Choosing an analytical data store in Azure

02/12/2018 • 5 minutes to read • Contributors

In this article

In a big data architecture, there is often a need for an analytical data store that serves processed data in a structured format that can be queried using analytical tools. Analytical data stores that support querying of both hot-path and cold-path data are collectively referred to as the serving layer, or data serving storage.

The serving layer deals with processed data from both the hot path and cold path. In the lambda architecture, the serving layer is subdivided into a *speed serving* layer, which stores data that has been processed incrementally, and a *batch serving* layer, which contains the batch-processed output. The serving layer requires strong support for random reads with low latency. Data storage for the speed layer should also support random writes, because batch loading data into this store would introduce undesired delays. On the other hand, data storage for the batch layer does not need to support random writes, but batch writes instead.

There is no single best data management choice for all data storage tasks. Different data management solutions are optimized for different tasks. Most real-world cloud apps and big data processes have a variety of data storage requirements and often use a combination of data storage solutions.

What are your options when choosing an analytical data store?

There are several options for data serving storage in Azure, depending on your needs:

- SQL Data Warehouse
- Azure SQL Database
- SQL Server in Azure VM
- HBase/Phoenix on HDInsight
- Hive LLAP on HDInsight
- Azure Analysis Services
- Azure Cosmos DB

These options provide various database models that are optimized for different types of tasks:

- Key/value databases hold a single serialized object for each key value. They're good for storing large volumes of data where you want to get one item for a given key value and you don't have to query based on other properties of the item.
- Document databases are key/value databases in which the values are *documents*. A "document" in this context is a collection of named fields and values. The database typically stores the data in a format such as XML, YAML, JSON, or BSON, but may use plain text. Document databases can query on non-key fields and define secondary indexes to make querying more efficient. This makes a document database more suitable for applications that need to retrieve data based on criteria more complex than the value of the document key. For example, you could query on fields such as product ID, customer ID, or customer name.
- Column-family databases are key/value data stores that structure data storage into collections of related columns called column families. For example, a census database might have one group of columns for a person's name (first, middle, last), one group for the person's address, and one group for the person's profile information (data of birth, gender). The database can store each column family in a separate partition, while keeping all of the data

for one person related to the same key. An application can read a single column family without reading through all of the data for an entity.

- Graph databases store information as a collection of objects and relationships. A graph database can efficiently perform queries that traverse the network of objects and the relationships between them. For example, the objects might be employees in a human resources database, and you might want to facilitate queries such as "find all employees who directly or indirectly work for Scott."

Key selection criteria

To narrow the choices, start by answering these questions:

- Do you need serving storage that can serve as a hot path for your data? If yes, narrow your options to those that are optimized for a speed serving layer.
- Do you need massively parallel processing (MPP) support, where queries are automatically distributed across several processes or nodes? If yes, select an option that supports query scale out.
- Do you prefer to use a relational data store? If so, narrow your options to those with a relational database model. However, note that some non-relational stores support SQL syntax for querying, and tools such as PolyBase can be used to query non-relational data stores.

Capability matrix

The following tables summarize the key differences in capabilities.

General capabilities

Capability	SQL Database	SQL Data Warehouse	HBase/Phoenix on HDInsight	Hive LLAP on HDInsight	Azure Analysis Services	Cosmos DB
Is managed service	Yes	Yes	Yes [1]	Yes [1]	Yes	Yes
Primary database model	Relational (columnar format when using columnstore indexes)	Relational tables with columnar storage	Wide column store	Hive/In-Memory	Tabular/MOLAP semantic models	Document store, graph, key-value store, wide column store
SQL language support	Yes	Yes	Yes (using Phoenix JDBC driver)	Yes	No	Yes
Optimized for speed serving layer	Yes [2]	No	Yes	Yes	No	Yes

[1] With manual configuration and scaling.

[2] Using memory-optimized tables and hash or nonclustered indexes.

Scalability capabilities

Capability	SQL Database	SQL Data Warehouse	HBase/Phoenix on HDInsight	Hive LLAP on HDInsight	Azure Analysis Services	Cosmos DB
Redundant regional servers for high availability	Yes	Yes	Yes	No	No	Yes
Supports query scale out	No	Yes	Yes	Yes	Yes	Yes
Dynamic scalability (scale up)	Yes	Yes	No	No	Yes	Yes
Supports in-memory caching of data	Yes	Yes	No	Yes	Yes	No

Security capabilities

Capability	SQL Database	SQL Data Warehouse	HBase/Phoenix on HDInsight	Hive LLAP on HDInsight	Azure Analysis Services	Cosmos DB
Authentication	SQL / Azure Active Directory (Azure AD)	SQL / Azure AD	local / Azure AD [1]	local / Azure AD [1]	Azure AD	database users / Azure AD via access control (IAM)
Data encryption at rest	Yes [2]	Yes [2]	Yes [1]	Yes [1]	Yes	Yes
Row-level security	Yes	Yes [3]	Yes [1]	Yes [1]	Yes (through object-level security in model)	No
Supports firewalls	Yes	Yes	Yes [4]	Yes [4]	Yes	Yes
Dynamic data masking	Yes	No	Yes [1]	Yes	No	No

[1] Requires using a domain-joined HDInsight cluster.

[2] Requires using transparent data encryption (TDE) to encrypt and decrypt your data at rest.

[3] Filter predicates only. See Row-Level Security

[4] When used within an Azure Virtual Network. See Extend Azure HDInsight using an Azure Virtual Network.