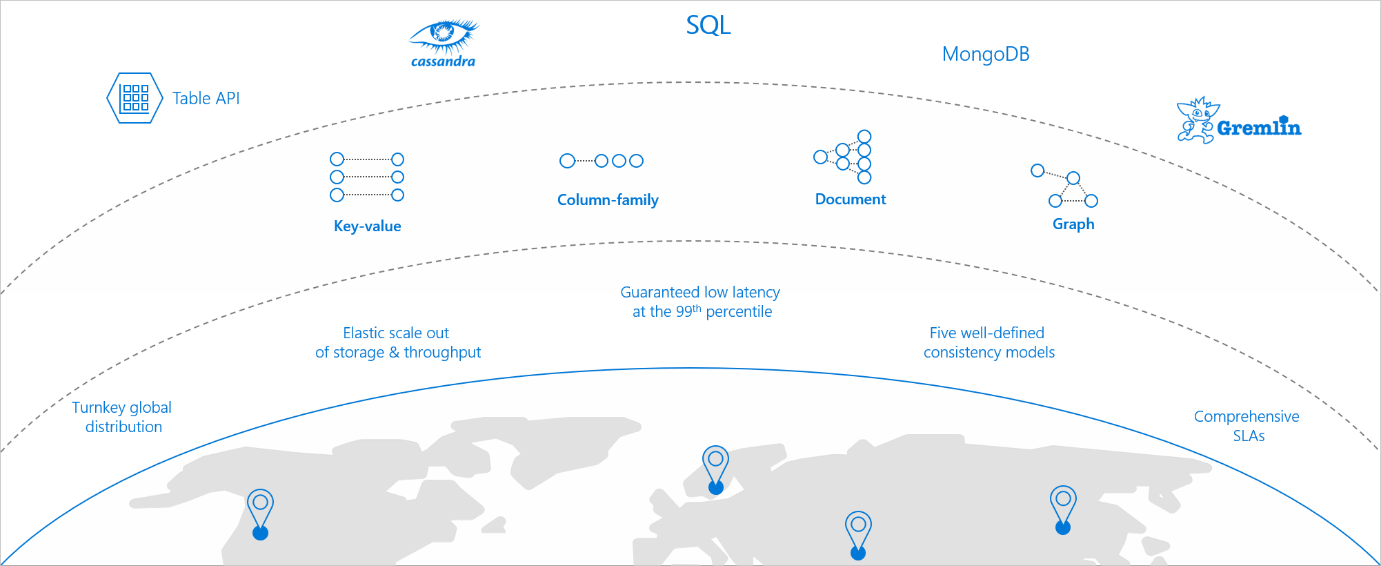
When you're trying to determine the right partition key and the solution isn't obvious, here are a few tips to keep in mind.

* Don’t be afraid of choosing a partition key that has a large number of values. The more values your partition key has, the more scalability you have.
* To determine the best partition key for a read-heavy workload, review the top three to five queries you plan on using. The value most frequently included in the WHERE clause is a good candidate for the partition key.
* For write-heavy workloads, you'll need to understand the transactional needs of your workload, because the partition key is the scope of multi-document transactions.

**Cosmos DB account 🡪 Cosmos DB 🡪 container.**

One cosmos DB can contain n number of containers.

At lowest level all the data stored in ARS format(atom record sequence)



Globally distributed – data consistency is an issue:

offering you five different consistency levels:

strong,

bounded staleness,

session,

consistent prefix,

and eventual.

Cosmos DB APIs (API is to be configured at Cosmos DB account level)

**SQL API**(default API)(JSON document format),

**MongoDB**(JSON document format),

**Cassandra API –** Cassandra Query Language (CQL) – supports version 4 of the CQL wire protocol.

Azure Table API:

Azure Cosmos DB's Azure Table API provides support for applications that are written for Azure Table Storage that need premium capabilities like global distribution, high availability, scalable throughput. The original Table API only allows for indexing on the Partition and Row keys; there are no secondary indexes. Storing table data in Comsos DB automatically indexes all the properties, and requires no index management.

Querying is accomplished by using OData and LINQ queries in code, and the original REST API for GET operations.

Gremlin(graph) API: data is either vertex(individual item in the database) or an edge(which is a relationship between items in the database).

Traversal langurage (azure Cosmos db supports Apache Tinderpop’s Gremlin language) . results will be returned in GraphSON format.

**Use the Azure Table API to store IoT data**

IoT data consists of key-value pairs.

Cosmos DB has a per document size limit of 2MB

Monitoring is general for every resource:

1. Alerts
2. Metrics
3. Diagnostic settings
4. Logs

Table storage Store data in a NoSQL database

## Differences between Azure Storage tables and Azure Cosmos DB tables

There are some differences in behavior between Azure Storage tables and Azure Cosmos DB tables to remember if you are considering a migration. For example:

* You are charged for the capacity of an Azure Cosmos DB table as soon as it is created, even if that capacity isn't used. This charging structure is because Azure Cosmos DB uses a reserved-capacity model to ensure that clients can read data within 10 ms. In Azure Storage tables, you are only charged for used capacity, but read access is only guaranteed within 10 seconds.
* Query results from Azure Cosmos DB are not sorted in order of partition key and row key as they are from Storage tables.
* Row keys in Azure Cosmos DB are limited to 255 bytes.
* Batch operations are limited to 2 MBs.
* Cross-Origin Resource Sharing (CORS) is not currently supported by Azure Cosmos DB.
* Table names are case-sensitive in Azure Cosmos DB. They are not case-sensitive in Storage tables.

|  |  |  |
| --- | --- | --- |
| **Priority** | **Azure Storage Tables** | **Azure Cosmos DB Tables** |
| Latency | Responses are fast, but there is no guaranteed response time. | < 10 ms for reads, < 15 ms for writes |
| Throughput | Maximum 20,000 operations/sec | No upper limit on throughput. Over 10 million operations/sec/table. |
| Global distribution | Single region for writes. A secondary read-only region is possible with read-access geo-redundant replication. | Replication of data for read and write to more than 30 regions. |
| Indexes | A single primary key on the partition key and the row key. No other indexes. | Indexes are created automatically on all properties. |
| Data consistency | Strong in the primary region. If you are using read-access geo-redundant replication, it may take time for changes to reach the secondary region. | You can choose from five different consistency levels depending on your needs for availability, latency, throughput, and consistency. |
| Pricing | Optimized for storage. | Optimized for throughput. |
| SLAs | 99.99% availability. | 99.99% availability for single region and relaxed consistency databases. 99.999% availability for multi-region databases. |

# Copy data from AZ table store to Cosomos DB (for table API)

# Migration tool

# Azcopy

# tables in Azure Storage accounts can rapidly migrate their data to Azure Cosmos DB to gain performance, scalability, and availability advantages. Another good reason to move to Azure Cosmos DB is that users can connect to writable replicas of the data in their own regions, and Azure Cosmos DB will replicate their changes throughout the world.

# Build a .NET Core app for Azure Cosmos DB in Visual Studio Code

* Must have [Visual Studio Code](https://code.visualstudio.com/) installed
* Must have [.NET Core 2.1 SDK or later versions](https://www.microsoft.com/net/download) installed
* Must have the [Azure Account](https://marketplace.visualstudio.com/items?itemName=ms-vscode.azure-account) extension installed
* Install Azure Cosmos DB extension

<https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-cosmosdb>

extensions will be done in VScode

LINQ is a .NET programming model that expresses computations as queries on streams of objects. You can create an **IQueryable** object that directly queries Azure Cosmos DB

**Cosmos DB performance:**

Azure Cosmos DB monitoring tools to observe and understand the following key metrics of Azure Cosmos DB:

* **Total requests** made per second, and the proportion of requests rejected because the provisioned throughput is exceeded
* **Total throughput**, measured in Request Units per second (RU/s), and the distribution of throughput across partitions
* **Total storage**, measured in kilobytes (KB), and the distribution of data across partitions

Querying documents from within the same partition is less expensive than querying across partitions.

Partitioning cant be changed after the database is created.

But indexing can be changed.

## **Avoiding hot partitions**

The Azure Cosmos DB throughput you've configured is divided evenly among partitions. A partition key design that doesn't evenly distribute throughput requests can create hot partitions. A hot partition is accessed more than the other partitions. The result is an inefficient use of the total configured throughput. If the demand on the hot partition is high enough, the partition becomes overloaded and traffic to the database is rate-limited.

A good partition design avoids hot partitions.

## **Partition design considerations**

Designing a partitioning strategy requires you to understand your data and its operational workloads. As you consider your design, we recommend that you do the following.

### Estimate the scale of your data needs

* What's the approximate size of your documents, or range of sizes?
* What's the required throughput in number of reads per second and writes per second?
* What's the volume of documents being queried?

### Understand the workload

* Do you have a read-heavy or write-heavy workload, or both?
* If it's read-heavy, what are the top five queries?
* If it's write-heavy, do you need transactions?

### Propose some partition key options

* Does the key choice have a large number of possible values or large cardinality?
* Do the values have a consistent spread across the data?
* Are some values accessed more than others?
* For read-heavy workloads, can the query be within a single partition?
* For write-heavy transactional workloads, can the transaction be within a single partition?

Partition key should be unique and should have high cardinality.

Indexing:

An index is extra information that sits alongside a collection to make querying more efficient. Queries use the index to locate documents.

Index modes: **Consistent, Lazy, and None.**

## **Global distribution**

Global distribution enables you to replicate data from one region into multiple Azure regions. You can add or remove regions in which your database is replicated at any time, and Azure Cosmos DB ensures that when you add an additional region, your data is available for operations within 30 minutes, assuming your data is 100 TBs or less.

There are two common scenarios for replicating data in two or more regions:

1. Delivering low-latency data access to end users no matter where they are located around the globe
2. Adding regional resiliency for business continuity and disaster recovery (BCDR)

There are three conflict resolution modes offered by Azure Cosmos DB.

* **Last-Writer-Wins (LWW)**, in which conflicts are resolved based on the value of a user-defined integer property in the document. By default \_ts is used to determine the last written document. Last-Writer-Wins is the default conflict handling mechanism.
* **Custom - User-defined function**, in which you can fully control conflict resolution by registering a User-defined function to the collection. A User-defined function is a special type of stored procedure with a specific signature. If the User-defined function fails or does not exist, Azure Cosmos DB will add all conflicts into the read-only conflicts feed they can be processed asynchronously.
* **Custom - Async**, in which Azure Cosmos DB excludes all conflicts from being committed and registers them in the read-only conflicts feed for deferred resolution by the user’s application. The application can perform conflict resolution asynchronously and use any logic or refer to any external source, application, or service to resolve the conflict.

|  |  |
| --- | --- |
| **Consistency levels and guarantees** |  |
|  |  |
| **Consistency Level** | **Guarantees** |
| Strong | Linearizability. Reads are guaranteed to return the most recent version of an item. |
| Bounded Staleness | Consistent Prefix. Reads lag behind writes by at most k prefixes or t interval. |
| Session | Consistent Prefix. Monotonic reads, monotonic writes, read-your-writes, write-follows-reads. |
| Consistent Prefix | Updates returned are some prefix of all the updates, with no gaps. |
| Eventual | Out of order reads. |

**Azure Data lake store Gen2**

A data lake is a repository of data that is stored in its natural format, usually as blobs or files. Azure Data Lake Storage is a comprehensive, scalable, and cost-effective data lake solution for big data analytics built into Azure.

ACL (access control lists enable file level access controls) hierarchical name space allows bigdata analytics.

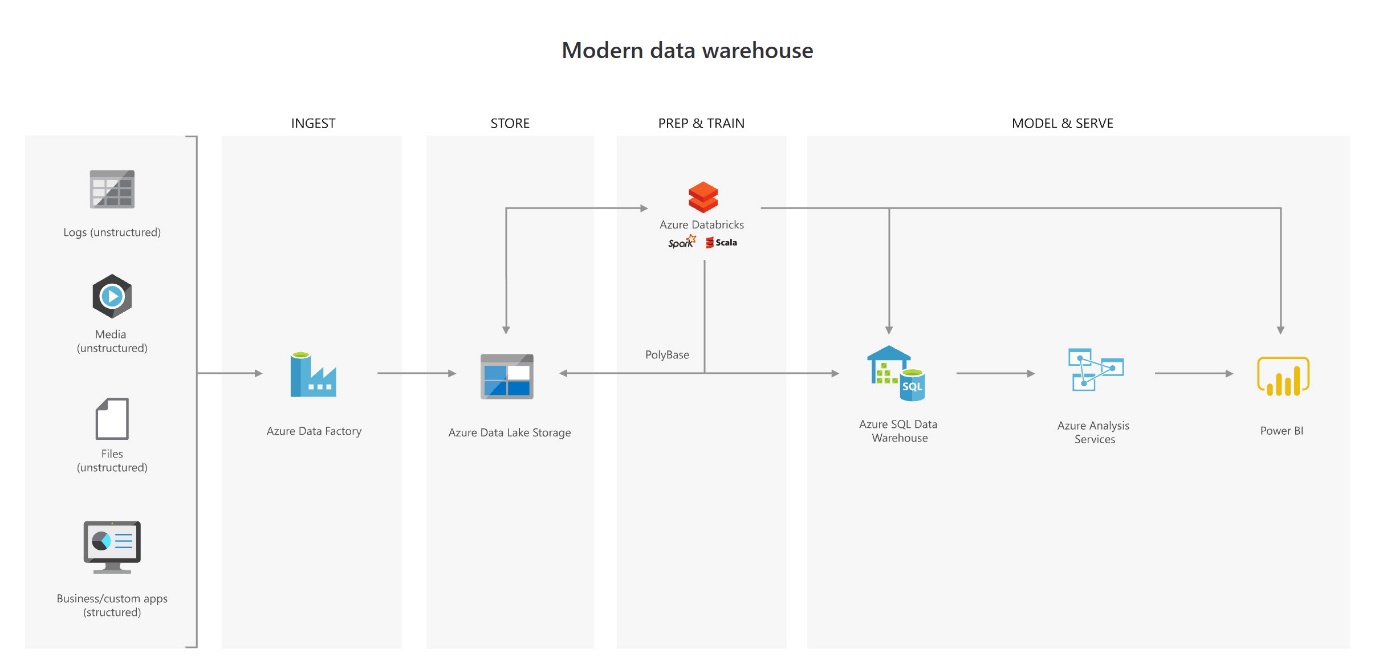
Blobstorage (flat namespace/single hierarchy) where as ADLS Gen 2 – has hierarchical name space – blob data into directories and stores metadata about each directory. Hierarchical namespaces keep the data organized, which yields better storage and retrieval performance for an analytical use case and lowers the cost of analysis.

* A modern data warehouse.
* Advanced analytics against big data.
* A real-time analytical solution.

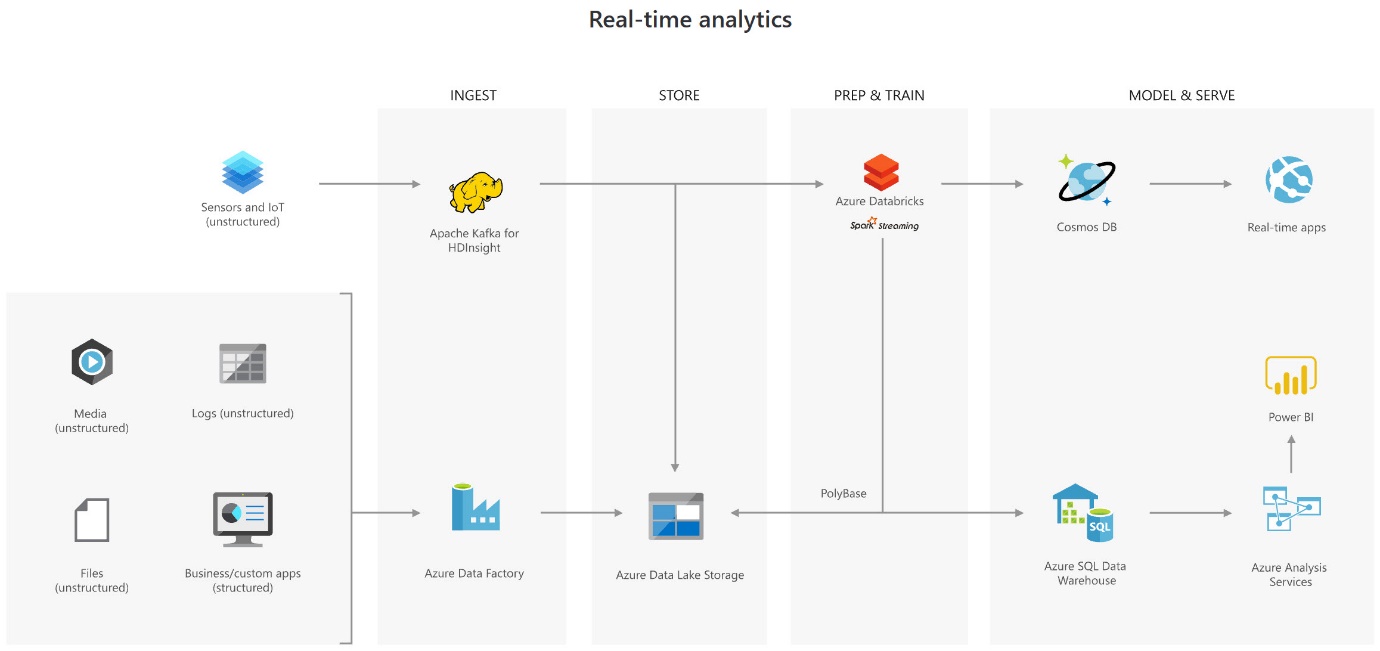
For example, for batch movement of data, Azure Data Factory may be the most appropriate technology to use. For real-time ingestion of data, Apache Kafka for HDInsight or Stream Analytics may be an appropriate technology to use.

There are four stages for processing big data solutions that are common to all architectures:

* **Ingestion** - The ingestion phase identifies the technology and processes that are used to acquire the source data. This data can come from files, logs, and other types of unstructured data that must be put into the Data Lake Store. The technology that is used will vary depending on the frequency that the data is transferred. For example, for batch movement of data, Azure Data Factory may be the most appropriate technology to use. For real-time ingestion of data, Apache Kafka for HDInsight or Stream Analytics may be an appropriate technology to use.
* **Store** - The store phase identifies where the ingested data should be placed. In this case, we're using Azure Data Lake Storage Gen2.
* **Prep and train** - The prep and train phase identifies the technologies that are used to perform data preparation and model training and scoring for data science solutions. The common technologies that are used in this phase are Azure Databricks, Azure HDInsight or Azure Machine Learning Services.
* **Model and serve** - Finally, the model and serve phase involves the technologies that will present the data to users. These can include visualization tools such as Power BI, or other data stores such as Azure SQL Data Warehouse, Azure Cosmos DB, Azure SQL Database, or Azure Analysis Services. Often, a combination of these technologies will be used depending on the business requirements.



Realtime analytics

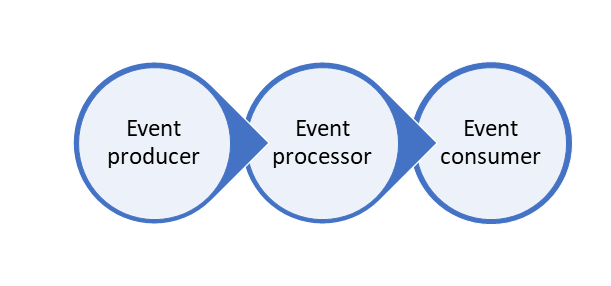


**Streaming Analytics**

Azure Stream Analytics is a PaaS service that integrates with your applications and Internet of Things (IoT) to gain insights with streaming data or static data held in a blob store. The process of consuming data streams, analyzing them, and deriving actionable insights out of them is called event processing. It requires an event producer, an event processor, and an event consumer. Azure Stream Analytics provides the event processing aspect to streaming that's fully managed and highly reliable.

The process of consuming data streams, analyzing them, and deriving actionable insights out of them is called **event processing**. An event processing pipeline has three distinct components:

* **Event producer**: Examples include sensors or processes that generate data continuously, such as a heart rate monitor or a highway toll lane sensor.
* **Event processor**: An engine to consume event data streams and derive insights from them. Depending on the problem space, event processors either process one incoming event at a time, such as a heart rate monitor, or process multiple events at a time, such as Azure Stream Analytics processing the highway toll lane sensor data.
* **Event consumer**: An application that consumes the data and takes specific action based on the insights. Examples of event consumers include alert generation, dashboards, or even sending data to another event processing engine.

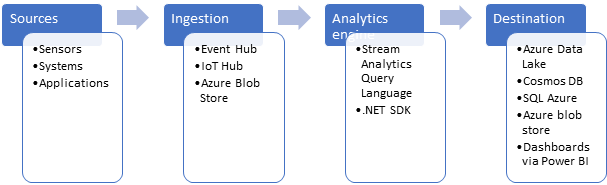


**Process events with Azure Stream Analytics**

* 5 minutes

Microsoft Azure Stream Analytics is an event processing engine. It enables the consumption and analysis of high volumes of streaming data generated by sensors, devices, or applications. Stream Analytics processes the data in real time. A typical event processing pipeline built on top of Stream Analytics consists of the following four components:

* **Event producer**: Any application, system, or sensor that continuously produces event data of interest. Examples can include a sensor that tracks the flow of water in a utility pipe to an application such as Twitter that generates tweets against a single hashtag.
* **Event ingestion system**: Takes the data from the source system or application to pass onto an analytics engine. Azure Event Hubs, Azure IoT Hub, or Azure Blob storage can all serve as the ingestion system.
* **Stream analytics engine**: Where compute is run over the incoming streams of data and insights are extracted. Azure Stream Analytics exposes the Stream Analytics query language (SAQL), a subset of Transact-SQL that's tailored to perform computations over streaming data. The engine supports windowing functions that are fundamental to stream processing and are implemented by using the SAQL.
* **Event consumer**: A destination of the output from the stream analytics engine. The target can be storage, such as Azure Data Lake, Azure Cosmos DB, Azure SQL Database, or Azure Blob storage, or dashboards powered by Power BI.



## **Azure Synapse Analytics (formerly SQL DW) (formerly SQL DWH)**

Azure Synapse Analytics is a limitless analytics service that brings together enterprise data warehousing and Big Data analytics.

It gives you the freedom to query data on your terms, using either serverless on-demand or provisioned resources-at scale. Azure Synapse brings these two worlds together with a unified experience to ingest, prepare, manage, and serve data for immediate BI and machine learning needs.

Simply put, Azure Synapse is Azure SQL Data Warehouse evolved. We have taken the [same industry leading data warehouse](https://azure.microsoft.com/en-ca/blog/analytics-in-azure-is-up-to-14x-faster-and-costs-94-less-than-other-cloud-providers-why-go-anywhere-else/) to a whole new level of performance and capabilities. Businesses can continue running their existing data warehouse workloads in production today with Azure Synapse and will automatically benefit from the new

Table Geometry in DWH:

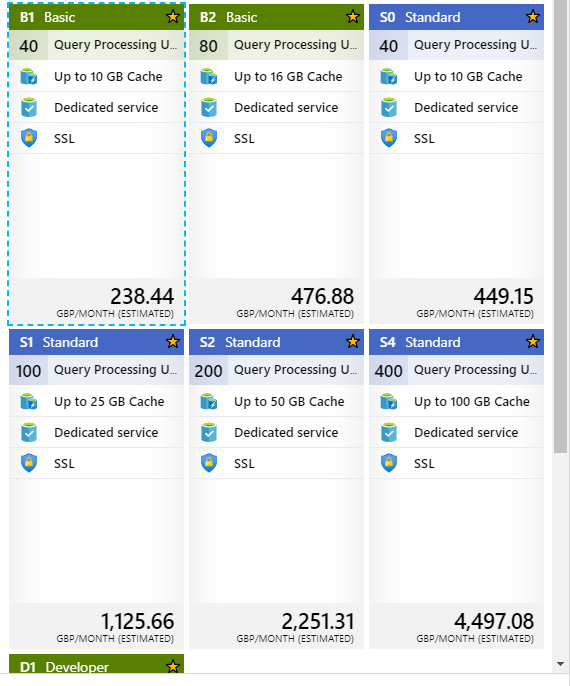
**Round-robin distribution is a table geometry that is useful for initial data loads.**

**Non-clustered is not a valid table geometry in Azure Synapse Analytics.**

**Replicated Table A replicated table is an appropriate table geometry choice because the size of the data in the table is less than 200 MB. The table will be replicated to every distribution node of a data warehouse in Azure Synapse Analytics to improve performance.**

Sample query: SELECT TOP (1000) \* FROM [dbo].[SalesByCategory]

Analysis services – performance measured in Query processing units (QPU)



Distribution/indexing info:

**Removing the CLUSTERED COLUMNSTORE INDEX for both tables reduces the performance of this query.**

**Correct. Placing Hash distribution on the ResellerKey on both the FactResellerSales and DimReseller improves the performance of the query.**

**Import data into DWH using PolyBase**

**In this case if a file of 1200 MB size to be loaded to dwh, synapsy analytics is G2 DW30000c.**

**Max size: Gen 2 DW30000c. it can’t be scaled beyond this size. It uses 60 compute nodes in parallel.**

**Azure Databricks**

Azure Databricks is a fully-managed version of the open-source [Apache Spark](https://spark.apache.org/) analytics and data processing engine. Azure Databricks is an enterprise-grade and secure cloud-based big data and machine learning platform.

### **Fast, easy, and collaborative Apache Spark-based analytics platform**

Accelerate innovation by enabling data science with a high-performance analytics platform that's optimized for Azure.

### Drive innovation and increase productivity

Bring teams together in an interactive workspace. From data gathering to model creation, use Databricks Notebooks to unify the process and instantly deploy to production. Launch your new Spark environment with a single click. Integrate effortlessly with a wide variety of data stores and services such as [Azure SQL Data Warehouse](https://azure.microsoft.com/en-us/services/sql-data-warehouse/), [Azure Cosmos DB](https://azure.microsoft.com/en-us/services/cosmos-db/), [Azure Data Lake Store](https://azure.microsoft.com/en-us/services/data-lake-store/), [Azure Blob storage](https://azure.microsoft.com/en-us/services/storage/blobs/), and [Azure Event Hub](https://azure.microsoft.com/en-us/services/event-hubs/). Add advanced artificial intelligence (AI) capabilities instantly and share your insights through rich integration with PowerBI.

### Build on secure, trusted cloud

Protect your data and business with Azure Active Directory integration, role-based controls, and enterprise-grade SLAs. Get peace of mind with fine-grained user permissions, enabling secure access to Databricks Notebooks, clusters, jobs, and data.

### Scale without limits

Globally scale your analytics and data science projects. Build and innovate faster using advanced machine learning capabilities. Add capacity instantly. Reduce cost and complexity with a fully managed, cloud-native platform. Target any size data or project using a complete set of analytics technologies including SQL, Streaming, MLlib, and GraphX.

## What is Apache Spark notebook?

A notebook is a collection of cells. These cells are run to execute code, to render formatted text, or to display graphical visualizations.

## What is a cluster?

The notebooks are backed by clusters, or networked computers, that work together to process your data. The first step is to create a cluster.

**Launch workspace 🡪 it will take you to databricks environment.**

**The supported Databricks notebook format is the DBC file type.**

**Chrome and Firefox are the recommended browsers to use with Databricks.**

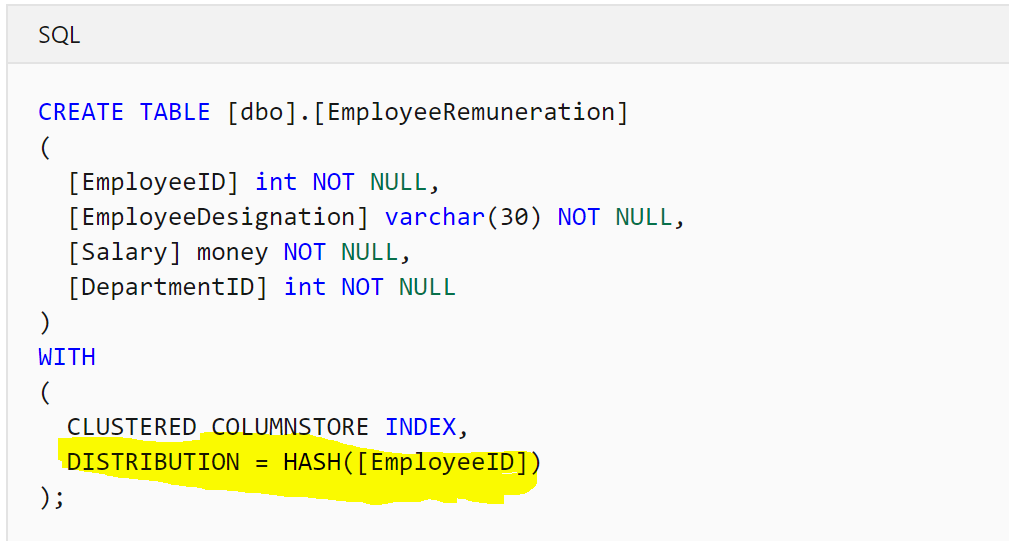
**SQL DWH(Synapsy analytics) to Databricks**

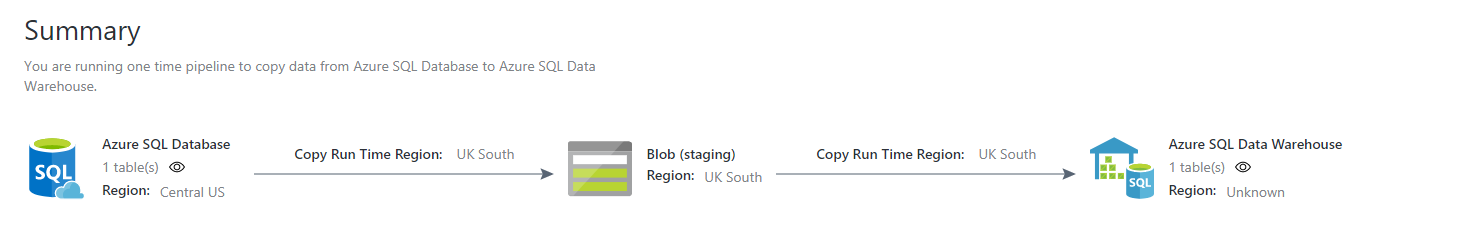
* Access Azure SQL Data Warehouse from Azure Databricks by using the SQL Data Warehouse connector.
* Use Apache Spark with Azure Blob storage and PolyBase in SQL Data Warehouse to transfer large volumes of data efficiently between a Databricks cluster and a SQL Data Warehouse instance.

The distribution method is defined at the table level. There are three choices:

* **Replicated**. The replicated table is fully copied to a distribution database on each Compute node. Replicating a table removes the need to transfer data among Compute nodes before a join or aggregation. Replicated tables are feasible only with small tables because of the extra storage required to store the full table on each Compute node.
* **Round robin**. Data is distributed evenly but randomly.
* **Hash distributed**. Data is distributed based on hashing values from a single column.

By default, when you don't define a data distribution method, your table will use the round robin distribution method. As you become more sophisticated in your implementation, you'll want to consider using hash distributed tables to minimize data movement, which will in turn optimize query performance.





**Source, destination and copy stings**

**Copy data using polybase:**

## **External data source 🡪 define file format 🡪 external table 🡪** Create a SQL Data Warehouse table and load data

**Eg table in dwh:**

CREATE TABLE [cso].[Transaction]

WITH

(

DISTRIBUTION = HASH([TransactionId])

)

AS

SELECT \* FROM [asb].[Transaction]

OPTION (LABEL = 'CTAS : Load [cso].[Transaction]');

What are the two prerequisites for connecting Azure Databricks with SQL Data Warehouse that apply to the SQL Data Warehouse instance?

Create a database master key and configure the firewall to enable Azure services to connect.

1. Which DataFrame method do you use to create a temporary view?

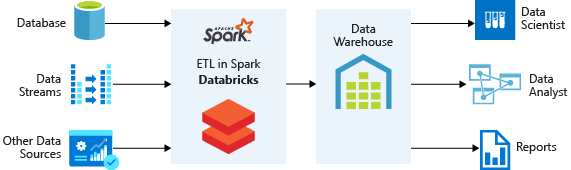
createOrReplaceTempView()

1. What is the DataFrame equivalent of the SQL statement SELECT count(\*) AS total?

.agg(count("\*").alias("total"))

Azure Databricks uses **DataFrames** to query, process, and analyze large volumes of data. The core structure of the data in a DataFrame is immutable. Being immutable means that the data structure can't be changed after it's created.

Databricks supports the end-to-end process of extracting data from the source, transforming that data, and finally loading the data into the target database. This entire process is known as extract, transform, and load (ETL).



By using Java Database Connectivity (JDBC), Databricks can virtually connect to any data store, including Azure Blob storage.

Spark ecosystem involves loading structured data back to the Databricks File System (DBFS) as a Parquet file

**Question:** Why are built-in functions faster?  
**Answer:** Reasons include:

* The catalyst optimizer knows how to optimize built-in functions
* They are written in highly optimized Scala
* There is no serialization cost at the time of running a built-in function
* **Question:** How do UDFs handle multiple column inputs and complex outputs?  
  **Answer:** UDFs allow for multiple column inputs. Complex outputs can be designated with the use of a defined schema encapsulate in a StructType() or a Scala case class.
* **Question:** How can I do vectorized UDFs in Python and are they as performant as built-in functions?  
  **Answer:** Spark 2.3 includes the use of vectorized UDFs using Pandas syntax. Even though they are vectorized, these UDFs will not be as performant built-in functions, though they will be more performant than non-vectorized Python UDFs.

## Review

**Question:** Why are joins expensive operations?  
**Answer:** Joins perform a large number of row-wise comparisons, making the cost associated with joining tables grow with the size of the data in the tables.

**Question:** What is the difference between a shuffle and broadcast join? How does Spark manage these differences?  
**Answer:** A shuffle join shuffles data between nodes in a cluster. By contrast, a broadcast join moves the smaller of two DataFrames to where the larger DataFrame sits, minimizing the overall data transfer. By default, Spark performs a broadcast join if the total number of records is below a certain threshold. The threshold can be manually specified or you can manually specify that a broadcast join should take place. Since the automatic determination of whether a shuffle join should take place is by number of records, this could mean that really wide data would take up significantly more space per record and should therefore be specified manually.

**Question:** What is a lookup table?  
**Answer:** A lookup table is small table often used for referencing commonly used data such as mapping cities to countries.

**Upserts are not supported in core Spark**

Databricks offers a data management system called Databricks Delta that does allow for upserts and other transactional functionality.

**Question:** How do you determine the number of connections to the database you write to?  
**Answer:** Spark makes one connection for each partition in your data. Increasing the number of partitions increases the database connections.

**Question:** How do you increase and decrease the number of partitions in your data?  
**Answer:** .repartitions(n) increases the number of partitions in your data. It can also decrease the number of partitions, but since this is a wide operation it should be used sparingly. .coalesce(n) decreases the number of partitions in your data. If you use .coalesce(n) with a number greater than the current paritions, this DataFrame method will have no effect.

**Question:** How can you change the default number of partitions?  
**Answer:** Changing the configuration parameter spark.sql.shuffle.partitions will alter the default number of partitions.

**Question:** What happens to the original data when I delete a managed table? What about an unmanaged table?  
**Answer:** Deleting a managed table deletes both the metadata and the data itself. Deleting an unmanaged table does not delete the original data.

**Question:** What is a metastore?  
**Answer:** A metastore is a repository of metadata such as the location of where data is and the schema information. A metastore does not include the data itself.

**Q:** What makes Spark different than Hadoop? **A:** Spark on Databricks performs 10-2000x faster than Hadoop Map-Reduce. It does this by providing a high-level query API which allows Spark to highly optimize the internal execution without adding complexity for the user. Internally, Spark employs a large number of optimizations such as pipelining related tasks together into a single operation, communicating in memory, using just-in-time code generation, query optimization, efficient tabular memory (Tungsten), caching, and more.

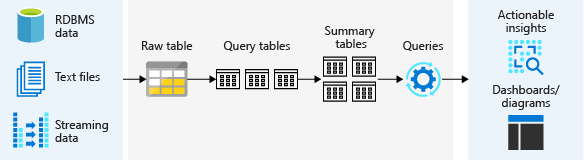
### Databricks Delta architecture

The Databricks Delta architecture is a vast improvement upon the traditional Lambda Architecture.

Text files, RDBMS data, and streaming data are collected into a raw table, also known as a bronze table at Databricks. A raw table is then parsed into query tables, also known as silver tables at Databricks. Query tables may be joined with dimension tables.

Summary tables, also known as gold tables at Databricks, are business-level aggregates. They're often used for reporting, dashboards, and aggregations such as daily active website users.

The final outputs are actionable insights, dashboards, and reports of business metrics.



**Q:** What is the Databricks Delta command to display metadata?  
**A:** Metadata is displayed through DESCRIBE DETAIL tableName.

**Q:** Where does the schema for a Databricks Delta data set reside?  
**A:** The table name, path, database info are stored in Hive metastore, the actual schema is stored in the \_delta\_logs directory.

**Q:** What is the general rule about partitioning and the cardinality of a set?  
**A:** We should partition on sets that are of small cardinality to avoid penalties incurred with managing large quantities of partition info meta-data.

**Q:** What is schema-on-read?  
**A:** It stems from Hive and roughly means: the schema for a data set is unknown until you perform a read operation.

**Q:** How does this problem manifest in Databricks assuming a parquet based data lake?  
**A:** It shows up as missing data upon load into a table in Databricks.

**Q:** How do you remedy this problem in Databricks above?  
**A:** To remedy, you repair the table using MSCK REPAIR TABLE or switch to Databricks Delta!

**Q:** What parameter do you need to add to an existing dataset in a Delta table?  
**A:** df.write...mode("append").save("..")

**Q:** What's the difference between .mode("append") and .mode("overwrite") ?  
**A:** append atomically adds new data to an existing Databricks Delta table and overwrite atomically replaces all of the data in a table.

**Q:** I've just repaired myTable using MSCK REPAIR TABLE myTable; How do I verify that the repair worked ?  
**A:** SELECT count(\*) FROM myTable and make sure the count is what I expected

**Q:** In exercise 2, why did we use .withColumn(.. cast(rand(5) ..) i.e. pass a seed to the rand() function ?  
**A:** In order to ensure we get the SAME set of pseudo-random numbers every time, on every cluster

**Q:** What does it mean to UPSERT?  
**A:** To UPSERT is to either INSERT a row, or if the row already exists, UPDATE the row.

**Q:** What happens if you try to UPSERT in a parquet-based data set?  
**A:** That's not possible due to the schema-on-read paradigm, you will get an error until you refresh the table.

**Q:** What is schema-on-read?  
**A:** It stems from Hive and roughly means: the schema for a data set is unknown until you perform a read operation.

**Q:** How to you perform UPSERT in a Databricks Delta dataset?  
**A:** Using the MERGE INTO my-table USING data-to-upsert.

**Q:** What is the caveat to USING data-to-upsert?  
**A:** Your source data has ALL the data you want to replace: in other words, you create a new dataframe that has the source data you want to replace in the Databricks Delta table.

**Q:** Why is Databricks Delta so important for a data lake that incorporates streaming data?  
**A:** Frequent meta data refreshes, table repairs and accumulation of small files on a secondly- or minutely-basis!

**Q:** What happens if you shut off your stream before it has fully initialized and started and you try to CREATE TABLE .. USING DELTA ?  
**A:** You will get this: Error in SQL statement: AnalysisException: The user specified schema is empty;.

**Q:** When you do a write stream command, what does this option do outputMode("append") ?  
**A:** This option takes on the following values and their respective meanings:

* **append**: add only new records to output sink
* **complete**: rewrite full output - applicable to aggregations operations
* **update**: update changed records in place

**Q:** What happens if you do not specify option("checkpointLocation", pointer-to-checkpoint directory)?  
**A:** When the streaming job stops, you lose all state around your streaming job and upon restart, you start from scratch.

**Q:** How do you view the list of active streams?  
**A:** Invoke spark.streams.active.

**Q:** How do you verify whether streamingQuery is running (boolean output)?  
**A:** Invoke spark.streams.get(streamingQuery.id).isActive.

 Why are many small files problematic when doing queries on data backed by these?  
**A:** If there are many files, some of whom may not be co-located the principal sources of slowdown are

* network latency
* (volume of) file metatadata

**Q:** What do OPTIMIZE and VACUUM do?  
**A:** OPTIMIZE creates the larger file from a collection of smaller files and VACUUM deletes the invalid small files that were used in compaction.

**Q:** What size files does OPTIMIZE compact to and why that value?  
**A:** Small files are compacted to around 1GB; this value was determined by the Spark optimization team as a good compromise between speed and performace.

**Q:** What should one be careful of when using VACUUM?  
**A:** Don't set a retention interval shorter than seven days because old snapshots and uncommitted files can still be in use by concurrent readers or writers to the table.

**Q:** What does ZORDER do?  
**A:** It is a technique to colocate related information in the same set of files.

**Q:** What is the difference between Lambda and Databricks Delta architecture?  
**A:** The principal difference is that with Databricks Delta architecture, output queries can be performed on streaming and historical data at the same time.

In Lambda architecture, streaming and historical data are treated as two separate branches feeding output queries.

**Q:** What is role of raw (bronze) tables?  
**A:** Raw tables capture streaming and historical data into a permanent record (streaming data tends to disappear after a short while). Though, it's generally hard to query.

**Q:** What is role of query (silver) tables?  
**A:** Query tables consist of normalized raw data that is easier to query.

**Q:** What is role of summary (gold) tables?  
**A:** Summary tables contain aggregated key business metrics that are queried frequently, but the silver queries themselves would take too long.

**Upsert:** Use MERGE INTO my-table USING data-to-upsert.

Show active streams: **spark.streams.active**

What size does OPTIMIZE compact small files to? Around 1 GB

**The Spark optimization team determined this value to be a good compromise between speed and performance.**

**Event Hubs**

Azure Event Hubs is a highly scalable publish-subscribe service that can ingest millions of events per second and stream them into multiple applications. This lets you process and analyze the massive amounts of data produced by your connected devices and applications.

Use Event Hubs to:

* Log millions of events per second in near real time.
* Connect devices using flexible authorization and throttling.
* Use time-based event buffering.
* Get a managed service with elastic scale.
* Reach a broad set of platforms using native client libraries.
* Pluggable adapters for other cloud services.

Eventhub Name Space 🡪 event hub

Kafka is messaging protocol

**Q:** Why is Databricks Delta so important for a data lake that incorporates streaming data?  
**A:** Frequent meta data refreshes, table repairs and accumulation of small files on a secondly- or minutely-basis!

**Q:** What happens if you shut off your stream before it has fully initialized and started and you try to CREATE TABLE .. USING DELTA ?  
**A:** You will get this: Error in SQL statement: AnalysisException: The user specified schema is empty;.

**Q:** When you do a write stream command, what does this option do outputMode("append") ?  
**A:** This option takes on the following values and their respective meanings:

* **append**: add only new records to output sink
* **complete**: rewrite full output - applicable to aggregations operations
* **update**: update changed records in place

**Q:** What happens if you do not specify option("checkpointLocation", pointer-to-checkpoint directory)?  
**A:** When the streaming job stops, you lose all state around your streaming job and upon restart, you start from scratch.

**Q:** How do you view the list of active streams?  
**A:** Invoke spark.streams.active.

**Q:** How do you verify whether streamingQuery is running (boolean output)?  
**A:** Invoke spark.streams.get(streamingQuery.id).isActive.

**Q:** Why is Databricks Delta so important for a data lake that incorporates streaming data?  
**A:** Frequent meta data refreshes, table repairs and accumulation of small files on a secondly- or minutely-basis!

**Q:** What happens if you shut off your stream before it has fully initialized and started and you try to CREATE TABLE .. USING DELTA ?  
**A:** You will get this: Error in SQL statement: AnalysisException: The user specified schema is empty;.

**Q:** When you do a write stream command, what does this option do outputMode("append") ?  
**A:** This option takes on the following values and their respective meanings:

* **append**: add only new records to output sink
* **complete**: rewrite full output - applicable to aggregations operations
* **update**: update changed records in place

**Q:** What happens if you do not specify option("checkpointLocation", pointer-to-checkpoint directory)?  
**A:** When the streaming job stops, you lose all state around your streaming job and upon restart, you start from scratch.

**Q:** How do you view the list of active streams?  
**A:** Invoke spark.streams.active.

**Q:** How do you verify whether streamingQuery is running (boolean output)?  
**A:** Invoke spark.streams.get(streamingQuery.id).isActive.

**Machine Learning**

**Models**

**Linear regression** – Linear regression predicts the value of a dependent variable based on its relationship with one or more independent variables.

Logistic regression

Support-vector machines:

Decision-tree classifier

Polynomial regression: advanced regression model

**Deep Learning**

Azure Databricks provides capabilities to build deep-learning algorithms that can be used to solve complex problems. Artificial neural networks make it possible to build such types of algorithms.

**Choose the right data store**

Modern business systems manage increasingly large volumes of data. Data may be ingested from external services, generated by the system itself, or created by users. These data sets may have extremely varied characteristics and processing requirements. Businesses use data to assess trends, trigger business processes, audit their operations, analyze customer behavior, and many other things.

This heterogeneity means that a single data store is usually not the best approach. Instead, it's often better to store different types of data in different data stores, each focused toward a specific workload or usage pattern. The term polyglot persistence is used to describe solutions that use a mix of data store technologies.

**RDBMS**

Relational databases organize data as a series of two-dimensional tables with rows and columns. Each table has its own columns, and every row in a table has the same set of columns. This model is mathematically based, and most vendors provide a dialect of the Structured Query Language (SQL) for retrieving and managing data. An RDBMS typically implements a transactionally consistent mechanism that conforms to the ACID (Atomic, Consistent, Isolated, Durable) model for updating information.

An RDBMS typically supports a schema-on-write model, where the data structure is defined ahead of time, and all read or write operations must use the schema. This is in contrast to most NoSQL data stores, particularly key/value types, where the schema-on-read model assumes that the client will be imposing its own interpretive schema on data coming out of the database, and is agnostic to the data format being written.

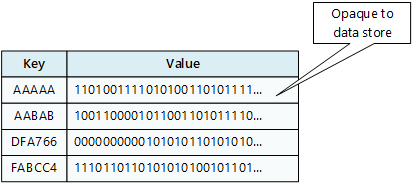
An RDBMS is very useful when strong consistency guarantees are important — where all changes are atomic, and transactions always leave the data in a consistent state. However, the underlying structures do not lend themselves to scaling out by distributing storage and processing across machines. Also, information stored in an RDBMS, must be put into a relational structure by following the normalization process. While this process is well understood, it can lead to inefficiencies, because of the need to disassemble logical entities into rows in separate tables, and then reassemble the data when running queries.

**Key/Value stores**

A key/value store is essentially a large hash table. You associate each data value with a unique key, and the key/value store uses this key to store the data by using an appropriate hashing function. The hashing function is selected to provide an even distribution of hashed keys across the data storage.

Most key/value stores only support simple query, insert, and delete operations. To modify a value (either partially or completely), an application must overwrite the existing data for the entire value. In most implementations, reading or writing a single value is an atomic operation. If the value is large, writing may take some time.

An application can store arbitrary data as a set of values, although some key/value stores impose limits on the maximum size of values. The stored values are opaque to the storage system software. Any schema information must be provided and interpreted by the application. Essentially, values are blobs and the key/value store simply retrieves or stores the value by key.



**Cosmos DB – Table API**

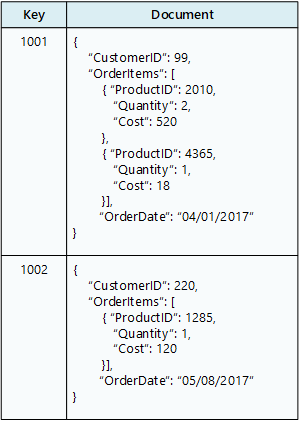
**Azure Cache - Redis**

## **Document databases**

A document database is conceptually similar to a key/value store, except that it stores a collection of named fields and data (known as documents), each of which could be simple scalar items or compound elements such as lists and child collections. The data in the fields of a document can be encoded in a variety of ways, including XML, YAML, JSON, BSON, or even stored as plain text. Unlike key/value stores, the fields in documents are exposed to the storage management system, enabling an application to query and filter data by using the values in these fields.

Typically, a document contains the entire data for an entity. What items constitute an entity are application specific. For example, an entity could contain the details of a customer, an order, or a combination of both. A single document may contain information that would be spread across several relational tables in an RDBMS.

A document store does not require that all documents have the same structure. This free-form approach provides a great deal of flexibility. Applications can store different data in documents as business requirements change.



The application can retrieve documents by using the document key. This is a unique identifier for the document, which is often hashed, to help distribute data evenly. Some document databases create the document key automatically. Others enable you to specify an attribute of the document to use as the key. The application can also query documents based on the value of one or more fields. Some document databases support indexing to facilitate fast lookup of documents based on one or more indexed fields.

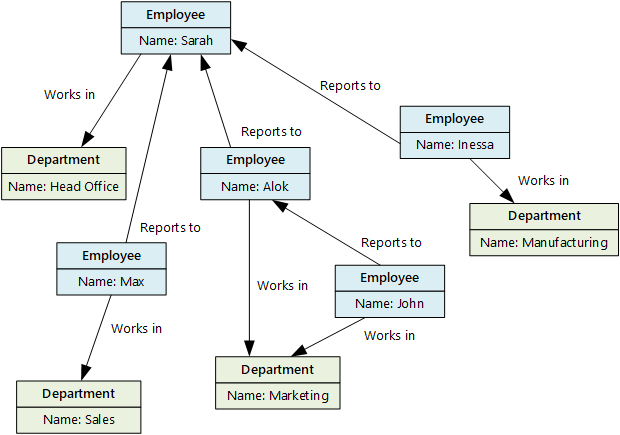
Many document databases support in-place updates, enabling an application to modify the values of specific fields in a document without rewriting the entire document. Read and write operations over multiple fields in a single document are usually atomic.

**Cosmos DB**

## **Graph databases**

A graph database stores two types of information, nodes and edges. You can think of nodes as entities. Edges which specify the relationships between nodes. Both nodes and edges can have properties that provide information about that node or edge, similar to columns in a table. Edges can also have a direction indicating the nature of the relationship.

The purpose of a graph database is to allow an application to efficiently perform queries that traverse the network of nodes and edges, and to analyze the relationships between entities. The following diagram shows an organization's personnel database structured as a graph. The entities are employees and departments, and the edges indicate reporting relationships and the department in which employees work. In this graph, the arrows on the edges show the direction of the relationships.



This structure makes it straightforward to perform queries such as "Find all employees who report directly or indirectly to Sarah" or "Who works in the same department as John?" For large graphs with lots of entities and relationships, you can perform very complex analyses very quickly. Many graph databases provide a query language that you can use to traverse a network of relationships efficiently.

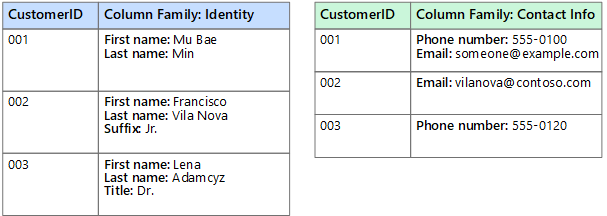
**Cosmos DB**

## **Column-family databases**

A column-family database organizes data into rows and columns. In its simplest form, a column-family database can appear very similar to a relational database, at least conceptually. The real power of a column-family database lies in its denormalized approach to structuring sparse data.

You can think of a column-family database as holding tabular data with rows and columns, but the columns are divided into groups known as *column families*. Each column family holds a set of columns that are logically related together and are typically retrieved or manipulated as a unit. Other data that is accessed separately can be stored in separate column families. Within a column family, new columns can be added dynamically, and rows can be sparse (that is, a row doesn't need to have a value for every column).

The following diagram shows an example with two column families, Identity and Contact Info. The data for a single entity has the same row key in each column-family. This structure, where the rows for any given object in a column family can vary dynamically, is an important benefit of the column-family approach, making this form of data store highly suited for storing structured, volatile data.



Unlike a key/value store or a document database, most column-family databases store data in key order, rather than by computing a hash. Many implementations allow you to create indexes over specific columns in a column-family. Indexes let you retrieve data by columns value, rather than row key.

Read and write operations for a row are usually atomic with a single column-family, although some implementations provide atomicity across the entire row, spanning multiple column-families.

Relevant Azure service: [HBase in HDInsight](https://docs.microsoft.com/en-us/azure/hdinsight/hdinsight-hbase-overview)

## **Data analytics**

Data analytics stores provide massively parallel solutions for ingesting, storing, and analyzing data. This data is distributed across multiple servers using a share-nothing architecture to maximize scalability and minimize dependencies. The data is unlikely to be static, so these stores must be able to handle large quantities of information, arriving in a variety of formats from multiple streams, while continuing to process new queries.

Relevant Azure services:

* [Azure Synapse Analytics](https://azure.microsoft.com/services/sql-data-warehouse/)
* [Azure Data Lake](https://azure.microsoft.com/solutions/data-lake/)
* [Azure Data Explorer](https://azure.microsoft.com/services/data-explorer/)

## **Search Engine Databases**

A search engine database supports the ability to search for information held in external data stores and services. A search engine database can be used to index massive volumes of data and provide near real-time access to these indexes. Although search engine databases are commonly thought of as being synonymous with the web, many large-scale systems use them to provide structured and ad-hoc search capabilities on top of their own databases.

The key characteristics of a search engine database are the ability to store and index information very quickly, and provide fast response times for search requests. Indexes can be multi-dimensional and may support free-text searches across large volumes of text data. Indexing can be performed using a pull model, triggered by the search engine database, or using a push model, initiated by external application code.

Searching can be exact or fuzzy. A fuzzy search finds documents that match a set of terms and calculates how closely they match. Some search engines also support linguistic analysis that can return matches based on synonyms, genre expansions (for example, matching dogs to pets), and stemming (matching words with the same root).

Relevant Azure service: [Azure Search](https://azure.microsoft.com/services/search/)

## **Time Series Databases**

Time series data is a set of values organized by time, and a time series database is a database that is optimized for this type of data. Time series databases must support a very high number of writes, as they typically collect large amounts of data in real time from a large number of sources. Updates are rare, and deletes are often done as bulk operations. Although the records written to a time-series database are generally small, there are often a large number of records, and total data size can grow rapidly.

Time series databases are good for storing telemetry data. Scenarios include IoT sensors or application/system counters.

Relevant Azure service: [Time Series Insights](https://azure.microsoft.com/services/time-series-insights/)

## **Object storage**

Object storage is optimized for storing and retrieving large binary objects (images, files, video and audio streams, large application data objects and documents, virtual machine disk images). Objects in these store types are composed of the stored data, some metadata, and a unique ID for accessing the object. Object stores enables the management of extremely large amounts of unstructured data.

Relevant Azure service: [Blob Storage](https://azure.microsoft.com/services/storage/blobs/)

## **Shared files**

Sometimes, using simple flat files can be the most effective means of storing and retrieving information. Using file shares enables files to be accessed across a network. Given appropriate security and concurrent access control mechanisms, sharing data in this way can enable distributed services to provide highly scalable data access for performing basic, low-level operations such as simple read and write requests.

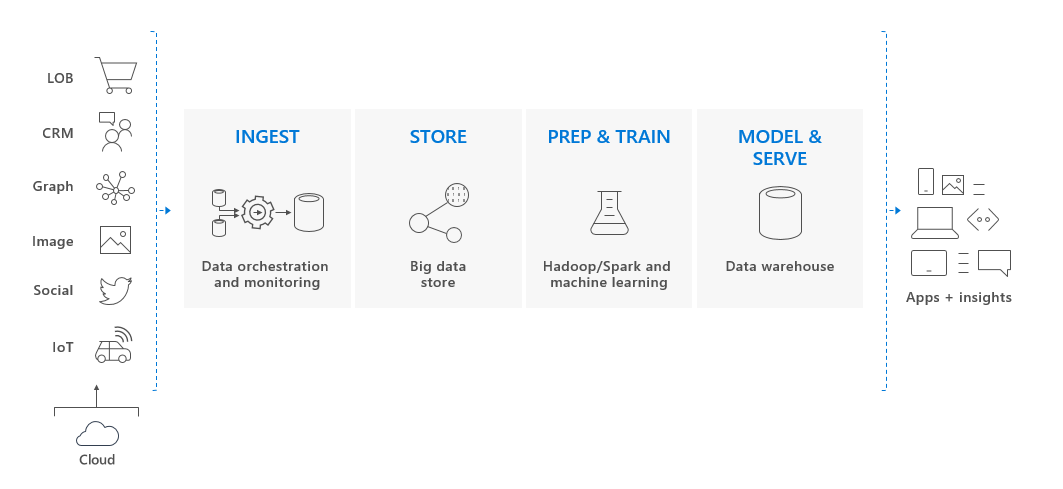
Relevant Azure service: [File Storage](https://azure.microsoft.com/services/storage/files/)

**Azure Cache for Redis**

Azure Cache for Redis is based on the popular software [Redis](https://redis.io/). It is typically used as a cache to improve the performance and scalability of systems that rely heavily on backend data-stores. Performance is improved by temporarily copying frequently accessed data to fast storage located close to the application. With [Azure Cache for Redis](https://redis.io/), this fast storage is located in-memory with Azure Cache for Redis instead of being loaded from disk by a database.

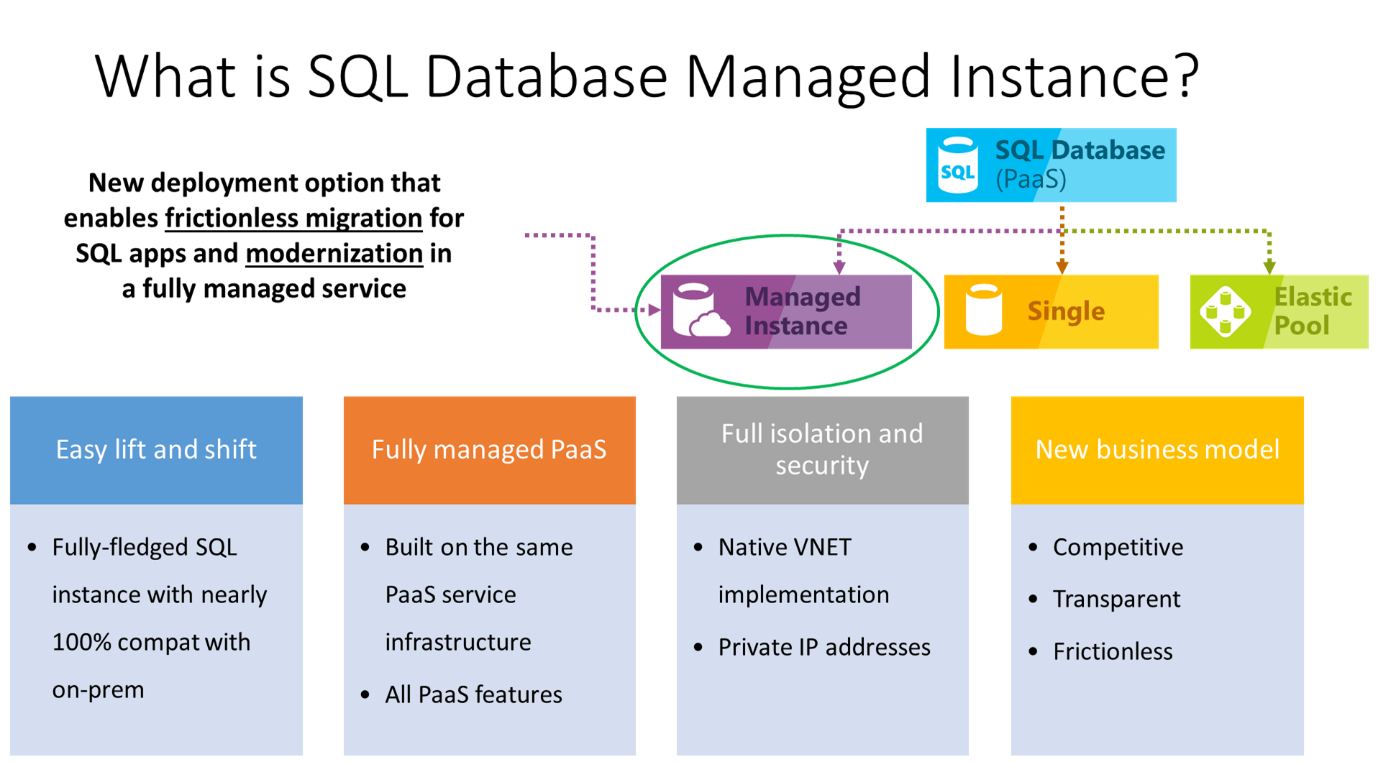
Azure Cache for Redis can also be used as an in-memory data structure store, a distributed **non-relational database**, and a message broker. Application performance is improved by taking advantage of the low-latency, high-throughput performance of the Redis engine.

Azure Cache for Redis provides you access to a secure, dedicated Redis cache. Azure Cache for Redis is managed by Microsoft, hosted within Azure, and accessible to any application within or outside of Azure.

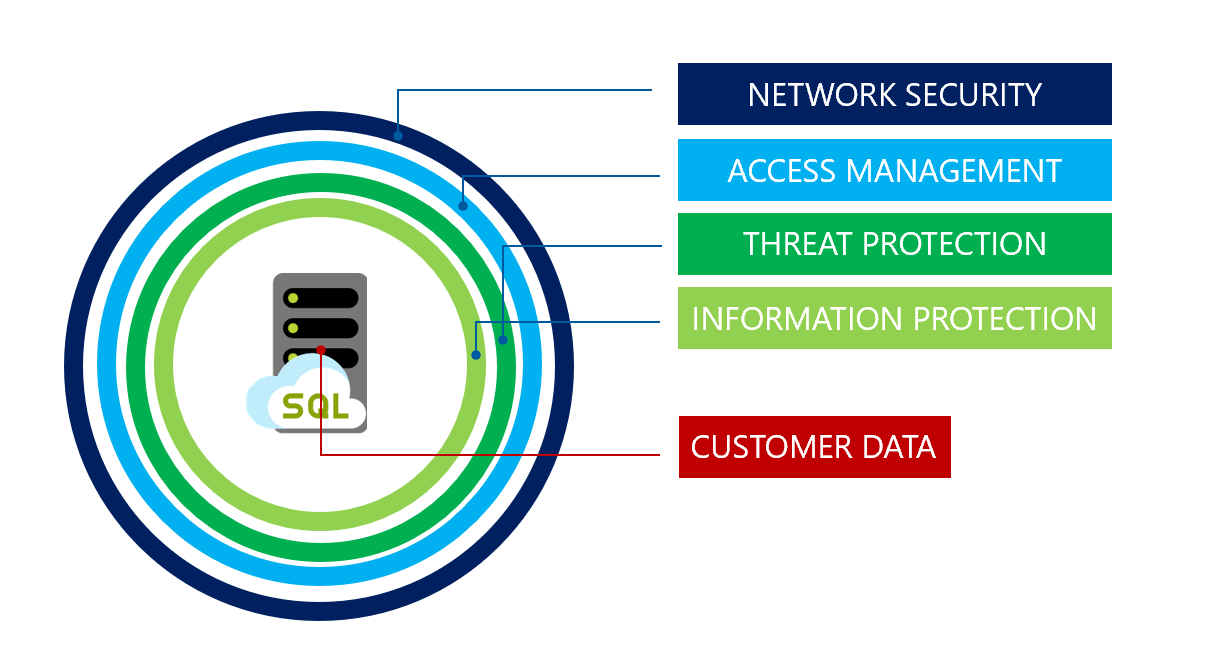


# What is Azure SQL Database managed instance?

Managed instance is a new deployment option of Azure SQL Database, providing near 100% compatibility with the latest SQL Server on-premises (Enterprise Edition) Database Engine, providing a native [virtual network (VNet)](https://docs.microsoft.com/en-us/azure/virtual-network/virtual-networks-overview) implementation that addresses common security concerns, and a [business model](https://azure.microsoft.com/pricing/details/sql-database/) favorable for on-premises SQL Server customers. The managed instance deployment model allows existing SQL Server customers to lift and shift their on-premises applications to the cloud with minimal application and database changes. At the same time, the managed instance deployment option preserves all PaaS capabilities (automatic patching and version updates, [automated backups](https://docs.microsoft.com/en-us/azure/sql-database/sql-database-automated-backups), [high-availability](https://docs.microsoft.com/en-us/azure/sql-database/sql-database-high-availability) ), that drastically reduces management overhead and TCO.



**SQL DB Security capabilities**



## **Network security**

Microsoft Azure SQL Database provides a relational database service for cloud and enterprise applications. To help protect customer data, firewalls prevent network access to the database server until access is explicitly granted based on IP address or Azure Virtual network traffic origin.

### **IP firewall rules**

IP firewall rules grant access to databases based on the originating IP address of each request. For more information, see [Overview of Azure SQL Database and SQL Data Warehouse firewall rules](https://docs.microsoft.com/en-us/azure/sql-database/sql-database-firewall-configure).

### **Virtual network firewall rules**

[Virtual network service endpoints](https://docs.microsoft.com/en-us/azure/virtual-network/virtual-network-service-endpoints-overview) extend your virtual network connectivity over the Azure backbone and enable Azure SQL Database to identify the virtual network subnet that traffic originates from. To allow traffic to reach Azure SQL Database, use the SQL [service tags](https://docs.microsoft.com/en-us/azure/virtual-network/security-overview) to allow outbound traffic through Network Security Groups.

[Virtual network rules](https://docs.microsoft.com/en-us/azure/sql-database/sql-database-vnet-service-endpoint-rule-overview) enable Azure SQL Database to only accept communications that are sent from selected subnets inside a virtual network.

**Authentication**

**SQL authentication**:

SQL database authentication refers to the authentication of a user when connecting to [Azure SQL Database](https://docs.microsoft.com/en-us/azure/sql-database/sql-database-technical-overview) using username and password. During the database server creation for the database, a "Server admin" login with a username and password must be specified. Using these credentials, a “server admin” can authenticate to any database on that database server as the database owner. After that, additional SQL logins and users can be created by the server admin, which enable users to connect using username and password.

**Azure Active Directory authentication**:

Azure Active Directory authentication is a mechanism of connecting to [Azure SQL Database](https://docs.microsoft.com/en-us/azure/sql-database/sql-database-technical-overview) and [SQL Data Warehouse](https://docs.microsoft.com/en-us/azure/sql-data-warehouse/sql-data-warehouse-overview-what-is) by using identities in Azure Active Directory (Azure AD). Azure AD authentication allows administrators to centrally manage the identities and permissions of database users along with other Microsoft services in one central location. This includes the minimization of password storage and enables centralized password rotation policies.

A server admin called the **Active Directory administrator** must be created to use Azure AD authentication with SQL Database. For more information, see [Connecting to SQL Database By Using Azure Active Directory Authentication](https://docs.microsoft.com/en-us/azure/sql-database/sql-database-aad-authentication). Azure AD authentication supports both managed and federated accounts. The federated accounts support Windows users and groups for a customer domain federated with Azure AD.

Additional Azure AD authentication options available are [Active Directory Universal Authentication for SQL Server Management Studio](https://docs.microsoft.com/en-us/azure/sql-database/sql-database-ssms-mfa-authentication) connections including [Multi-Factor Authentication](https://docs.microsoft.com/en-us/azure/active-directory/authentication/concept-mfa-howitworks) and [Conditional Access](https://docs.microsoft.com/en-us/azure/sql-database/sql-database-conditional-access).

SSPR: self service password reset.

|  |  |
| --- | --- |
| **Authentication Method** | **Usage** |
| Password | MFA and SSPR |
| Security questions | SSPR Only |
| Email address | SSPR Only |
| Microsoft Authenticator app | MFA and SSPR |
| OATH Hardware token | Public preview for MFA and SSPR |
| SMS | MFA and SSPR |
| Voice call | MFA and SSPR |
| App passwords | MFA only in certain cases |

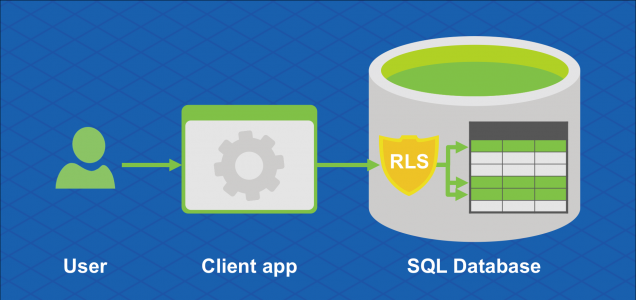
## Authorization

Authorization refers to the permissions assigned to a user within an Azure SQL Database, and determines what the user is allowed to do. Permissions are controlled by adding user accounts to [database roles](https://docs.microsoft.com/en-us/sql/relational-databases/security/authentication-access/database-level-roles) and assigning database-level permissions to those roles or by granting the user certain [object-level permissions](https://docs.microsoft.com/en-us/sql/relational-databases/security/permissions-database-engine). For more information, see [Logins and users](https://docs.microsoft.com/en-us/azure/sql-database/sql-database-manage-logins)

As a best practice, create custom roles when needed. Add users to the role with the least privileges required to do their job function. Do not assign permissions directly to users. The server admin account is a member of the built-in db\_owner role, which has extensive permissions and should only be granted to few users with administrative duties. For Azure SQL Database applications, use the [EXECUTE AS](https://docs.microsoft.com/en-us/sql/t-sql/statements/execute-as-clause-transact-sql) to specify the execution context of the called module or use [Application Roles](https://docs.microsoft.com/en-us/sql/relational-databases/security/authentication-access/application-roles) with limited permissions. This practice ensures that the application that connects to the database has the least privileges needed by the application. Following these best practices also fosters separation of duties.

### Row-level security

Row-Level Security enables customers to control access to rows in a database table based on the characteristics of the user executing a query (for example, group membership or execution context). Row-Level Security can also be used to implement custom Label-based security concepts. For more information, see [Row-Level security](https://docs.microsoft.com/en-us/sql/relational-databases/security/row-level-security).



## Threat protection

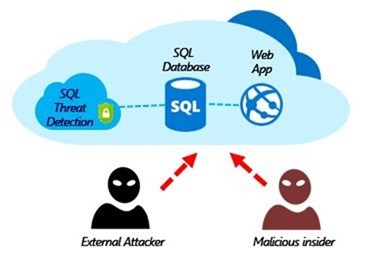
SQL Database secures customer data by providing auditing and threat detection capabilities.

### SQL auditing in Azure Monitor logs and Event Hubs

SQL Database auditing tracks database activities and helps to maintain compliance with security standards by recording database events to an audit log in a customer-owned Azure storage account. Auditing allows users to monitor ongoing database activities, as well as analyze and investigate historical activity to identify potential threats or suspected abuse and security violations. For more information, see Get started with [SQL Database Auditing](https://docs.microsoft.com/en-us/azure/sql-database/sql-database-auditing).

### **Advanced Threat Protection**

Advanced Threat Protection is analyzing your SQL Server logs to detect unusual behavior and potentially harmful attempts to access or exploit databases. Alerts are created for suspicious activities such as SQL injection, potential data infiltration, and brute force attacks or for anomalies in access patterns to catch privilege escalations and breached credentials use. Alerts are viewed from the [Azure Security Center](https://azure.microsoft.com/services/security-center/), where the details of the suspicious activities are provided and recommendations for further investigation given along with actions to mitigate the threat. Advanced Threat Protection can be enabled per server for an additional fee. For more information, see [Get started with SQL Database Advanced Threat Protection](https://docs.microsoft.com/en-us/azure/sql-database/sql-database-threat-detection).



## **Information protection and encryption**

### Transport Layer Security TLS (Encryption-in-transit)

SQL Database secures customer data by encrypting data in motion with [Transport Layer Security](https://support.microsoft.com/help/3135244/tls-1-2-support-for-microsoft-sql-server).

Sql Server enforces encryption (SSL/TLS) at all times for all connections. This ensures all data is encrypted "in transit" between the client and server irrespective of the setting of **Encrypt** or **TrustServerCertificate** in the connection string.

As a best practice, recommend that in your application's connection string you specify an encrypted connection and ***not*** trust the server certificate. This forces your application to verify the server certificate and thus prevents your application from being vulnerable to man in the middle type attacks.

For example when using the ADO.NET driver this is accomplished via **Encrypt=True** and **TrustServerCertificate=False**. If you obtain your connection string from the Azure portal, it will have the correct settings.

### **Transparent Data Encryption (Encryption-at-rest)**

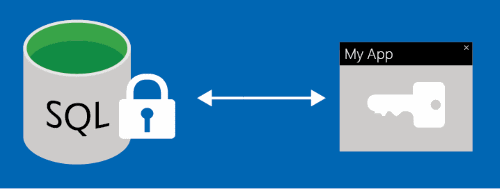
[Transparent Data Encryption (TDE) for Azure SQL Database](https://docs.microsoft.com/en-us/azure/sql-database/transparent-data-encryption-azure-sql) adds a layer of security to help protect data at rest from unauthorized or offline access to raw files or backups. Common scenarios include datacenter theft or unsecured disposal of hardware or media such as disk drives and backup tapes. TDE encrypts the entire database using an AES encryption algorithm, which doesn’t require application developers to make any changes to existing applications.

In Azure, all newly created SQL databases are encrypted by default and the database encryption key is protected by a built-in server certificate. Certificate maintenance and rotation are managed by the service and requires no input from the user. Customers who prefer to take control of the encryption keys can manage the keys in [Azure Key Vault](https://docs.microsoft.com/en-us/azure/key-vault/key-vault-secure-your-key-vault).

### Key management with Azure Key Vault

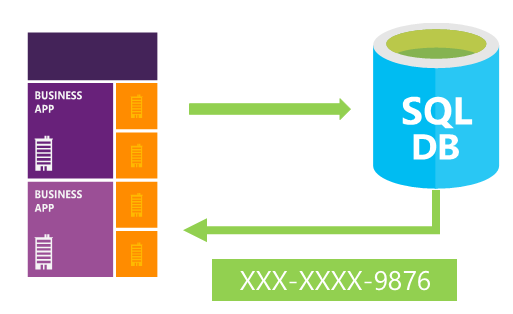
[Bring Your Own Key](https://docs.microsoft.com/en-us/azure/sql-database/transparent-data-encryption-byok-azure-sql) (BYOK) support for [Transparent Data Encryption](https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/transparent-data-encryption) (TDE) allows customers to take ownership of key management and rotation using [Azure Key Vault](https://docs.microsoft.com/en-us/azure/key-vault/key-vault-secure-your-key-vault), Azure’s cloud-based external key management system. If the database's access to the key vault is revoked, a database cannot be decrypted and read into memory. Azure Key Vault provides a central key management platform, leverages tightly monitored hardware security modules (HSMs), and enables separation of duties between management of keys and data to help meet security compliance requirements.

### **Always Encrypted (Encryption-in-use)**



[Always Encrypted](https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/always-encrypted-database-engine) is a feature designed to protect sensitive data stored in specific database columns from access (for example, credit card numbers, national identification numbers, or data on a *need to know* basis). This includes database administrators or other privileged users who are authorized to access the database to perform management tasks, but have no business need to access the particular data in the encrypted columns. The data is always encrypted, which means the encrypted data is decrypted only for processing by client applications with access to the encryption key. The encryption key is never exposed to SQL and can be stored either in the [Windows Certificate Store](https://docs.microsoft.com/en-us/azure/sql-database/sql-database-always-encrypted) or in [Azure Key Vault](https://docs.microsoft.com/en-us/azure/sql-database/sql-database-always-encrypted-azure-key-vault).

### Dynamic data masking



### Dynamic data masking policy

* **SQL users excluded from masking** - A set of SQL users or AAD identities that get unmasked data in the SQL query results. Users with administrator privileges are always excluded from masking, and see the original data without any mask.
* **Masking rules** - A set of rules that define the designated fields to be masked and the masking function that is used. The designated fields can be defined using a database schema name, table name, and column name.
* **Masking functions** - A set of methods that control the exposure of data for different scenarios.

Data masking functions:

**Default:** **Full masking according to the data types of the designated fields**

**Eg: number: 0, varchar: xxxx**

**Credit Card: Masking method, which exposes the last four digits of the designated fields** and adds a constant string as a prefix in the form of a credit card.

XXXX-XXXX-XXXX-1234

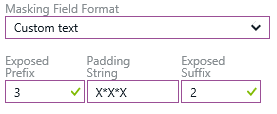
Email: **Masking method, which exposes the first letter and replaces the domain with XXX.com** using a constant string prefix in the form of an email address.

Eg:

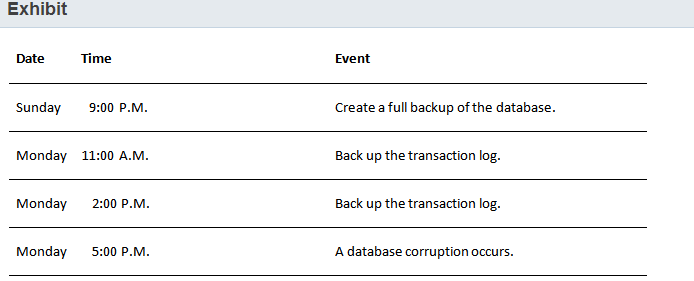
|  |
| --- |
| aXX@XXXX.com |
|  |

**Random number Masking method, which generates a random number** according to the selected boundaries and actual data types. If the designated boundaries are equal, then the masking function is a constant number.

**Custom Text:** **Masking method, which exposes the first and last characters** and adds a custom padding string in the middle. If the original string is shorter than the exposed prefix and suffix, only the padding string is used.  
prefix[padding]suffix



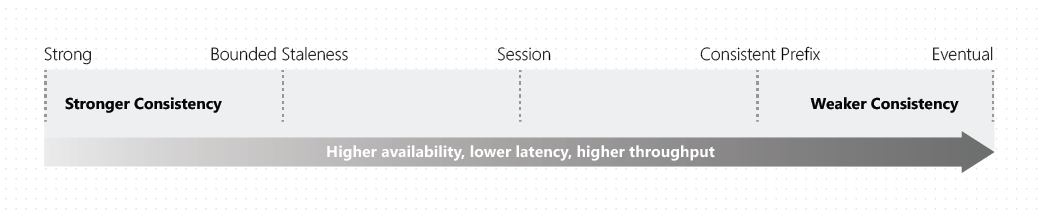
Backup scenario:



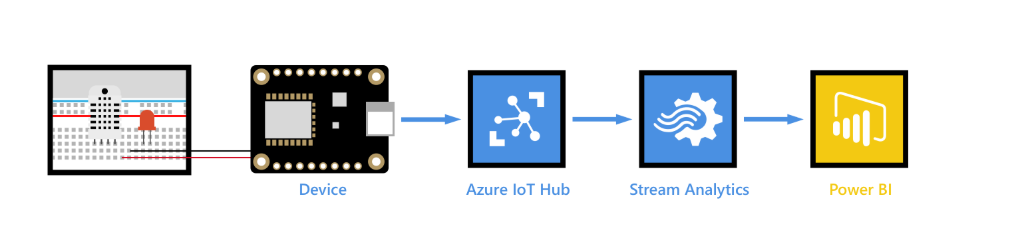
* 1. Create a tail log back up of the current transaction log
  2. Restore the full back up
  3. Restore the 11:00 AM transaction log back up
  4. Restore the 2:00 pm transaction log back up
  5. Restore the tail-log backup

Data Processing capabilities:

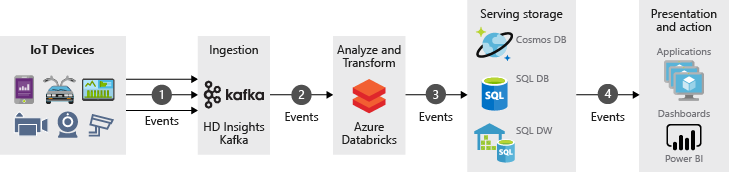
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Capability** | **Azure Data Lake Analytics** | **Azure Synapse** | **HDInsight with Spark** | **HDInsight with Hive** | **HDInsight with Hive LLAP** | **Azure Databricks** |
| Autoscaling | No | No | No | No | No | Yes |
| Scale-out granularity | Per job | Per cluster | Per cluster | Per cluster | Per cluster | Per cluster |
| In-memory caching of data | No | Yes | Yes | No | Yes | Yes |
| Query from external relational stores | Yes | No | Yes | No | No | Yes |
| Authentication | Azure AD | SQL / Azure AD | No | Azure AD1 | Azure AD1 | Azure AD |
| Auditing | Yes | Yes | No | Yes 1 | Yes 1 | Yes |
| Row-level security | No | Yes2 | No | Yes 1 | Yes 1 | No |
| Supports firewalls | Yes | Yes | Yes | Yes 3 | Yes 3 | No |
| Dynamic data masking | No | Yes | No | Yes 1 | Yes 1 | No |



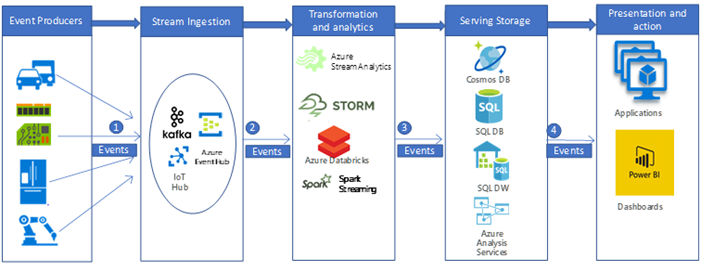
Real time data:



# Ingestion and processing of real-time automotive IoT data



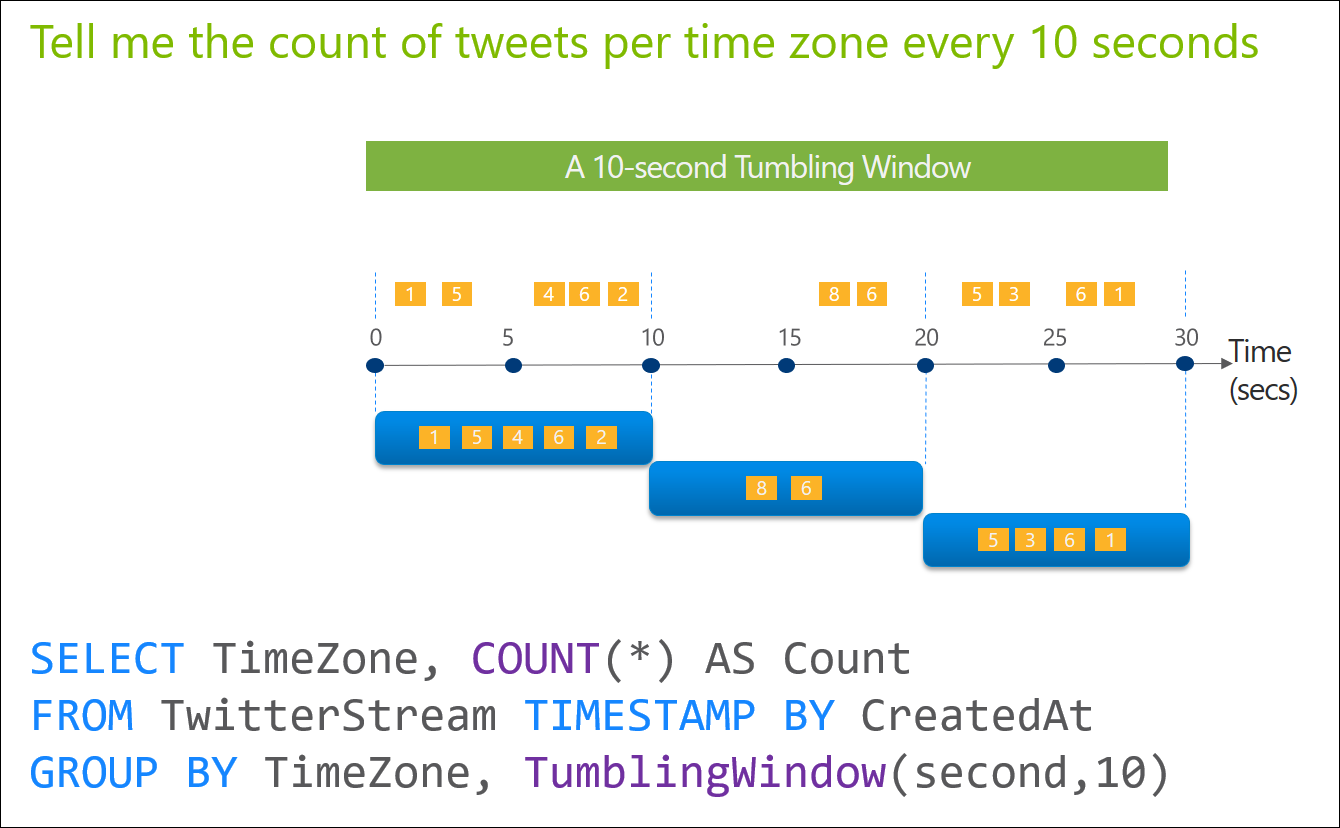
Alternatives:



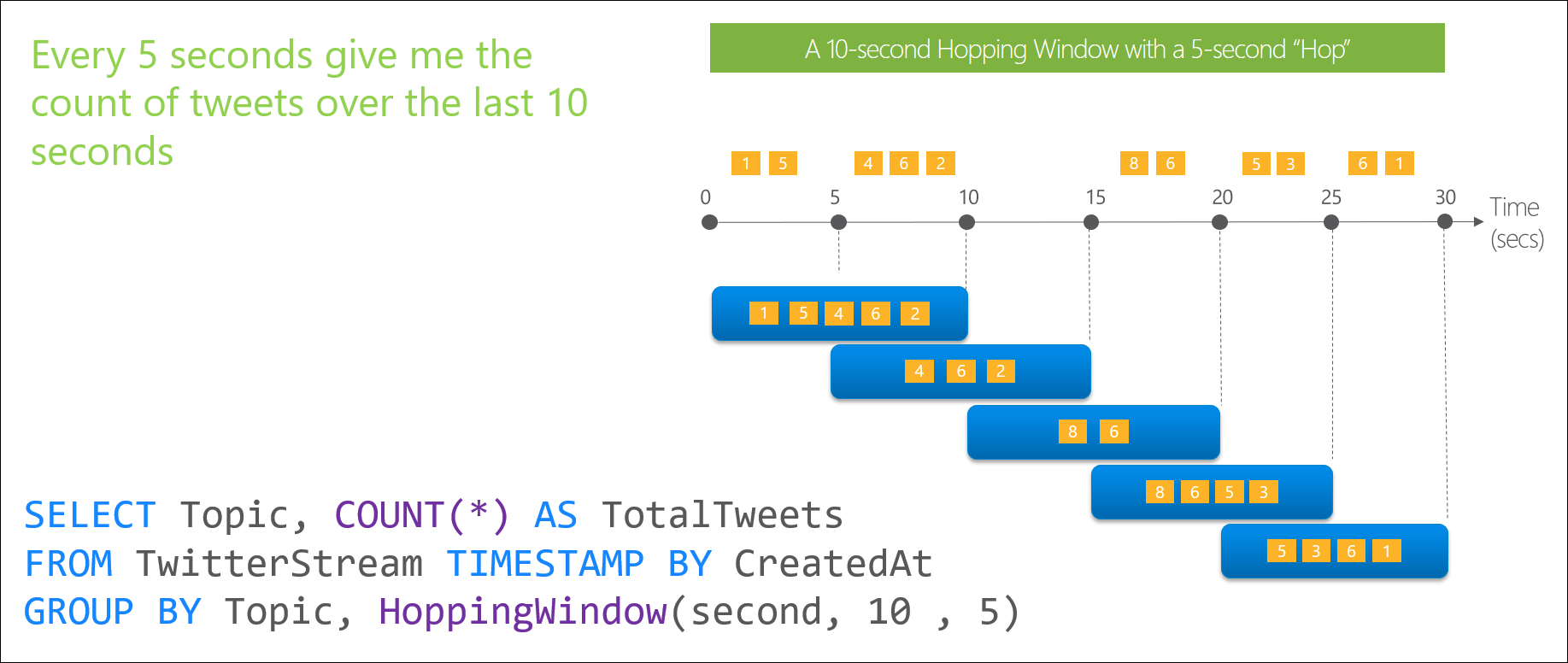
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Azure Store | Shared Key (storage account key) | Shared access signature (SAS) | Azure Active Directory (Azure AD) | Anonymous public read access |
| Azure Blobs | Supported | Supported | Supported | Supported |
| Azure Files (SMB) | Supported | Not supported | Supported, only with AAD Domain Services | Not supported |
| Azure Files (REST) | Supported | Supported | Not supported | Not supported |
| Azure Queues | Supported | Supported | Supported | Not supported |
| Azure Tables | Supported | Supported | Not supported | Not supported |

Windowing functions

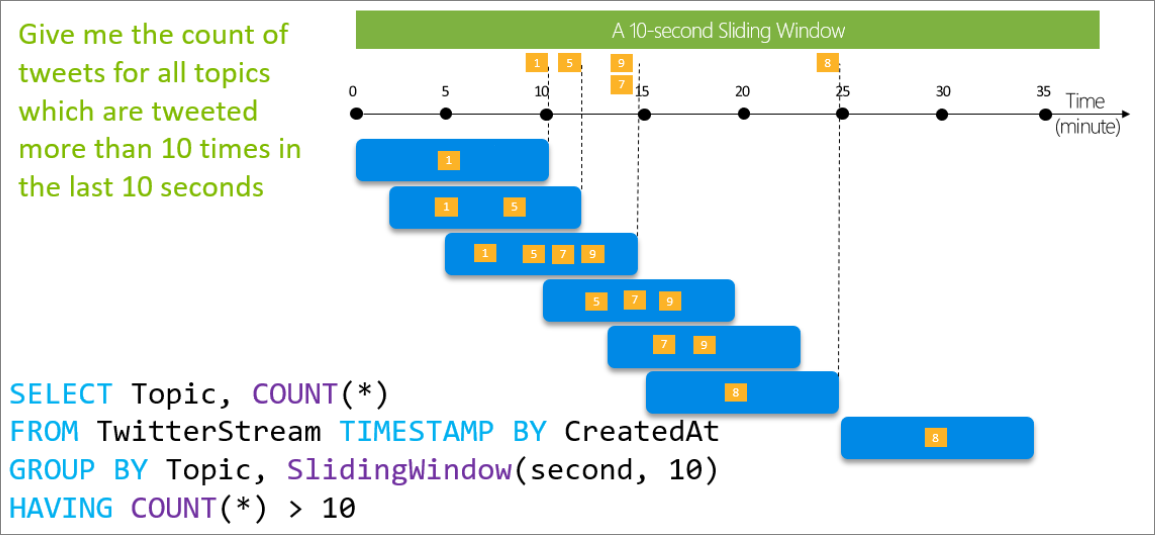
Tubbling: segmenation



Hopping: lookback and overlap



**Sliding: only when event occurs**



**Session:** Session window functions group events that arrive at similar times, filtering out periods of time where there is no data

