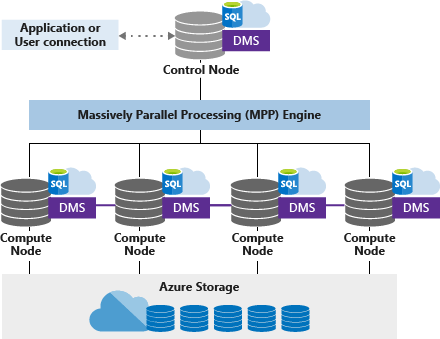
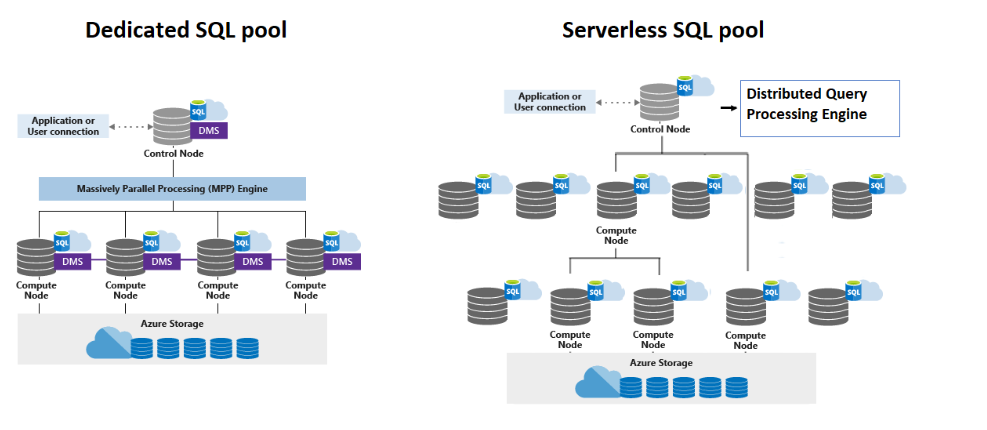
**Dedicated SQL pool (formerly SQL DW) architecture in Azure Synapse Analytics**

Synapse is an analytics service that brings together enterprise data warehousing and Big Data analytics. It gives you the freedom to query data on your terms.

**Synapse SQL architecture components**

Dedicated SQL pool (formerly SQL DW) leverages a scale-out architecture to distribute computational processing of data across multiple nodes. The unit of scale is an abstraction of compute power that is known as a data warehouse unit. Compute is separate from storage, which enables you to scale compute independently of the data in your system.





Dedicated SQL pool (formerly SQL DW) uses a node-based architecture. Applications connect and issue T-SQL commands to a Control node. The Control node hosts the distributed query engine, which optimizes queries for parallel processing, and then passes operations to Compute nodes to do their work in parallel.

The Compute nodes store all user data in Azure Storage and run the parallel queries. The Data Movement Service (DMS) is a system-level internal service that moves data across the nodes as necessary to run queries in parallel and return accurate results.

With decoupled storage and compute, when using a dedicated SQL pool (formerly SQL DW) one can:

* Independently size compute power irrespective of your storage needs.
* Grow or shrink compute power, within a dedicated SQL pool (formerly SQL DW), without moving data.
* Pause compute capacity while leaving data intact, so you only pay for storage.
* Resume compute capacity during operational hours.

**Azure Storage**

Dedicated SQL pool SQL (formerly SQL DW) leverages Azure Storage to keep your user data safe. Since your data is stored and managed by Azure Storage, there is a separate charge for your storage consumption. The data is sharded into **distributions** to optimize the performance of the system. You can choose which sharding pattern to use to distribute the data when you define the table. These sharding patterns are supported:

* Hash
* Round Robin
* Replicate

**Control node**

The Control node is the brain of the architecture. It is the front end that interacts with all applications and connections. The distributed query engine runs on the Control node to optimize and coordinate parallel queries. When you submit a T-SQL query, the Control node transforms it into queries that run against each distribution in parallel.

**Compute nodes**

The Compute nodes provide the computational power. Distributions map to Compute nodes for processing. As you pay for more compute resources, distributions are remapped to available Compute nodes. The number of compute nodes ranges from 1 to 60, and is determined by the service level for Synapse SQL.

Each Compute node has a node ID that is visible in system views. You can see the Compute node ID by looking for the node\_id column in system views whose names begin with sys.pdw\_nodes..

**Data Movement Service**

Data Movement Service (DMS) is the data transport technology that coordinates data movement between the Compute nodes. Some queries require data movement to ensure the parallel queries return accurate results. When data movement is required, DMS ensures the right data gets to the right location.

**Distributions**

A distribution is the basic unit of storage and processing for parallel queries that run on distributed data. When Synapse SQL runs a query, the work is divided into 60 smaller queries that run in parallel.

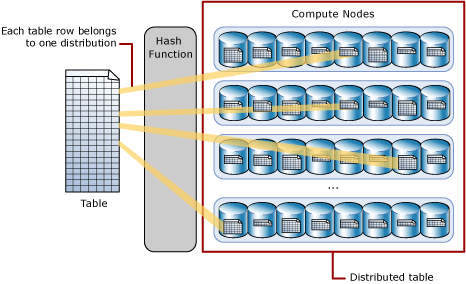
Each of the 60 smaller queries runs on one of the data distributions. Each Compute node manages one or more of the 60 distributions. A dedicated SQL pool (formerly SQL DW) with maximum compute resources has one distribution per Compute node. A dedicated SQL pool (formerly SQL DW) with minimum compute resources has all the distributions on one compute node.

**Hash-distributed tables**

A hash distributed table can deliver the highest query performance for joins and aggregations on large tables.

To shard data into a hash-distributed table, a hash function is used to deterministically assign each row to one distribution. In the table definition, one of the columns is designated as the distribution column. The hash function uses the values in the distribution column to assign each row to a distribution.

The following diagram illustrates how a full (non-distributed table) gets stored as a hash-distributed table.



* Each row belongs to one distribution.
* A deterministic hash algorithm assigns each row to one distribution.
* The number of table rows per distribution varies as shown by the different sizes of tables.

There are performance considerations for the selection of a distribution column, such as distinctness, data skew, and the types of queries that run on the system.

**Round-robin distributed tables**

A round-robin table is the simplest table to create and delivers fast performance when used as a staging table for loads.

A round-robin distributed table distributes data evenly across the table but without any further optimization. A distribution is first chosen at random and then buffers of rows are assigned to distributions sequentially. It is quick to load data into a round-robin table, but query performance can often be better with hash distributed tables. Joins on round-robin tables require reshuffling data, which takes additional time.

**Replicated Tables**

A replicated table provides the fastest query performance for small tables.

A table that is replicated caches a full copy of the table on each compute node. Consequently, replicating a table removes the need to transfer data among compute nodes before a join or aggregation. Replicated tables are best utilized with small tables. Extra storage is required and there is additional overhead that is incurred when writing data, which make large tables impractical.

The diagram below shows a replicated table that is cached on the first distribution on each compute node.

