Criteria for choosing a data store

06/01/2018 • 8 minutes to read • +1

In this article

Azure supports many types of data storage solutions, each providing different features and capabilities. This article describes the comparison criteria you should use when evaluating a data store. The goal is to help you determine which data storage types can meet your solution's requirements.

General Considerations

To start your comparison, gather as much of the following information as you can about your data needs. This information will help you to determine which data storage types will meet your needs.

Functional requirements

- **Data format**. What type of data are you intending to store? Common types include transactional data, JSON objects, telemetry, search indexes, or flat files.

- **Data size**. How large are the entities you need to store? Will these entities need to be maintained as a single document, or can they be split across multiple documents, tables, collections, and so forth?

- **Scale and structure**. What is the overall amount of storage capacity you need? Do you anticipate partitioning your data?

- **Data relationships**. Will your data need to support one-to-many or many-to-many relationships? Are relationships themselves an important part of the data? Will you need to join or otherwise combine data from within the same dataset, or from external datasets?

- **Consistency model**. How important is it for updates made in one node to appear in other nodes, before further changes can be made? Can you accept eventual consistency? Do you need ACID guarantees for transactions?

- **Schema flexibility**. What kind of schemas will you apply to your data? Will you use a fixed schema, a schema-on-write approach, or a schema-on-read approach?

- **Concurrency**. What kind of concurrency mechanism do you want to use when updating and synchronizing data? Will the application perform many updates that could potentially conflict. If so, you may require record locking and pessimistic concurrency control. Alternatively, can you support optimistic concurrency controls? If so, is simple timestamp-based concurrency control enough, or do you need the added functionality of multi-version concurrency control?

- **Data movement**. Will your solution need to perform ETL tasks to move data to other stores or data warehouses?

- **Data lifecycle**. Is the data write-once, read-many? Can it be moved into cool or cold storage?

- **Other supported features**. Do you need any other specific features, such as schema validation, aggregation, indexing, full-text search, MapReduce, or other query capabilities?

Non-functional requirements

- **Performance and scalability**. What are your data performance requirements? Do you have specific requirements for data ingestion rates and data processing rates? What are the acceptable response times for querying and aggregation of data once ingested? How large will you need the data store to scale up? Is your workload more read-heavy or write-heavy?

- **Reliability**. What overall SLA do you need to support? What level of fault-tolerance do you need to provide for data consumers? What kind of backup and restore capabilities do you need?

- **Replication**. Will your data need to be distributed among multiple replicas or regions? What kind of data replication capabilities do you require?

- **Limits**. Will the limits of a particular data store support your requirements for scale, number of connections, and throughput?

Management and cost

- **Managed service**. When possible, use a managed data service, unless you require specific capabilities that can only be found in an IaaS-hosted data store.
- **Region availability**. For managed services, is the service available in all Azure regions? Does your solution need to be hosted in certain Azure regions?
- **Portability**. Will your data need to be migrated to on-premises, external datacenters, or other cloud hosting environments?
- **Licensing**. Do you have a preference of a proprietary versus OSS license type? Are there any other external restrictions on what type of license you can use?
- **Overall cost**. What is the overall cost of using the service within your solution? How many instances will need to run, to support your uptime and throughput requirements? Consider operations costs in this calculation. One reason to prefer managed services is the reduced operational cost.
- **Cost effectiveness**. Can you partition your data, to store it more cost effectively? For example, can you move large objects out of an expensive relational database into an object store?

Security

- **Security**. What type of encryption do you require? Do you need encryption at rest? What authentication mechanism do you want to use to connect to your data?
- **Auditing**. What kind of audit log do you need to generate?
- **Networking requirements**. Do you need to restrict or otherwise manage access to your data from other network resources? Does data need to be accessible only from inside the Azure environment? Does the data need to be accessible from specific IP addresses or subnets? Does it need to be accessible from applications or services hosted on-premises or in other external datacenters?

DevOps

- **Skill set**. Are there particular programming languages, operating systems, or other technology that your team is particularly adept at using? Are there others that would be difficult for your team to work with?
- **Clients** Is there good client support for your development languages?

The following sections compare various data store models in terms of workload profile, data types, and example use cases.

Relational database management systems (RDBMS)

Workload

- Both the creation of new records and updates to existing data happen regularly.
- Multiple operations have to be completed in a single transaction.
- Requires aggregation functions to perform cross-tabulation.
- Strong integration with reporting tools is required.
- Relationships are enforced using database constraints.
- Indexes are used to optimize query performance.
- Allows access to specific subsets of data.

Data type

- Data is highly normalized.
- Database schemas are required and enforced.
- Many-to-many relationships between data entities in the database.
- Constraints are defined in the schema and imposed on any data in the database.
- Data requires high integrity. Indexes and relationships need to be maintained accurately.
- Data requires strong consistency. Transactions operate in a way that ensures all data are 100% consistent for all users and processes.
- Size of individual data entries is intended to be small to medium-sized.

Examples

- Line of business (human capital management, customer relationship management, enterprise resource planning)
- Inventory management
- Reporting database
- Accounting
- Asset management
- Fund management
- Order management

Document databases

Workload

- General purpose.
- Insert and update operations are common. Both the creation of new records and updates to existing data happen regularly.
- No object-relational impedance mismatch. Documents can better match the object structures used in application code.
- Optimistic concurrency is more commonly used.
- Data must be modified and processed by consuming application.
- Data requires index on multiple fields.
- Individual documents are retrieved and written as a single block.

Data type

- Data can be managed in de-normalized way.
- Size of individual document data is relatively small.
- Each document type can use its own schema.
- Documents can include optional fields.
- Document data is semi-structured, meaning that data types of each field are not strictly defined.
- Data aggregation is supported.

Examples

- Product catalog
- User accounts
- Bill of materials
- Personalization
- Content management
- Operations data
- Inventory management
- Transaction history data
- Materialized view of other NoSQL stores. Replaces file and Blob indexing.

Key/value stores

Workload

- Data is identified and accessed using a single ID key, like a dictionary.
- Massively scalable.
- No joins, lock, or unions are required.
- No aggregation mechanisms are used.
- Secondary indexes are generally not used.

Data type

- Data size tends to be large.

- Each key is associated with a single value, which is an unmanaged data Blob.
- There is no schema enforcement.
- No relationships between entities.

Examples

- Data caching
- Session management
- User preference and profile management
- Product recommendation and ad serving
- Dictionaries

Graph databases

Workload

- The relationships between data items are very complex, involving many hops between related data items.
- The relationship between data items are dynamic and change over time.
- Relationships between objects are first-class citizens, without requiring foreign-keys and joins to traverse.

Data type

- Data is comprised of nodes and relationships.
- Nodes are similar to table rows or JSON documents.
- Relationships are just as important as nodes, and are exposed directly in the query language.
- Composite objects, such as a person with multiple phone numbers, tend to be broken into separate, smaller nodes, combined with traversable relationships

Examples

- Organization charts
- Social graphs
- Fraud detection
- Analytics
- Recommendation engines

Column-family databases

Workload

- Most column-family databases perform write operations extremely quickly.

- Update and delete operations are rare.
- Designed to provide high throughput and low-latency access.
- Supports easy query access to a particular set of fields within a much larger record.
- Massively scalable.

Data type

- Data is stored in tables consisting of a key column and one or more column families.
- Specific columns can vary by individual rows.
- Individual cells are accessed via get and put commands
- Multiple rows are returned using a scan command.

Examples

- Recommendations
- Personalization
- Sensor data
- Telemetry
- Messaging
- Social media analytics
- Web analytics
- Activity monitoring
- Weather and other time-series data

Search engine databases

Workload

- Indexing data from multiple sources and services.
- Queries are ad-hoc and can be complex.
- Requires aggregation.
- Full text search is required.
- Ad hoc self-service query is required.
- Data analysis with index on all fields is required.

Data type

- Semi-structured or unstructured
- Text
- Text with reference to structured data

Examples

- Product catalogs
- Site search
- Logging
- Analytics

- Shopping sites

Data warehouse

Workload	• Data analytics • Enterprise BI
Data type	• Historical data from multiple sources. • Usually denormalized in a "star" or "snowflake" schema, consisting of fact and dimension tables. • Usually loaded with new data on a scheduled basis. • Dimension tables often include multiple historic versions of an entity, referred to as a *slowly changing dimension*.
Examples	An enterprise data warehouse that provides data for analytical models, reports, and dashboards.

Time series databases

Workload	• An overwhelming proportion of operations (95-99%) are writes. • Records are generally appended sequentially in time order. • Updates are rare. • Deletes occur in bulk, and are made to contiguous blocks or records. • Read requests can be larger than available memory. • It's common for multiple reads to occur simultaneously. • Data is read sequentially in either ascending or descending time order.
Data type	• A timestamp that is used as the primary key and sorting mechanism. • Measurements from the entry or descriptions of what the entry represents. • Tags that define additional information about the type, origin, and other information about the entry.
Examples	• Monitoring and event telemetry. • Sensor or other IoT data.

Object storage

Workload	• Identified by key. • Objects may be publicly or privately accessible. • Content is typically an asset such as a spreadsheet, image, or video file. • Content must be durable (persistent), and external to any application tier or virtual machine.
Data type	• Data size is large. • Blob data. • Value is opaque.
Examples	• Images, videos, office documents, PDFs • CSS, Scripts, CSV • Static HTML, JSON • Log and audit files • Database backups

Shared files

Workload	• Migration from existing apps that interact with the file system. • Requires SMB interface.
Data type	• Files in a hierarchical set of folders. • Accessible with standard I/O libraries.
Examples	• Legacy files • Shared content accessible among a number of VMs or app instances

Is this page helpful?

Yes No