

Azure Data Warehouse In-A-Day

Steve Young - Data & AI CSA
Rebecca Young - Data & AI CSA



PRESENTATION OBJECTIVES

01 Introduction to SQL DW

- Introduction to the Day and Logistics
 - Washrooms, Lunch, EOD, Azure Pass, Keep DW Small
- How SQL DW fits into the Azure Data Services
- SQL DW in the modern data solution architecture
- Various Important Features and definitions

Agenda

Agenda:

Time	Topic	Description	Materials
09:00am - 09:15am	Introductions & Logistics (15min)	Welcome	N/A
09:15am - 10:00am	Datawarehouse Patterns in Azure & SQL DW Overview (45min)	Slide Deck 01	N/A
10:00am - 10:45am	SQL DW Gen2 New Features & Planning Your Project Build (45min)	Slide Deck 02	N/A
10:45am - 11:00pm	Break (15min)	Please take a break	N/A
11:00am - 12:00pm	Demo & Lab 01 (60 Min)	Setting up the LAB environment	Lab 01
12:00pm - 1:00pm	Lunch (60 Min)	Lunch and complete lab 01	N/A
01:00pm - 1:30pm	SQLDW Loading Best Practices (30 Min)	Lecture	N/A
01:30pm - 02:15pm	Lab 02/03: User IDs & Data loading scenarios and best practices (45min)	Loading different scenarios	Lab 02/03
02:15am - 2:30pm	Break (15min)	Please take a break	N/A
02:30pm - 3:00pm	SQLDW Operational Best Practices (30 Min)	Lecture	N/A
03:00pm - 03:45pm	Lab 04: Performance Tuning best practices (45min)		Lab 04
03:45pm - 4:15pm	Lab 05: Lab 3: Monitoring, Maintenance and Security (30min)		Lab 05
4:15pm - 5:00pm	Q&A and Wrap-up (45min)	final remarks or takeaways/next steps	Survey

Important Info

AzurePass Instructions - <https://www.microsoftazurepass.com/Home/HowTo>

Azure Pass - <https://www.microsoftazurepass.com/>

File Location - <https://github.com/steveyoungca/SQLDWinaDayWorkshop>



Microsoft Azure

Productive + Hybrid + Open + Trusted

Azure and the Modern Data Estate

Modern Environment will have many tools → Use the open source tools you like

DevOps

Nagios



Management



Applications



Pivotal



App frameworks & tools



nodeJS



Databases & middleware



cloudera



Couchbase



Infrastructure



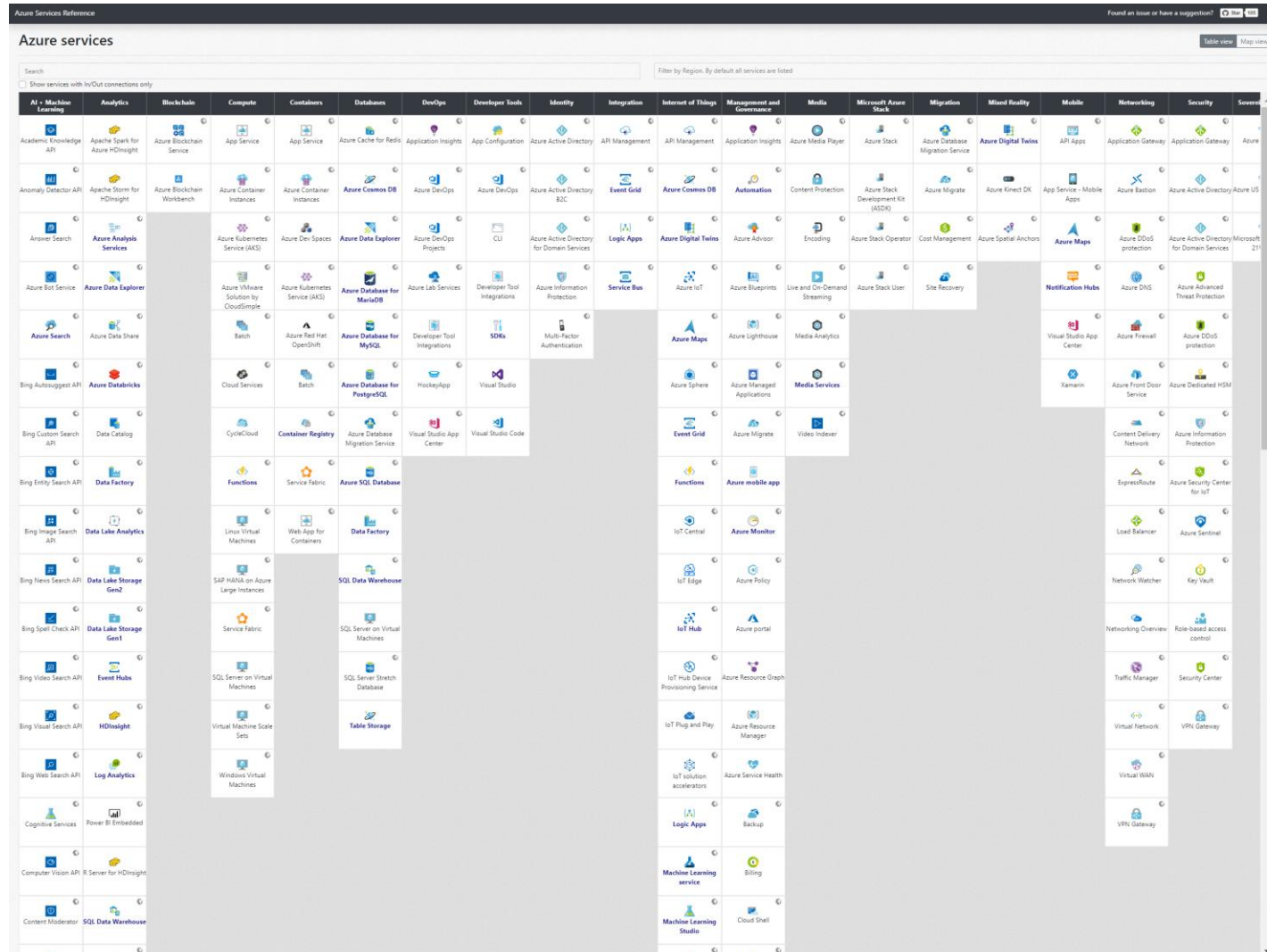
ORACLE
LINUX



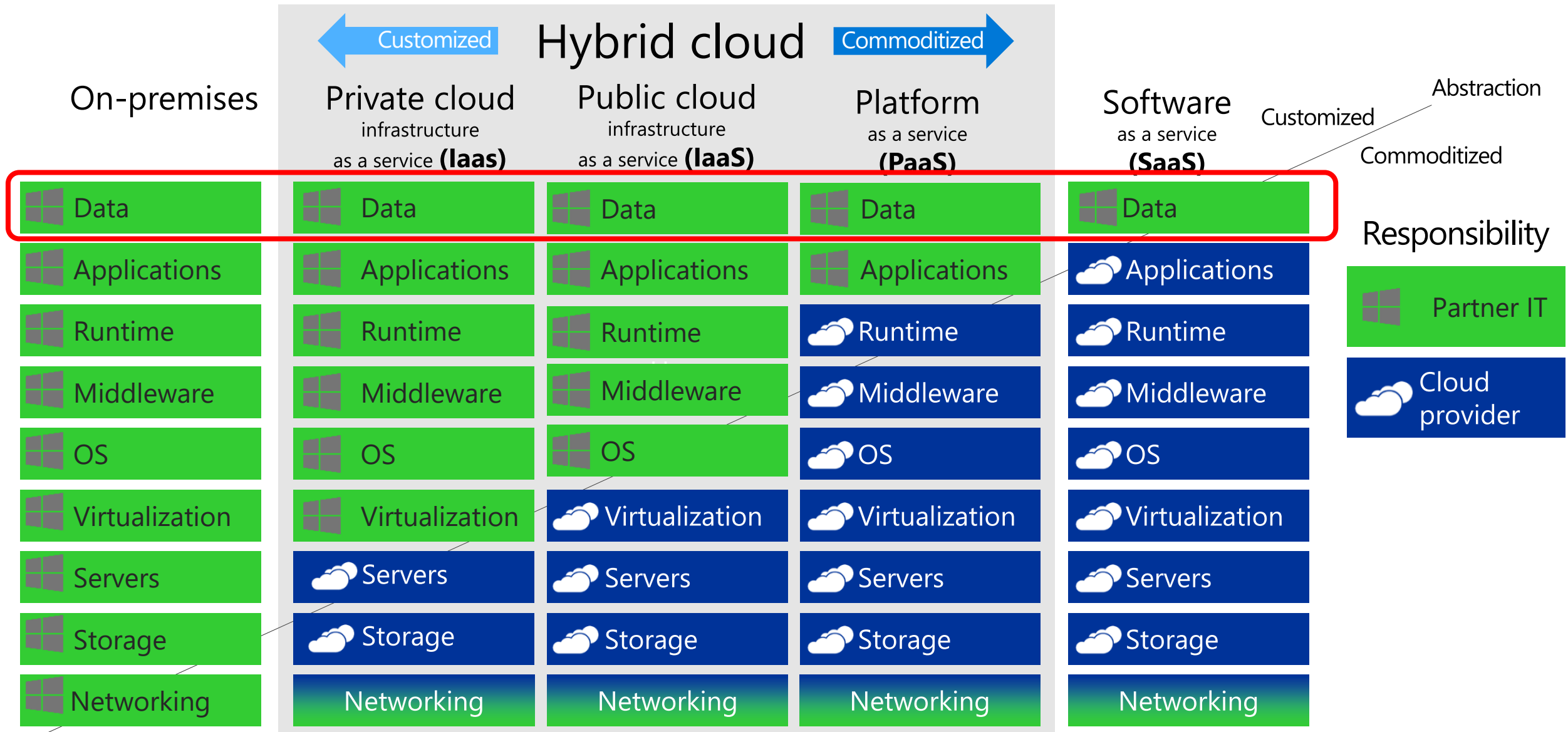
<https://nnmer.github.io/azure-services-map/dist/>

Anton Romanov - <https://github.com/nmmer>

Azure Services Map



Moving up the stack

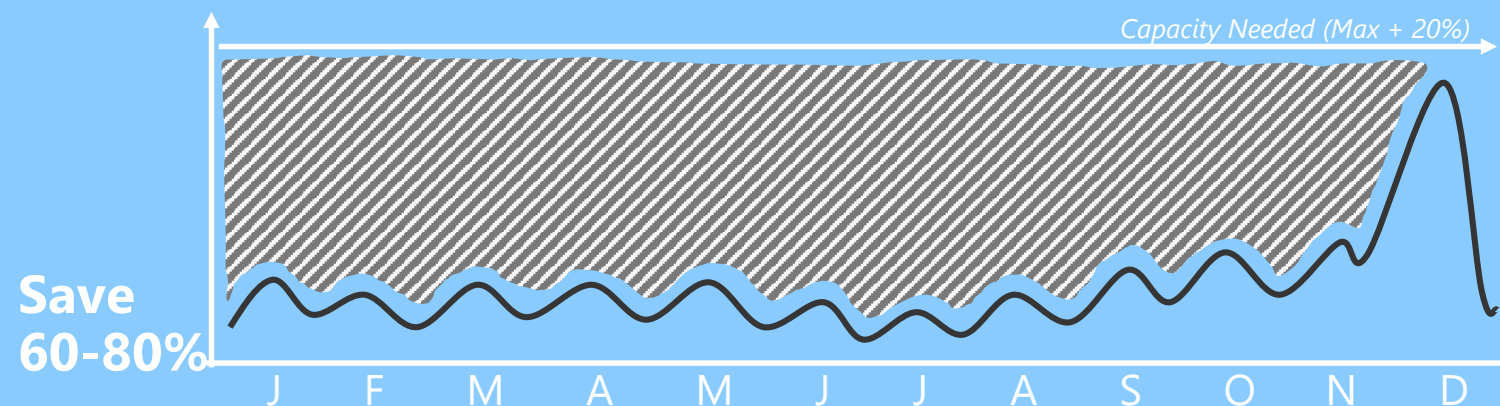
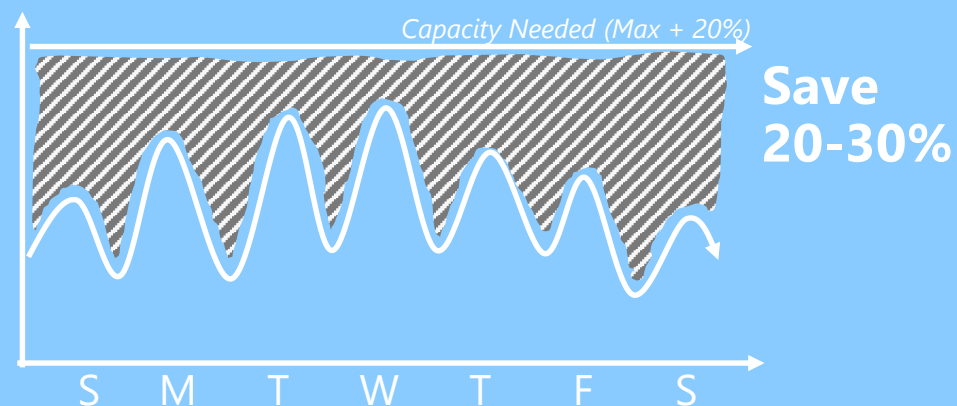
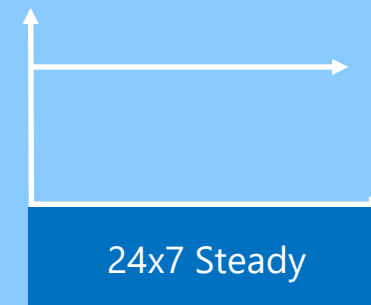
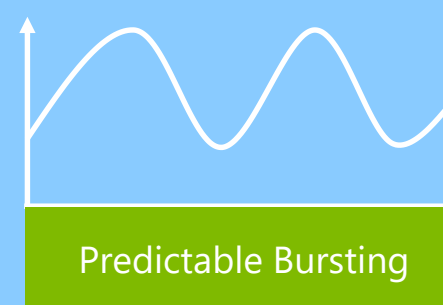
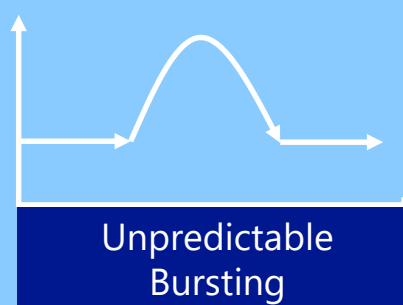
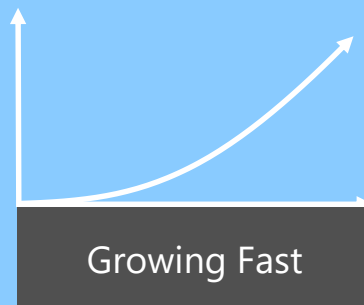
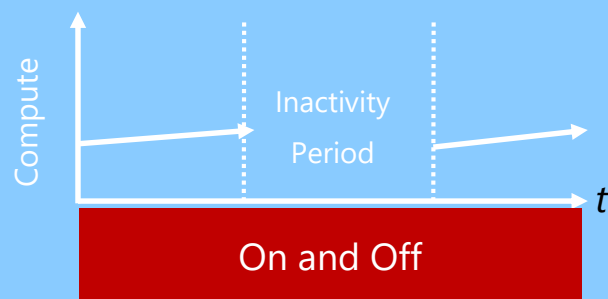


Cloud principles

A person is seen from the side, sitting at a desk and looking at a large computer monitor. The monitor displays a complex dashboard with various charts, including a pie chart and several tables of data. The person is wearing a light-colored shirt. The background is a blurred office environment with some framed pictures on the wall.

- **Freedom of choice**
- **Marketplaces**
- **Cloud Inspired Infrastructure**
- **Multi Vendor**
- **Hybrid**
- **Hyper scale**
- **Self-service**
- **Build in and on top of Security**
- **Build in Compliancy**
- **Automation**
- **Continuous Change**
- **Software defined**
- **Scalable**
- **Pay per Use**
- **Lock in Reduction**
- **Open- and closed source**

Saving cost



On and Off
(30%)

Growing Fast
(15%)

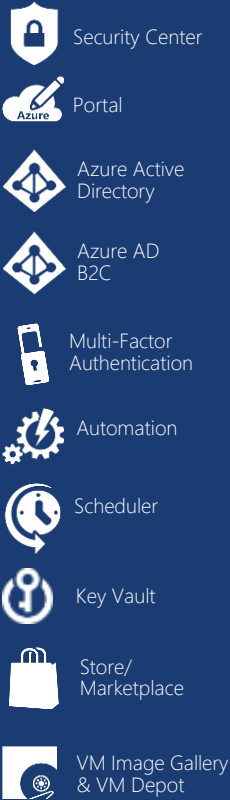
Unpredictable Bursting
(25%)

Predictable Bursting
(20%)

24x7 Steady
(10%)

Your Application Portfolio – What Does it Look Like..?

Security & Management



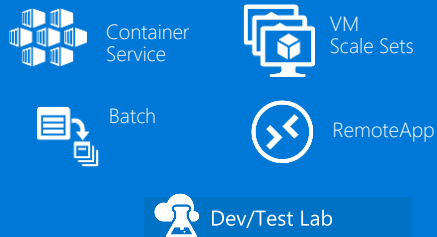
Media & CDN



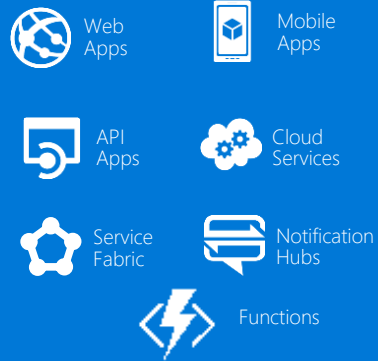
Integration



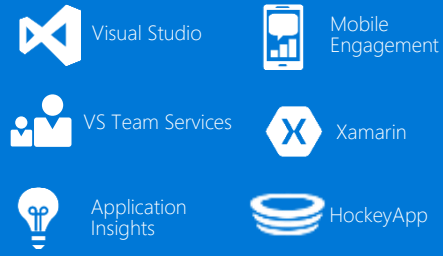
Compute Services



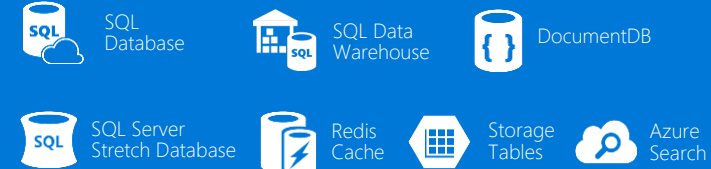
Application Platform



Developer Services



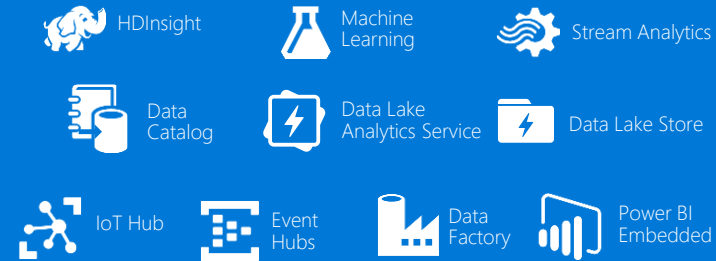
Data



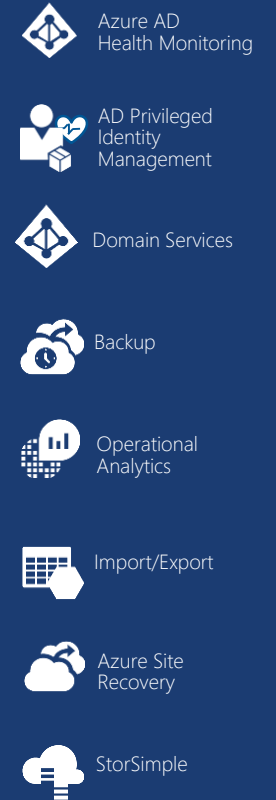
Intelligence



Analytics & IoT



Hybrid Cloud



Infrastructure Services

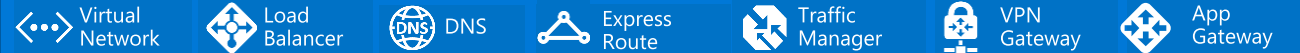
Compute



Storage



Networking



Datacenter Infrastructure



Data services



SQL Database



DocumentDB



Data Warehouse



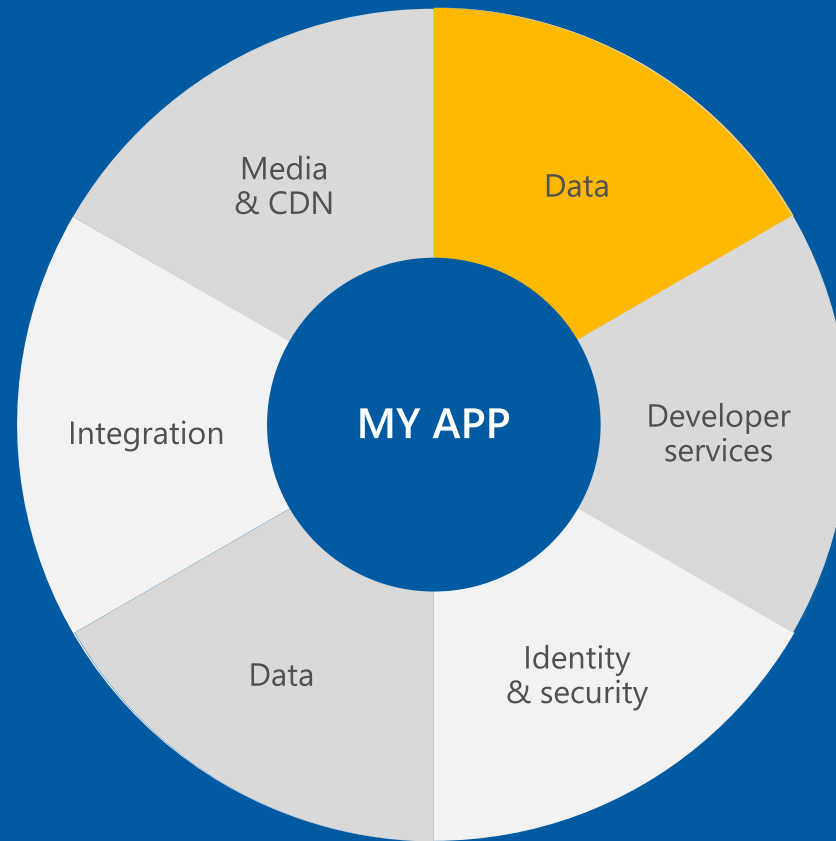
Storage Tables



Azure Search
























Redis Cache

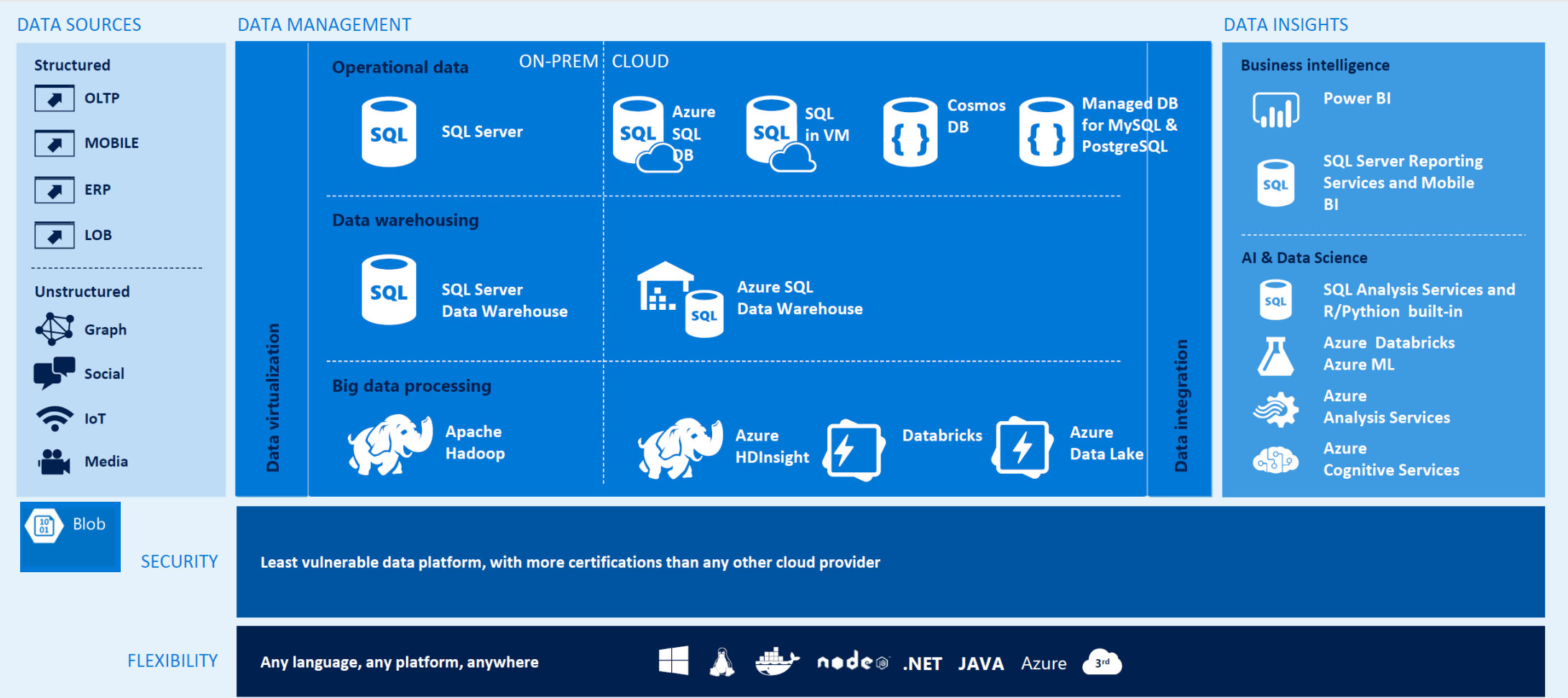


- ✓ Database as a service
- ✓ SQL/no-SQL data stores
- ✓ Big Data capabilities
- ✓ Search
- ✓ Caching

Microsoft Azure BI & Analytics services portfolio

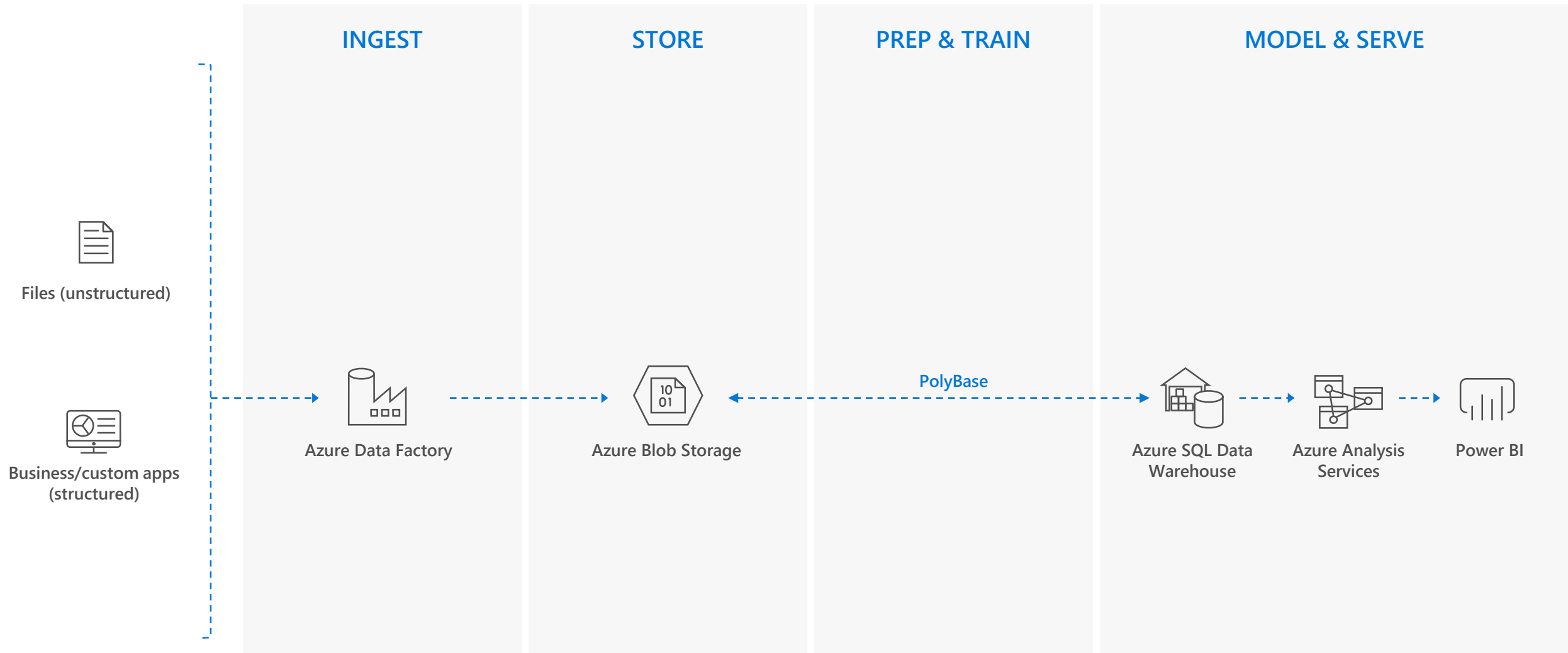
Devices	Device Connectivity	Storage	Analytics	Presentation & Action
	 IoT Hub	 SQL Database	 Machine Learning	 App Service
	 Event Hub	 Table/Blob Storage	 Stream Analytics	 Power BI
	 Service Bus	 DocumentDB	 HDInsight	 Notification Hubs
	 External Data Sources	 3 rd party Databases	 Data Factory	 Mobile Services
			 Data Lake	 BizTalk Services

Azure and Data Estate Data View



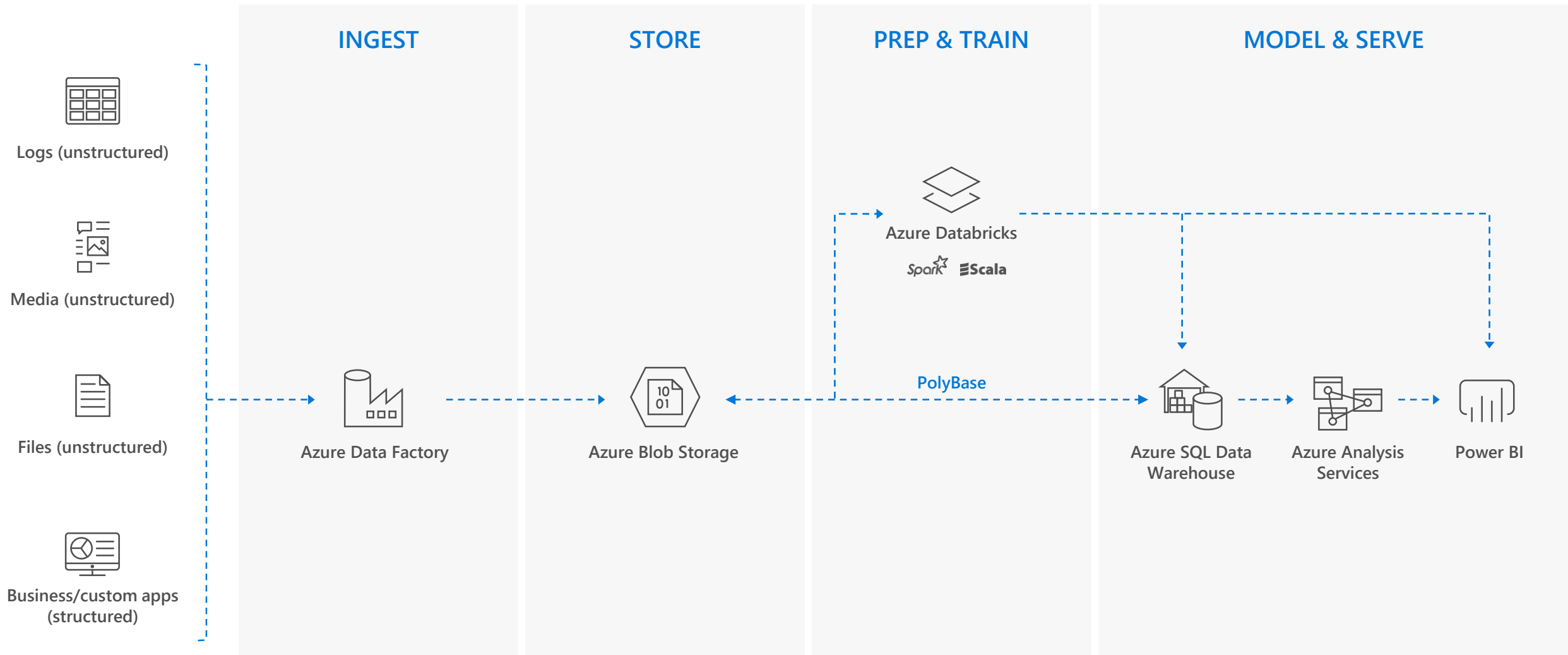
Modern Data Solution Architectures

CLOUD DATA WAREHOUSE



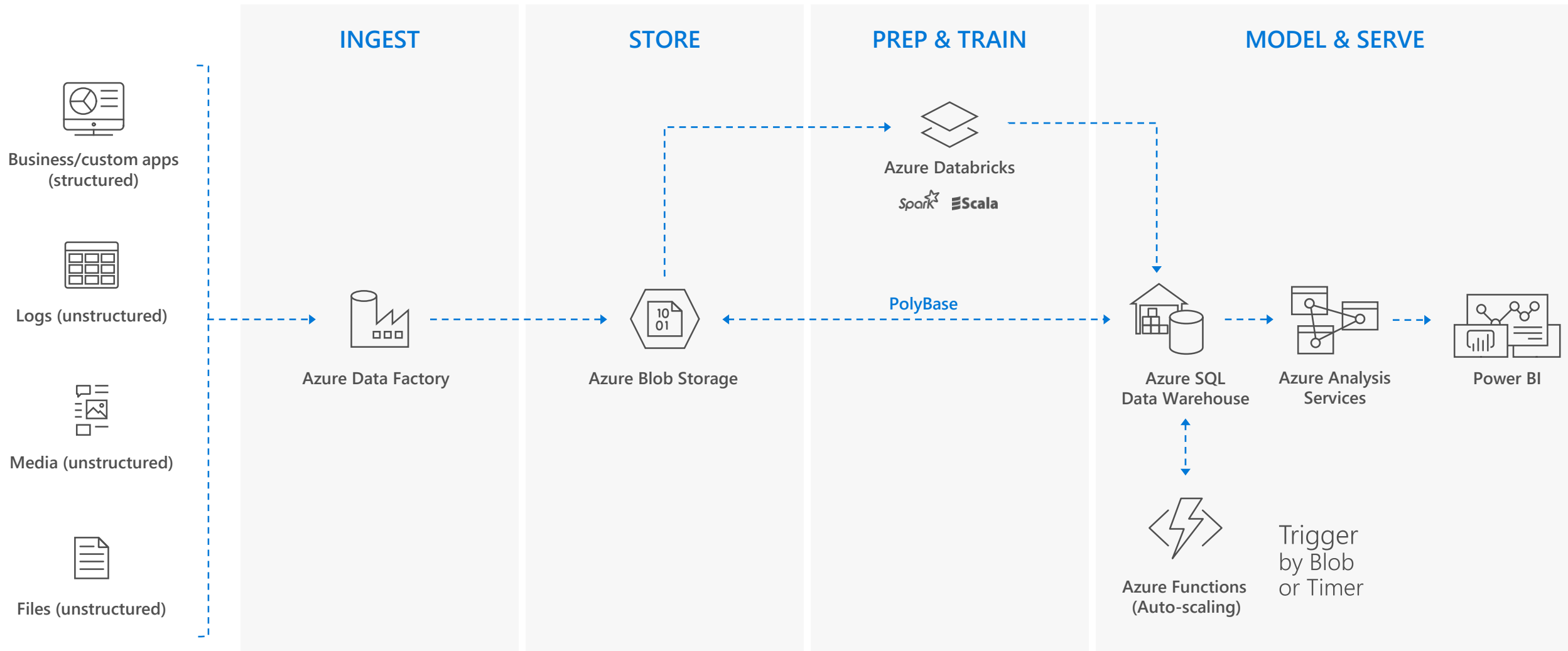
Microsoft Azure also supports other Big Data services like Azure HDInsight and Azure Data Lake to allow customers to tailor the above architecture to meet their unique needs.

MODERN DATA WAREHOUSE



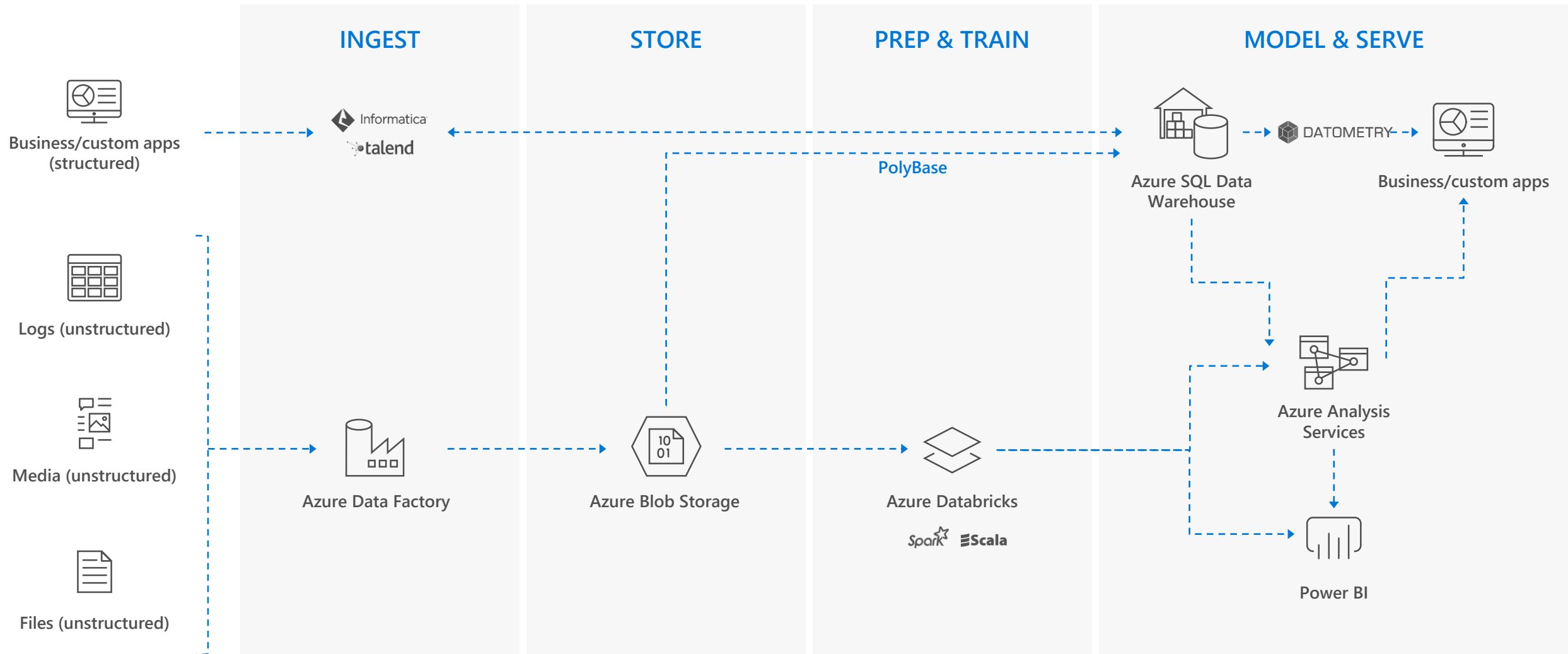
Microsoft Azure also supports other Big Data services like Azure HDInsight and Azure Data Lake to allow customers to tailor the above architecture to meet their unique needs.

AUTO SCALING DATA WAREHOUSE



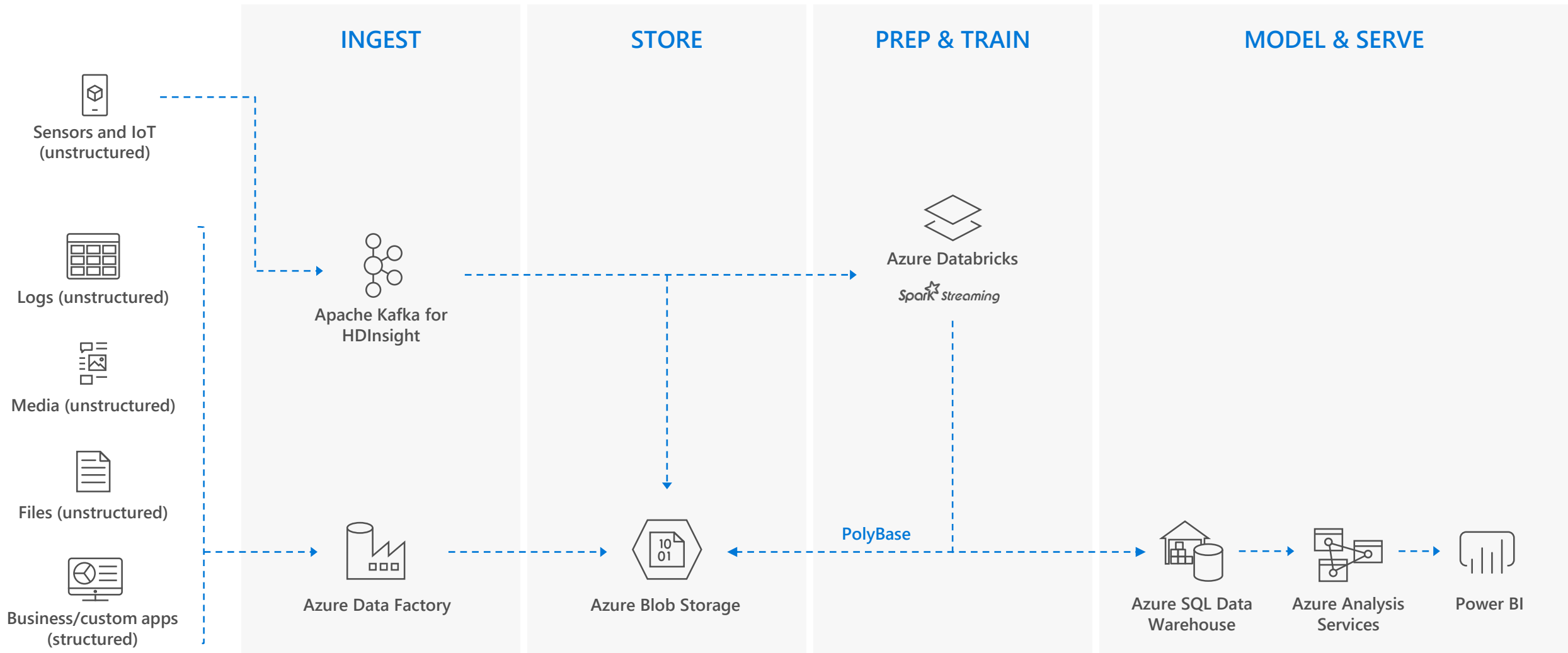
Use Azure Functions to manage compute resources in Azure SQL Data Warehouse - <https://docs.microsoft.com/en-us/azure/sql-data-warehouse/manage-compute-with-azure-functions>

DATA WAREHOUSE MIGRATION



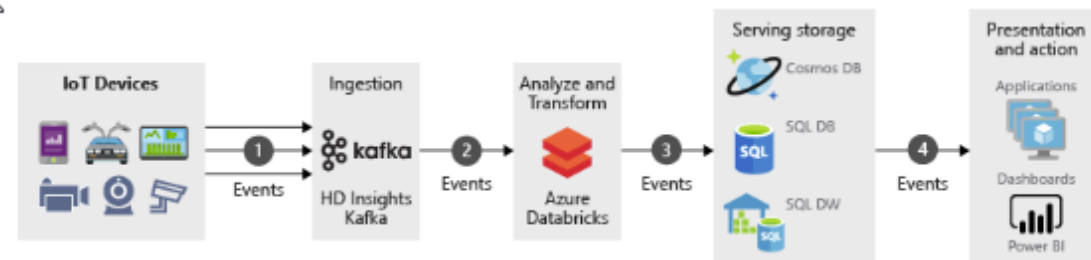
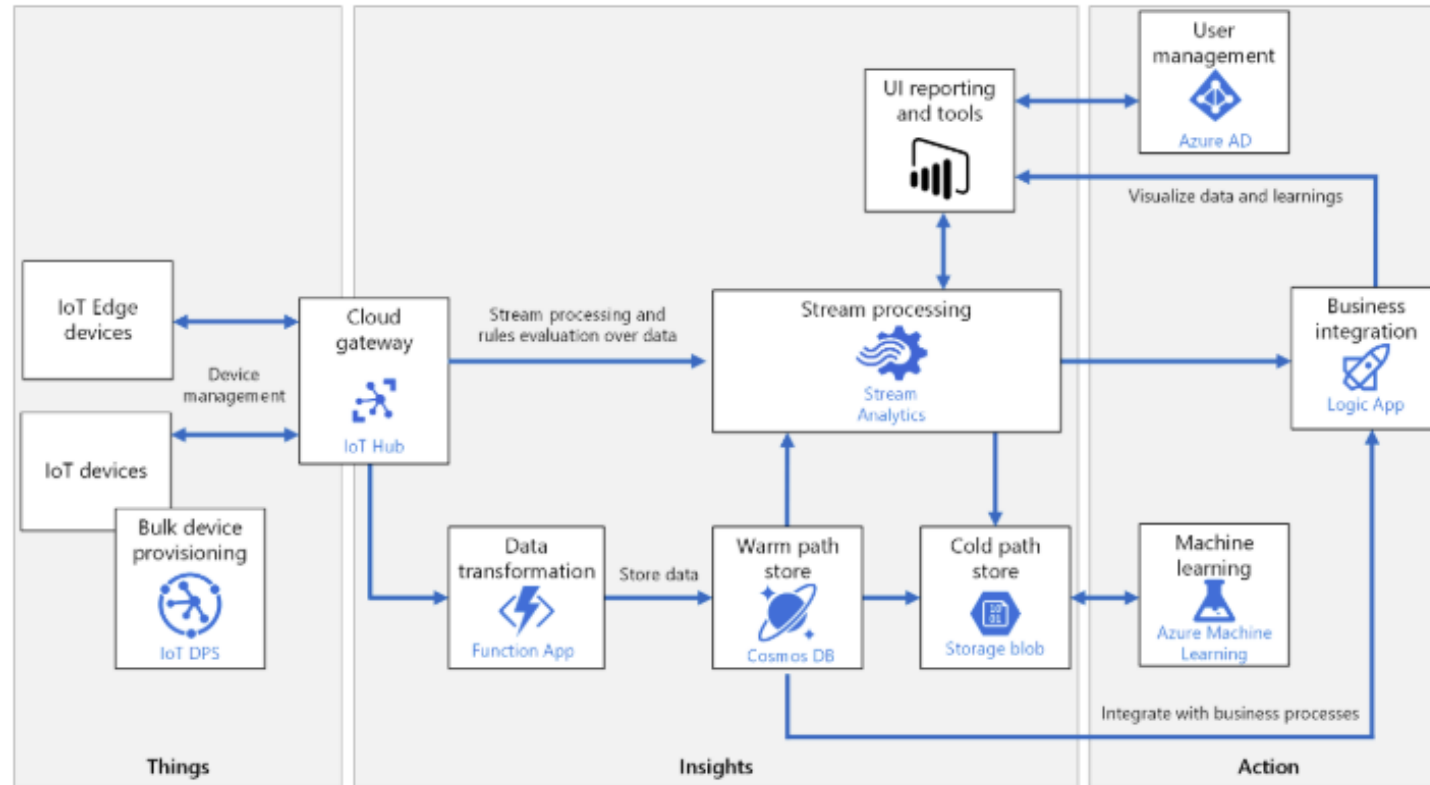
Azure also supports other Big Data services like Azure HDInsight and Azure Data Lake to allow customers to tailor the architecture to meet their unique needs.

REAL TIME ANALYTICS

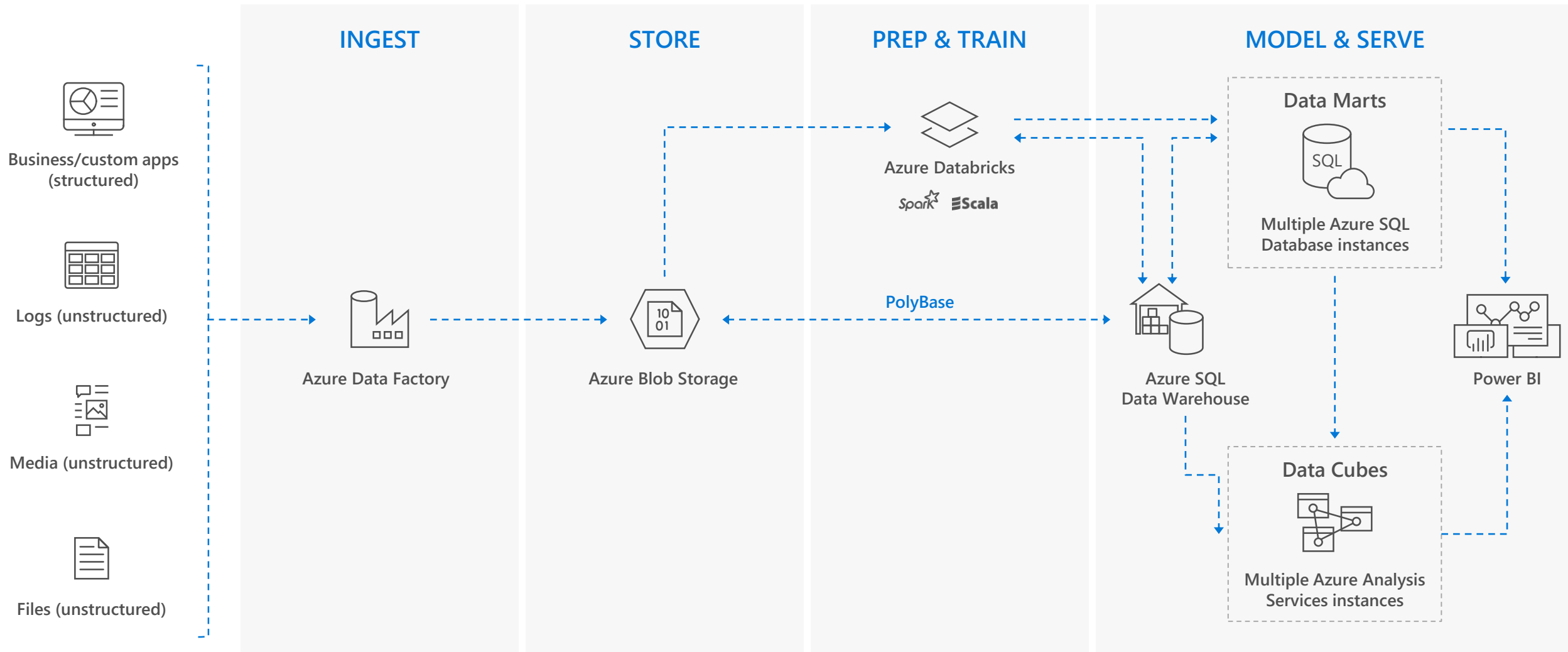


Microsoft Azure also supports other Big Data services like Azure IoT Hub, Azure Event Hubs, Azure Machine Learning and Azure Data Lake to allow customers to tailor the above architecture to meet their unique needs.

REAL TIME ANALYTICS



HUB & SPOKE ARCHITECTURE FOR BI



CONSUMPTION PATTERNS

Pattern 1: Direct Access

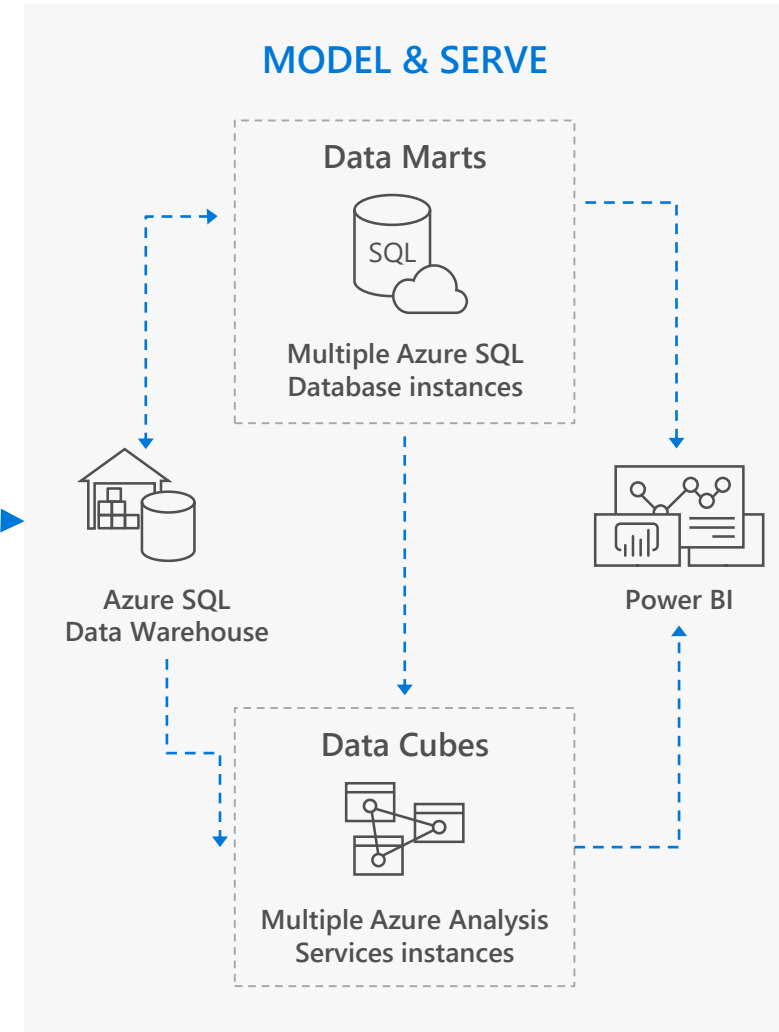
- PBI Direct Query
- Client Tools (SSMA, Data Tools etc.)
- Third-party reporting tools (Talend, Qlik etc.)

Pattern 2: Semantic Layer.

- Tabular models using AAS
- OLAP models through SSAS
- Data Marts (SQL DB, SQL VM etc.)

Pattern 3: Translation Layer

- APIs for custom apps



Suitable workloads

Analytics

- Store large volumes of data.
- Consolidate disparate data into a single location.
- Shape, model, transform and aggregate data.
- Perform query analysis across large datasets.
- Ad-hoc reporting across large data volumes.
- All using simple SQL constructs.

“SQL on SQL”

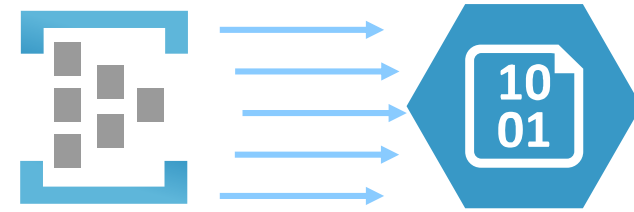
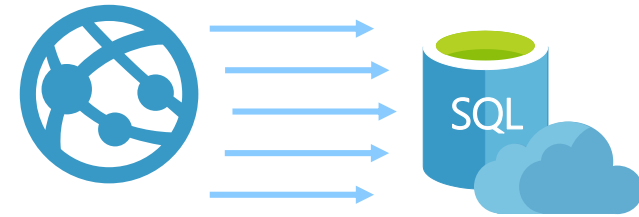
Unsuitable workloads

Operational workloads (OLTP)

High frequency reads and writes.

Large numbers of singleton selects.

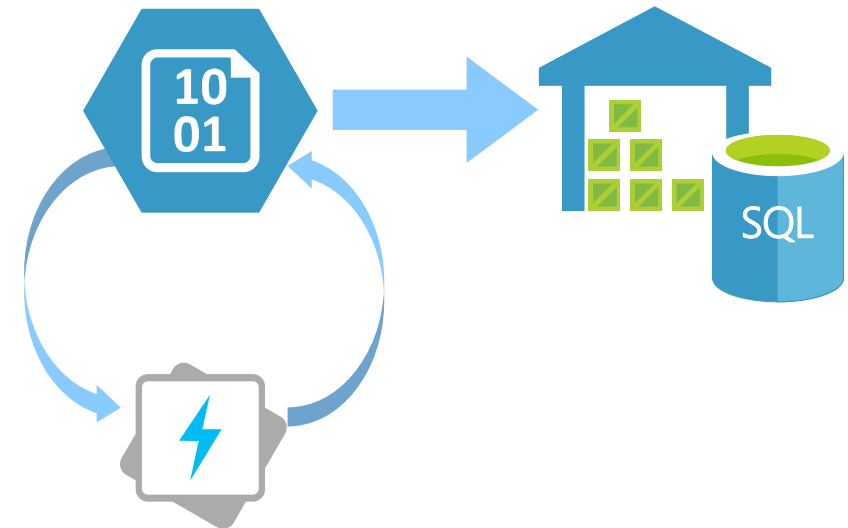
High volumes of single row inserts.



Data Preparation

Row by row processing needs.

Incompatible formats (JSON, XML).

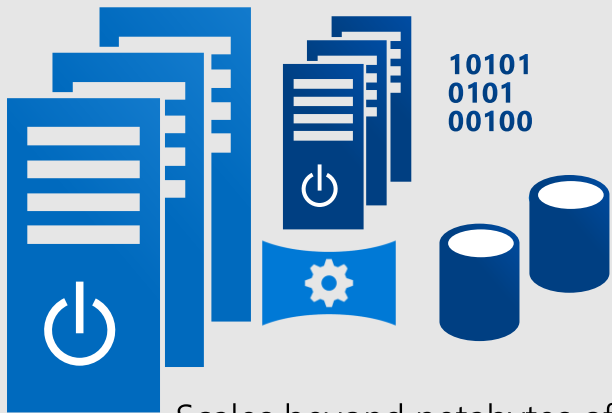


Azure SQL Data Warehouse Architecture & Overview

Azure SQL DW Service

A relational data warehouse-as-a-service, fully managed by Microsoft.
Industries first elastic cloud data warehouse with proven SQL Server capabilities.
Support your smallest to your largest data storage needs.

Elastic scale & performance

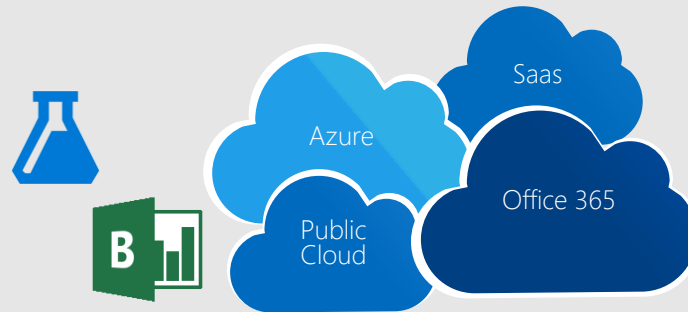


- Scales beyond petabytes of data
- Cloud DW Scale-out Processing
- Instant-on compute scales in seconds
- Query Relational / Non-Relational

Powered by the Cloud

Get started in minutes

Integrated with Azure ML, PowerBI, ADB & ADF



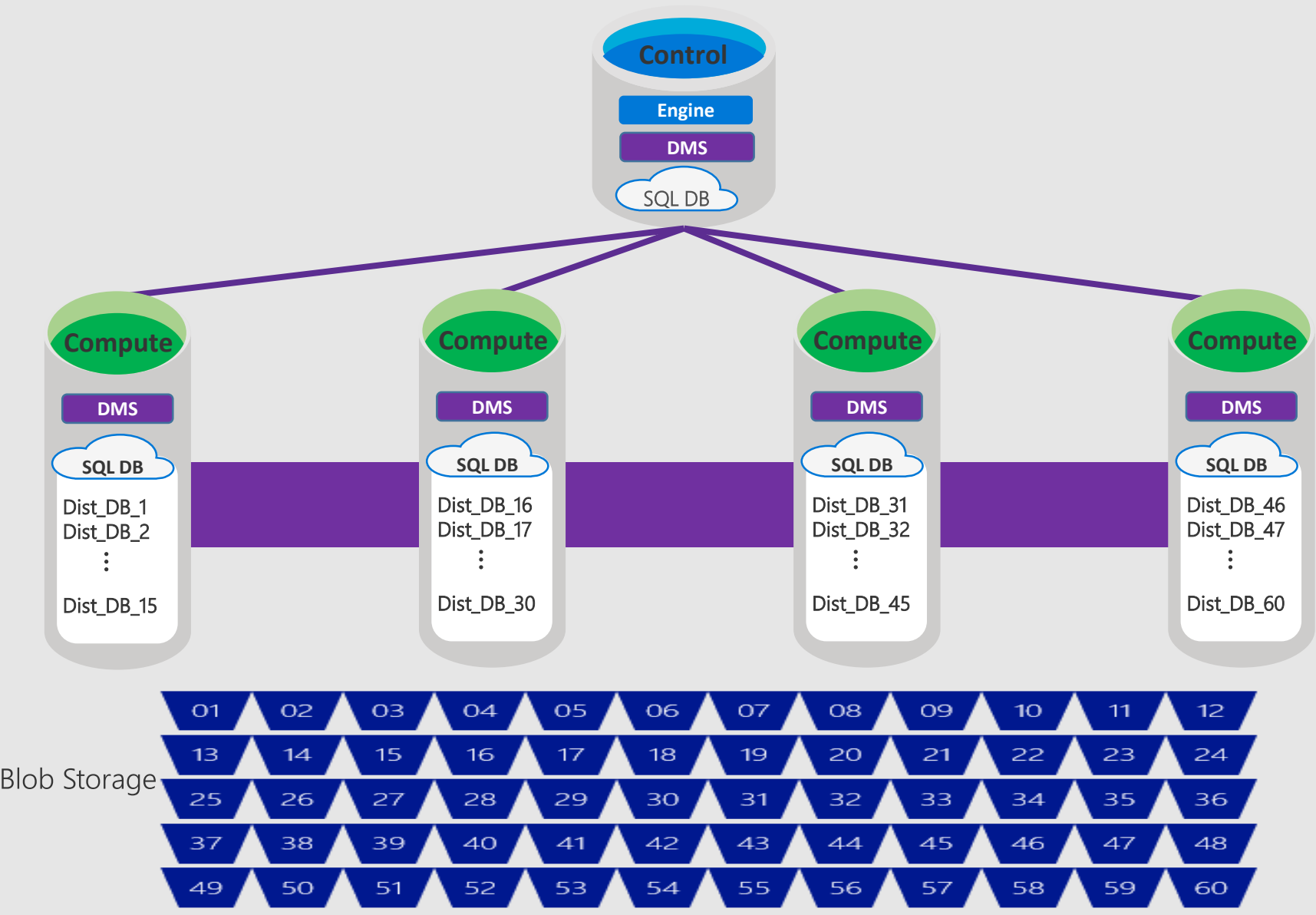
Market Leading Price & Performance



Simple billing compute & storage

Pay for what you need, when you need it
with dynamic pause

SQL DW Fundamentals



Control

Connection and tool endpoint.
Coordinates compute activity.

Compute

Handles query processing,
ability to scale up/down

DMS: Data Movement Services

Coordinates data movement
between nodes

Storage

Remote storage. Scales
independently of compute

Azure SQL Data Warehouse Architecture & Overview

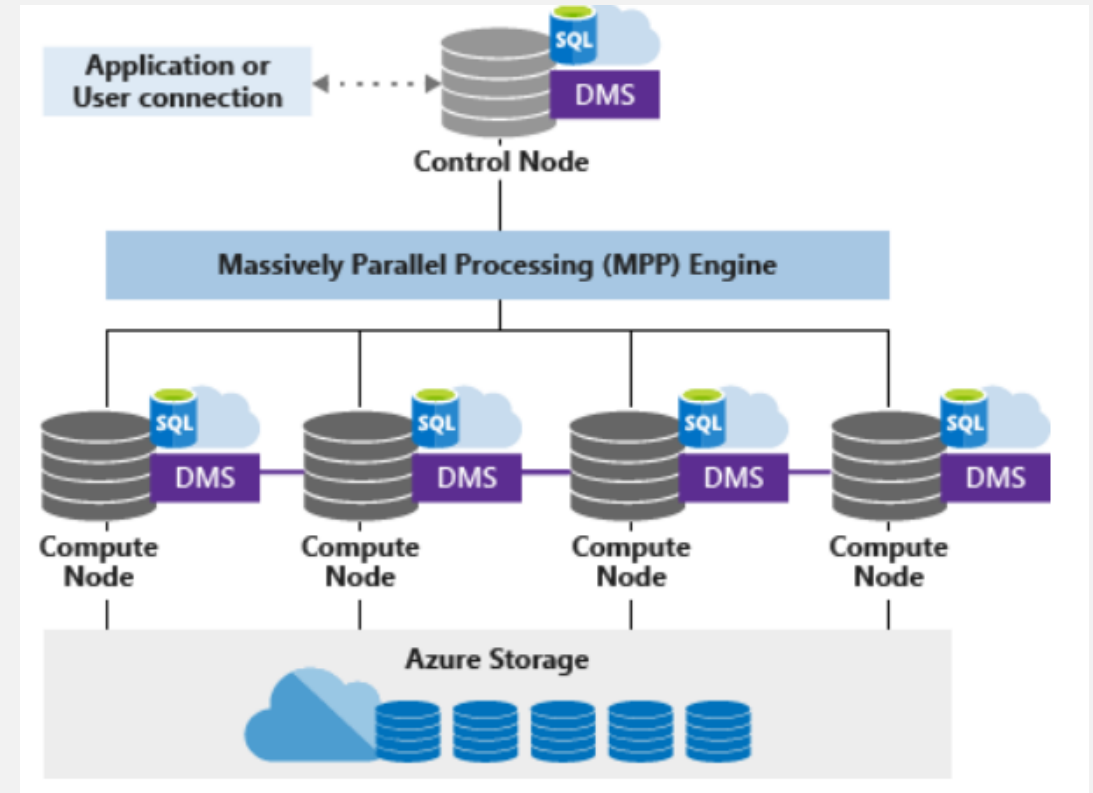
Azure SQL Data Warehouse Current release notes

<https://docs.microsoft.com/en-us/azure/sql-data-warehouse/release-notes-10-0-10106-0>

ARCHITECTURE - OVERVIEW

Decoupled Storage allows:

- Independently size compute power irrespective of your storage needs.
- Grow or shrink compute power without moving data.
- Pause compute capacity while leaving data intact, so you only pay for storage.
- Resume compute capacity during operational hours.

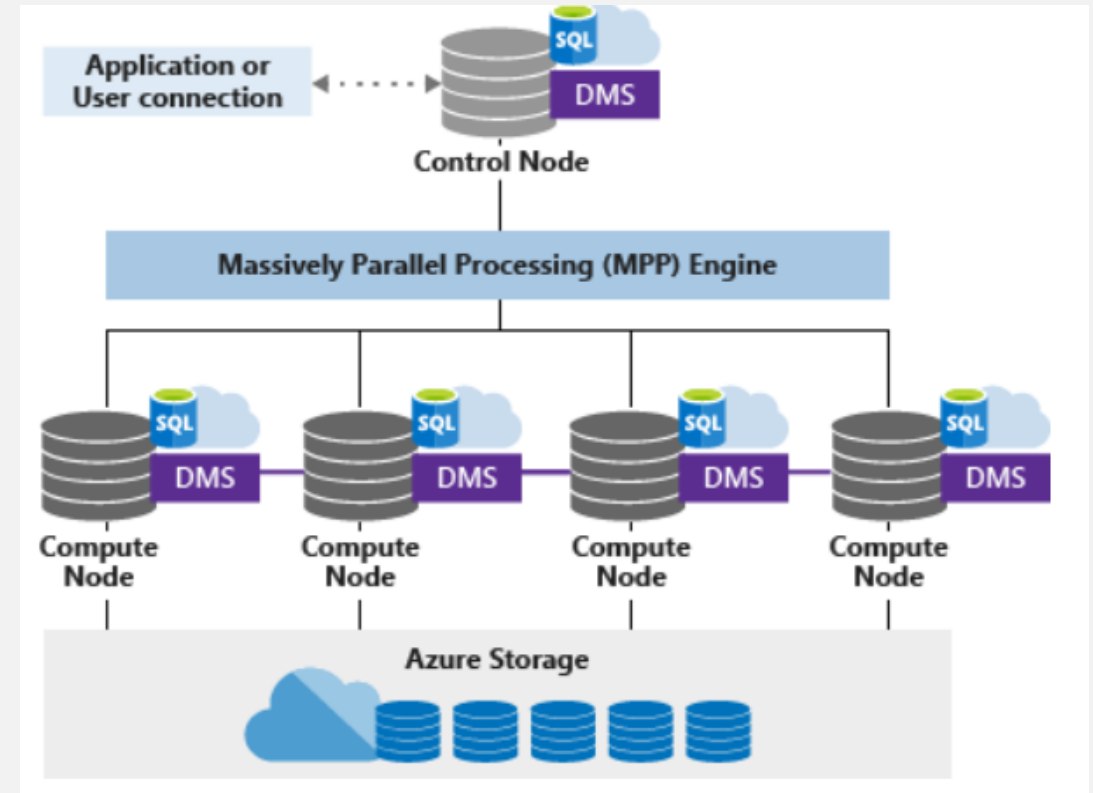


Reference - <https://docs.microsoft.com/en-us/azure/sql-data-warehouse/massively-parallel-processing-mpp-architecture>

ARCHITECTURE - OVERVIEW

Overview

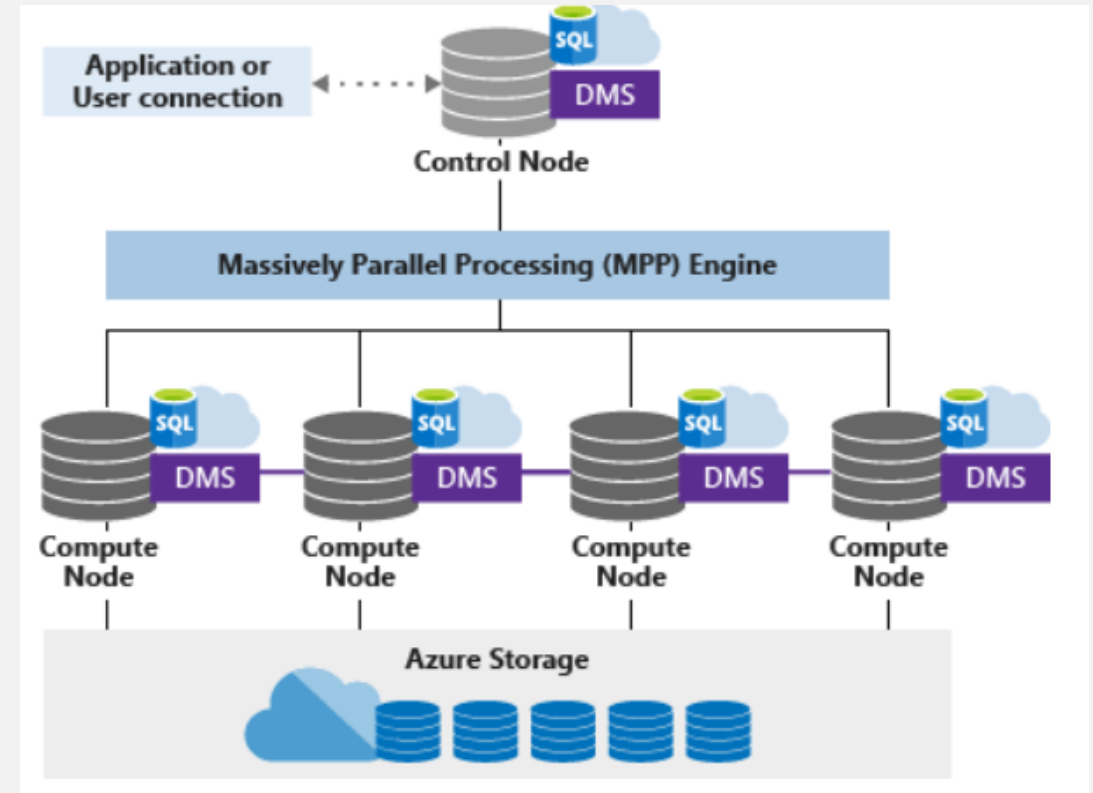
- SQL Data Warehouse uses a node-based architecture.
- Applications connect and issue T-SQL commands to a Control node, single point of entry
- The Control node runs the MPP engine which optimizes queries for parallel processing, and then passes operations to Compute nodes to do their work in parallel.
- The Compute nodes store all user data in Azure Storage and run the parallel queries.
- The Data Movement Service (DMS) moves data across the nodes as necessary to run queries in parallel and return accurate results.



ARCHITECTURE – AZURE STORAGE

Overview

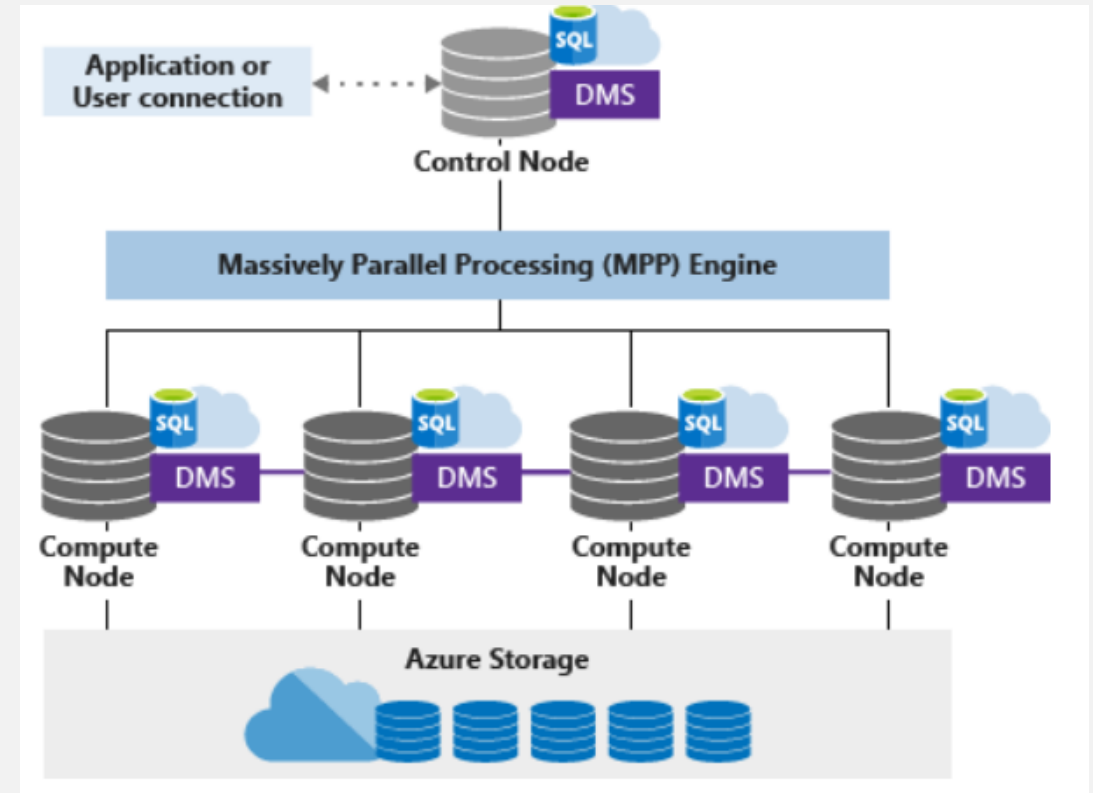
- Data is stored and managed by Azure storage
- The data itself is **sharded into distributions** to optimize the performance of the system.
- You can choose which sharding pattern to use to distribute the data when you define the table. SQL Data Warehouse supports these sharding patterns:
 - Hash
 - Round Robin
 - Replicate



ARCHITECTURE – CONTROL NODE

Overview

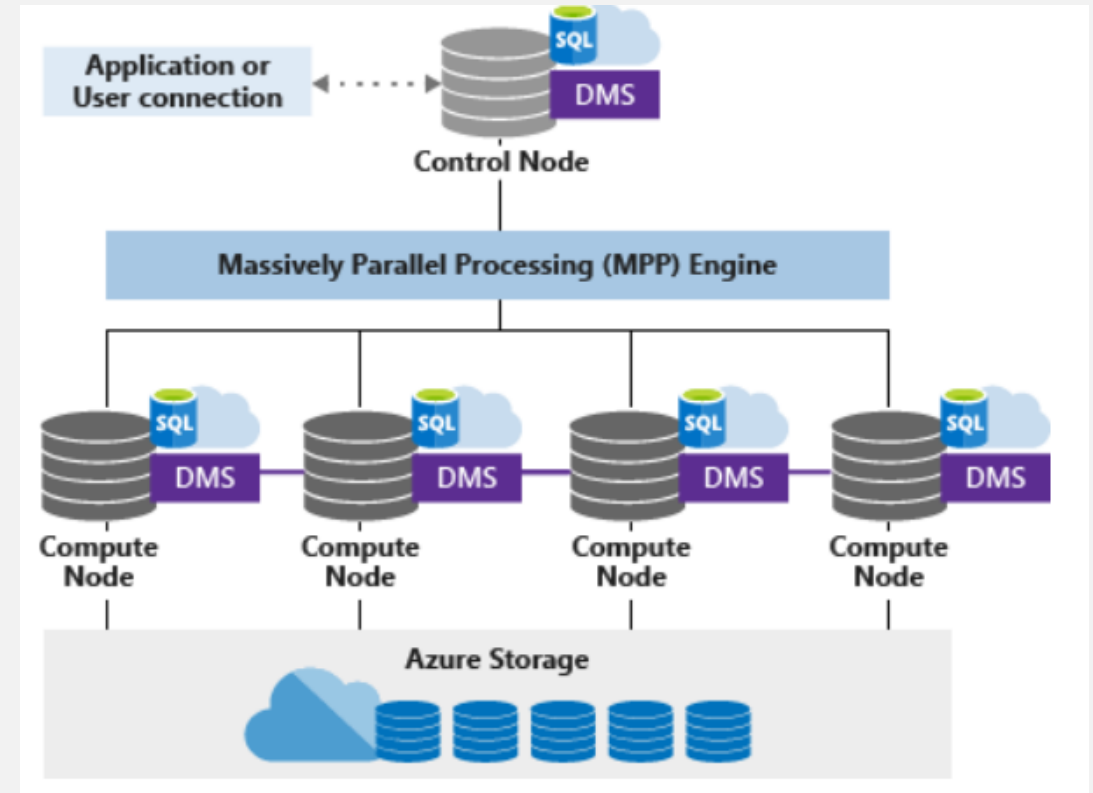
- The brain of the data warehouse
- The front end that interacts with all applications and connections.
- MPP engine runs on the Control node to optimize and coordinate parallel queries.
- The Control node transforms a submitted T-SQL query into queries that run against each distribution in parallel.



ARCHITECTURE – COMPUTE NODES

Overview

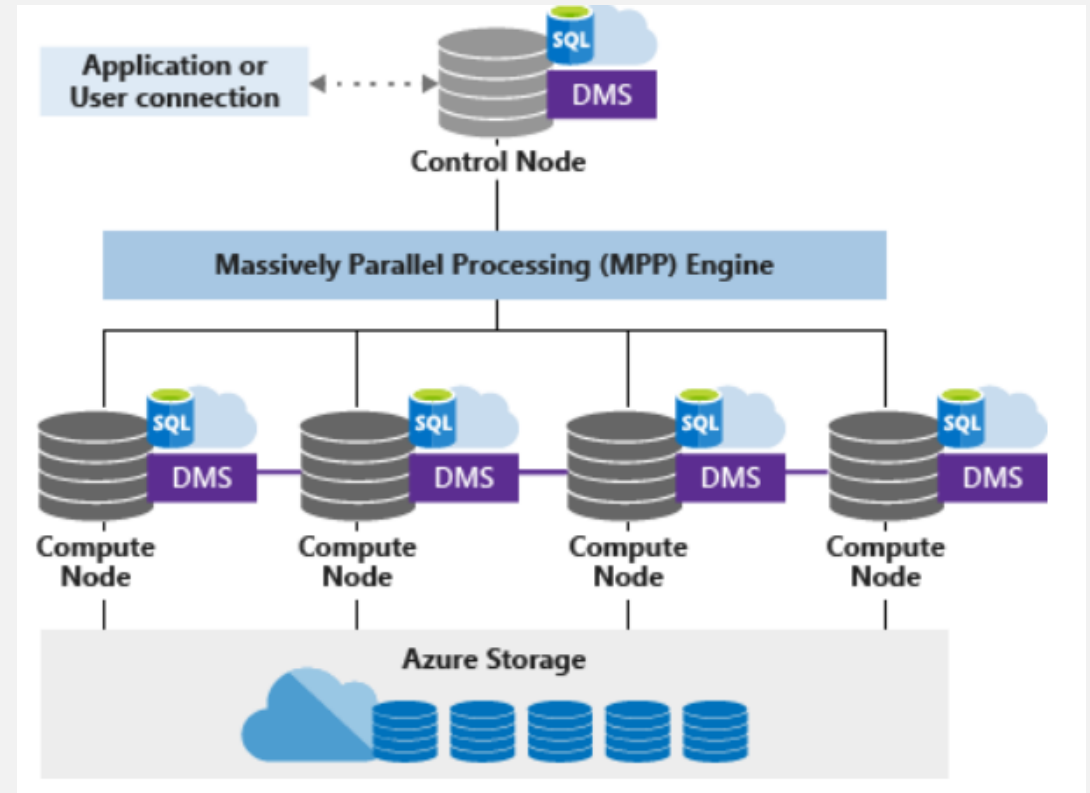
- Provide the computational power.
- Distributions map to Compute nodes for processing.
- Scale Up compute resources, SQL Data Warehouse re-maps the distributions to the available Compute nodes which ranges from 1 to 60
- Each Compute node has a node ID that is visible in system views. <https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sql-data-warehouse-and-parallel-data-warehouse-catalog-views?view=aps-pdw-2016-au7>



ARCHITECTURE – DATA MOVEMENT SERVICE

Overview

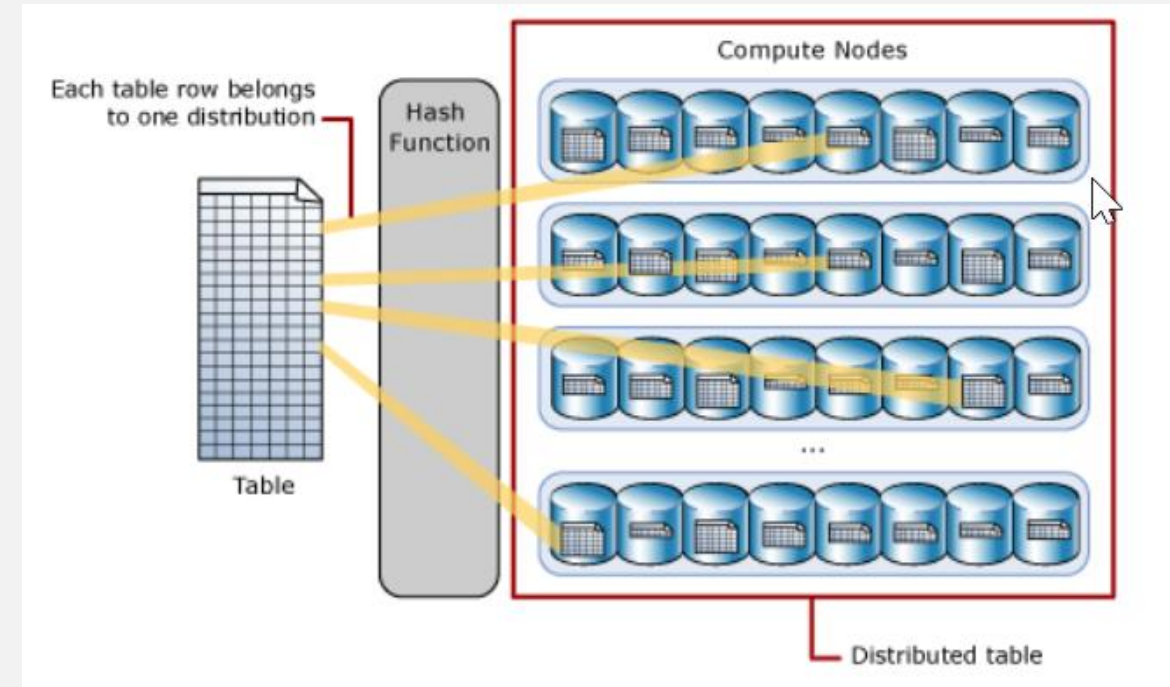
- The data transport technology that coordinates data movement between the Compute nodes.
- Some queries require data movement to ensure the parallel queries return accurate results.



ARCHITECTURE – TABLE DISTRIBUTIONS

Hash Table Distributions

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



ARCHITECTURE – TABLE DISTRIBUTIONS

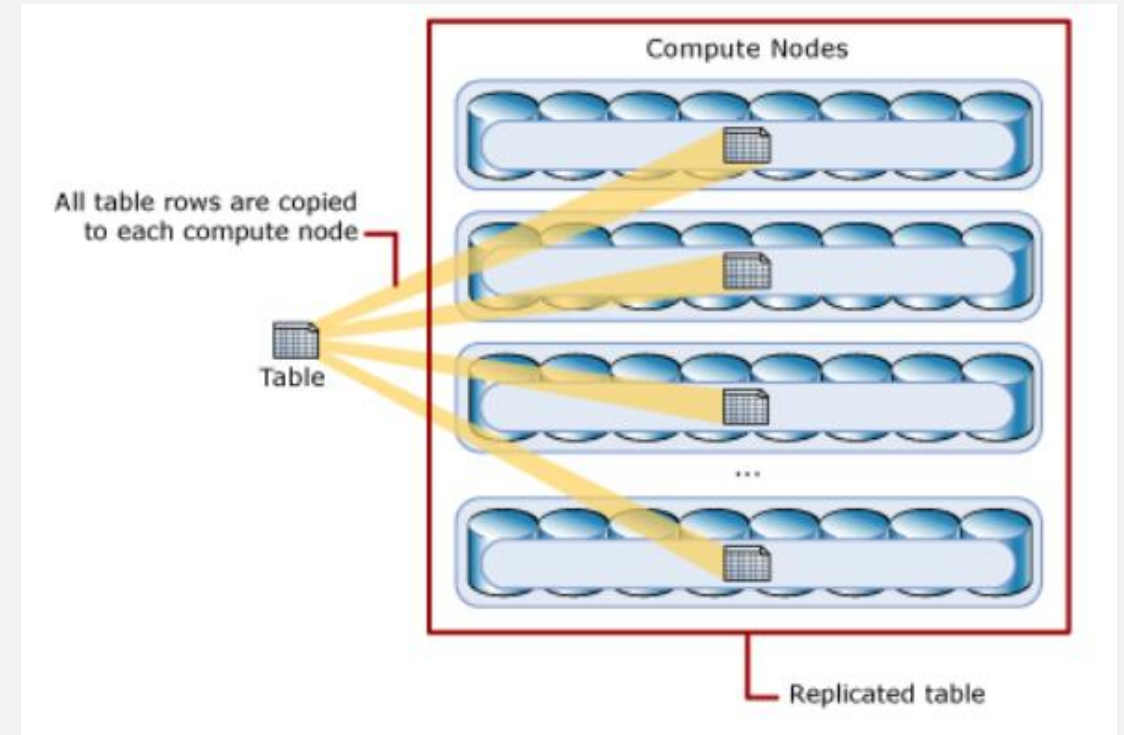
Round-robin distributed Table Distributions

- 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.
- Query performance can often be better with hash distributed tables.
- Joins on round-robin tables require reshuffling data and this takes additional time.

ARCHITECTURE – TABLE DISTRIBUTIONS

Round-robin distributed Table Distributions

- 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.

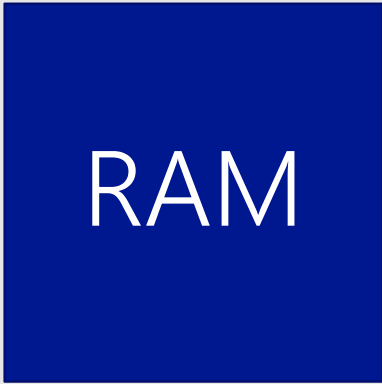


SQL Datawarehouse Architecture

Data Warehouse Units

Normalized amount of compute

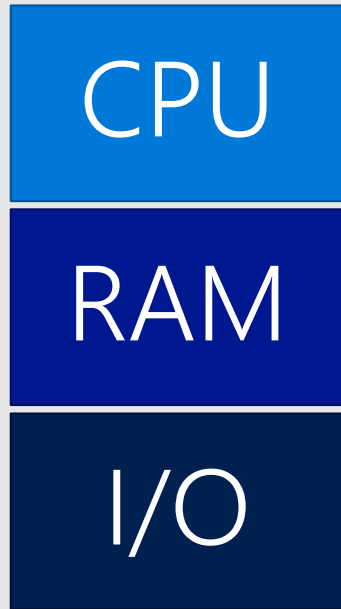
Converts to billing units i.e. what you pay



DWUc	Nodes	Dist/ Node
100	1	60
200	1	60
300	1	60
400	1	60
500	1	60
1000	2	30
1500	3	20
2000	4	15
2500	5	12
3000	6	10
5000	10	6
...		
30000	60	1

Data Warehouse Units

Normalized amount of compute
Converts to billing units
i.e. what you pay



Gen2 provides 2.5x more memory per query than the Gen1. This extra memory helps the Gen2 deliver its fast performance. The performance levels for the Gen2 range from DW100c to DW30000c.

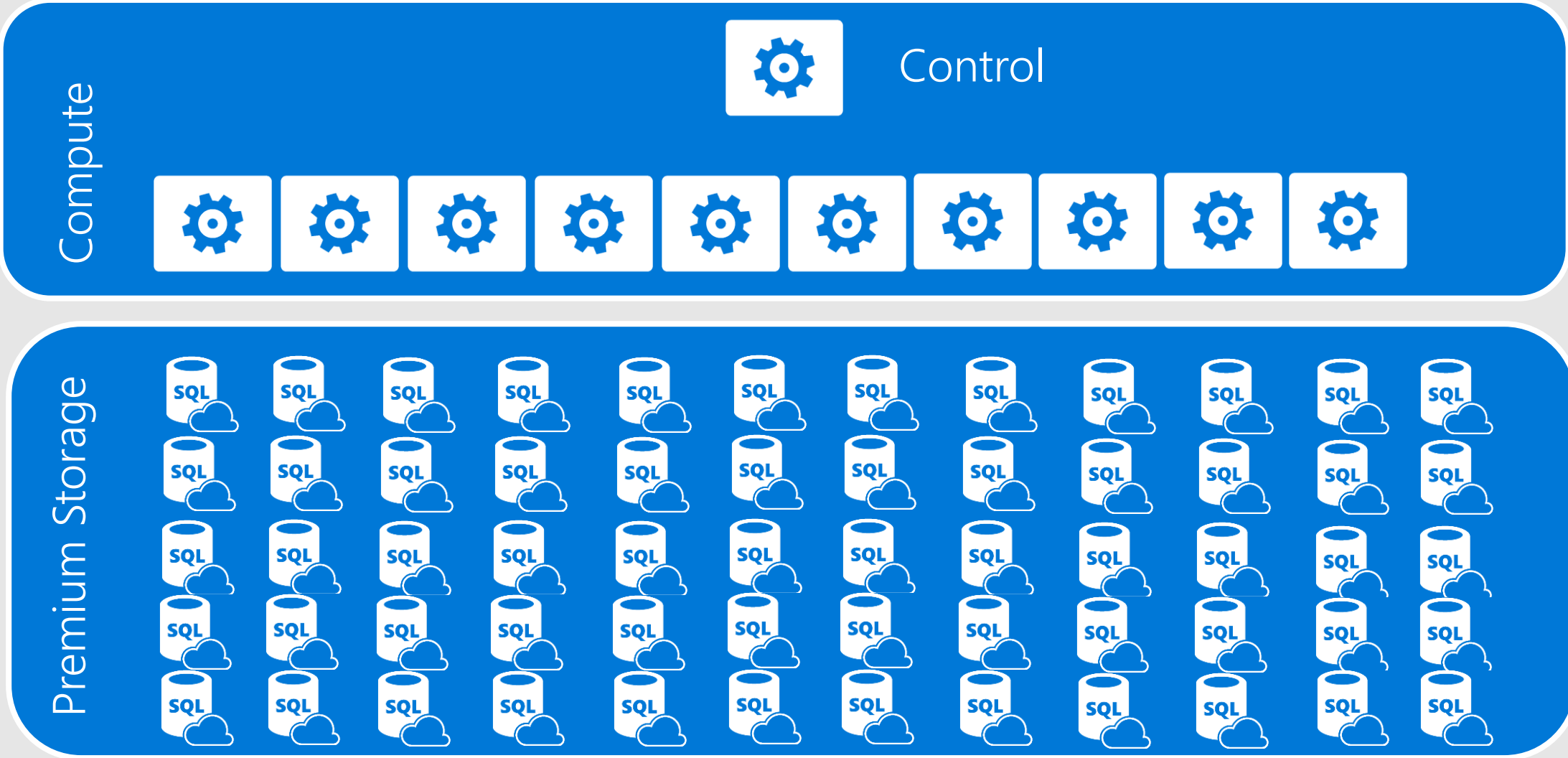
Performance level	Compute nodes	Distributions per Compute node	Memory per data warehouse (GB)
DW100c	1	60	60
DW200c	1	60	120
DW300c	1	60	180
DW400c	1	60	240
DW500c	1	60	300
DW1000c	2	30	600
DW1500c	3	20	900
DW2000c	4	15	1200
DW2500c	5	12	1500
DW3000c	6	10	1800
DW5000c	10	6	3000
DW6000c	12	5	3600
DW7500c	15	4	4500
DW10000c	20	3	6000
DW15000c	30	2	9000
DW30000c	60	1	18000

The maximum Gen2 DWU is DW30000c, which has 60 Compute nodes and one distribution per Compute node. For example, a 600 TB data warehouse at DW30000c processes approximately 10 TB per Compute node.

<https://docs.microsoft.com/en-us/azure/sql-data-warehouse/memory-and-concurrency-limits>

Important to note for Labs !!

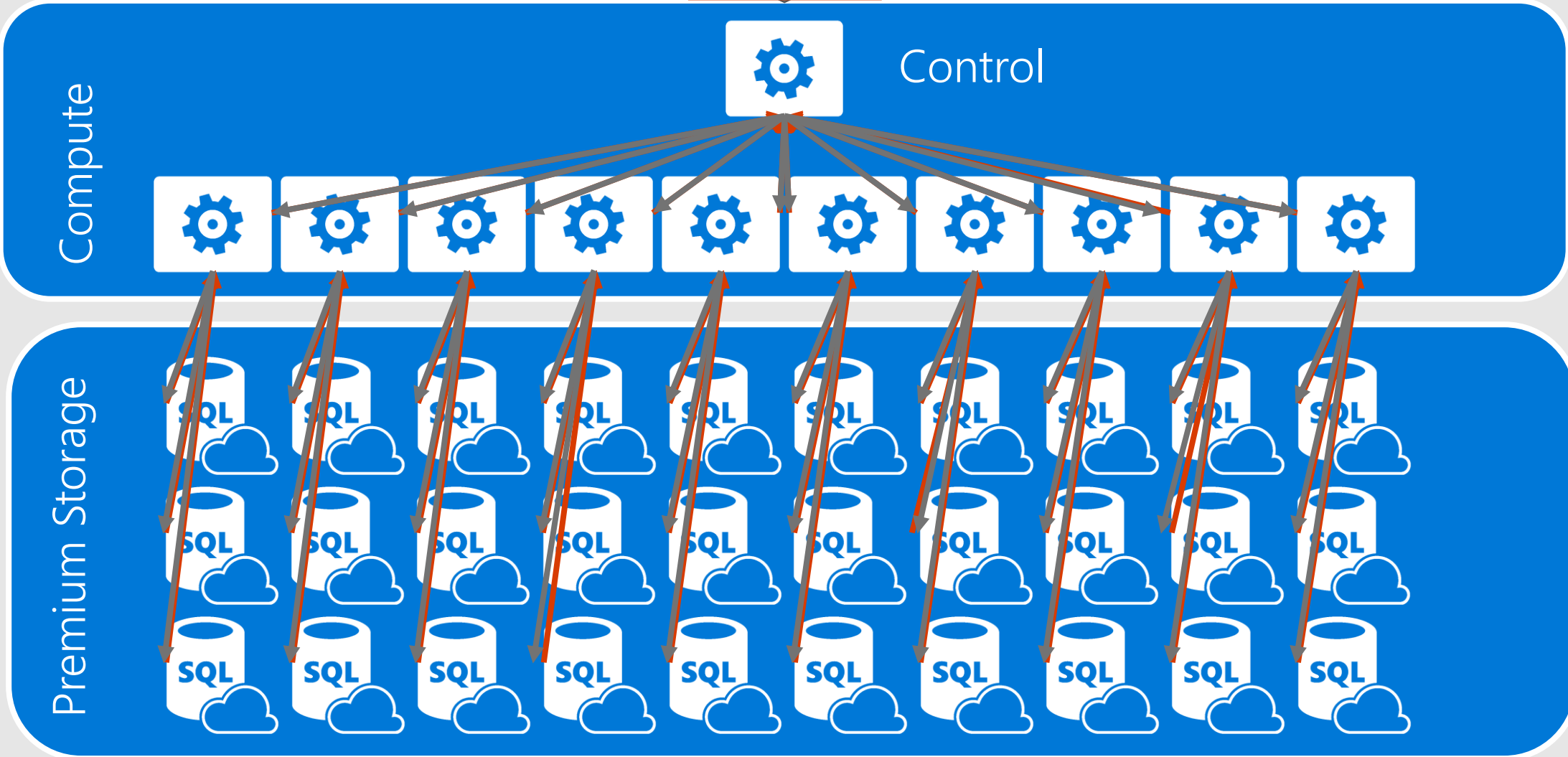
Separate compute from storage



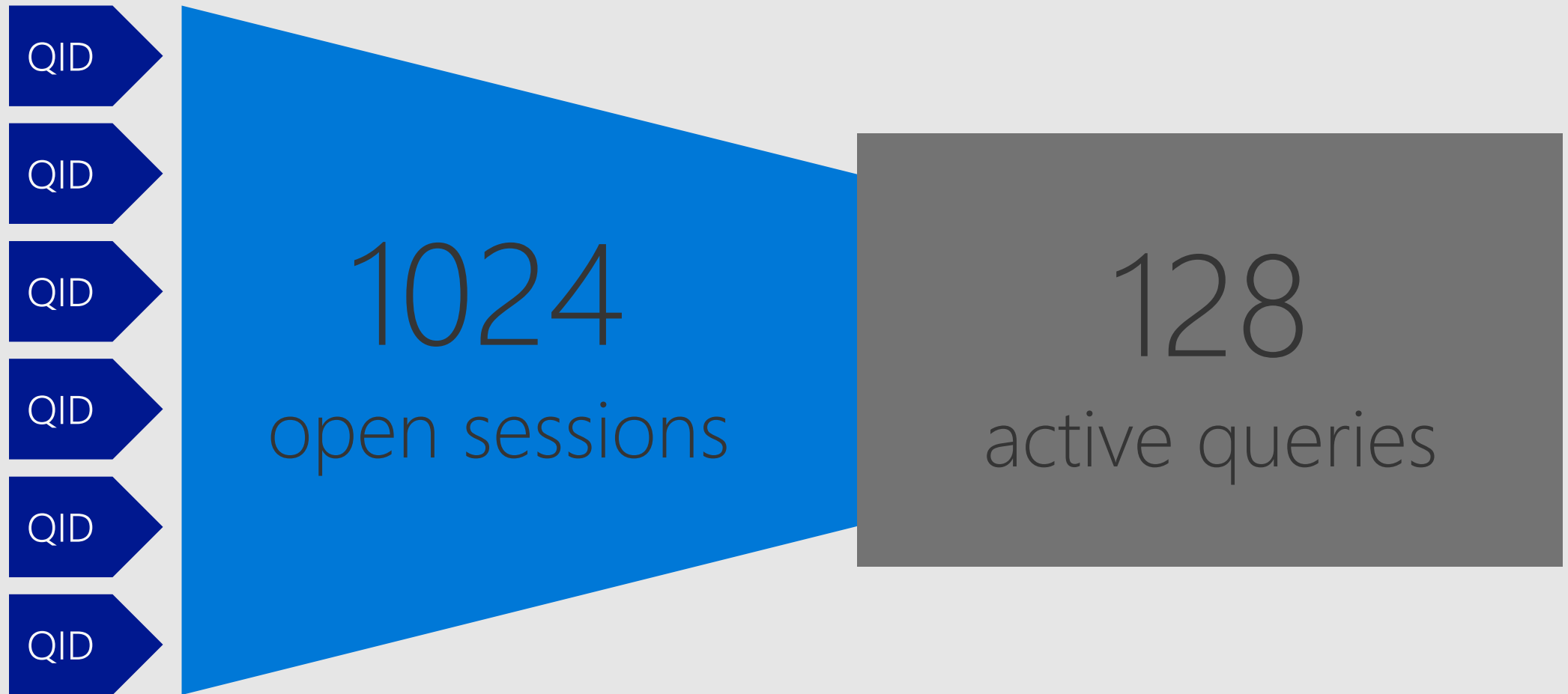
Query Execution



Result



Concurrent queries

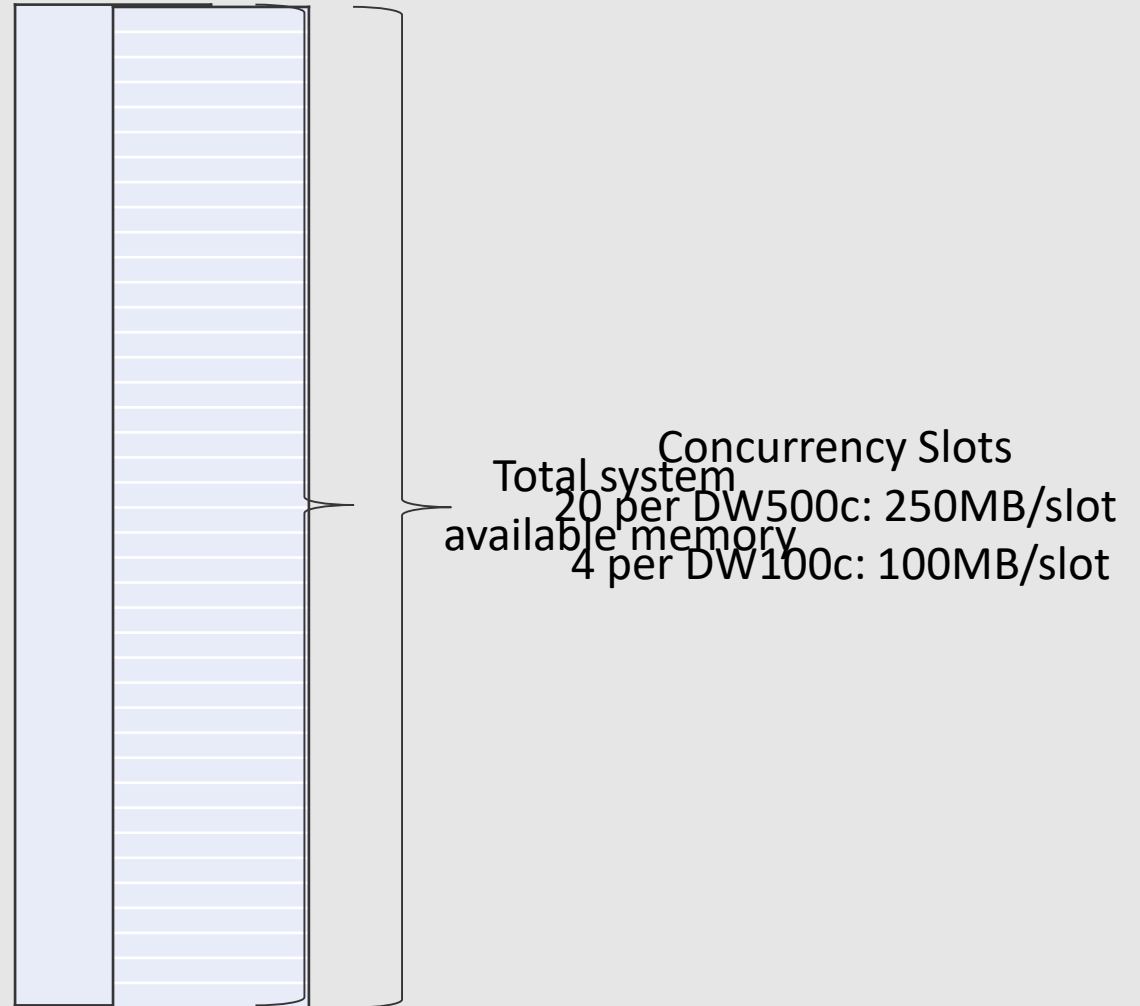


How many queries and slots are running in my Azure SQL Data Warehouse?

https://blogs.msdn.microsoft.com/sql_dw/2018/03/08/how-many-queries-and-slots-are-running-in-my-azure-sql-data-warehouse/

Statistics - <https://docs.microsoft.com/en-us/azure/sql-data-warehouse/sql-data-warehouse-tables-statistics>

Concurrency slots



Concurrent Query and Slots

Resource Class

Service Level	Maximum concurrent queries	Concurrency slots available	Slots used by staticrc10	Slots used by staticrc20	Slots used by staticrc30	Slots used by staticrc40	Slots used by staticrc50	Slots used by staticrc60
DW100c	4	4	1	2	4	4	4	4
DW200c	8	8	1	2	4	8	8	8
DW300c	12	12	1	2	4	8	8	8
DW400c	16	16	1	2	4	8	16	16
DW500c	20	20	1	2	4	8	16	16
DW1000c	32	40	1	2	4	8	16	32
DW1500c	32	60	1	2	4	8	16	32
DW2000c	48	80	1	2	4	8	16	32
DW2500c	48	100	1	2	4	8	16	32
DW3000c	64	120	1	2	4	8	16	32
DW5000c	64	200	1	2	4	8	16	32
DW6000c	128	240	1	2	4	8	16	32
DW7500c	128	300	1	2	4	8	16	32
DW10000c	128	400	1	2	4	8	16	32
DW15000c	128	600	1	2	4	8	16	32
DW30000c	128	1200	1	2	4	8	16	32



Memory and concurrency limits for Azure SQL Data Warehouse - <https://docs.microsoft.com/en-us/azure/sql-data-warehouse/memory-and-concurrency-limits>

Resource classes

Dynamic

Increases resource consumption as you scale

No increase in concurrency as you scale

Static

Maintain resource consumption as you scale

Increase concurrent queries as you scale

Consume Slots

Increase memory
Isolate resources

Resource Classes – Dynamic

Allocates variable amounts of memory depending on the scale of the DW instance.

- ✓ Beneficial for variable sized workloads that scale to meet demand.
There is no increase in concurrency with scaling.



Scaling up →

Dynamic Resource Classes

With Gen2, dynamic resource pools were introduced with a 3-10-22-70 model for resource allocations.

Resource Class	Percent Resources	Concurrency
SmallRc	3%	32
MediumRc	10%	10
LargeRc	22%	4
XLargeRc	70%	1

Resource Classes – Static

Allocates a fixed amount of memory regardless of the scale level.

- ✓ Essential for high query concurrency workloads.
Queries may run the same regardless of the service level.



Scaling up →

Gen 2 Concurrency – Static RC

Service Level	Maximum concurrent queries	Concurrency slots available	staticrc10	staticrc20	staticrc30	staticrc40	staticrc50	staticrc60	staticrc70	staticrc80
DW100c	4	4	1	2	4	4	4	4	4	4
DW200c	8	8	1	2	4	8	8	8	8	8
DW300c	12	12	1	2	4	8	8	8	8	8
DW400c	16	16	1	2	4	8	16	16	16	16
DW500c	20	20	1	2	4	8	16	16	16	16
DW1000c	32	40	1	2	4	8	16	32	32	32
DW1500c	32	60	1	2	4	8	16	32	32	32
DW2000c	48	80	1	2	4	8	16	32	64	64
DW2500c	48	100	1	2	4	8	16	32	64	64
DW3000c	64	120	1	2	4	8	16	32	64	64
DW5000c	64	200	1	2	4	8	16	32	64	128
DW6000c	128	240	1	2	4	8	16	32	64	128
DW7500c	128	300	1	2	4	8	16	32	64	128
DW10000c	128	400	1	2	4	8	16	32	64	128
DW15000c	128	600	1	2	4	8	16	32	64	128
DW30000c	128	1200	1	2	4	8	16	32	64	128

Data Distribution

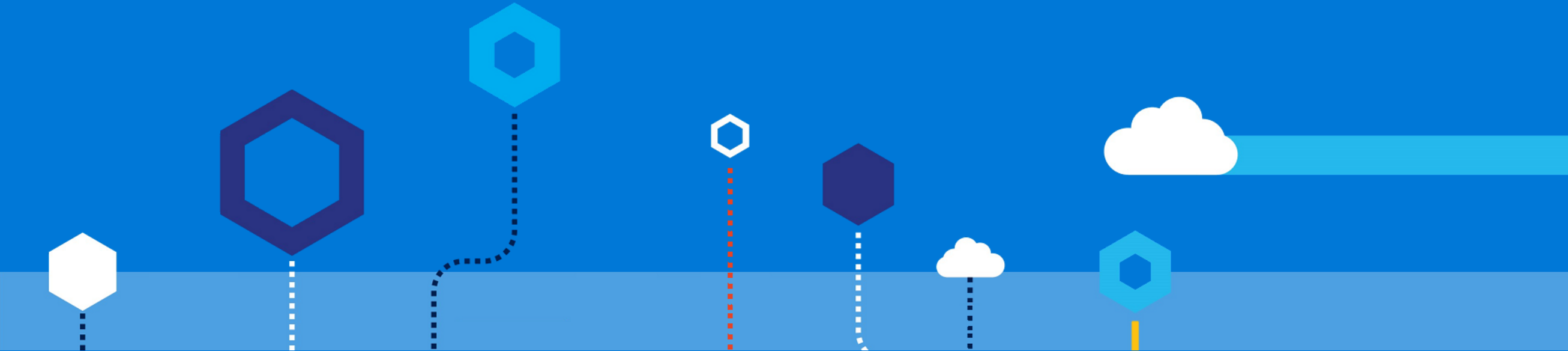


Table Distribution Options

Hash Distributed

Data divided across nodes based on hashing algorithm

Same value will always hash to same distribution

Single column only

Check for Data Skew,
NULLS, -1

Round Robin (Default)

Data distributed evenly across nodes

Easy place to start, don't need to know anything about the data

Simplicity at a cost

Will incur more data movement at query time

Replicated

Data repeated on every node

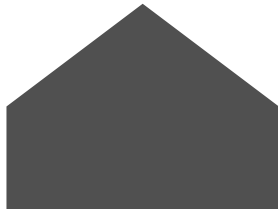
Simplifies many query plans and reduces data movement

Best with joining hash table

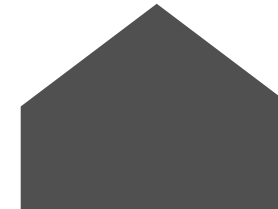
Consumes more space
Joining two Replicated
Table runs
on one node

Creating tables

```
CREATE TABLE [build].[FactOnlineSales]
(
    [OnlineSalesKey]      int          NOT NULL
,   [DateKey]            datetime     NOT NULL
,   [StoreKey]           int          NOT NULL
,   [ProductKey]         int          NOT NULL
,   [PromotionKey]       int          NOT NULL
,   [CurrencyKey]       int          NOT NULL
,   [CustomerKey]        int          NOT NULL
,   [SalesOrderNumber]   nvarchar(20) NOT NULL
,   [SalesOrderLineNumber] int        NULL
,   [SalesQuantity]      int          NOT NULL
,   [SalesAmount]        money        NOT NULL
)
WITH
( CLUSTERED COLUMNSTORE INDEX
,  DISTRIBUTION = ROUND_ROBIN
)
;
```



```
CREATE TABLE [build].[FactOnlineSales]
(
    [OnlineSalesKey]      int          NOT NULL
,   [DateKey]            datetime     NOT NULL
,   [StoreKey]           int          NOT NULL
,   [ProductKey]         int          NOT NULL
,   [PromotionKey]       int          NOT NULL
,   [CurrencyKey]       int          NOT NULL
,   [CustomerKey]        int          NOT NULL
,   [SalesOrderNumber]   nvarchar(20) NOT NULL
,   [SalesOrderLineNumber] int        NULL
,   [SalesQuantity]      int          NOT NULL
,   [SalesAmount]        money        NOT NULL
)
WITH
( CLUSTERED COLUMNSTORE INDEX
,  DISTRIBUTION = HASH([ProductKey])
)
;
```



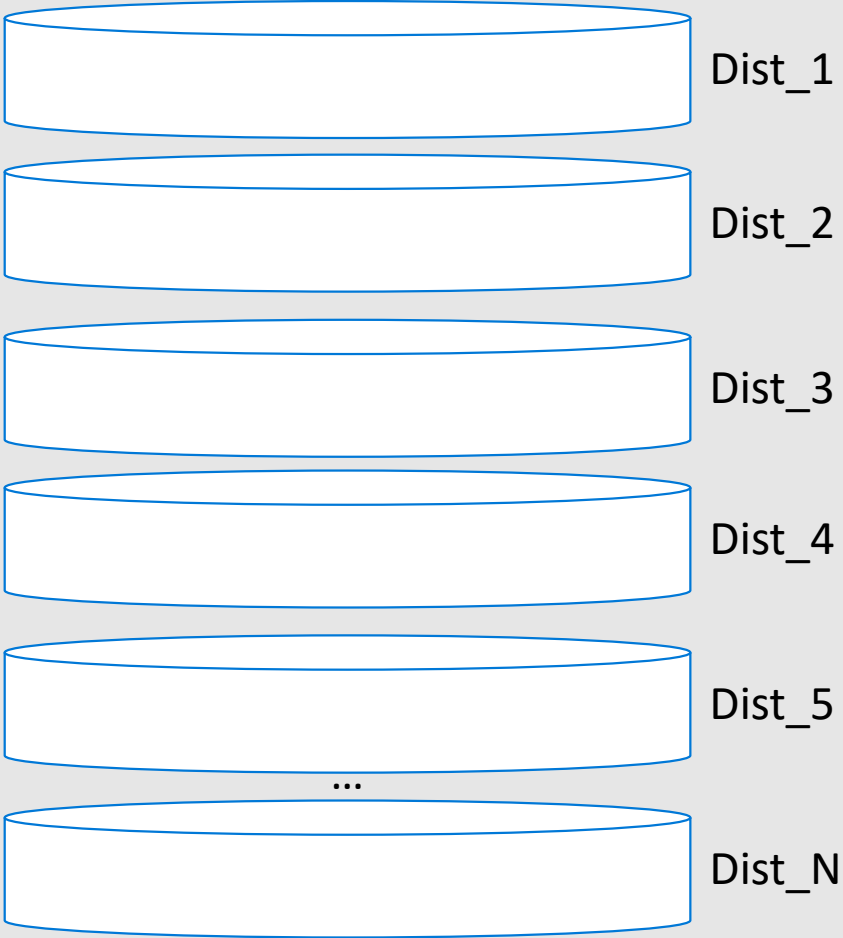
Hash Distributed

```
CREATE TABLE ProductSales
WITH (DISTRIBUTION=HASH(AccountID))
AS...
```

ProductSales – Raw Data

AccountID	SalesAmt	...
47	\$1,234.36	...
36	\$2,345.47	...
14	\$3,456.58	...
25	\$4,567.69	...
48	\$5,678.70	...
37	\$6,789.81	...
...

Hash(36)

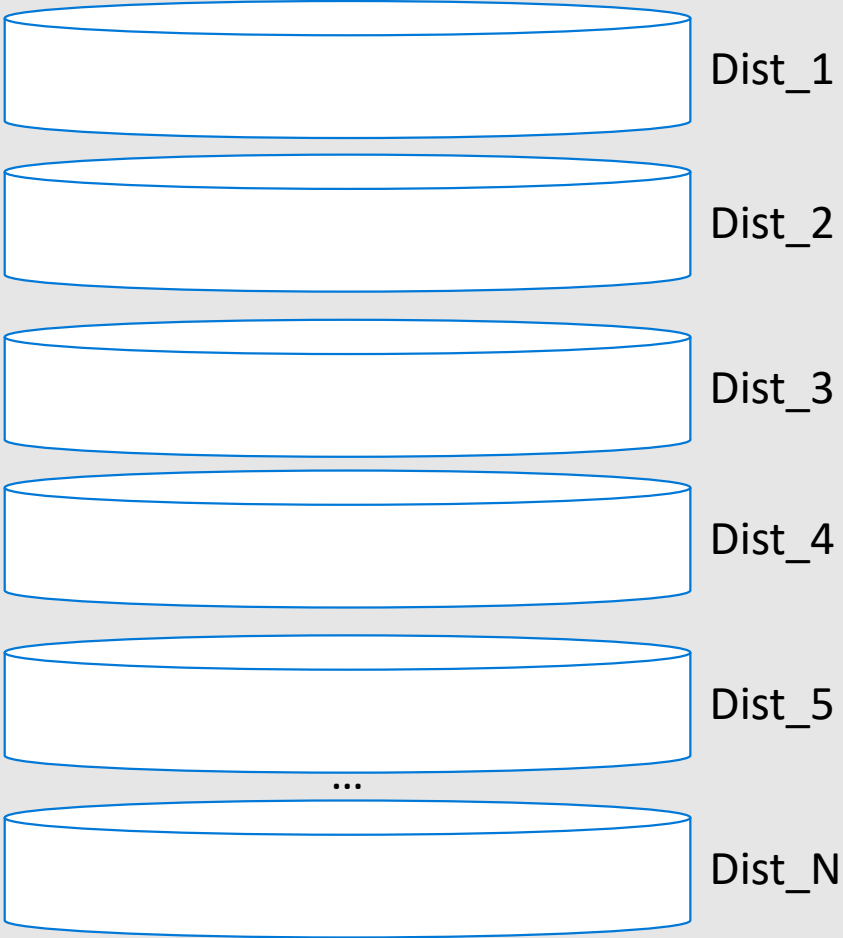


Round Robin Distributed

```
CREATE TABLE ProductSales
WITH (DISTRIBUTION = ROUND_ROBIN)
AS ...
```

ProductSales – Raw Data

AccountID	SalesAmt	...
47	\$1,234.36	...
36	\$2,345.47	...
14	\$3,456.58	...
25	\$4,567.69	...
48	\$5,678.70	...
37	\$6,789.81	...
42	\$1,632.25	...
42	\$4,453.21	...
52	\$7,892.81	...
91	\$9,549.64	...
66	\$2,498.14	...
23	\$3,145.99	...



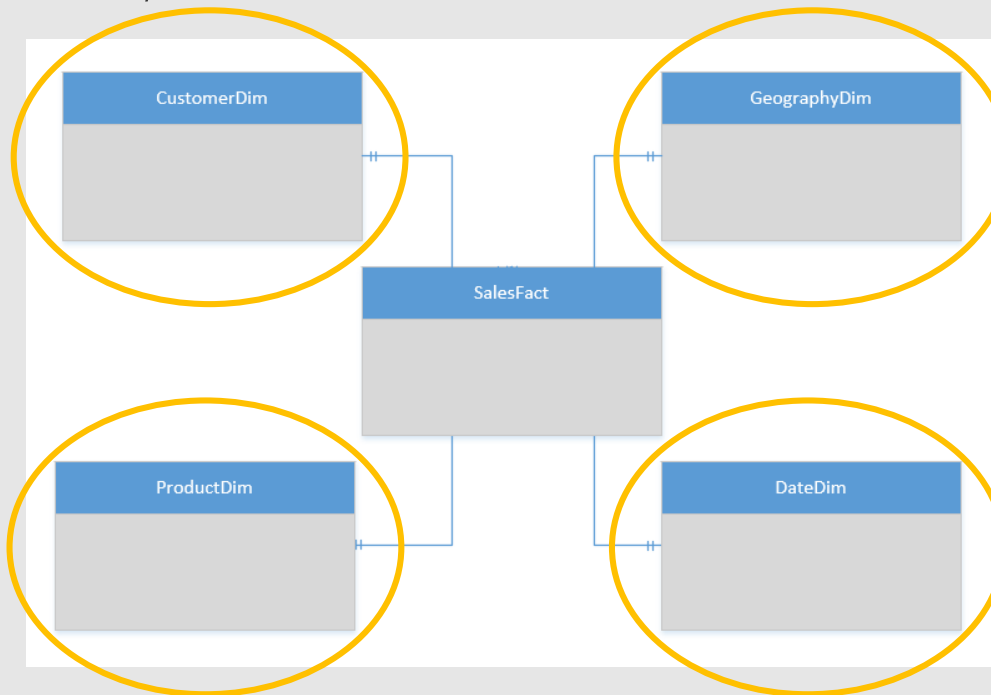
Replicated Table Scenarios

Scenarios to consider using Replicated tables:

Star schema reporting

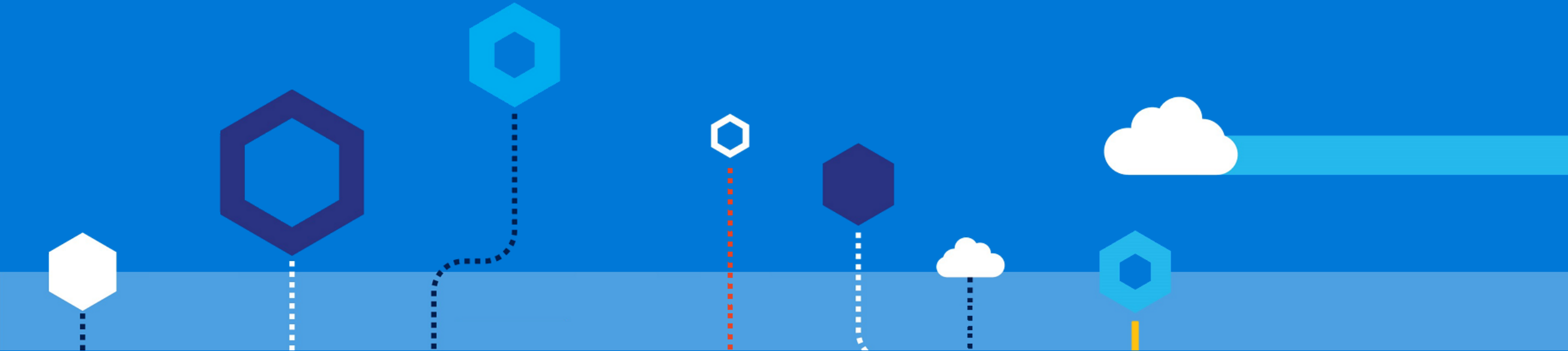
Dimensions – descriptive entities about fact data

ETL master data, common domain data used during transaction loading



```
CREATE TABLE dbo.DimCustomer
(
    CustomerKey      int      NOT NULL
    , GeographyKey   int      NULL
)
WITH
(
    CLUSTERED COLUMNSTORE INDEX
    , DISTRIBUTION = REPLICATED
)
```

Indexing



Indexing Choices

- Row Store : Clustered Index
- Column Store : Clustered Columnstore Index (CCI)
- Heap : No index
- Non-Clustered Index

Indexing tables

```
CREATE TABLE [dbo].[DimStore]
```

```
(  
    [StoreKey]          int          NOT NULL  
, [GeographyKey]       int          NOT NULL  
, [StoreName]          nvarchar(100) NOT NULL  
, [StoreType]          nvarchar(15)  NULL  
, [StoreDescription]   nvarchar(300) NOT NULL  
, [Status]             nvarchar(20)  NOT NULL  
, [OpenDate]           datetime     NOT NULL  
, [CloseDate]          datetime     NULL  
, [ETLLoadID]          int          NULL  
, [LoadDate]           datetime     NULL  
, [UpdateDate]         datetime     NULL  
)
```

```
WITH
```

```
( CLUSTERED INDEX([StoreKey])  
  DISTRIBUTION = ROUND_ROBIN  
)  
;
```

Row

```
CREATE TABLE [dbo].[FactOnlineSales]
```

```
(  
    [OnlineSalesKey]    int          NOT NULL  
, [DateKey]            datetime     NOT NULL  
, [StoreKey]           int          NOT NULL  
, [ProductKey]         int          NOT NULL  
, [PromotionKey]       int          NOT NULL  
, [CurrencyKey]        int          NOT NULL  
, [CustomerKey]        int          NOT NULL  
, [SalesOrderNumber]   nvarchar(20) NOT NULL  
, [SalesOrderLineNumber] int        NULL  
, [SalesQuantity]      int          NOT NULL  
, [SalesAmount]        money        NOT NULL  
)
```

```
WITH
```

```
( CLUSTERED COLUMNSTORE INDEX  
  DISTRIBUTION = HASH([ProductKey])  
)  
;
```

Column

Distribution

Clustered Columnstore & Partitioning

```
CREATE TABLE [dbo].[FactInternetSales]
(
    [ProductKey] int NOT NULL ,
    [OrderDateKey] int NOT NULL ,
    [CustomerKey] int NOT NULL ,
    [PromotionKey] int NOT NULL ,
    [SalesOrderNumber] nvarchar(20) NOT NULL ,
    [OrderQuantity] smallint NOT NULL ,
    [UnitPrice] money NOT NULL ,
    [SalesAmount] money NOT NULL
)
WITH ( CLUSTERED COLUMNSTORE INDEX ,
        DISTRIBUTION = HASH([ProductKey]) ,
        PARTITION ( [OrderDateKey] RANGE RIGHT FOR
VALUES (20000101,20010101,20020101,20030101,20040101,20050101)
        )
)
;
```

Column Store vs Row Store - Definitions

Columnstore

A columnstore is data that's logically organized as a table with rows and columns, and physically stored in a column-wise data format.

Rowstore

A rowstore is data that's logically organized as a table with rows and columns, and **physically stored in a row-wise data format**. This format is the traditional way to store relational table data.

In SQL Server, rowstore refers to a table where the underlying data storage format is a heap, a clustered index, or a memory-optimized table. (Everything Else)

Rowgroup

A rowgroup is a group of rows that are compressed into columnstore format at the same time. A rowgroup usually contains the maximum number of rows per rowgroup, which is 1,048,576 rows.

Delta rowgroup

A delta rowgroup is a clustered index that's used only with columnstore indexes. It improves columnstore compression and performance by storing rows until the number of rows reaches a threshold and are then moved into the columnstore.

Column Store vs Row Store

Why should I use a columnstore index?

A columnstore index can provide a very high level of data compression, typically by 10 times, to significantly reduce your data warehouse storage cost.

Reasons why columnstore indexes are so fast:

- Columns store values from the same domain and commonly have similar values, which result in high compression rates. I/O bottlenecks in your system are minimized or eliminated, and memory footprint is reduced significantly.
- High compression rates improve query performance by using a smaller in-memory footprint. In turn, query performance can improve because SQL Server can perform more query and data operations in memory.
- Batch execution improves query performance, typically by two to four times, by processing multiple rows together.
- Queries often select only a few columns from a table, which reduces total I/O from the physical media.

Column Store vs Row Store

When should I use a columnstore index?

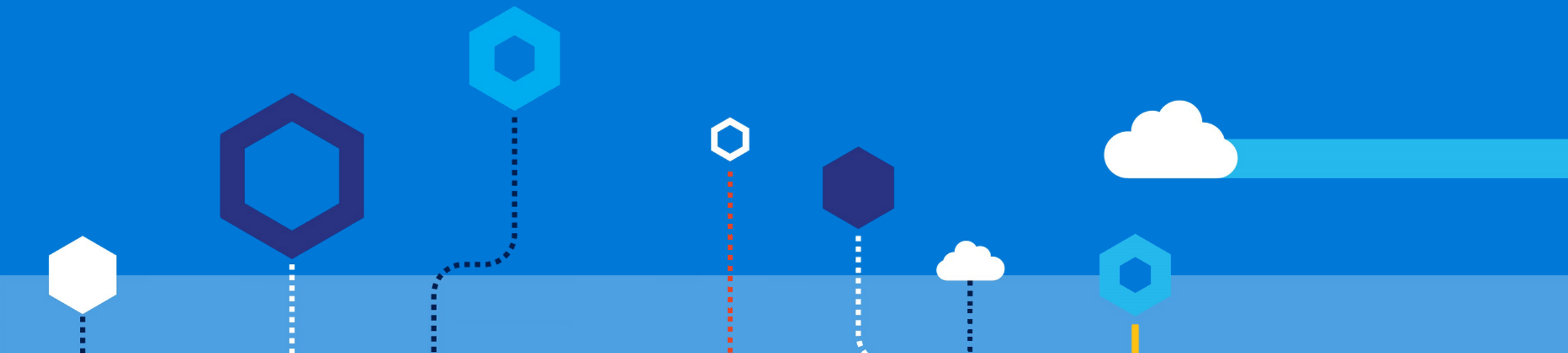
Recommended use cases:

- Use a clustered columnstore index to store fact tables and large dimension tables for data warehousing workloads
- Improves query performance and data compression by up to 10 times
- Use a nonclustered columnstore index to perform analysis in real time on an OLTP workload.

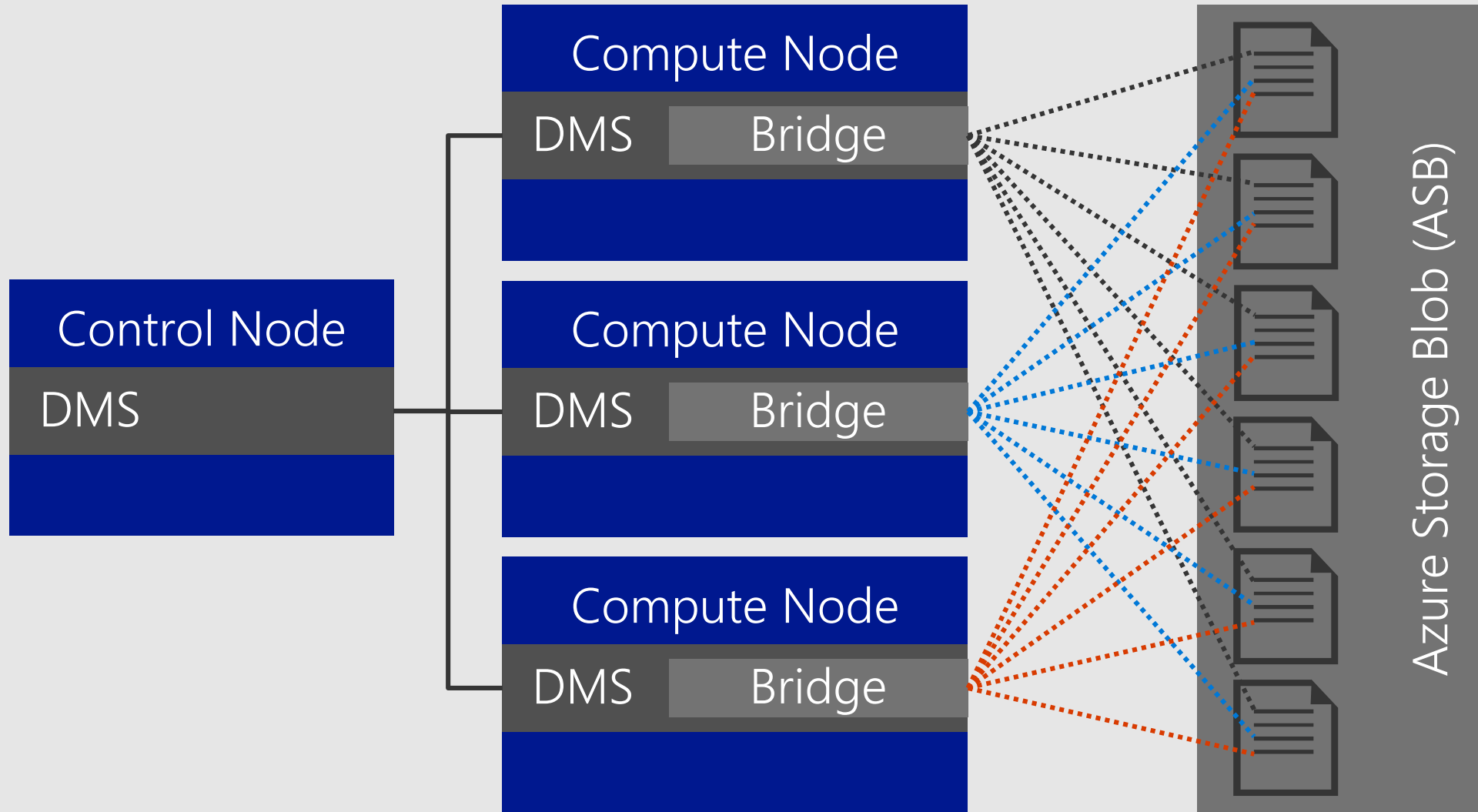
How do I choose between a rowstore index and a columnstore index?

- Rowstore indexes perform best on queries that seek into the data
- When searching for a particular value, or for queries on a small range of values.
- With transactional workloads because they tend to require mostly table seeks instead of table scans.
- Columnstore indexes give high performance gains for analytic queries that scan large amounts of data, especially on large tables. Use columnstore indexes on data warehousing and analytics workloads, especially on fact tables, because they tend to require full table scans rather than table seeks.

Polybase



Polybase parallel load from Azure Storage



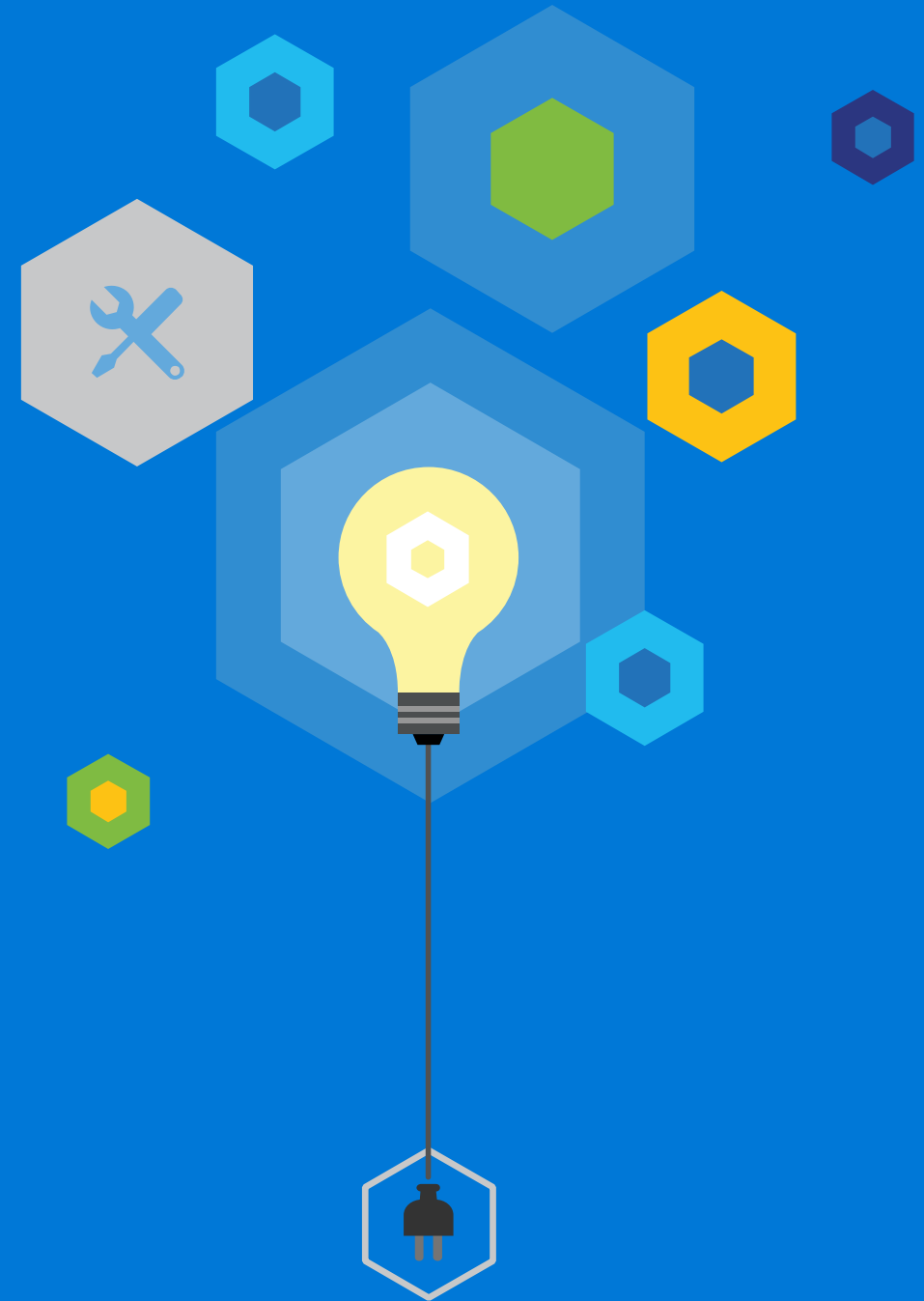
Best practices and considerations when using PolyBase

Here are a few more things to consider when using PolyBase for SQL Data Warehouse loads:

- A single PolyBase load operation provides best performance.
- The load performance scales as you increase DWUs.
- PolyBase automatically parallelizes the data load process, so you don't need to explicitly break the input data into multiple files and issue concurrent loads, unlike some traditional loading practices. Each reader automatically read 512MB for each file for Azure Storage BLOB and 256MB on Azure Data Lake Storage.
- Multiple readers will not work against gzip files. Only a single reader is used per gzip compressed file since uncompressing the file in the buffer is single threaded. Alternatively, generate multiple gzip files. The number of files should be greater than or equal to the total number of readers.
- Multiple readers will work against compressed columnar/block format files (e.g. ORC, RC) since individual blocks are compressed independently.

Demo – Polybase examples

C:\Users\styoung\Desktop\Extra_Credit_CreateConnectivity_B
lob_Export_DW_AsExternal_Tables.sql



Q&A

